LESŁAW SIENIAWSKI

Oficyna Wydawnicza
Państwowej Wyższej Szkoły Zawodowej
w Nysie

PAŃSTWOWA WYŻSZA SZKOŁA ZAWODOWA W NYSIE
SKRYPT NR 11

Lesław Sieniawski

SYSTEM *N*X

OFICYNA WYDAWNICZA PWSZ W NYSIE
NYSA 2006

SEKRETARZ OFICYNY:
Tomasz Drewniak

RECENZENT:
Adam Grzech

PROJEKT GRAFICZNY OKŁADKI:
Ryszard Szymończyk

SKŁAD I ŁAMANIE:
Lesław Sieniawski

KOREKTA I ADJUSTACJA:
Konrad Szcześniak

ISBN 83 - 60081 - 09 - 3

OFICYNA WYDAWNICZA PWSZ W NYSIE
48-300 Nysa, ul. Grodzka 19
Tel.: (077) 409 08 55
e-mail: oficyna@pwsz.nysa.pl
http://www.pwsz.nysa.pl/oficyna

Wydanie I

Druk i oprawa: SOWA - druk na życzenie
tel. 22 431 81 40

Spis treści

Przedmowa

Celem niniejszego opracowania jest przedstawienie podstawowych właściwości użytkowych systemów operacyjnych z rodziny UNIX, ich funkcji i sposobu użytkowania.

Podręcznik ten przeznaczony jest zwłaszcza dla początkujących i średnio zaawansowanych użytkowników systemu operacyjnego UNIX oraz systemów pochodnych, tj. wszystkich tych, którzy chcą świadomie i skutecznie korzystać z systemu komputerowego wyposażonego w UNIX lub odpowiednik, a zwłaszcza nim zarządzać, ale nie są zaangażowani w jego tworzenie lub modyfikację. Omawiana jest wyłącznie praca w trybie tekstowym, który stanowi podstawową formę kontaktu administratorów z systemem tego typu. Zdaniem autora, po opanowaniu niełatwej sztuki pracy w środowisku tekstowym UNIX-a, korzystanie z graficznego interfejsu użytkownika nie powinno stanowić większego problemu.

Szczególną uwagę zwrócono na systemy klasy Linux, które ilościowo dominują w klasie systemów UNIX-opodobnych i są łatwo dostępne dla każdego posiadacza komputera. Zamieszczone przykłady wykonano w systemie Linux z rodziny RedHat, najpopularniejszej i jednej z najbardziej przystępnych dystrybucji systemu Linux, zarazem jednej z nielicznych instytucjonalnie wspieranych przez producenta. Opis podstawowych właściwości użytkowych uzupełniają przykłady zastosowań. Przy definiowaniu pojęć podana została terminologia w języku angielskim, który jest językiem informatyki. Przy opracowywaniu tekstu autor wykorzystał własne materiały przygotowane do wykładu pt. "System UNIX". Treść podręcznika uzupełnia dołączona płyta CD-ROM.

Autor pozostał przy klasycznej pisowni nazw systemów: *UNIX* i *Linux*. Czytelnik powinien jednak zwrócić uwagę na coraz powszechniejsze stosowanie spolszczeń: *Uniks* i *Linuks oraz uniks i linuks*. Konsekwencją stosowania konkretnej konwencji pisowni jest zapis form fleksyjnych i pochodnych: *UNIX-a*/*UNIX-owy* i *Linux-a*/*Linux-owy* bądź *uniksa*/*uniksowy* i *linuksa*/*linuksowy*.

Zakłada się, że Czytelnik posiada podstawową znajomość pojęć z zakresu systemów komputerowych, w tym ogólnych zasad budowy i funkcjonowania systemów operacyjnych oraz praktyczną znajomość systemu operacyjnego klasy MS Windows.

Reguły typograficzne

Dla ułatwienia interpretacji treści tego opracowania zastosowano szereg konwencji notacyjnych, z których podstawowe zilustrowano tabelami 1. i 2.

Tabela 1. Reguły typograficzne

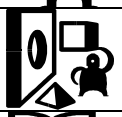
Krój czcionki	Zastosowanie	Przykład użycia
Times Roman, prosty	Zwykły tekst	To jest zwykły tekst.
Times Roman, pochylony	Definiowany termin, objaśnienie nazwy obcojęzycznej	Określenie *komputer* (z ang. *computer*) weszło na stałe do języka polskiego w końcu lat osiemdziesiątych XX wieku.
Times Roman, pogrubiony	Nazwy klawiszy, kombinacje klawiszowe	**<Ctrl>**
`Courier, prosty`	Tekst wprowadzany przez użytkownika lub wyprowadzany przez system i aplikacje	`ls -la`
`Courier, pochylony`	Elementy składniowe definicji poleceń, stanowiące parametry poleceń	`groupdel` *`nazwa_grupy`*

Tabela 2. Inne oznaczenia stosowane w tekście

Symbol	Zastosowanie	Przykład użycia
z	Wizualne wyróżnienie znaku **z**	Pojedyncza kropka . wskazuje katalog bieżący.
↵	Wizualne oznaczenie przeniesienia fragmentu tekstu drukowanego do następnego wiersza (nie dotyczy wprowadzania poleceń z konsoli systemu)	passwd ↵ [*nazwa_użytkownika*]

W niektórych miejscach tekstu zostały umieszczone tzw. *ramki*, stanowiące samodzielne akapity pośrednio wiążące się z tekstem poprzedzającym lub następującym po danej ramce. Rodzaj zawartości ramki oznaczany jest symbolem graficznym, według klasyfikacji przedstawionej w tabeli 3.

Tabela 3. Klasyfikacja ramek w tekście

Symbol	Rodzaj zawartości ramki
	Przykłady
	Zadania i problemy do samodzielnego rozwiązania
	Dodatkowe źródła wiedzy: definicje, informacje uzupełniające, komentarze itp.

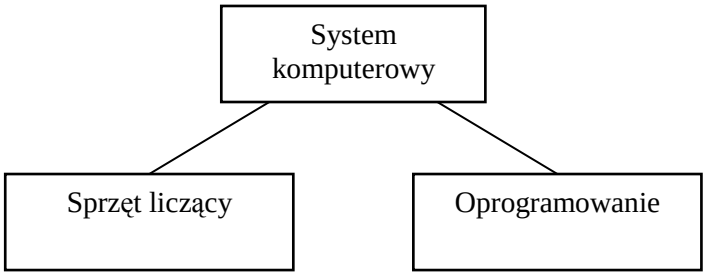
Inne konwencje notacyjne będą omawiane sukcesywnie.

1 Za górami, za lasami... czyli Wprowadzenie

1.1 Komputery i ich oprogramowanie

Rozwój konstrukcji nowożytnych urządzeń obliczeniowych rozpoczął się w latach 40. ubiegłego wieku. Równolegle ze zmianami technologicznymi, dotyczącymi zwłaszcza elementów przełączających (lamp elektronowych, a później tranzystorów i układów scalonych) oraz nośników pamięci, postępowało doskonalenie struktury wewnętrznej sprzętu liczącego i oprogramowania. Nieustające poszukiwania sposobów na to, aby uzyskiwać coraz większą szybkość działania i coraz większą pojemność pamięci, spowodowały konieczność zadbania również o to, aby sprzęt liczący był używany efektywnie. W związku z tym rozwiązania służące wyłącznie do umieszczenia programu obliczeń w pamięci urządzenia i jego uruchomienie zostały sukcesywnie zastąpione przez rozbudowane środki do zarządzania zasobami maszyny i ich wykorzystaniem.

Określenie *komputer* (z ang. *computer*) weszło na stałe do języka polskiego w końcu lat osiemdziesiątych XX wieku. Wcześniej urządzenie do automatycznego wykonywania działań na danych było (nie tylko w języku polskim) nazywane *maszyną liczącą* (ang. *computing machine*), na podobieństwo innych znanych nazw, jak *maszyna do pisania*, *maszynka do mięsa* itp. Współcześnie mówiąc *komputer*, mamy na myśli system złożony z elementów należących do dwóch wzajemnie uzupełniających się klas: sprzętu liczącego (ang. *hardware*) i oprogramowania (ang. *software*).



Rys. 1. Podstawowa struktura systemu komputerowego

Sprzęt odpowiedzialny jest za wykonywanie czynności wprowadzania/wyprowadzania danych do/z swojego otoczenia, przechowywanie, przetwarzanie i prezentację danych oraz komunikację z innymi systemami. Rolą oprogramowania natomiast jest kontrolowanie tego co, gdzie, w jaki sposób i w jakiej kolejności ma być wykonane.

Szczególną rolę ma tu do spełnienia część oprogramowania zwana *systemem operacyjnym*. Na pewnym poziomie abstrakcji system operacyjny można traktować jako *superprogram*, dla którego danymi są inne programy, a jego działanie polega na wprowadzaniu tych programów do pamięci, przechowywaniu, przetwarzaniu i prezentacji zewnętrznych przejawów ich działania (Tabela 4.).

Tabela 4. Analogie pomiędzy „zwykłym" programem a systemem operacyjnym

Cecha	„Zwykły" program	System operacyjny
Dane	liczby, teksty, symbole, obrazy itp.	„zwykłe" programy
Czynności przetwarzania	Wytwarzanie nowych danych, modyfikacja lub usuwanie z pamięci danych istniejących	Wytwarzanie nowych lub zmodyfikowanych „zwykłych" programów przy wykorzystaniu kompilatorów i innych narzędzi do tworzenia oprogramowania; usuwanie tych programów z pamięci
Przechowywanie	Czasowe lub trwałe umieszczenie danych w pamięci	Czasowe lub trwałe umieszczenie „zwykłych" programów w pamięci
Prezentacja	Przedstawienie danych w postaci czytelnej dla użytkownika, np. zestawień, wykresów, rysunków, animacji itp.	Przedstawienie zewnętrznych przejawów działania zwykłych" programów w postaci czytelnej dla użytkownika (tzw. *interfejs użytkownika*)
Komunikacja między programami	Przekazywanie danych pomiędzy programami	Przekazywanie „zwykłych" programów lub ich części pomiędzy systemami operacyjnymi, wykonywanie fragmentów zadań obliczeniowych na różnych komputerach
Komunikacja z użytkownikiem	Wprowadzanie danych wejściowych, wyprowadzanie wyników działania	Przekazywanie poleceń dotyczących sposobu i warunków wykonywania „zwykłych" programów oraz wyprowadzanie związanych z tym komunikatów

Miano „zwykły" program obejmuje również wszelkie programy nieprzeznaczone do bezpośredniego wykorzystania przez użytkownika, np. programy stanowiące części samego systemu operacyjnego.

Użytkownicy są skłonni traktować komputer jako całość, bez zwracania uwagi na funkcje jego poszczególnych elementów. Ten sposób postrzegania jest odpowiedni, o ile spełnione są następujące warunki:

- Wykorzystywane są względnie nieskomplikowane funkcje komputera,
- Komputer (jako całość) pracuje w sposób stabilny,
- Komputer jest dla użytkownika narzędziem, a nie przedmiotem pracy.

W każdym innym przypadku zrozumienie podziału zadań – i zarazem odpowiedzialności za ich prawidłowe wykonanie – pomiędzy poszczególne elementy składowe komputera jest warunkiem koniecznym dla uzyskiwania pożądanych rezultatów i skutecznego pokonywania problemów, których istnienia nie można nigdy wykluczyć.

1.2 Powstanie i genealogia UNIX-a

Historia systemu operacyjnego UNIX sięga końca lat sześćdziesiątych ubiegłego wieku i wiąże się z projektem systemu MULTICS, realizowanym na potrzeby obronności USA [11, 17]. Ponieważ ukończenie prac nad MULTICS-em, pomyślanym jako system modularny, o rozbudowanych funkcjach związanych z bezpieczeństwem (w tym wysokiej dostępności) i opartym o wiele nowatorskich koncepcji przesuwało się w czasie, powstała myśl, aby ze złożonego projektu wydzielić opracowanie czegoś, co ma nie być aż tak doskonałe, ale po prostu działać i wykonywać programy. W roku 1969 Ken Thompson, Dennis Ritchie i inni rozpoczęli prace na "mało używanym komputerze PDP-7 stojącym w rogu pokoju[1]" w AT&T Bell Laboratories, które miały w przyszłości zaowocować UNIX-em. W roku 1973 większość kodu UNIX-a została przepisana w języku C, opracowanym przez D. Ritchiego, co odtąd stało się podstawą jego *przenośności* (ang. *portability*) na maszyny o różnej architekturze i parametrach. W roku 1977 UNIX działał już w ok. 500 firmach i organizacjach. W rok później na Uniwersytecie Berkeley w Kalifornii (USA) powstała wersja pochodna względem oryginału z AT&T, nazwana BSD UNIX, która zaczęła się rozwijać samodzielnie i szybko wyprzedziła swój wzorzec pod względem funkcjonalnym.

W latach 80. rozwój systemu UNIX skomercjalizował się, zainteresowali się nim producenci komputerów, m.in. firmy IBM, Hewlett-Packard, Silicon Graphics, Sun Microsystems, a także producenci oprogramowania - Santa Cruz Operation (SCO), a przejściowo również Microsoft. Konkurencyjne wyścigi spowodowały, iż istniało wiele koncepcji normalizacyjnych dotyczących systemów UNIX (np. zestaw standardów POSIX – obecnie norma ISO/IEC 9945). Jednak wbrew obawom różnice w budowie poszczególnych realizacji systemów UNIX-opodobnych nie są na tyle znaczące, aby były przeszkodą dla użytkow-

[1] Ang. "little-used PDP-7 in a corner".

ników. Tym bardziej, że stosunkowo dobrze został znormalizowany interfejs i funkcje jądra.

Nazwa *UNIX* jest nazwą zastrzeżoną. Prawo do kodu - należące pierwotnie do twórcy systemu, firmy AT&T Bell Laboratories, zostało sprzedane w roku 1993 firmie Novell Inc., a w roku 1995 przejęte przez firmę Santa Cruz Operation (SCO). Znak handlowy UNIX-a jest obecnie w dyspozycji X/Open Consortium. Producenci, którzy opracowane przez siebie systemy operacyjne chcą określić mianem UNIX, muszą uzyskać odpowiednią licencję od właściciela praw do znaku handlowego, po uprzednim wykazaniu zgodności swojego systemu ze specyfikacją systemu UNIX, to jest po pozytywnym przejściu 1170 testów zgodności. Przykładami takich produktów są np. *SunOs/Solaris* firmy Sun Microsystems, *HP UX* firmy Hewlett-Packard i *AIX* firmy International Business Machines. Brak miana *UNIX* nie oznacza jednak automatycznie, iż system operacyjny jest niezgodny ze specyfikacją.

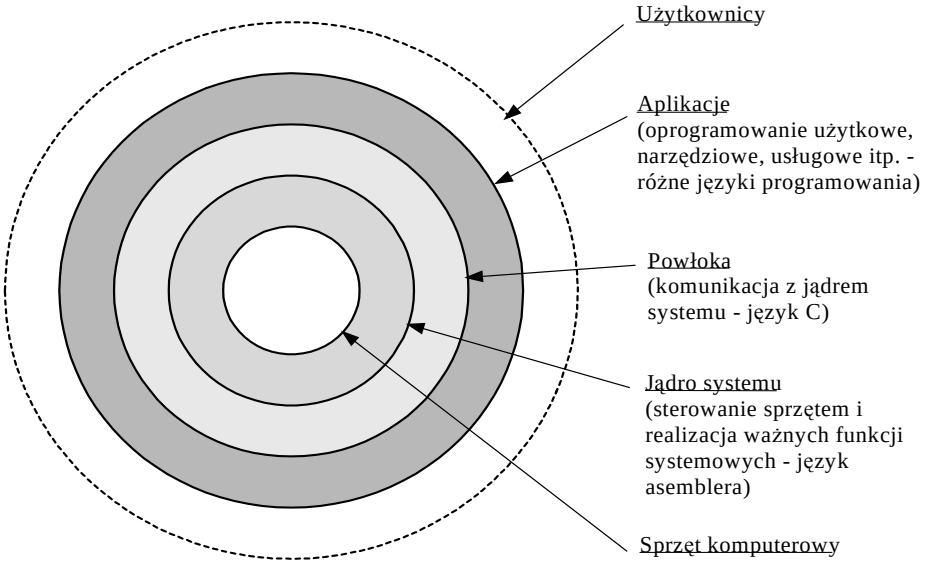
W dalszej części podręcznika, przez **N*X* będziemy rozumieli całą klasę systemów operacyjnych posiadających analogiczną strukturę i cechy funkcjonalne jak system UNIX, niezależnie od producentów i nazw, pod którymi są one rozpowszechniane. Klasa ta obejmuje w szczególności systemy Linux dystrybuowane na zasadach licencji GPL[2] i systemy BSD rozpowszechniane według własnej licencji. Lista ta nie wyczerpuje jednak wszystkich możliwości.

[2] *GNU General Public License* – Powszechna Licencja Publiczna.

2 Kamień na kamieniu, na kamieniu kamień... czyli Struktura i ogólne właściwości systemów *N*X

2.1 Struktura

System UNIX został zaprojektowany w strukturze warstwowej. Warstwą najniższą, bezpośrednio stykającą się ze sprzętem, jest tzw. *jądro systemu* (ang. *system kernel*). Jest ono odpowiedzialne za kontrolowanie realizacji podstawowych operacji wykonywanych przez sprzęt i monopolizuje do niego dostęp. Oznacza to, że żaden z programów użytkowych, ale też żaden z programów systemu operacyjnego nienależący do jądra, nie ma możliwości bezpośredniej ingerencji w działanie urządzeń zewnętrznych, wpływanie na gospodarkę zasobami systemu takimi jak pamięć operacyjna, system plików, media komunikacyjne itd. Dzięki przejęciu pełnej kontroli nad fizycznymi i logicznymi zasobami systemu wyeliminowana jest możliwość wzajemnego zakłócania pracy działających w nim programów i dezorganizacji pracy samego systemu operacyjnego. Cecha ta nie jest wyróżnikiem systemu UNIX; jest ona wymagana w przypadku każdego dojrzałego systemu operacyjnego, przeznaczonego do pracy wielozadaniowej i wieloużytkownikowej. Jądro wykonuje usługi na rzecz pozostałych składników oprogramowania działającego w systemie.



Rys. 2. Warstwy systemu UNIX i jego otoczenie

Kolejną warstwę systemu stanowi tzw. *powłoka* (ang. *shell*). Jest ona swoistym pośrednikiem pomiędzy jądrem a kolejną warstwą obejmującą aplikacje i umożliwia zarządzanie działaniem systemu przez użytkownika (np. operatora) oraz monitorowanie stanu samego systemu i realizowanych w nim zadań. Powłoka udostępnia mechanizm tekstowej komunikacji użytkownika z systemem, na który składa się:

- język poleceń (kierunek: od użytkownika do systemu),
- system komunikatów (kierunek: od systemu i wykonywanych zadań do użytkownika).

Powłoka komunikuje się z jądrem systemu poprzez wywoływanie wbudowanych w nie funkcji. Istnieje możliwość posługiwania się wieloma wersjami powłoki, stosownie do potrzeb i upodobań. Domyślna wersja powłoki uruchamiana jest w chwili rozpoczynania przez użytkownika pracy w systemie (po otwarciu tzw. *sesji*). W ramach sesji użytkownik może jednak posługiwać się różnymi powłokami. W szczególności, wydając polecenie, może określić, za pomocą której powłoki ma ono być zinterpretowane i wykonane. Korzystanie z powłoki wymaga dostępności tzw. *terminala*, który stanowi wyspecjalizowane urządzenie, bądź jest emulowany poprzez odpowiednią aplikację działającą w tym samym lub innym systemie komputerowym. Terminal, rzeczywisty lub emulowany, pozwala na wprowadzanie tekstów poleceń i wyświetlanie komunikatów wymienianych pomiędzy powłoką a użytkownikiem, w tym na wprowadzanie poleceń wyrażanych specjalnymi kombinacjami klawiszy. Współczesna technika prezentacji tekstów poleceń i komunikatów polega na ich wyświetlaniu na ekranie monitora komputerowego.

Najwyższą warstwę UNIX-a stanowią aplikacje, czyli programy użytkowe i pomocnicze. O ile liczba znaczących wersji powłoki nie przekracza dziesięciu, o tyle liczba aplikacji jest o kilka rzędów większa; dotyczą one wszelkich dających się wyobrazić dziedzin zastosowań.

Ponad aplikacjami rozpościera się *warstwa użytkowników* (ang. *userware* lub *peopleware*). Aczkolwiek (podobnie jak sprzęt komputerowy) nie stanowi ona części systemu operacyjnego, należy o jej istnieniu pamiętać. Reprezentuje ona bowiem potrzeby dotyczące przetwarzania, dla realizacji których istnieją i funkcjonują wszystkie warstwy usytuowane niżej, a jakość działania całego systemu komputerowego jest oceniana w stosunku do wymagań warstwy najwyższej. Innymi słowy, bez warstwy użytkowników sens istnienia warstw niższych jest wątpliwy.

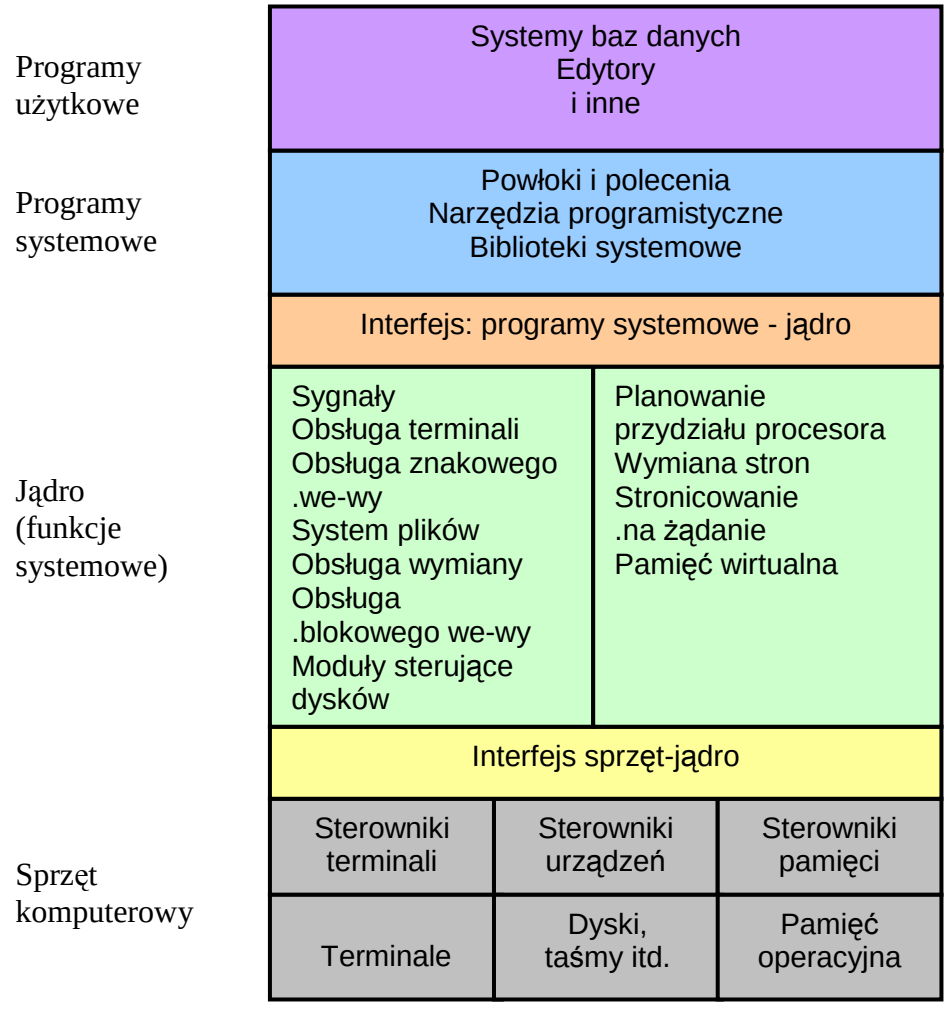
Opisany sposób budowy systemu UNIX został później przeniesiony na całą klasę systemów podobnych.

2.2 Pierwotny system UNIX

Pierwsze wersje systemu UNIX nosiły na sobie znamiona ograniczeń wynikających z niewielkich możliwości sprzętu. Podstawowymi elementami tworzącymi tę nowatorską koncepcję były:

- jądro (interfejsy i programy obsługi urządzeń),
- programy systemowe (w tym powłoka).

Architektura wczesnej wersji systemu przedstawiona jest na rys. 3 (na podstawie [11]).



Rys. 3. Struktura systemu 4.3BSD

2.3 Ogólne właściwości systemów *N*X

W klasie UNIX-opodobnych systemów operacyjnych istnieją również wersje przeznaczone do pracy w bezpośrednim połączeniu z obiektami fizykalnymi, np. urządzeniami przemysłowymi i laboratoryjnymi oraz stanowiskami pomiarowymi. Są to tzw. *systemy czasu rzeczywistego* (ang. *real-time system*, RT)[3]. Systemy klasy *N*X są również stosowane w urządzeniach mobilnych. Nie będą one jednak tutaj opisywane.

Systemy *N*X z założenia są:

- wielozadaniowe (ang. *multitasking*), co oznacza, że potrafią współbieżnie przetwarzać wiele zadań,
- wieloużytkownikowe (ang. *multiuser*) i wielodostępne (ang. *multiaccess*), tj. umożliwiają wspólne korzystanie z systemu przez wielu użytkowników, przy czym każdy z nich ma wrażenie, jakby korzystał z komputera w sposób wyłączny.

Równoczesność a współbieżność

Mówimy, że procesy A i B zachodzą **współbieżnie**, gdy realizacja drugiego z nich została rozpoczęta przed zakończeniem realizacji pierwszego, lecz bez usuwania go z systemu. Dla zewnętrznego obserwatora procesy A i B wydają się być wykonywane w tym samym czasie. Mechanizm uzyskiwania współbieżności zależy od technicznych możliwości systemu komputerowego: w przypadku systemu jednoprocesorowego (ogólnie: systemu, w którym liczba procesorów jest mniejsza od liczby procesów), współbieżność jest realizowana poprzez podział czasu procesora pomiędzy wiele procesów. Można powiedzieć, że współbieżność jest *wrażeniem równoczesności* (patrz dalej).

Mówimy, że procesy A i B zachodzą **równocześnie**, gdy w każdej chwili obydwa są wykonywane w sposób nieprzerwany, tj. bez konieczności podziału czasu procesora pomiędzy te procesy. Np. w dwuprocesorowym systemie komputerowym równocześnie mogą być realizowane co najwyżej dwa procesy, każdy na innym procesorze.

[3] Np. systemy QNX, RT Linux.

Innymi cechami systemów *N*X są:

- hierarchiczny system plików, obejmujący w jednym drzewie wszystkie woluminy (fizyczne i logiczne) oraz pliki specjalne, w tym pliki reprezentujące urządzenia zewnętrzne,
- obsługa komunikacji międzyprocesowej, w tym zdalnej,
- duża liczba funkcji wbudowanych, narzędzi, programów usługowych i zaawansowanych aplikacji z różnych dziedzin,
- *przenośność* (ang. *portability*) uzyskana dzięki architekturze wewnętrznej systemu i powszechnie dostępnemu językowi C,
- dobra *skalowalność* (ang. *scalability*), w tym możliwość eksploatacji w konfiguracjach wieloprocesorowych oraz wielomaszynowych (składających się z większej liczby komputerów),
- otwartość standardów, pozwalająca na zastąpienie danej wersji systemu *N*X przez inną, o większych możliwościach.

Trzy ostatnie z wymienionych wyżej zapewniają tzw. *bezpieczeństwo inwestycji*, tj. istotnie redukują ryzyko zaistnienia sytuacji, w której jeden lub więcej elementów systemu informatycznego zbudowanego w oparciu o system klasy *N*X przestanie spełniać swoją rolę (np. z powodu zwiększenia liczby przetwarzanych danych wydajność jednoprocesorowego serwera bazy danych przestanie być wystarczająca) i trzeba go będzie zastąpić innym.

3 Na początku był chaos... czyli System plików

3.1 Dane i pliki w systemach komputerowych

Działanie systemu komputerowego na ogół polega na przetwarzaniu jednych zestawów danych (nazywanych danymi wejściowymi) w inne zestawy (nazywane danymi wyjściowymi). Niezależnie od postaci i interpretacji danych, występuje potrzeba ich przechowywania. Ze względu na znaczną i ciągle rosnącą objętość danych, wymagane jest posiadanie skutecznego mechanizmu zarządzania obszarem nośnika przeznaczonego do ich przechowywania oraz zarządzania umieszczaniem danych na tym nośniku w sposób pozwalający m.in. na:

- jednoznaczną ich lokalizację i skuteczne odnajdowanie,
- określenie czasu utworzenia i modyfikacji, a niekiedy również czasu ostatniego dostępu,
- kontrolę uprawnień do korzystania z poszczególnych elementów danych przez użytkowników systemu komputerowego,
- sterowanie współbieżnym dostępem do elementów danych przez różne zadania realizowane w systemie,
- kontrolę wielkości obszaru zajętego przez elementy danych, zwłaszcza podczas tworzenia, usuwania i zmiany wielkości plików, a w konsekwencji -
- zarządzanie niewykorzystanym obszarem nośnika.

Pamiętajmy przy tym o specyficznym rodzaju danych, którymi są programy – one też muszą być przechowywane w systemie komputerowym.

Zalew informacji

Na każdego z 6,3 miliarda mieszkańców Ziemi przypada rocznie ok. 800 MB nowo zarejestrowanych informacji. Stacje telewizyjne na świecie produkują rocznie 31 milionów godzin programów, co stanowi 70 tys. TB danych. Światowe zasoby serwisów WWW obejmują 170 TB danych, a poczta elektroniczna generuje rocznie 400 tys. TB danych. 40% wszystkich przechowywanych na świecie danych wytwarzanych jest w USA [13].

Zob. ramkę pt. „Komputerowe jednostki pojemności”.

Jak wiadomo, w systemach komputerowych dane przechowywane są w *plikach* (ang. *file*). Plik posiada skończoną objętość, a przechowywane w nim dane posiadają zdefiniowaną interpretację. Z punktu widzenia systemu operacyjnego, plik stanowi całość, która opisana jest zbiorem atrybutów (w tym: na-

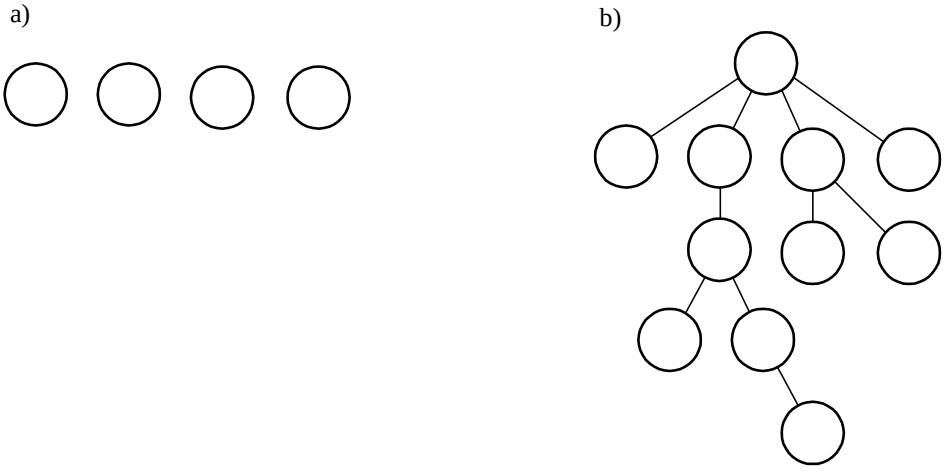
zwa, typ, prawa własności, prawa dostępu, informacja o lokalizacji na nośniku, status pliku). Z punktu widzenia zawartości można wyróżnić pliki: wykonywalne, tekstowe, skrypty (pliki wsadowe) i inne. Plik jest pojęciem logicznym i nie definiuje sposobu fizycznego przechowywania danych na nośniku. Struktura służąca do przechowywania na nośniku plików i ich atrybutów oraz związane z nią oprogramowanie zarządzające nazywane są łącznie *systemem plików* (ang. *file system*).

Komputerowe jednostki pojemności (KJP)

1 KB (1 Kilobajt) = 2^{10} bajtów	1 TB (1 Terabajt) = 2^{40} bajtów	1 ZB (1 Zettabajt) = 2^{70} bajtów
1 MB (1 Megabajt) = 2^{20} bajtów	1 PB (1 Pentabajt) = 2^{50} bajtów	1 YB (1 Yettabajt) = 2^{80} bajtów
1 GB (1 Gigabajt) = 2^{30} bajtów	1 EB (1 Exabajt) = 2^{60} bajtów	

Uwaga: $2^{10} = 1024 \approx 10^3 = 1000$

Przy niewielkich liczbach przechowywanych plików i nieznacznej ich łącznej objętości, mogą one być identyfikowane wyłącznie za pośrednictwem swojej nazwy. W każdym innym przypadku potrzebne jest uporządkowanie plików przez zbudowanie struktury drzewiastej, w której do identyfikacji pliku, oprócz jego nazwy (tzw. *nazwy względnej*), niezbędne jest określenie tzw. *ścieżki*, wskazującej umiejscowienie pliku w tej strukturze.



Rys. 4. Systemy plików: a) jednopoziomowy, b) wielopoziomowy

Do ewidencji plików zlokalizowanych w danym miejscu tej struktury służą pliki specjalne nazywane *katalogami* (ang. *directory*), które przechowują atrybuty odpowiednich plików. O ile system UNIX od samego początku posiadał drzewiasty (wielopoziomowy) system plików, o tyle w systemie operacyjnym MS DOS dla komputerów IBM PC pierwotnie funkcjonował tylko płaski (jednopoziomowy) system plików i dopiero pojawienie się pojemniejszych urządzeń pamięci zewnętrznej na dyskach elastycznych[4] spowodowało udostępnienie wielopoziomowego systemu plików (MS DOS wersja 2.0).

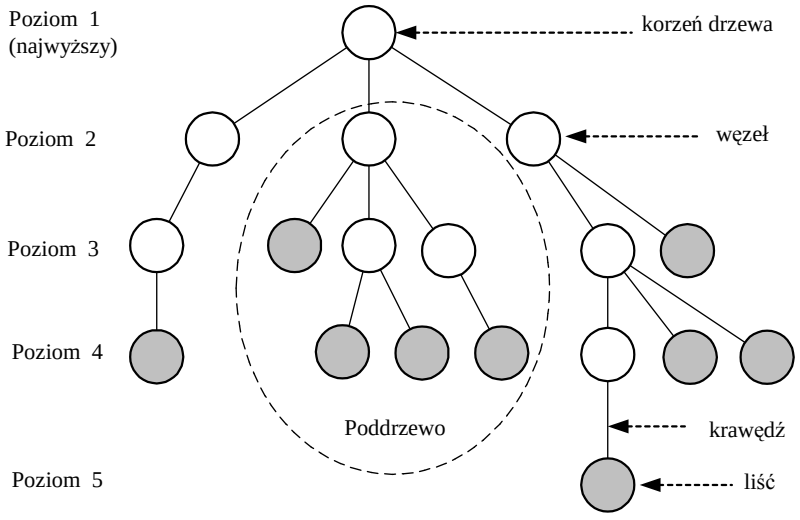
3.2 System plików w *N*X

Jak wspomniano wcześniej, systemy *N*X wyposażone są w system plików o strukturze hierarchicznej, inaczej - *strukturze drzewa* (ang. *tree*). Zawiera ona punkty nazywane *wierzchołkami* (ang. *vertex*) lub *węzłami* (ang. *node*), które połączone są tzw. *krawędziami* (*gałęziami*). Krawędzie reprezentują zależności typu nadrzędny-podrzędny pomiędzy węzłami. Początkowy, wyróżniony węzeł tej struktury nazywany jest *korzeniem* (ang. *root*). Cechą charakterystyczną drzewa jest to, że dwa dowolne węzły łączy dokładnie jedna trasa, zwana *ścieżką* (ang. *path*). Węzły drzewa można uporządkować według poziomów, które z kolei reprezentują hierarchię węzłów.

Podzbiór węzłów i łączących je krawędzi, mający swój początek w danym węźle drzewa i zawierający wszystkie węzły hierarchicznie zależne od danego węzła nazywany jest *poddrzewem* (ang. *subtree*); dany węzeł stanowi korzeń tego poddrzewa. Ostatni węzeł na ścieżce rozpoczynającej się w korzeniu drzewa nazywany jest *liściem*.

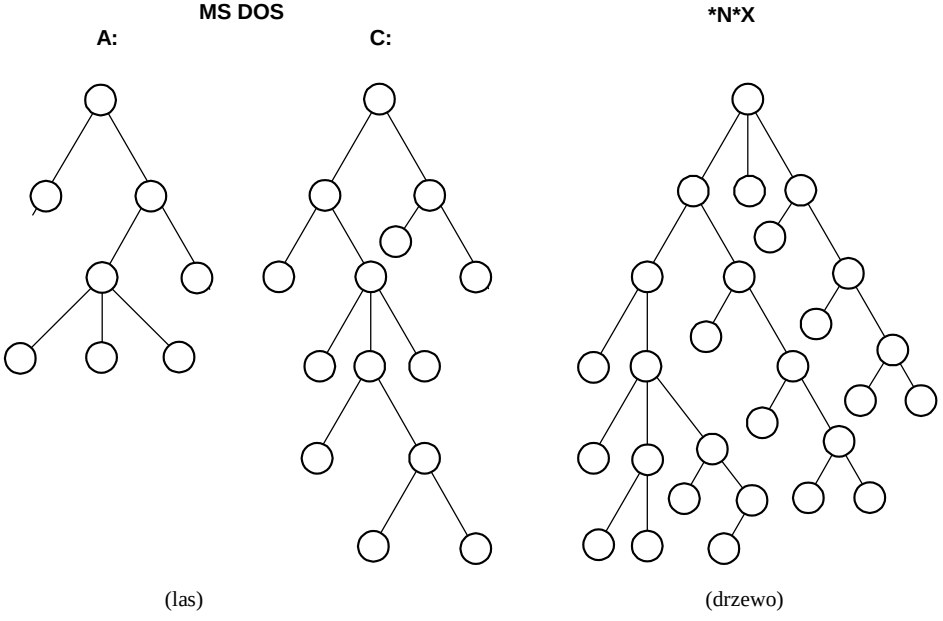
Zwyczajowo graficzna reprezentacja systemu plików posiada korzeń umieszczony u góry (rys. 5.).

[4] Tzw. *dyski twarde* pojawiły się w komputerach IBM PC o wiele później niż w systemach UNIX, ale za to jako urządzenia o zasadniczo udoskonalonej konstrukcji. Pierwszy model komputera osobistego z dyskiem (10 MB), znany jako IBM PC XT, wszedł na rynek w 1983 r. [15].



Rys. 5. Elementy struktury drzewiastej. Pokolorowano węzły będące liśćmi.

W odróżnieniu od systemów plików wywodzących się z MS DOS, system plików *N*X stanowi pojedyncze drzewo, a nie tzw. *las* (rys. 6.).



Rys. 6. Porównanie struktury systemu plików systemów operacyjnych MS DOS i pochodnych z systemami *N*X

3.3 Pliki i katalogi

Węzły systemu plików reprezentują różne rodzaje zawartości:

- katalogi,
- pliki zwykłe,
- pliki specjalne.

Katalogi[5] (ang. *directories*) stanowią wykazy innych plików. Katalog odpowiadający korzeniowi drzewa nazywany jest *katalogiem głównym* (ang. *root directory*). O plikach umieszczonych w danym wykazie mówimy, że *znajdują się* w odpowiadającym mu katalogu. Na odwrót, atrybuty opisujące każdy plik (a więc i katalog) znajdują się w jego katalogu nadrzędnym, przy czym katalogiem nadrzędnym względem katalogu głównego jest on sam. W konsekwencji każdy plik znajduje się w jakimś katalogu. Nazwy plików zlokalizowanych w danym katalogu muszą być różne.

Pliki zwykłe (ang. *regular files*) służą do przechowywania informacji użytkowych, takich jak dane, programy, parametry sterujące itp. Pojęcie pliku zwykłego w systemie *N*X odpowiada pojęciu pliku w systemach MS DOS i pochodnych. Najczęściej pliki zwykłe zawierają dane w postaci tekstowej (*pliki tekstowe*, ang. *text files*) lub binarnej (*pliki binarne*, ang. *binary files*). Zawartość pliku tekstowego jest wyrażona za pomocą znaków (liter, cyfr, znaków specjalnych) i bezpośrednio czytelna dla użytkownika. Plik binarny z kolei może zawierać kod programu w postaci zrozumiałej tylko dla komputera lub zakodowane dane, np. obraz nieruchomy lub ruchomy, dźwięk, a także zawartość skompresowaną i/lub zarchiwizowaną itp. Wśród plików binarnych należy wyróżnić pliki *modułów ładowalnych*, tj. takich, które zawierają kod programu nadający się do bezpośredniego wykonania przez komputer – bez dodatkowych czynności przygotowawczych, np. kompilacji. *Plik wykonywalny* zaś to plik zawierający moduł ładowalny lub *skrypt*, tj. zestaw poleceń powłoki.

Pliki specjalne (ang. *special files*) reprezentują urządzenia systemu *N*X. Korzystając z tych plików, system operacyjny uzyskuje dostęp do pamięci operacyjnej, urządzeń wejścia/wyjścia, woluminów pamięci zewnętrznej i innych.

Jeżeli nie podano inaczej, terminem *plik* (bez przymiotnika) będziemy określać plik dowolnego rodzaju, w tym katalog.

Na każdym poziomie struktury systemu plików mamy do czynienia z jakimiś danymi: katalogi zawierają atrybuty znajdujących się w nich plików, pliki zaś przechowują dane będące przedmiotem lub wynikiem przetwarzania. System plików zawiera więc pewną strukturę danych, która wymaga adekwatnego mechanizmu dostępu do nich. Dla identyfikacji plików stosuje się nazwy i ścieżki.

[5] Używany jest również termin *folder*.

Nazwa pliku jest ciągiem znaków z ograniczonego zbioru znaków ASCII, tj. obejmuje:

- małe i duże litery,
- cyfry,
- inne znaki, przy czym ich użycie wymaga szczególnej ostrożności, ponieważ mają one niekiedy dla powłoki specjalne znaczenie.

Znaki odstępu (spacji) i lewej kreski ukośnej \ nie mogą występować w nazwach. Duże i małe litery są traktowane jak różne znaki.

Zwróćmy uwagę na to, że kropka nie posiada specjalnego znaczenia, jak to jest w systemie MS DOS i pochodnych. Wynika to z tego, że w systemach *N*X nie istnieje pojęcie *rozszerzenia nazwy pliku*, bowiem zawartość i sposób interpretacji pliku nie są określane poprzez jego nazwę. Nazwa rozpoczynająca się od kropki i zawierająca więcej niż jeden znak oznacza *plik ukryty*. Pliki takie są pomijane przy zwykłym wyświetlaniu zawartości katalogu.

Przykłady

Poprawnymi nazwami plików systemu *N*X są:

- **`Linus`**
- **`.co_to_jest`**
- **`mar-cheffka`**
- **`PanToGraf`**

Niepoprawnymi nazwami plików są:

- **`Pan Janek`** (niedozwolony odstęp w nazwie)
- **Znak\Zorro** (niedozwolona kreska ukośna)

W nazwie pliku kropka może występować dowolnie wiele razy, przy czym nazwy składające się wyłącznie z jednej kropki . lub dwóch kropek .. mają znaczenie specjalne[6].

[6] Oznaczenia te zarezerwowane są dla nawigacji pomiędzy katalogami, o czym będzie mowa w pkt. 3.4.

Tworzenie nazw obiektów

Nazwa powinna możliwie dokładnie odzwierciedlać znaczenie obiektu, który sobą reprezentuje. Powinno się unikać nazw jednoznakowych, np. a, i nazywania obiektów w sposób stereotypowy, np. osoba, dane, projekt. Jeżeli w danym zbiorze występuje wiele podobnych obiektów, ich nazwy powinny zawierać dodatkowe określenia wyróżniające.

Przy tworzeniu nazwy obiektu, którego opis słowny zawiera wiele wyrazów, można posłużyć się jedną z następujących metod:

- Zestawienie wyrazów stanowiących opis słowny lub ich skrótów w jeden ciąg znaków (tj. pominięcie odstępów między wyrazami),
- Zestawienie skrótów wyrazów stanowiących opis słowny lub ich skrótów w jeden ciąg znaków, z rozdzieleniem ich za pomocą znaku podkreślenia _,
- Zestawienie skrótów wyrazów stanowiących opis słowny lub ich skrótów w jeden ciąg znaków, z wyróżnieniem inicjałów tych wyrazów dużą literą (tzw. *metoda węgierska*) lub kombinacją powyższych metod.

Ze względów praktycznych zalecane jest, aby przed przystąpieniem do przekształcenia słownego opisu obiektu w nazwę, dokonać skrótu tego opisu przez wyeliminowanie mniej istotnych wyrazów. Należy przy tym zadbać, aby skojarzenia powstające podczas posługiwania się taką nazwą, nie odbiegały zbyt daleko od znaczenia samego obiektu.

Kluczowymi kryteriami oceny jakości nazw są więc wygoda korzystania i jednoznaczność interpretacji przez grono użytkowników, które tymi nazwami się posługuje.

W szczególności, powyższe zasady stosują się do nazw plików w systemie operacyjnym. W tym przypadku możliwość umieszczenia pliku użytkowego w odpowiednio nazwanym katalogu (ścieżce) stanowi dodatkowy sposób reprezentacji znaczenia tego pliku.

Przykład tworzenia nazwy obiektu

Dla obiektu nazwanego „Sprawozdanie z działalności zarządu w roku finansowym 2004” można utworzyć następujące nazwy:

- `Sprawozdaniezdzialalnoscizarzaduwroku↵ finansowym2004`
- `sprawozda- nie_z_dzialalnosci_zarzadu_w_roku_↵ finansowym_2004`
- `SprawZarzRokFin2004`

Nazwy przedstawione w pkt. 1-3 zostały utworzone przy wykorzystaniu metod oznaczonych odpowiednio jako a-c. Nazwa z punktu 1 jest niepraktyczna z powodu długości i nieczytelności. Nazwa z punktu 2 jest czytelna, ale jest jeszcze dłuższa niż poprzednia. Nazwa z punktu 3 jest akceptowalna, aczkolwiek można ją ulepszyć poprzez skrócenie do `SprawZarz2004`. Zakładamy przy tym, że osoby korzystające z obiektu o nazwie `SprawZarz2004` rozpoznają jednoznacznie, iż chodzi o rok finansowy, a nie np. o rok sporządzenia sprawozdania.

Przykład tworzenia nazwy pliku

Załóżmy, że tworzymy szablon (wzór) nazwy, jaką będziemy nadawali kolejnym plikom zawierającym kopie bezpieczeństwa danych.

Jeżeli pliki kopii bezpieczeństwa będą umieszczane w katalogu niededykowanym specjalnie do tego celu (tj. oprócz plików kopii bezpieczeństwa będą się tu znajdować pliki o innej zawartości), to w nazwie pliku kopii powinniśmy zawrzeć co najmniej informacje o tym:

- że jest to plik kopii bezpieczeństwa,
- jaki podzbiór systemu plików zawiera kopia,
- kiedy kopia została wykonana.

Wzór nazwy takiego pliku mógłby mieć postać `BackAll_`*`rrrr-mm-dd`*, gdzie element `Back`[7] został użyty do wskazania, że jest to właśnie kopia bezpieczeństwa, element `All` - że jest to kopia całego systemu plików, a symbole *`rrrr`*, *`mm`* i *`dd`* reprezentują odpowiednio rok, miesiąc i dzień w miesiącu, kiedy kopia została wykonana.

Jeżeli przewidujemy, że w ciągu jednego dnia będzie wykonywana więcej niż jedna kopia, to powyższy wzór można rozbudować np. do postaci `BackAll_`*`rrrr-mm-dd.k`*, gdzie *`k`* oznacza kolejny numer kopii w danym dniu.

[7] Od ang. *backup* - kopia bezpieczeństwa.

	Jeżeli administrator systemu postanowi utworzyć katalog dedykowany do przechowywania plików kopii bezpieczeństwa, to część informacji na temat kopii można zawrzeć w nazwie tego katalogu, np. `Backups`. Wtedy poszczególne pliki kopii mogłyby mieć nazwy postaci `All_`*`rrrr-mm-dd`* lub podobne.

3.4 Nawigacja w systemie plików

Ścieżka do pliku (*bezwzględna nazwa pliku)* stanowi ciąg nazw plików, którego pierwszym elementem jest oznaczenie katalogu głównego w postaci pojedynczej kreski ukośnej /. Kolejne elementy ścieżki, oprócz ostatniego, są nazwami kolejnych katalogów występujących w drzewie systemu plików na trasie pomiędzy katalogiem głównym a danym plikiem. Ostatnim elementem ścieżki jest nazwa danego pliku. Poszczególne nazwy są oddzielone od siebie separatorem w postaci kreski ukośnej. Katalog bezpośrednio poprzedzający dany plik na ścieżce do tego pliku nazywany jest jego *katalogiem nadrzędnym* (ang. *parent directory*). Nazwa pliku w swoim katalogu nadrzędnym jest jego *nazwą lokalną* w tym katalogu. Katalog następujący na ścieżce do pliku bezpośrednio po danym katalogu nazywany jest jego *podkatalogiem* (ang. *subdirectory*).

	**Separator a ogranicznik** **Separator** jest symbolem, który oddziela od siebie kolejne obiekty, np. pola tekstowe, tj. występuje pomiędzy nimi. Przykładem separatora jest znak odstępu stosowany pomiędzy poszczególnymi elementami polecenia powłoki, a także dwukropek w niektórych systemowych bazach danych. **Ogranicznik** (**terminator**) jest symbolem umieszczanym po obiektach danego typu. Przykładem ogranicznika jest znak nowej linii lub średnik kończący zapis polecenia w systemie *N*X.

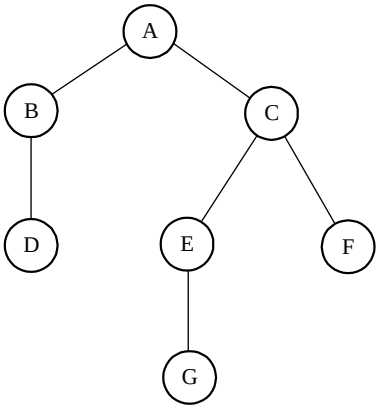
	**Przykłady ścieżek w systemie *N*X** • `/` • `/etc/init.d/network` • `/usr/lib/` • `/var/www/html/error.log`

Ciąg znaków będący względną lub bezwzględną nazwą pliku i zakończony kreską ukośną oznacza katalog. Jeżeli względna lub bezwzględna nazwa pliku nie jest zakończona kreską ukośną, ustalenie czy plik ten jest katalogiem, jest możliwe jedynie przez sprawdzenie.

Aby jednoznacznie zidentyfikować dany plik w drzewie systemu plików, wystarczy podać:

- ścieżkę do pliku, albo
- ścieżkę do wybranego katalogu leżącego na ścieżce do danego pliku oraz pozostałą część ścieżki do pliku.

Przykłady identyfikacji plików w drzewie systemu plików



W drzewie plików przedstawionym obok:

Plik o nazwie `F` można zidentyfikować np. nazwą bezwzględną `/A/C/F`.

Plik o nazwie `G` można zidentyfikować np. przez wskazanie ścieżki do katalogu nadrzędnego o nazwie `C`, tj. `/A/C/` i pozostałej części ścieżki do `G` poniżej katalogu `C`, tj. `E/G`.

Drugi ze sposobów pozwala na istotne ograniczenie liczby znaków wprowadzanych z terminala w celu identyfikacji plików niezbędnych do wykonania polecenia. W chwili rozpoczynania przez użytkownika pracy w systemie *N*X ustalany jest dla niego pewien katalog, nazywany *katalogiem bieżącym* (*domyślnym*), stanowiący wspomniany wyżej katalog nadrzędny. Zazwyczaj jest to tzw. *katalog prywatny* użytkownika (ang. *home directory*). Podczas pracy na terminalu katalog bieżący może być zmieniany dowolnie wiele razy[8]. Dla zidentyfikowania pliku należącego do poddrzewa, którego korzeń jest katalogiem bieżącym, wystarczy podać nazwę tego pliku. Jeżeli natomiast potrzebne jest wskazanie pliku spoza tego poddrzewa, można posłużyć się bezwzględną nazwą pliku.

Do nawigacji w drzewie systemu plików użyteczne są *specjalne nazwy plików*:

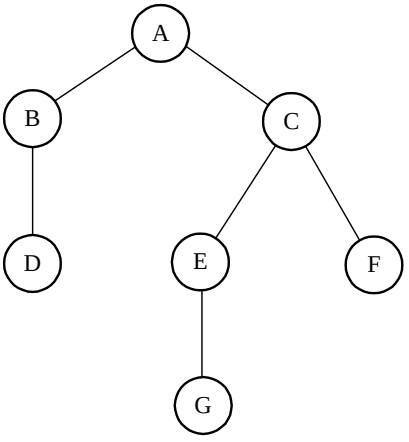
- pojedyncza kropka `.` wskazuje katalog bieżący,
- podwójna kropka `..` wskazuje katalog nadrzędny w stosunku do bieżącego,
- tylda `~` oznacza prywatny (własny) katalog użytkownika.

Podobnie jak w systemach MS DOS, grupy plików o zbliżonych nazwach można wskazywać za pomocą tzw. *symboli wieloznacznych* (ang. *wildcards*):

[8] Niektóre katalogi mogą być dla konkretnego użytkownika niedostępne, o czym będzie szerzej mowa w rozdziale 4.

- gwiazdka * zastępuje dowolny ciąg znaków, również *ciąg pusty*, tj. niezawierający żadnego znaku,
- znak zapytania ? zastępuje dokładnie jeden dowolny znak.

Przykłady użycia specjalnych nazw plików



Załóżmy, że katalogiem bieżącym jest **E**

Katalog **A** można wskazać jako **../..**

Katalog **B** można wskazać jako **../../B**

Katalog **C** można wskazać jako **..** .

Plik (katalog) **D** można wskazać jako **../../B/D**

Katalog **E** można wskazać jako **./**

Plik (katalog) **F** można wskazać jako **../F**

Plik (katalog) **G** można wskazać jako **G** lub **./G**

3.5 Podstawowe katalogi systemu *N*X

Aczkolwiek ogólna struktura systemu plików jest podobna, poszczególne wersje systemów *N*X różnią się między sobą zarówno liczbą i nazwami katalogów drugiego poziomu, tj. znajdujących się w katalogu głównym, jak i strukturą poddrzew, których korzenie znajdują się na tym poziomie.

Najczęściej w katalogu głównym systemu *N*X znajdują się katalogi:

- `/bin`
- `/dev`
- `/etc`
- `/lib`
- `/lost+found`
- `/mnt`

- /sbin
- /tmp
- /usr
- /var

Katalog /bin (od ang. *binary* – binarny) przeznaczony jest do przechowywania programów w postaci wykonywalnej, w tym wielu poleceń systemu *N*X.

Katalog /dev (od ang. *device* – urządzenie) zawiera systemowe pliki specjalne (zob. 3.6). Ich zawartość przeważnie nie nadaje się do bezpośredniego wykorzystania, np. urządzenia przechowujące systemy plików wymagają tzw. *zamontowania*, dzięki czemu ich zawartość logiczna staje się dostępna poprzez wskazany katalog (przeważnie /mnt).

Katalog /etc (od łac. *et cetera* – i tak dalej) zawiera przede wszystkim pliki konfiguracyjne systemu i skrypty powłokowe służące do zarządzania nim. Poprzez ich modyfikację administrator systemu *N*X uzyskuje pożądany sposób działania.

Katalog /lib (od ang. *library* – biblioteka) zawiera standardowe składniki oprogramowania, w tym ładowalne moduły biblioteczne, wywoływane przez działające w systemie aplikacje.

Katalog /lost+found (od ang. *lost* – zgubione i *found* – znalezione[9]) jest używany podczas naprawy uszkodzonego systemu plików, jako magazyn plików odzyskanych.

Katalog /mnt (od ang. *mount* – montuj) służy do logicznego udostępnienia w postaci podkatalogów struktur plików czasowo użytkowanych woluminów pamięci zewnętrznej o bezpośrednim dostępie, np. dyskietek, płyt CD-ROM lub DVD, partycji dysku.

Katalog /sbin (od ang. *system binaries* – binarne programy systemowe) zawiera programy w postaci wykonywalnej, przeznaczone do użycia wyłącznie przez administratora systemu.

Katalog /tmp (od ang. *temporary* – tymczasowy) zawiera pliki robocze, tworzone przez system i działające w nim programy.

[9] Ang. *Lost and found* oznacza biuro rzeczy znalezionych.

Katalog `/usr` (od ang. *user* – użytkownik) zazwyczaj zawiera rozbudowaną strukturę podkatalogów, które można traktować jako rozszerzenia innych katalogów systemu, w tym `/bin`, `/lib` i `/tmp`.

Katalog `/var` (od ang. *variable* – zmienny) zawiera pliki, które powstają w toku eksploatacji systemu i poszczególnych aplikacji lub mają tendencję do zmiany swojej objętości, a nie stanowią plików tymczasowych. Są to m.in. dane samych aplikacji, pliki kronik systemowych.

W niektórych realizacjach systemów *N*X występują dodatkowe katalogi, np. w systemach RedHat Linux i pochodnych są to katalogi: `/boot`, `/home`, `/root`, `/misc`, `/opt`, `/proc` i inne.

3.6 Katalog /dev

W tym miejscu dokonamy przeglądu zawartości podkatalogu `/dev`, reprezentującego fizyczne i logiczne urządzenia systemowe. Urządzeniem fizycznym jest to, któremu wprost odpowiada określony fragment sprzętu komputerowego, np. zegar czasu rzeczywistego, urządzeniem logicznym natomiast to, które jest w pełni realizowane programowo.

Niektóre z plików urządzeń przedstawiono poniżej, przy czym zmienne elementy nazw zaznaczono czcionka pochyłą:

- `/dev/tty0` - terminal nr 0,
- `/dev/fd0` - czytnik dyskietek nr 0,
- `/dev/cdrom` - czytnik CD-ROM nr 0,
- `/dev/null` - plik pusty (“czarna dziura”),
- `/dev/zero` - urządzenie logiczne generujące binarne zera,
- `/dev/hd`*x* - dysk IDE (urządzenie fizyczne); *x* oznacza symbol dysku, *x* = a, b, ...
- `/dev/sd`*x* - dysk SCSI (urządzenie fizyczne)[10]; *x* oznacza symbol dysku, *x* = a, b, ...
- `/dev/hd`*xy* - partycja (dysk logiczny) na dysku IDE; *x* oznacza symbol dysku, *y* oznacza numer partycji na dysku, *x* = a, b, ... *y* = 1, 2, ...
- `/dev/sd`*xy* - partycja (dysk logiczny) na dysku SCSI; *x* oznacza symbol dysku, *y* - numer partycji na dysku *x* = a, b, ...*y* = 1, 2, ...
- `/dev/md`*r* - partycja logiczna RAID[11]; *r* oznacza numer partycji RAID, *r* = 0,1, ...

[10] W systemach Linux tak samo jak dyski SCSI obsługiwane są również pamięci wymienne USB (ang. *flash disk*).

Podczas dołączania do systemu nowych urządzeń, nadawane im są kolejne identyfikatory.

Należy zwrócić uwagę na to, że zawartość plików specjalnych jest bezpośrednio nieprzydatna dla użytkownika - nie podlegają one tworzeniu, modyfikacji i użyciu w taki sposób, jak to się czyni z plikami zwykłymi. Nie oznacza to jednak, iż mogą być one z systemu usunięte, po prostu wykorzystuje się je poprzez odpowiednie polecenia systemowe, potrafiące ich zawartość prawidłowo zinterpretować. Np. w celu udostępnienia woluminu o bezpośrednim dostępie musi być wykonane systemowe polecenie `mount`.

Specyficzną rolę w systemie *N*X pełni plik `/dev/null` reprezentujący dość nietypowe urządzenie wejścia lub wyjścia. Jako źródło danych wejściowych dla programu dostarcza ono zawartości pustej, tj. nie dostarcza żadnych danych. Jako miejsce przeznaczenia danych wyjściowych zachowuje się jak „czarna dziura" – dane skierowane do`/dev/null` nie są wyświetlane ani nigdzie zapamiętywane. Z tej drugiej możliwości korzysta się np. wtedy, gdy istnieje potrzeba ograniczenia objętości plików wytwarzanych podczas wykonywania programu, a kierowany do `/dev/null` strumień danych nie ma wartości informacyjnej.

[11] Ang. *Redundant Array of Independent Disks* (nadmiarowa macierz niezależnych dysków) - systemowa struktura dyskowa o podwyższonej wydajności lub niezawodności.

3.7 Porównanie systemów plików w MS DOS i *N*X

Zamieszczona poniżej tabela zawiera zestawienie podstawowych cech systemów plików wywodzących się z MS DOS i systemów klasy UNIX.

Tabela 5. Porównanie systemów plików MS DOS i *N*X

Cecha	MS DOS[12] (w nawiasie MS Windows 9x i następne)	*N*X
Nazwy plików i katalogów		
Rozróżnianie dużych i małych liter	Nie	Tak
Spacje w nazwach dozwolone	Nie (Tak)	Nie
Znaki narodowe w nazwach dozwolone	Nie (Tak)	Nie
Terminator nazwy katalogu	\	/
Oznaczenie woluminu fizycznego	brak	`/dev/xxx`
Struktura zawartości	Las (wiele drzew)	Pojedyncze drzewo
Wolumin logiczny	Odrębne drzewo	Poddrzewo
Oznaczenie woluminu logicznego	**A:, B:, ...**	brak

Nazwy plików w mieszanych środowiskach operacyjnych

Tam, gdzie obok siebie funkcjonują systemy operacyjne MS Windows 9x i następne oraz systemy *N*X, poprzez wprowadzenie mało dolegliwych ograniczeń można doprowadzić do zgodności nazw w systemach plików.

W tym celu we wzajemnie udostępnianych strukturach systemów plików należy:
W systemach MS Windows wyeliminować nazwy zawierające odstępy i znaki narodowe.
W systemach obydwu rodzin korzystać z nazw zawierających zbiór znaków ograniczony np. wyłącznie do dużych lub wyłącznie małych liter alfabetu łacińskiego oraz cyfr, kropki ., łącznika - i znaku podkreślenia _.
Drugie z ograniczeń można nieco złagodzić, aczkolwiek zaproponowany tu zbiór dopuszczalnych znaków jest wystarczający dla większości zastosowań.

[12] Dotyczy również MS Windows 3.x, stanowiącego nakładkę na MS DOS.

3.8 Zadania i ćwiczenia

1. Wyznacz kolejne potęgi liczby 2, tj. oblicz wartość 2^x, dla x = 1, 2, …, 16 i sprawdź, czy potrafisz operować tymi wartościami z pamięci. Np. ile wynosi 2^{11}?
2. Oblicz procentowy błąd zastąpienia wartości 2^{10} (1 K) przez wartość przybliżoną 10^3 (1 K).
3. Korzystając z definicji komputerowych jednostek pojemności (ramka powyżej) oblicz, ilu bajtom odpowiada 1 KB, 1 MB, 1 GB, 1 TB, 1 PB i 1 EB.
4. Ze względów marketingowych producenci dysków twardych podają ich pojemność posługując się metrycznymi wartościami przedrostków jednostek miary. 1 MB oznacza wtedy „zwykły" milion, tj. 10^6, a nie 2^{20}. Pojemność dysku typu WD800JB producent określił na 80 GB. Wyraź tę wartość w komputerowych jednostkach pojemności. O ile druga wartość jest mniejsza od pierwszej? Wyraź tę różnicę w KJP.
5. Na podstawie danych z ramki „Zalew informacji" oraz liczby mieszkańców Polski określ statystyczną objętość nowych informacji rejestrowanych w kraju w ciągu roku.
6. Zakładając, że lokalna nazwa pliku może mieć długość 1-8 znaków, z których pierwszy jest literą alfabetu łacińskiego [A-Z], a pozostałe mogą być literami lub cyframi [0-9], oblicz liczbę różnych nazw możliwych do uzyskania. Nie rozróżniamy liter dużych i małych.
7. Rozwiąż zadanie 6 przyjmując, że litery duże [A-Z] i małe [a-z] są traktowane odrębnie.
8. Dlaczego względna nazwa pliku nie może rozpoczynać się kreską ukośną?
9. Czy ścieżka do pliku może składać się z identycznych nazw, np. `/A/A/A/A`? Narysuj przykład takiej struktury plikowej.
10. Czy nazwy `./Alga` i `Alga` wskazują ten sam plik? Dlaczego?
11. Czy nazwy `.Bulga` i `Bulga` wskazują ten sam plik? Dlaczego?
12. Przyjmując, że katalog główny znajduje się na 1. poziomie hierarchii systemu plików, oblicz numer poziomu, na którym znajduje się plik o nazwie `/wisla/tu/odra/jak_mnie/slyszysz/odbior`. Narysuj odpowiedni fragment struktury systemu plików.

	13. Dane są dwie bezwzględne nazwy plików: `/wisla/tu/odra/jak_mnie/slyszysz/odbior` oraz `/rzeka/wisla/najwieksza/z/polskich/rzek`. Wskaż katalog stanowiący korzeń najmniejszego poddrzewa systemu plików, które zawiera obydwa podane pliki. Swoją odpowiedź sprawdź za pomocą rysunku.

4 Wolnoć Tomku w swoim domku… czyli Podstawowy system uprawnień

4.1 Wprowadzenie

W znanym wierszu pt. „Paweł i Gaweł”[13] Aleksander hrabia Fredro w humorystyczny sposób pokazał skutki nieliczenia się z otoczeniem. Jeżeli tytułowych bohaterów wiersza potraktować w tym miejscu jako metaforycznych reprezentantów zadań (procesów) realizowanych w systemie komputerowym, to następstwa postępowania na zasadzie „Wolnoć Tomku w swoim domku” mogą wcale nie być humorystyczne.

Każde zadanie wykonywane w systemie komputerowym korzysta z jego zasobów, takich jak procesor, pamięć operacyjna i zewnętrzna, urządzenia wejścia/wyjścia, funkcje wbudowane itp. Zasoby używane przez dane zadanie nie mogą być jednak dowolnie wykorzystywane przez inne zadania. Groziłoby to ich niekontrolowanymi interferencjami, zmianą zawartości istotnych obszarów pamięci, zakłóceniem pracy urządzeń zewnętrznych, a nawet zakłóceniem lub zatrzymaniem pracy samego systemu operacyjnego.

O ile kontrola nad prawidłowym współużytkowaniem przez różne zadania procesora, pamięci operacyjnej i urządzeń wejścia/wyjścia w całości spoczywa na systemie operacyjnym, to dla zapewnienia prawidłowego współdziałania uruchamianych zadań konieczne jest umożliwienie korzystania przez nie ze wspólnych danych, bibliotek modułów programowych, plików parametrów sterujących, itp. Podobnie jak zadaniom, również użytkownikom systemu komputerowego niezbędna jest możliwość wspólnego posługiwania się określonymi fragmentami struktury systemu plików.

W systemach operacyjnych MS DOS i wcześniejszych wersjach systemów MS Windows[14] każdy z użytkowników komputera miał jednakowy i pełny dostęp do wszystkich plików systemu. W ten sposób w pełni realizowany był postulat dostępu do wspólnych danych. Z drugiej strony jednak, w tego typu systemie plików są one dostępne również użytkownikom i zadaniom, dla których nie jest to niezbędne. Powszechna dostępność plików wywołuje zagrożenia w postaci:

- ujawnienia danych podlegających ochronie, np. utraty prywatności,
- nieupoważnionej lub niezamierzonej zmiany zawartości lub usunięcia pliku.

[13] Np. „Paweł i Gaweł i inne opowieści”, Wydawnictwo Podsiedlik-Raniowski i Spółka, 1990.

[14] W tym MS Windows 3.x i pochodnych oraz MS Windows 9x, z systemami plików typu FAT i FAT32.

System UNIX, jako wieloużytkownikowy i wielozadaniowy system operacyjny, został zaprojektowany tak, aby umożliwić selektywne przydzielanie praw do korzystania z plików użytkownikom i uruchamianym przez nich zadaniom.

4.2 Podstawowy system kontroli dostępu do plików

4.2.1 Użytkownicy i grupy użytkowników

Z komputerów, a więc i z systemów *N*X korzystają konkretne osoby. Nie są one częścią systemu, a więc ich każdorazowy kontakt z systemem wymaga uprzedniego potwierdzenia tożsamości.

Klasyczna metoda potwierdzania tożsamości (ang. *authentication*, *uwierzytelnianie*[15]) wymaga, aby dla nowego użytkownika utworzony został tzw. *identyfikator* (ang. *identifier*) i aby za jego pomocą użytkownik ten przedstawiał się za każdym razem, gdy przystępuje do pracy w systemie. Ponieważ identyfikator jest jawny, to łatwo sobie wyobrazić, że może być wykorzystany przez osoby nieupoważnione. W związku z tym używa się dodatkowej - tym razem poufnej - danej nazywanej *hasłem* (ang. *password*). Dopiero para danych składająca się z (jawnego) identyfikatora użytkownika oraz odpowiedniego (poufnego) hasła wystarcza do potwierdzenia tożsamości[16].

Identyfikator użytkownika w systemie *N*X (nazwijmy go *identyfikatorem zewnętrznym*) ma postać nazwy. Jest z nią skojarzony identyfikator będący liczbą naturalną (nazwijmy go *identyfikatorem wewnętrznym*), który oznaczany jest skrótem UID (and. *user identifier*). Skojarzenie to jest jednoznaczne w tym sensie, że zewnętrznemu identyfikatorowi użytkownika zarejestrowanemu w systemie odpowiada dokładnie jedna liczba. Z drugiej strony jednak danemu wewnętrznemu identyfikatorowi użytkownika może odpowiadać wiele różnych nazw. Początkowe numery UID są zarezerwowane dla celów systemowych. Nowo tworzeni użytkownicy otrzymują identyfikatory wewnętrzne począwszy od pewnej wartości, zależnej od wersji systemu, np. powyżej 100 lub powyżej 500. Istnieje możliwość nadania użytkownikowi konkretnej wartości UID. Baza danych o użytkownikach systemu znajduje się w pliku tekstowym `/etc/passwd`.

Mówimy, że dany użytkownik posiada *konto* (ang. *account*) w systemie *N*X, jeśli jego identyfikator jest znany systemowi, a sam użytkownik posiada nadane uprawnienia do korzystania z określonych plików.

---

[15] Określenie *autentykacja* jest kalką językową i powinno być traktowane jak żargon.

[16] W rozdziale 12. Czytelnik znajdzie informacje na temat jakości zabezpieczeń systemu przed nieupoważnionym dostępem.

Ze względu na konieczność wspólnego dostępu do plików, użytkownicy są organizowani w tzw. grupy (ang. *groups*). Grupy użytkowników są identyfikowane analogicznie jak użytkownicy – poprzez nazwę i skojarzony z nią numer, oznaczany skrótem GID (ang. *group identifier*). Każdy użytkownik należy przynajmniej do jednej grupy; w szczególności może nią być grupa domyślna nazwana `users`, `other` lub podobnie. Szereg grup tworzonych jest automatycznie w czasie instalowania systemu *N*X, np. `root`, `bin`, `mail`, `daemon`, `adm.` Lista grup może być różna dla różnych wersji systemu *N*X. Baza danych o grupach użytkowników systemu znajduje się w pliku tekstowym `/etc/group`.

Definiowanie nowych grup oraz przypisywanie do nich użytkowników należy do kompetencji użytkownika o identyfikatorze zewnętrznym `root` i wewnętrznym `UID=0`, który należy do grupy o nazwie `root` i identyfikatorze wewnętrznym `GID=0`. Polecenia służące do zarządzania użytkownikami i ich grupami oraz uprawnieniami dostępu do plików zostały przedstawione w punkcie 0. Użytkownik o identyfikatorze `root`[17] nie może być usunięty z systemu. Każdy użytkownik z `UID=0` i `GID=0` ma uprawnienia identyczne jak `root` (aczkolwiek może mieć odrębne hasło).

4.2.2 Uprawnienia dostępu do plików

UNIX-owy system zarządzania uprawnieniami dostępu do plików dzieli te uprawnienia na trzy klasy:

- Uprawnienia odnoszące się do właściciela pliku – oznaczenie `u` (od ang. *user*),
- Uprawnienia odnoszące się do grupy, do której należy właściciel pliku, tzw. *grupy właścicielskiej* – oznaczenie `g` (od ang. *group*),
- Uprawnienia dla wszystkich pozostałych użytkowników zarejestrowanych w systemie (tj. poza właścicielem i członkami grupy właścicielskiej) – oznaczenie `o` (od ang. *other*).

Do zarządzania prawami własności plików służy polecenie `chown` (od ang. *change owner* – zmień użytkownika), które zostanie przedstawione w pkt. 7.7.

Dla danego pliku, każdej klasie nadaje się niezależnie uprawnienia (tzw. *tryby dostępu*, ang. *access mode*) do:

- Odczytu pliku (np. wyświetlenia zawartości lub użycia jako danych wejściowych dla programu) – oznaczenie `r` (od ang. *read*),
- Zapisu do pliku (w tym użycia pliku do zapisania danych wyjściowych, jego modyfikacji lub skasowania) – oznaczenie `w` (od ang. *write*),

[17] Zwany również *superużytkownikiem* (ang. *superuser*).

- Wykonania pliku (zinterpretowania jego zawartości) – oznaczenie `x` (od ang. *execute*). Jeżeli jest to plik zwykły i zawiera program w postaci wykonywalnej lub skrypt powłokowy – uprawnienie dotyczy jego wykonania. Jeżeli plik jest katalogiem, uprawnienie dotyczy udostępnienia jego zawartości, np. w celu sporządzenia wykazu.

Zestaw danych o podstawowych uprawnieniach dostępu do pliku zapisuje się w postaci:

u(ser)	g(roup)	o(ther)
`rwx`	`rwx`	`rwx`

gdzie `r`, `w` i `x` oznaczają wartości `0` (zero), gdy odpowiednie uprawnienie nie zostało, lub `1` (jeden), gdy zostało udzielone. Pierwsza triada uprawnień `rwx` dotyczy właściciela, druga – grupy właścicielskiej, trzecia natomiast - wszystkich pozostałych użytkowników. Całość danych o podstawowych uprawnieniach dostępu do danego pliku można więc zapisać w postaci trzycyfrowej liczby w systemie ósemkowym; przyjmuje ona wartości od `000` do `777`.

Alternatywny sposób zapisu tych danych polega na użyciu liter `r`, `w` i `x`, gdy odpowiednie uprawnienie zostało nadane, i łącznika `-`, gdy zostało odebrane.

Przykład – kodowanie podstawowych uprawnień dostępu do pliku

Kodowanie ósemkowe	Kodowanie symboliczne
777	rwxrwxrwx
711	rwx--x--x
644	rw-r--r--
640	rw-r-----

Do zarządzania trybami dostępu do plików służy polecenie `chmod`, które zostanie przedstawione w pkt. 7.7.

Tabela 6. Znaczenie trybów dostępu dla różnych rodzajów plików

Rodzaj pliku	Tryb dostępu	Sterowanie wykonalnością operacji
Katalog		
	r (odczyt)	Odczytu katalogu
	w (zapis)	Tworzenia, usuwania i modyfikacji elementów katalogu
	x (wykonanie)	Dostępu do katalogu
Plik zwykły		
	r (odczyt)	Odczytu pliku
	w (zapis)	Modyfikacji i usuwania pliku
	x (wykonanie)	Wykonania zawartości pliku
Plik specjalny		
	r (odczyt)	Odczytu pliku
	w (zapis)	Modyfikacji i usuwania pliku
	x (wykonanie)	Wykonania zawartości pliku

Dociekliwy użytkownik systemu *N*X podczas sprawdzania trybu dostępu do plików może będzie miał okazję zauważyć, że niekiedy zamiast x pojawia się litera s lub S. Wbrew pozorom, nie jest to błąd, tylko sygnalizacja dodatkowego uprawnienia dostępu do pliku, zakodowana w nietypowy sposób.

Aby to wyjaśnić, omówimy dwa dodatkowe tryby dostępu, oznaczane SUID i GUID. Tryb SUID steruje prawem do wykonania pliku z uprawnieniami jego właściciela, GUID natomiast – z uprawnieniami nadanymi grupie właścicielskiej. Wydanie przez danego użytkownika polecenia wykonania zawartości pliku z nadanym uprawnieniem SUID powoduje, iż na czas wykonywania identyfikator tego użytkownika zostanie zmieniony na identyfikator właściciela pliku, tj. inny użytkownik „wejdzie w prawa" właściciela i będzie mógł skutecznie ten plik wykonać nawet wtedy, gdy sam nie posiada do niego prawa dostępu x. Analogiczna sytuacja dotyczy trybu GUID, wtedy jednak skutkiem wywołania pliku będzie chwilowa zmiana grupy, do której należy użytkownik wywołujący plik na grupę właściciela samego pliku.

Przy symbolicznym kodowaniu uprawnień tryb SUID kodowany jest w polu x właściciela, tryb GUID natomiast w polu x grupy właścicielskiej pliku. Zasady kodowania tych dwóch trybów na jednej pozycji znakowej przedstawia tabela 7.

Tabela 7. Łączne kodowanie trybów x oraz SUID/GUID

Tryb x	Tryb SUID/GUID	Oznaczenie symboliczne
-	wyłączony	-
x	wyłączony	x
-	włączony	S
x	włączony	s

Wykrycie włączonego trybu *SUID* lub *GUID*

W systemie *N*X liczba plików, dla których istnieje rzeczywista potrzeba włączenia trybu SUID lub GUID, jest bardzo niewielka. Jeżeli podczas tworzenia wykazu plików zostanie stwierdzone pojawienie się nowych pozycji z włączonym trybem dostępu SUID lub GUID (litera S, a zwłaszcza S w oznaczeniu uprawnień), należy to potraktować jako sygnał alarmowy o zagrożeniu bezpieczeństwa systemu komputerowego.
Bez wyraźnej potrzeby trybów tych nie należy też włączać.

W starszych wersjach systemów UNIX, działających na komputerach o niewielkiej pojemności pamięci operacyjnej, intensywnie korzystano z tzw. *wymiatania* (ang. *swap*) nieaktywnych w danej chwili programów z pamięci operacyjnej do pamięci o bezpośrednim dostępie, w celu zapewnienia dostatecznie dużego obszaru dla programów aktywnych. Aby uniknąć wymiatania często wywoływanych modułów systemowych, co powodowało znaczne opóźnienia czasowe, używano trybu dostępu nazywanego *sticky*[18]. Włączenie trybu *sticky* powodowało, iż dany plik wykonywalny nie podlegał wymiataniu, dzięki czemu w razie potrzeby zawarty w nim kod był natychmiast osiągalny w pamięci operacyjnej.

Przy symbolicznym kodowaniu uprawnień tryb *sticky* kodowany jest w polu x pozostałych użytkowników w sposób przedstawiony w tabeli poniżej.

Tabela 8. Łączne kodowanie trybów x i *sticky*

Tryb x	Tryb *sticky*	Oznaczenie symboliczne
-	wyłączony	-
x	wyłączony	x
-	włączony	T
x	włączony	t

We współczesnych komputerach wyposażonych w system operacyjny *N*X tryb *sticky* ma już tylko znaczenie historyczne. Należy jednak sądzić, że dla zachowania wstecznej zgodności jest on nadal implementowany. Omyłkowe włączenie trybu *sticky* nie powinno mieć wpływu na bezpieczeństwo systemu komputerowego.

[18]Ang. sticky, tu: przyklejający się do swojego miejsca. W języku polskim spotyka się określenie *bit lepkości*.

Uwzględniając tryby SUID, GUID oraz *sticky* otrzymujemy następujący – rozszerzony schemat kodowania symbolicznego:

u(ser)	g(roup)	o(ther)
αβγ	αβγ	αβδ

gdzie:

- w miejscu symbolu α może wystąpić znak `r` lub `-`,
- w miejscu symbolu β - znak `w` lub `-`,
- w miejscu symbolu γ - znak `x`, `-`, `S` lub `s`,
- w miejscu symbolu δ - znak `x`, `-`, `T` lub `t`.

Łatwo zauważyć, że schematu rozszerzonego nie da się przedstawić w postaci numerycznej za pomocą trzech cyfr ósemkowych. W zapisie liczbowym dla przedstawienia dodatkowych trzech trybów potrzebne są jeszcze trzy pozycje binarne, tj. jedna cyfra ósemkowa. Ostatecznie rozszerzony schemat uprawnień w postaci numerycznej składa się z czterech cyfr ósemkowych:

extra	u(ser)	g(roup)	o(ther)
SGT	RWX	RWX	RWX

gdzie:

- `S`, `G` i `T` reprezentują bitowe wskaźniki trybów, odpowiednio SUID, GUID i *sticky*,
- grupy `RWX` reprezentują bitowe wskaźniki trybów, odpowiednio np. (*r*)*ead*, (*w*)*rite* i *e(x)ecute*, kolejno dla właściciela pliku, grupy właścicielskiej i pozostałych użytkowników systemu.

Przykład – kodowanie rozszerzonych uprawnień dostępu do pliku

Kodowanie ósemkowe	Kodowanie symboliczne
777	rwxrwxrwx
2755	rwxr-sr-x
4711	rws--x--x
50	---r-x---
3645	rw-r-Sr-t
7640	rwSr-S--T

W zapisie numerycznym nieznaczące zera można pominąć.

4.2.3 Maskowanie uprawnień

Po utworzeniu nowego pliku ma on ustawiony domyślny zestaw praw dostępu. Dla umożliwienia użytkownikowi sprawowania kontroli nad tymi prawami, a zwłaszcza dotyczącymi dostępu do jego plików przez innych użytkowników, systemy *N*X posiadają tzw. m*askę użytkownika* (ang. *user mask*). Maska ta ma postać trzycyfrowej liczby ósemkowej i określa uprawnienia dostępu, które podczas tworzenia pliku zostaną wyłączone. Poszczególne pozycje bitowe maski odpowiadają symbolicznemu kodowi uprawnień `rwxrwxrwx`. Pierwotna wartość maski jest ustalana podczas tworzenia konta użytkownika i przybiera wartość od `000` (pełne zaufanie do wszystkich użytkowników), przez `022` (słabo restrykcyjna polityka bezpieczeństwa) do `077` (polityka bezpieczeństwa maksymalnie restrykcyjna w stosunku do użytkowników innych niż właściciel). Maski różnych użytkowników mogą być różne.

Rola maski podczas kształtowania uprawnień do nowo utworzonego pliku przedstawiona jest poniżej. Załóżmy, że maska użytkownika ma wartość `022`.

plik danych			
podstawowe prawa:	`666`	tj.	`rw-rw-rw-`
maska:	`022`		
(po zamaskowaniu)	`644`	tj.	`rw-r--r--`
plik wykonywalny lub katalog			
podstawowe prawa:	`777`	tj.	`rwxrwxrwx`
maska	`022`		
(po zamaskowaniu)	`755`	tj.	`rwxr-xr-x`

Aktualna wartość maski użytkownika może być sprawdzona lub zmieniona poleceniem `umask`[19].

Inne sposoby kontroli dostępu do plików

W obliczu współcześnie istniejących zagrożeń bezpieczeństwa, pierwotny sposób kontroli dostępu do plików przestał już dziś wystarczać. Wprowadzono rozbudowane systemy kontroli, bazujące na tzw. listach sterowania uprawnieniami (ang. *Access Control Lists*; ACL), które pozwalają na precyzyjne sterowanie uprawnieniami dostępu dla wielu różnych użytkowników.
Pierwotny sposób kontroli dostępu do plików nadal jest jednak w systemach *N*X rozwiązaniem podstawowym.

[19] Polecenie `umask` będzie dokładniej omówione w rozdziale 7.

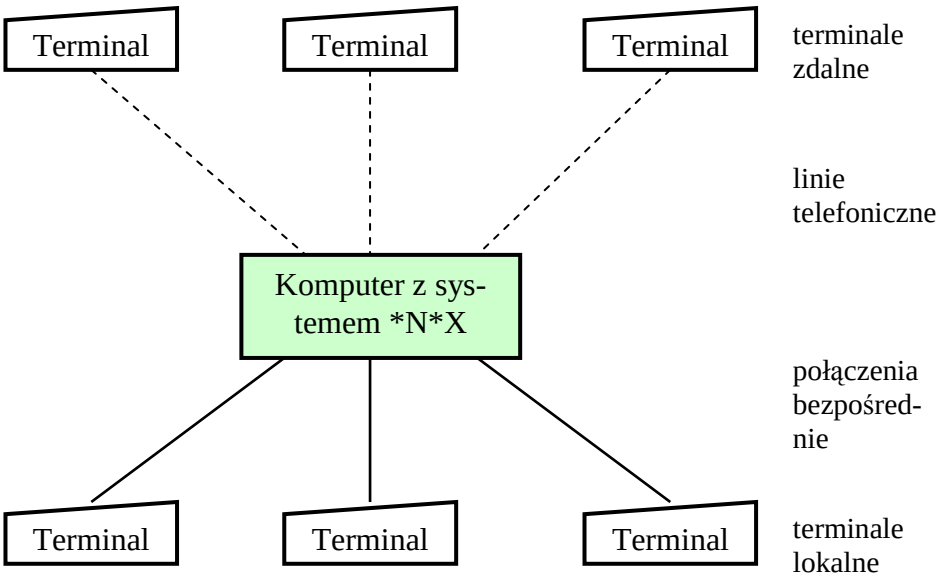
4.3 Zadania i ćwiczenia

1. Przećwicz konwersję w pamięci liczb z zapisu ósemkowego na dwójkowy i odwrotnie. Zamień na postać ósemkową liczbę w zapisie dwójkowym 11101001001. Liczbę 1535_8 przedstaw w zapisie dwójkowym.
2. Określ symboliczne odpowiedniki trybów dostępu: 711, 401, 644, 100.
3. Przedstaw w postaci symbolicznej tryby dostępu: 7744, 16, 1234, 601.
4. Jaka powinna być maska użytkownika, aby tworzyć pliki niedostępne dla nikogo, poza właścicielem?
5. Jaka powinna być maska użytkownika, aby tworzyć pliki dostępne wyłącznie dla właściciela i to tylko do odczytu?
6. Jakie będą uprawnienia do pliku tekstowego utworzonego z maską `120`?
7. Jaka była maska użytkownika, jeśli plik wykonywalny utworzono z uprawnieniami `rwxr-----`?
8. Napisz polecenie, które policzy użytkowników systemu i wyprowadzi komunikat postaci:
   `Ogólna liczba użytkowników systemu wynosi:` *`liczba`*`.`
9. Wykonaj zadanie 8., lecz w odniesieniu do grup użytkowników.

5 Najtrudniejszy pierwszy krok… czyli Praca z terminalem

5.1 Terminal

Kontakt użytkownika z systemem *N*X odbywa się za pośrednictwem tzw. *terminala*. Początkowo było nim urządzenie elektromechaniczne podobne do dalekopisu (rodzaj elektrycznej maszyny do pisania wysyłającej i odbierającej znaki poprzez łącze szeregowe).



Rys. 7. Terminale w systemie *N*X

Repertuar znaków dostępnych na klawiaturze ograniczony był do liter, cyfr, odstępu i znaków przestankowych. Klawiatura wyposażona była w klawisz **<Shift>** służący do zmiany rejestru (przejście z liter małych na duże i dostęp do znaków przestankowych) oraz klawisz **<Control>**[20] służący do generowania tzw. *znaków sterujących* (ang. *control character*). Dla formalności należy przypomnieć, że klawiszy **<Shift>** i **<Control>** używa się łącznie ze zwykłymi klawiszami. Przebieg konwersacji użytkownika z komputerem rejestrowany był na szerokiej papierowej wstędze. Z tego też okresu wywodzi się wiele pojęć używanych do nazwania lub opisania czynności systemu, pochodzących od słowa *print* (ang. – *drukuj*), jak np. polecenie `pwd` (ang. *print working directory*

[20] Oznaczany również jako **<Ctrl>**.

– drukuj katalog roboczy) i inne[21]. W czasach współczesnych, mimo zastąpienia maszyny do pisania urządzeniami ekranowymi, klasyczne nazewnictwo obowiązuje nadal. Wiele funkcji uzyskiwanych dawniej za pomocą kombinacji klawisza **<Control>** z innymi znakami realizują obecnie dodatkowe klawisze specjalne.

W dalszej części niniejszego podręcznika ograniczymy się do terminali znakowych, poprzestając jedynie na informacji, że – znacznie wcześniej od systemów operacyjnych MS Windows – systemy UNIX posiadały własne graficzne środowisko pracy[22], pozwalające na kształtowanie swojego wyglądu przez użytkownika, działające przy tym w układzie klient-serwer.

5.2 Repertuar znaków

Systemy *N*X korzystają ze zbioru znaków ASCII (zobacz Załącznik) zawierającego:

- znaki sterujące (kody dziesiętne 0-31),
- podstawowy zbiór znaków, w tym:
  - znak odstępu (spacji) (kod 32),
  - znaki specjalne (kody 33-47, 58-64, 91-96, 123-127),
  - cyfry (kody 48-57),
  - duże litery alfabetu łacińskiego (kody 65-90),
  - małe litery alfabetu łacińskiego (kody 97-122).

Nieprzypadkowo kody odpowiadających sobie małych i dużych liter różnią się o stałą wartość równą 32.

Terminal systemu *N*X umożliwia dwukierunkowe przesyłanie informacji tekstowych, wykorzystując do tego celu co najmniej podstawowy zbiór znaków ASCII oraz zbiór znaków sterujących. Oprócz tego terminale systemów *N*X rozpoznają niektóre kombinacje uzyskiwane przez wciśnięcie klawisza z przypisanym kodem ASCII łącznie z klawiszem **<Control>** albo **<Shift>**. Równoczesne użycie klawiszy **<Control>** i **<Shift>** jest przy tym interpretowane jak **<Control>**. Niektóre terminale rozpoznają również klawisze funkcyjne **<F1>** do **<F12>** i klawisze sterowania kursorem.

[21] W wyniku wykonania polecenia `apropos print` w systemie Slax 1.4.1 uzyskuje się 137 poleceń, które słowo *print* zawierają w nazwie bądź opisie swojej funkcji.

[22] X Window System opracowany został w Laboratory for Computer Science, Massachusetts Institute of Technology. Obecnie prawa autorskie do X Window System posiada The Open Group [wg polecenia `man X`].

5.3 Otwarcie sesji

Pracę użytkownika w systemie *N*X poprzedza procedura identyfikacji jego tożsamości zwana *logowaniem* (ang. *log-in*), które rozpoczyna się zwykle od wyprowadzenia napisu identyfikującego system oraz napisu `Login:`, po którym następuje oczekiwanie na wprowadzenie przez użytkownika jego identyfikatora. Następnie system wyprowadza tekst `Password:` i oczekuje na podanie hasła. W przeciwieństwie do identyfikatora wprowadzone hasło nie jest widoczne na terminalu (rys. 8.).

```
vhomer Rambo.3.4
Login: les              ← zewnętrzny identyfikator (nazwa) użytkownika
Password:               ← znaki hasła niewidoczne
$
```

Rys. 8. Przebieg logowania do systemu.

Pomyślne przeprowadzenie logowania rozpoczyna tzw. *sesję* pracy w systemie. Uruchamiana jest (wcześniej ustalona) powłoka, której zadaniem jest zapewnienie możliwości konwersacji użytkownika z systemem. Po ewentualnych dodatkowych informacjach na terminalu pojawia się wiersz zawierający tzw. *tekst zaproszenia* (ang. *prompt – zachęta*). W najprostszej postaci, dla superużytkownika tekst ten jest pojedynczym znakiem `#`, dla wszystkich zaś pozostałych użytkowników – znakiem `$`. Tekst zachęty może też zawierać inne informacje, np. identyfikator użytkownika, nazwę komputera i nazwę katalogu roboczego. O postaci tekstu zachęty decydują tzw. *zmienne środowiska* (ang. *environment variables*) o nazwach `$PS1` i `$PS2`, o czym będzie mowa w rozdziale 6.

```
    [les@vhomer dokum]$
     ↗      ↑       ↑
użytkownik komputer ostatni katalog na bieżącej ścieżce
```

Rys. 9. Rozbudowany tekst zachęty.

Każdorazowo po pojawieniu się zachęty użytkownik może wydać polecenie i śledzić rezultaty jego wykonania. W szczególności, w ramach otwartej sesji, można zalogować się do tego samego lub innego komputera używając polecenia `telnet`[23]:

[23] O ile na docelowym komputerze jest ono obsługiwane (zobacz rozdział 12.).

```
# telnet venus.priv.pl
Trying 127.0.0.1...
Connected to venus.
Escape character is '^]'.
Red Hat Linux Release 9 (Shrike)
Kernel 2.4.20-8 on an i686
login: les
Password:            ← znaki hasła niewidoczne
Last login: Thu Oct  9 22:09:06 from venus
[les@venus les]$
```

Rys. 10. Zdalne logowanie do systemu *N*X

Od tej chwili aż do wylogowania się z komputera `venus` konwersacja użytkownika `les` będzie dotyczyła komputera `venus` i będzie realizowana za pomocą powłoki określonej dla użytkownika w tym właśnie komputerze.

5.4 Polecenia klawiszowe

Oprócz wydawania poleceń opisanych wyżej zachodzi również potrzeba sygnalizowania systemowi różnorodnych sytuacji szczególnych.

Na przykład:

- wykonywanie polecenia trwa zbyt długo, a na podstawie otrzymanych komunikatów nie można ustalić, jaka jest tego przyczyna,
- uruchomiony program przestał wprowadzać dane wejściowe lub informować użytkownika o stanie obliczeń, np. realizuje nieskończoną pętlę programową,
- użytkownik zorientował się, że użył niewłaściwego polecenia, lub polecenie zostało wydane w niewłaściwy sposób (złe opcje lub parametry),
- użytkownik wprowadził już z terminala wszystkie dane wejściowe a program nie potrafi sam rozpoznać tego faktu i oczekuje na kolejne dane,
- zachodzi potrzeba wydania polecenia w taki sposób, aby zostało odebrane inaczej niż fragment wprowadzanych lub modyfikowanych właśnie danych.

Ponieważ użytkownik otrzyma kolejną zachętę dopiero po zakończeniu wykonywania bieżącego polecenia, w pewnych sytuacjach mógłby nie mieć możliwości przekazania sygnałów, o których mowa wyżej, poprzez polecenie o postaci opisanej w pkt. 6.1. Możliwość taką udostępniają polecenia wydawane w inny sposób – jako kombinacje klawiszy terminala; dalej będziemy je nazywać *poleceniami klawiszowymi*.

Tabela 9. Ważniejsze polecenia klawiszowe terminala

Kombinacja klawiszy	Umowny kod polecenia	Znaczenie
<Ctrl>+A	**home** (ang. *home* – tu: początek)	Ustawienie kursora na początku wiersza (tuż za tekstem zachęty)
<Ctrl>+C	**intr** (ang. *interrupt* – przerwanie)	Przerwanie aktualnie wykonywanego polecenia
<Ctrl>+D	**eof** (ang. e*nd of file*)	Koniec pliku – zakończenie wprowadzania danych w trybie interaktywnym
<Ctrl>+L	**clear** (ang. *clear* - wyczyść)	To samo, co polecenie powłoki **clear**[24]
<Ctrl>+Q	**start**	Wznowienie zatrzymanego wyświetlania (listowania) danych
<Ctrl>+S	**stop**	Chwilowe zatrzymanie wyświetlanych (listowanych) danych
<Ctrl>+U lub **<Ctrl>+X**	**kill**[25] (ang. *kill* - zabij)	Skasowanie całego wiersza polecenia (polecenie stosowane zamiast wielokrotnego użycia BSP)
<Ctrl>+W	**werase** (ang. *word erase* – skasuj słowo)	Usunięcie ostatniego słowa z wiersza polecenia
<Ctrl>+Y		Zawieszenie programu pierwszoplanowego, gdy podejmie próbę czytania z terminala
<Ctrl>+Z	**susp** (ang. *suspend* - zawieś)	Zawieszenie programu pierwszoplanowego
<Ctrl>+[	**esc** (ang. *escape*)	Przerwanie bieżącej sesji

W kolejnej tabeli zamieszczono niektóre inne polecenia klawiszowe, mające już jednak - z powodu istnienia specjalnych klawiszy - znaczenie historyczne.

[24] W powłoce GNU bash 3.00.15(1) RedHat polecenie klawiszowe `clear` wydane po `home` skutkuje tylko usunięciem wszystkich poprzednich wierszy, z pozostawieniem bieżącego.

[25] Polecenia klawiszowego `kill` (służącego do usuwania tekstu polecenia w trakcie jego redagowania) nie należy mylić z nazwą polecenia powłoki (służącego do usuwania procesu).

Tabela 10. Niektóre przestarzałe polecenia klawiszowe terminala

Kombinacja klawiszy	Umowny kod polecenia	Znaczenie
<Ctrl>+H	BSP (ang. *backspace*)	Cofnięcie ze skasowaniem poprzedzającego znaku
<Ctrl>+I	TAB (ang. *tabulator*)	Przeskok do następnej kolumny wydruku (standardowo co 8 znaków)
<Ctrl>+J	LF (ang. *line feed*)	Przesunięcie wydruku do następnego wiersza
<Ctrl>+M	CR (ang. *carriage return*)	Powrót do początku bieżącego wiersza

Alternatywnym sposobem oznaczania klawisza **<Control>** jest znak ^. W tej pisowni kombinacja klawiszy **<Ctrl>+D** oznaczona będzie jako ^D. Domyślny sposób realizacji poleceń klawiszowych przez powłokę można zmienić poleceniem `stty`.

Polecenia klawiszowe nie są bezpośrednio odzwierciedlane na terminalu, obserwujemy tylko wywoływane nimi skutki.

5.5 Przykłady realizacji poleceń

Systematyczny przegląd poleceń systemu *N*X znajdzie Czytelnik w rozdziale 7. W tym miejscu posłużymy się niektórymi z nich, jednak bez bardziej wnikliwego opisu.

Mając otwartą sesję, na początek sprawdźmy, które urządzenie pełni rolę bieżącego terminala[26]:

```
$ tty
/dev/pts/1
$
```

Rys. 11. Sprawdzanie urządzenia będącego terminalem

Aby dowiedzieć się, jakie są aktualne ustawienia terminala, użyjemy polecenia `stty` z opcją `–a`. Pośród wielu różnych parametrów, znajdziemy tu definicje znanych nam poleceń klawiszowych, m.in. `intr`, `erase`, `kill`, `eof` i in. Omawianie znaczenia pozostałych parametrów terminala pominiemy.

[26] Ilustracje zamieszczone w tym podręczniku wykonano w środowisku systemu Linux, dystrybucja Fedora Core 4, w oknie terminala `xterm`.

```
$ stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol
= M-^?; eol2 = M-^?;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V;
flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -
igncr icrnl ixon -ixoff
-iuclc ixany imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0
cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase
-tostop -echoprt
echoctl echoke
$
```

Rys. 12. Badanie parametrów terminala

Polecenie `stty` użyte bez parametru podaje ograniczony zbiór danych terminala.

```
$ stty
speed 38400 baud; line = 0;
eol = M-^?; eol2 = M-^?;
-brkint ixany
$
```

Rys. 13. Podstawowy format polecenia `stty`

Podczas pracy na terminalu kolejno wprowadzane i wyprowadzane wiersze dopisywane są w naturalny sposób poniżej tych, które pojawiły się wcześniej. Na terminalach typu elektryczna maszyna do pisania było to związane z sukcesywnym wysuwem papieru, na którym utrwalany był przebieg konwersacji. W przypadku terminala symulowanego na monitorze ekranowym, po wypełnieniu całego widocznego obszaru roboczego, wiersze położone najwyżej są kolejno usuwane, a pozostała zawartość przesuwana odpowiednio w górę. Terminale symulowane w środowiskach graficznych posiadają na ogół bufor, co umożliwia przewijanie obszaru roboczego w oknie terminala. Polecenie `clear`[27] przesuwa zawartość okna w buforze tak, że w najwyżej położonym wierszu zostanie wyświetlona tylko zachęta, a pozostała część okna będzie pusta.

[27] Analogiczne do `CLS` w systemie MS DOS.

Jeśli zawartość okna terminala jest jak poniżej,

```
$ stty
speed 38400 baud; line = 0;
eol = M-^?; eol2 = M-^?;
-brkint ixany
$ stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol
= M-^?; eol2 = M-^?;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V;
flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -
igncr icrnl ixon -ixoff
-iuclc ixany imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0
cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase
-tostop -echoprt
echoctl echoke
$
```

Rys. 14. Zawartość okna terminala przed wydaniem polecenia `clear`

to bezpośrednio po wykonaniu polecenia `clear` będzie ono wyglądać tak:

```
$
```

Rys. 15. Okno terminala po wykonaniu polecenia `clear`

W trakcie sesji użytkownik systemu *N*X może czasowo wejść w rolę innego użytkownika. Służy do tego polecenie `su` (ang. *switch user* – przełącz użytkownika). Jeżeli polecenie to jest wydawane przez zwykłego użytkownika, system zażąda podania hasła. Po poprawnym uwierzytelnieniu dotychczasowe prawa użytkownika zostaną przysłonięte prawami użytkownika wskazanego w poleceniu `su`. Sytuacja ta tworzy możliwość pomyłek, zwłaszcza po wejściu w rolę superużytkownika, ponieważ polecenie `su root` nie prowadzi do zmiany znaku zachęty[28]. Aby sprawdzić, który użytkownik pierwotnie otworzył sesję, użyjemy polecenia `logname`, w celu zaś ustalenia efektywnych uprawnień – polecenia `whoami`. Powrót do poprzedniej roli uzyskujemy wydając polecenie `exit`.

```
$ su root
Password:     ← tu zostało wpisane hasło (niewidoczne)
$ logname
skrypt
$ whoami
root
$ exit
$ logname
skrypt
$ whoami
skrypt
$
```

Rys. 16. Użycie polecenia `su`. Użytkownik nominalny i efektywny.

Polecenie `whoami` jest szczególną postacią polecenia `who am i` (ang. *kim jestem*), dającego pełniejszą informację o efektywnym użytkowniku. Polecenie `who` (bez parametrów) podaje zaś informacje o wszystkich sesjach otwartych przez użytkowników systemu.

W ramach wprowadzenia do korzystania z terminala wspomnimy o poleceniu `bc`, które wywołuje interaktywny kalkulator o dowolnej precyzji, udostępniający również operacje logiczne i funkcje matematyczne. Poniżej przytoczono przykładowy przebieg konwersacji z programem. W celu przerwania pracy kalkulatora należy użyć polecenia klawiszowego eof (^D) – niewidocznego na ekranie.

[28] Na ogół polecenia tego używa się w postaci `su - nazwa_użytkownika`. Dzięki użyciu znaku łącznika - m.in. będzie podawany znak zachęty właściwy dla superużytkownika.

```
[root@localhost skrypt]# bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
a
0
a=2*5
a
10
b=2^8
b
256
a<b
1
a>b
0
sqrt(b)
16
[root@localhost skrypt]#
```

Rys. 17. Przykład użycia kalkulatora `bc`

Użytkownik systemu *N*X może korzystać z wielu możliwości uzyskiwania pomocy na temat dostępnych poleceń i sposobu ich użycia. Klasycznym mechanizmem jest tzw. *podręcznik* (ang. *manual*), dostępny poprzez polecenie `man`. Polecenie to, wydane z parametrem będącym nazwą interesującego polecenia, prezentuje jego opis sformatowany według ustalonych zasad[29].

```
HOSTNAME(1)              Narzędzia powłokowe GNU 2.0             HOSTNAME(1)

NAZWA
       hostname - pokazuje lub ustawia nazwę hosta systemu

SKŁADNIA
       hostname [nazwahosta]

OPIS
       Hostname  jest  programem służącym do nadawania bądź wyświetlania nazwy
       hosta.  Wywołany bez  argumentów  program  wyświetla  nazwę  aktualnego
       hosta.   Z jednym argumentem opisywane polecenie ustawia nazwę hosta na
       podany łańcuch.  Do ustawienia nazwy hosta  niezbędne  jest  posiadanie
       odpowiednich  Nazwa  hosta  jest  zwykle  nadawana  jednorazowo podczas
       startu systemu - zazwyczaj przez odczytanie zawartości pliku zawieraja-
       cego nazwę hosta, np. /etc/hostname.

OPCJE
       --help Wyświetla informację o stosowaniu programu i dostępnych opcjach,
              kończy pracę.

       --version
              Wyświetla numer wersji programu i kończy pracę.

ZGŁASZANIE BŁĘDÓW
       Błędy proszę zgłaszać, w jęz.ang., do <bug-sh-utils@gnu.org>.
```

[29] Na ogół opisy poszczególnych poleceń dostępne są w języku angielskim; wersje polskojęzyczne występują w niektórych dystrybucjach systemów *N*X.

```
COPYRIGHT
       Copyright © 1999 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is
       NO  warranty;  not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
       PURPOSE.

ZOBACZ TAKŻE
       uname(1), printenv(1), sh-utils(1).

OD TŁUMACZA
       Zaktualizowano i poszerzono wg dokumentacji Texinfo dla narzędzi powło-
       kowych GNU wersji 2.0.

maj 2000                              FSF                           HOSTNAME(1)
(END)
```

Rys. 18. Strona podręcznika dotycząca polecenia `hostname`

Szerszy opis źródeł informacji na temat poleceń systemu znajdzie Czytelnik w rozdziale 7.

5.6 Wylogowanie

Sesję kończy tzw. *wylogowanie się* (ang. *log-out*) użytkownika, które prowadzi do zlikwidowania jego logicznego połączenia z systemem, wyłączenia aktywnej powłoki i przywrócenia stanu, jaki istniał przed otwarciem sesji. Jeżeli stanem tym była otwarta sesja na komputerze `vhomer`, to przebieg wylogowania z komputera `venus` będzie wyglądał jak niżej:

```
[les@venus les]$ logout
$
```

Rys. 19. Przebieg wylogowania użytkownika

Ten sam efekt uzyskamy używając polecenia `exit` zamiast `logout`.

5.7 Zadania i ćwiczenia

Do wykonania przedstawionych tu zadań użytkownik winien posiadać co najmniej jedno własne konto w systemach *N*X, a w razie potrzeby powinien mieć możliwość skorzystania z uprawnień użytkownika root.

1. Zaloguj się do swojego systemu *N*X jako zwykły użytkownik. Sprawdź, jaki kod realizuje polecenie klawiszowe `kill`. Sprawdź działanie tego polecenia.
2. Mając otwartą sesję, po pojawieniu się zachęty użyj polecenia klawiszowego ^D. Uzasadnij zaobserwowane działanie systemu.
3. Poleceniem `cat` *`nazwa_pliku`* uruchom listowanie zawartości dowolnego obszernego pliku tekstowego. W trakcie listowania zatrzymaj je na chwilę, a potem wznów.
4. Znak F posiada kod ASCII 70_{10} (notacja dziesiętna). Oblicz kod znaku f.
5. Znak m posiada kod 109_{10}. Oblicz kod znaku M.
6. Kodem ASCII znaku sterującego ^Q jest 17_{10}. Przedstaw ten kod ósemkowo i szesnastkowo.
7. Czy znak o kodzie ASCII 16_{10} należy do podstawowego zbioru znaków?
8. Zapoznaj się bliżej z poleceniem `bc`, które stanowi tekstowy kalkulator. Za jego pomocą wyznacz liczby odpowiadające wielkościom 1 MB, 1 GB, 1 TB, 1 PB, 1 EB, 1 ZB, 1 YB. Zobacz ramkę "Komputerowe jednostki pojemności" w rozdziale 3.
9. Porównaj sposób działania poleceń `whoami` oraz `who am i`. Sprawdź, jak działają polecenia `who` i `w`.
10. Zapoznaj się z poleceniem `cal`. Wywołując je wielokrotnie z odpowiednimi parametrami, uzyskaj kalendarz na bieżący miesiąc, bieżący semestr i na przyszły rok akademicki. Poniedziałek ma być pierwszym dniem tygodnia.
11. Zapoznaj się z poleceniami `apropos` i `whatis`. Opracuj krótkie „ściągi” na temat składni i przeznaczenia tych poleceń w formacie takim, jaki wykorzystuje polecenie `man`.

6 Szata zdobi *N*X-a... czyli Powłoka

6.1 Postać polecenia powłoki

Polecenie powłoki (dalej: *polecenie*) wskazuje pojedynczą czynność lub sekwencję czynności[30] do wykonania przez system operacyjny komputera.

Polecenie systemu *N*X (ang. *command*) ma postać:

```
nazwa_polecenia [opcja]… [parametr]…
```

gdzie:

`nazwa_polecenia` wskazuje polecenie wbudowane powłoki, bądź jest nazwą pliku zawierającego program w postaci wykonywalnej lub skrypt powłokowy,

`opcja` jest rodzajem dwustanowego przełącznika, włączającego lub wyłączającego określoną funkcję polecenia; opcje kodowane są znakiem łącznika `-`, bezpośrednio po którym następuje jeden znak i/lub za pomocą podwójnego łącznika `--`, bezpośrednio po którym następuje ciąg znaków,

`parametr` najczęściej wskazuje obiekt, którego dotyczy wykonywane polecenie, np. plik zawierający dane wejściowe lub wyjściowe dla realizującego to polecenie programu.

Poszczególne elementy polecenia oddzielone są od siebie co najmniej jednym znakiem odstępu. Zmienne elementy polecenia zaznaczono drukiem pochylonym. Nawiasy kwadratowe `[` i `]` służą do zaznaczenia tych elementów polecenia, które mogą być pomijane. Wielokropek `...` wskazuje, iż poprzedzający go element składniowy może być powtórzony dowolnie wiele razy.

Opcje służą do modyfikowania sposobu wykonania polecenia. Opcją bardzo często dostępną w poleceniach systemów *N*X jest `--help` oznaczana również niekiedy jako `–h`. Wywołanie polecenia z opcją `--help` powoduje wyświetlenie skróconej informacji o sposobie jego użycia.

Separatorem poleceń jest średnik. Pozwala on oddzielić od siebie wiele poleceń umieszczonych w jednym wierszu. Z uwagi na czytelność zapisu nie jest to jednak zalecane. Wiersz polecenia kończymy wciśnięciem klawisza **<Enter>**.

[30] Takie polecenie nosi nazwę *skryptu powłokowego*. Będzie o tym szerzej mowa w rozdziale 8.

6.2 Przegląd powłok

Pierwszą dostępną w systemie UNIX była powłoka o nazwie `sh` opracowana przez S.R. Bourne'a. W kolejnych latach koncepcja ta doczekała się licznych modyfikacji, które zostały opublikowane przez D. Korna w postaci programu o nazwie `ksh`, a po dalszych usprawnieniach - przez B. Foxa pod nazwą `bash`. Ponieważ powłoka `bash` zawiera w sobie funkcje powłoki `sh`, to zastosowanie tej ostatniej zostało ograniczone do tych przypadków, w których rozbudowane możliwości nie są konieczne, za to istotne są małe wymagania co do zasobów systemu komputerowego.

Tabela 11. Ważniejsze powłoki systemów *N*X

Nazwa programu	Nazwa powłoki	Autor (instytucja, data opracowania)
`sh`	Bourne Shell	Stephen R. Bourne; (AT&T; lata 70.)
`csh`	C-Shell	Bill Joy; (University of California, Berkeley)
`ksh`	Korn Shell	Dawid Korn (AT&T; opublikowana w 1986 r.)
`tcsh`	---	Rozszerzona powłoka `csh`; Ken Greer (Carnegie Mellon University; 1981), Paul Placeway (Ohio State University; 1987) i in.
`bash`	Bourn Again Shell	„Ponowiona" powłoka Bourne'a; Brian Fox (Free Software Foundation – project GNU; 1989 r.)
`zsh`	Z-shell	Udoskonalona powłoka `bash`

Równolegle rozwijane były powłoki o składni poleceń wzorowanej na języku C. Powłoka `csh` dawała liczne ułatwienia w pracy interaktywnej, posiadała jednak wady powodujące, że do zastosowań nieinteraktywnych i tak była używana powłoka `sh`.

Spośród wymienionych powłok `bash`, `tcsh` i `zsh` rozwijane są na zasadach OpenSource. W tabeli poniżej podamy ocenę porównawczą funkcji dostępnych w różnych powłokach, opracowaną przez A. Taddei [21].

W dalszym ciągu skryptu skupimy się na właściwościach powłoki `bash` jako tej, z którą Czytelnik będzie się zapewne spotykał najczęściej.

Tabela 12. Ocena porównawcza powłok

Kryterium	Rodzina sh				Rodzina csh	
	sh	ksh	bash	zsh	csh	tcsh
Podatność na konfigurowanie	2	3	4	5	2	3
Wykonywanie poleceń	3	3	3	4	3	4
Uzupełnianie	0	3	4	5	3	4
Edycja wiersza	2	3	4	4	2	4
Podstawianie nazw	3	3	4	4	3	4
Historia poleceń	0	3	4	4	3	4
Przeadresowywanie i potoki	3	3	3	4	3	3
Poprawianie błędów maszynowych	0	0	0	3	0	3
Konfigurowanie zachęty	3	3	3	4	3	4
Sterowanie zadaniami	0	3	3	3	3	3
Sterowanie wykonaniem	3	3	3	3	3	3
Obsługa sygnałów	3	3	3	3	2	2

OCENY: 5 – bardzo dobra, 4 – dobra, 3 – dostateczna, 2 – słaba, 0 – brak właściwości

6.3 Powłoka domyślna

Bezpośrednio po uruchomieniu sesji uruchamiana jest powłoka określona w parametrach konta użytkownika jako siódme pole odpowiedniego wiersza pliku `/etc/passwd`. W przykładzie podanym poniżej dla użytkownika `skrypt` domyślna powłoka uruchamiana jest z pliku `/bin/bash`.

```
skrypt:x:501:100:skrypt:/home/skrypt:/bin/bash
```

Rys. 20. Dane konta użytkownika `skrypt`

Domyślny program powłoki może być wskazany podczas tworzenia konta użytkownika i zmieniony później we własnym zakresie przez użytkownika lub przez superużytkownika. Służy do tego celu polecenie `chsh`.

`chsh`	`chsh` *`nowy_program_powłoki`*
	Polecenie `chsh` (od ang. *change shell* – zmień powłokę) zmienia istniejący wpis w odpowiednim polu pliku /etc/passwd, umieszczając tam ścieżkę podaną jako *`nowy_program_powłoki`*. Od tej chwili powłoka określona w parametrze polecenia staje się powłoką domyślną użytkownika. Nie zmienia to jednak powłoki bieżącej. Wykaz powłok dostępnych w danym systemie znajduje się w pliku `/etc/shells`.

```
$ more /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/ksh
/bin/tcsh
/bin/csh
/bin/zsh
$ chsh /bin/sh
$
```

Rys. 21. Przykład użycia polecenia `chsh`

W przykładzie zamieszczonym powyżej najpierw sprawdziliśmy, jakie powłoki są dostępne w systemie. Wśród innych występuje tu program `/sbin/nologin`, który jest używany dla kont, do których logowanie się jest wykluczone; po uruchomieniu program ten wypisuje tylko odpowiedni komunikat i kończy pracę. Następnie zmieniliśmy powłokę domyślną na `/bin/sh`. Efekt zmiany będzie widoczny po następnym zalogowaniu się.

W czasie trwania sesji użytkownik może wywołać dowolną powłokę, a każda następna uruchamiana jest jako polecenie bieżącej powłoki. Wywołane powłoki tworzą stos. Wyjście z bieżącej powłoki wymaga wydania polecenia wbudowanego `exit`.

6.4 Znaki specjalnie traktowane przez powłokę

Następujące znaki są traktowane przez powłokę w sposób specjalny:

`;` `&` `(` `)` `|` `^` `<` `>` `newline` `space` `tab`
`$` `#` `'` `"` `` ` `` `\` `@` `*` `?` `[` `]` `-` `!`

W powłoce `csh` znakami specjalnymi są również `%` `{` `}` `~`.

Jeżeli zachodzi konieczność użycia znaku specjalnego w jego zwykłym znaczeniu (jako elementu napisu), znak ten musi być podany w sposób, który nazywamy *cytowaniem* (ang. *quotation*).

Najprostszym sposobem cytowania znaku jest jego poprzedzenie lewym ukośnikiem `\`. W przykładzie zamieszczonym poniżej pokazano różnice w interpretacji znaku gwiazdki. Użyta jako parametr polecenia, jest rozwijana przez powłokę do listy plików znajdujących się w katalogu domyślnym.

```
$ echo *
anakonda-ks.cfg grep install.log (pominięte)
$ # Komentarz: gwiazdka została zastąpiona nazwami wszyst-
kich plików
$ # w aktualnym katalogu
$ echo \*
*
$ # Komentarz: znaki \* zostały zinterpretowane jako *
$
```

Rys. 22. Zmiana znaczenia gwiazdki przez cytowanie

Gwiazdka cytowana przy pomocy lewego ukośnika stała się zwykłym znakiem gwiazdki. W podobny sposób pokażemy, jak zmienia się funkcja znaku średnika.

```
$ echo Pierwsze; echo Drugie
Pierwsze
Drugie
$ echo Pierwsze \; echo Drugie
Pierwsze; echo Drugie
$ echo \\
\
$
```

Rys. 23. Zmiana znaczenia średnika i lewego ukośnika poprzez cytowanie

Średnik użyty bez cytowania jest separatorem poleceń powłoki zapisanych w jednym wierszu, ale wykonywanych niezależnie. Średnik cytowany jest zwykłym znakiem przestankowym, a więc nie oddziela od siebie poleceń. Aby użyć znaku lewego ukośnika w jego pierwotnym znaczeniu, wystarczy go zacytować.

W ten sam sposób przywracamy podstawowe znaczenie znakowi zmiany wiersza, używając lewego ukośnika bezpośrednio przed wciśnięciem klawisza <**Enter**>.

```
$ echo \↵
> 123
123
$
```

Rys. 24. Kontynuacja zapisu polecenia w nowym wierszu

Fakt kontynuacji wiersza oznaczony został przez powłokę znakiem > (zamiast $ lub #).

Cytowanie za pomocą lewego ukośnika wymaga, aby był nim poprzedzony każdy cytowany znak. Przy większej liczbie cytowań zapis polecenia staje się jednak nieczytelny. Stosuje się wtedy cytowanie nie pojedynczych znaków, tylko fragmentów tekstów, poprzez umieszczenie ich w apostrofach. Dzięki temu wszystkie znaki pomiędzy apostrofami mają swoje zwykłe znaczenie.

```
$ echo Ile \(średnio\) kosztuje 1\$ \?
Ile (średnio) kosztuje 1$?
$ echo Ile '(średnio)' kosztuje '1$?'
Ile (średnio) kosztuje 1$?
$
```

Rys. 25. Przykłady cytowania dłuższych tekstów

Odmianą cytowania za pomocą apostrofów jest cytowanie za pomocą cudzysłowów. W tekście ujętym w cudzysłowy zachowują swoje znaczenie wszystkie znaki oprócz znaków dolara $ i akcentu `. Znaczenie znaku dolara zostanie omówione w związku ze zmiennymi powłoki. Znaki akcentu służą do zaznaczenia wyniku wykonania polecenia, który ma być w postaci tekstu wstawiony do zapisu innego polecenia. Dokładnie rzecz biorąc, cytowanie wyniku polecenia umieszcza w miejscu cytowania tekst wyprowadzany przez to polecenie do jego standardowego strumienia wyjściowego (zobacz 6.5.2).

```
$ echo Teraz mamy: `date`
Teraz mamy: czw lis 13 11:39:39 CET 2003-11-13
$ echo Jestem: `id`
Jestem: uid=500(les) gid=500(les) grupy=500(les)
$ echo Katalog bieżący: `pwd`
Katalog bieżący: /home/les/pliki
$
```

Rys. 26. Przykłady cytowania wyników poleceń

Tabela 13. Zestawienie sposobów cytowania

Znak cytowania	Interpretacja	Przykład użycia
\	Cytowanie jednego znaku	`\*`
‘	Cytowanie całego tekstu	`‘To są nawiasy ( i )‘`
″	Cytowanie całego tekstu, z wyjątkiem znaków $ i `	`″Zalogowany: $LOGNAME″`
`	Wstawianie wyniku wykonania polecenia	`echo `pwd``

W odniesieniu do cytowania z użyciem apostrofu, cudzysłowu i akcentu, znak otwierający cytowany tekst musi być identyczny ze znakiem zamykającym.

6.5 Mechanizmy powłoki

6.5.1 Warunkowe wykonanie polecenia

Powłoka posiada możliwość uzależnienia wykonania polecenia od kodu powrotu poprzednio zrealizowanego polecenia. Służą do tego operatory `&&` oraz `||`.

Tabela 14. Wartości kodu powrotu

Kod powrotu	Znaczenie	Uwagi
`0`	Wykonanie bezbłędne	
`>0`	Podczas wykonywania polecenia wystąpił błąd	Wartość kodu powrotu określa rodzaj błędu
`128+n`	Podczas wykonywania polecenia wystąpił sygnał o kodzie n	

`&&`	*`polecenie1 && polecenie2`*
	`polecenie2` będzie wykonane, o ile kod powrotu z polecenia *`polecenie1`* jest równy zero.

`||`	*`polecenie1 || polecenie2`*
	`polecenie2` będzie wykonane, o ile kod powrotu z polecenia *`polecenie1`* jest różny od zera.

Możliwości dostarczane przez operatory zilustrujemy poniżej.

```
$ mkdir katalog
$
$ mkdir katalog
mkdir: nie można utworzyć katalogu ‘katalog’: Plik istnieje
$ rmdir katalog
$ mkdir katalog && echo Katalog utworzono.
Katalog utworzono.
$ mkdir katalog || echo Niestety...
mkdir: nie można utworzyć katalogu ‘katalog’: Plik istnieje
Niestety...
$
```

Rys. 27. Przykłady warunkowego wykonania poleceń

Ponowna próba utworzenia katalogu o nazwie `katalog` z oczywistych względów nie mogła się powieść. Po usunięciu tego katalogu do jego tworzenia zastosowano polecenia warunkowe. Pierwsze sygnalizuje poprawne wykonanie,

drugie zaś – odmowę. Jeśli potraktujemy polecenie warunkowe jak każde inne polecenie i dokonamy złożenia warunków, to otrzymamy polecenie uniwersalne.

```
$ rmdir katalog
$ mkdir katalog && echo Katalog utworzono. || echo Niestety...
Katalog utworzono.
$ mkdir katalog && echo Katalog utworzono. || echo Niestety...
mkdir: nie można utworzyć katalogu ‘katalog’: Plik istnieje
Niestety...
$
```

Rys. 28. Złożone użycie poleceń warunkowych

6.5.2 Język powłoki

Powłoka pozwala na konstruowanie złożonych struktur programowych, udostępniając m.in.:

- zmienne powłoki,
- struktury sterujące wykonywaniem poleceń,
- polecenia wbudowane w powłokę
- mechanizm funkcji,
- mechanizm skryptów.

Dla opisu języka programowania akceptowanego przez powłokę wprowadzimy tzw. *metajęzyk*. Stanowi on zbiór definicji elementów składniowych, z których budowane są polecenia powłoki.

Tabela 15. Symbole metajęzyka powłoki

Nazwa	Definicja
puste	`SPACJA TAB`
słowo (token)	Ciąg znaków traktowany przez powłokę jako jeden element składniowy
nazwa (identyfikator)	Słowo zawierające tylko litery, cyfry i znak podkreślenia
metaznak	Każdy ze znaków służących do separacji słów: `| & ; ( ) < > SPACJA TAB`
operator sterujący	Każdy z symboli: `|| & && ; ;; ( ) | NEWLINE`
słowo zarezerwowane	`! case do done elif else esac fi for function if in select then until while { } time [ ]`
potok	`[time [-p]] [!] polecenie1 [| polecenie2]...` Opcja `!` neguje kod powrotu ostatniego polecenia

Znaki `{` i `}` są traktowane przez powłokę jak polecenia – obowiązuje separacja za pomocą `SPACJI`.

Kolejne definicje metajęzyka przedstawimy w ramkach.

`lista`	Ciąg jednego lub wielu potoków oddzielonych jednym z operatorów: `;` `&` `&&` `||` i opcjonalnie zakończony jednym z operatorów: `;` `&` `NEWLINE` • Polecenie zakończone znakiem `&` jest realizowane niezależnie w tle – powłoka ustawia kod powrotu na 0 i nie czeka na jego zakończenie. • Polecenia zakończone znakiem `;` są wykonywane sekwencyjnie – powłoka czeka na zakończenie każdego polecenia przed rozpoczęciem wykonywania następnego; kod powrotu listy jest kodem powrotu ostatniego wykonanego polecenia • Dwa polecenia połączone operatorem `&&` - wykonanie drugiego z poleceń następuje wyłącznie po **poprawnym** wykonaniu pierwszego. • Dwa polecenia połączone operatorem `||` - wykonanie drugiego z poleceń następuje wyłącznie po **niepoprawnym** wykonaniu pierwszego.

Powłoka udostępnia mechanizm zmiennych, dzięki czemu możliwa jest realizacja różnorodnych algorytmów przetwarzania, w tym podejmowanie decyzji na podstawie badania warunków. Wśród zmiennych powłoki wyróżniamy:

- *Zmienne standardowe* (o wartościach ustalanych przez powłokę),
- *Zmienne specjalne* (związane ze środowiskiem wykonywania poleceń powłoki),
- *Zmienne użytkownika* (definiowane w treści poleceń powłoki).

Tabela 16. Przegląd zmiennych standardowych

Nazwy zmiennych standardowych	
`?`	kod powrotu ostatnio wykonanego polecenia (zapis dziesiętny); badanie kodu powrotu np. za pomocą poleceń `if` i `while`
`#`	liczba parametrów pozycyjnych powłoki (dziesiętnie)
`$`	dziesiętny numer procesu bieżącej powłoki; ponieważ numery procesów są unikalne, wartość ta może być wykorzystana do tworzenia unikalnych nazw plików tymczasowych
`!`	dziesiętny numer ostatniego procesu wykonywanego w tle
`0`	nazwa wykonywanego polecenia („zerowy” argument)
`1, 2, ...`	pierwszy, drugi,... argument polecenia
`*`	wszystkie argumenty polecenia jako jeden ciąg znaków
`@`	wszystkie argumenty polecenia jako osobne ciągi znaków

Niektóre z powyższych zmiennych mają praktyczne znaczenie tylko wewnątrz skryptów.

W celu odwołania się do wartości zmiennej należy użyć jej nazwy poprzedzonej znakiem dolara `$`.

```
$ echo $?
0
$ echo $$
781
$
```

Rys. 29. Przykłady użycia zmiennych standardowych

Tabela 17. Przegląd zmiennych specjalnych

Nazwy zmiennych specjalnych	
MAIL	ścieżka do pliku otrzymanej poczty bieżącego użytkownika
HOME	ścieżka do katalogu macierzystego (prywatnego) bieżącego użytkownika
PATH	lista katalogów przeglądanych w kolejności zapisu, w celu odnalezienia pliku wykonywalnego; separatorem katalogów jest średnik
PS1	Podstawowy ciąg znaków wyprowadzany jako zachęta (np. ‘$‘)
PS2	Pomocniczy ciąg znaków wyprowadzany jako zaproszenie do kontynuacji wprowadzania (zachęta niższego poziomu)

Poniżej pokażemy przykłady wykorzystania zmiennych specjalnych LOGNAME, HOME i PS1.

```
[root@vhomer ~] # who am i
root   tty1     Nov 19 11:26
[root@vhomer ~] # echo Katalog macierzysty uzytkownika \
”$LOGNAME\”: $HOME
Katalog macierzysty uzytkownika "root": /root
[root@vhomer ~] # echo $PS1
[\u@\h \W] \#
[root@vhomer ~] #
```

Rys. 30. Przykłady użycia zmiennych specjalnych

Symbole \u, \h i \W w definicji zachęty PS1 oznaczają odpowiednio nazwę hosta, nazwę zalogowanego użytkownika i lokalną nazwę katalogu roboczego.

$*nazwa*	$*nazwa*
	Jest to definicja zmiennej użytkownika. *nazwa* musi być różna od zmiennych standardowych i specjalnych (zalecane jest pisanie nazw zmiennych dużymi literami).

Na zmiennych użytkownika, oprócz posługiwania się ich wartościami, można wykonywać operacje deklarowania i przypisywania wartości, do czego służy polecenie wbudowane set.

set	[set] *nazwa_zmiennej*=[*wartość*]
	Polecenie `set` (ang. *set* – ustal) jest wbudowanym poleceniem powłoki, służącym do deklarowania wskazanej zmiennej i/lub nadania jej wartości podanej jako parametr. Wartościami zmiennych są ciągi znaków. Brak *wartości* oznacza pusty ciąg znaków.

```
$ set PI=3.14
$ echo $PI
3.14
$ ALFA=127.234
$ echo $ALFA
127.234
$ cd /user/local
$ UZYTK=$LOGNAME
$ KAT=`pwd`
echo Nazywasz się $UZYTK. Używasz katalogu $KAT.
$ Nazywasz się skrypt. Używasz katalogu /usr/local.
$
```

Rys. 31. Przykłady użycia zmiennych użytkownika

Polecenia powłoki można składać w struktury bardziej skomplikowane, zwane *poleceniami złożonymi*.

Tabela 18. Struktura poleceń złożonych

Polecenia złożone	
(*lista*)	Lista poleceń jest wykonywana w podpowłoce. Skutki dokonanych podstawień wartości zmiennych nie są widoczne po zakończeniu wykonywania polecenia złożonego. Kod powrotu listy jest równy kodowi powrotu ostatniego wykonanego polecenia listy.
{ *lista;* }	Lista jest wykonywana w środowisku bieżącej powłoki (polecenie grupowe). Znaki `{}` muszą być oddzielone od listy białą spacją.
((*wyrażenie*))	Obliczana jest wartość wyrażenia; jeśli jest ona niezerowa to kod powrotu jest równy `0`, w przeciwnym razie równy `1`.

```
$ (cd /home; ls)
les karen miki
$ { cd ~; mkdir nowe; chmod 600 nowe; ls –l nowe }
razem 8
drw------   2 root root 4096 lis 19 13:58 nowe
$
```

Rys. 32. Przykłady list poleceń

W pierwszym przykładzie do wykonania listy poleceń została użyta odrębna podpowłoka. W drugim zaś nawiasy klamrowe posłużyły tylko do zgrupowania poleceń we wspólny blok.

Język powłoki byłby niekompletny, gdyby zabrakło w nim możliwości powtarzania poleceń i ich grup oraz konstrukcji pozwalających na sterowanie przebiegiem wykonywania.

Polecenia iteracyjne	
`for` *(postać 1)*	`for` *nazwa* [`in` *słowo*]`...;` `do` *lista*`;` `done`
	Lista wymieniona po słowie „in” jest rozwijana, tworząc listę elementów. Pod *nazwa* podstawiana jest kolejno każda wartość z listy elementów, a następnie wykonywana jest *lista* poleceń. Pominięcie fragmentu „in” powoduje, że pętla wykonywana jest jeden raz dla każdego parametru pozycyjnego (konstrukcja równoważna *in $**).
`for` *(postać 2)*	`for` *((wyr1;wyr2;wyr3));* `do` *lista;* `done`
	1. obliczana jest wartość *wyr1*, 2. obliczana jest wartość *wyr2*; 3. jeśli *wyr2* ≠0, to wykonywana jest *lista*, obliczana wartość *wyr3* i następuje powrót do pkt. 2 4. jeśli *wyr2* =0, to wykonywanie polecenia `for` kończy się; kod powrotu z polecenia for jest kodem powrotu ostatniego polecenia. UWAGA: opuszczenie *wyrX* jest interpretowane jako 1.
`while`	`while` *lista1;* `do` *lista2;* `done`
	lista2 jest wykonywana tak długo, jak długo *lista1* zwraca zero
`until`	`until` *lista1;* `do` *lista2;* `done`
	lista2 jest wykonywana dopóty, dopóki *lista1* zwraca wartość niezerową

W pierwszym przykładzie zliczamy i wyświetlamy liczbę wierszy w plikach o nazwach określonych wzorcem `/etc/*.conf`. Liczba takich plików nie jest z góry znana.

```
$ for i in /etc/*.conf; do wc –l $i; done
 4     /etc/esd.conf
 2     /etc/fam.conf
 1     /etc/host.conf

(pozostałe wiersze pominięte)

$ for i in /etc/passwd /etc/shadow; do wc –l $i; done
 32 /etc/shadow
```

```
$ set A=5
$ for ((B=A+5; B<17; B++)); do echo B=$B; done
B=10
B=11
B=12
B=13
B=14
B=15
B=16
$
```

Rys. 33. Przykłady użycia konstrukcji `for`

W przykładzie drugim wykonaliśmy analogiczną operację, lecz zbiór plików określony był przez wyliczenie. Przykład trzeci obrazował zastosowanie wyrażenia do sterowania iterowaniem poleceń.

Polecenia sterujące	
case	`case` *`słowo`* `in [ [(] `*`wzorzec`*` [|`*`wzorzec`*`]...) `*`lista`*`;; ] ... esac` Polecenie `case` (ang. *case* – przypadek) poleca porównywać podane *`słowo`* z kolejnymi wzorcami. Jeśli zostanie znaleziony `wzorzec` pasujący do *`słowa`*, to wykonywana jest lista poleceń umieszczona bezpośrednio po tym wzorcu. Jeśli wzorzec nie zostanie dopasowany, to wykonywana jest *`lista`* umieszczona bezpośrednio po wzorcu zastępczym `*)`, o ile został zdefiniowany. W pozostałych przypadkach nie jest wykonywane żadne polecenie.

```
$ PRZELACZNIK=2
$ case $PRZELACZNIK in \
> 1) echo Jedynka! \
> ;; \
> 2) echo Dwójka! \
> ;;\
> *) echo Coś całkiem innego! \
> esac
Dwójka!
$
$ PRZELACZNIK=12
$ case $PRZELACZNIK in \
> [1-10]) echo Pierwsza dziesiątka.;; \
> *) echo Poza konkurencją!;; \
> esac
Poza konkurencją!
$
```

Rys. 34. Przykłady użycia konstrukcji `case`

W drugim z przykładów zaprezentowano możliwość posługiwania się listą wzorców: dowolna wartość badanej zmiennej o nazwie PRZELACZNIK w zakresie od 1 do 10 powoduje wyświetlenie napisu „Pierwsza dziesiątka.”. Każda inna natomiast – napisu „Poza konkurencją!”.

Polecenia sterujące (ciąg dalszy)	
if	`if` *`lista1;`* `then` *`lista2;`* `[elif` *`lista3;`* `then` *`lista4`*`;]...` `[else` *`lista5;`*`] fi`
	1. wykonywana jest *`lista1`*, 2. jeśli kod powrotu =0, to wykonywana jest *`lista2`*, 3. w przeciwnym razie wykonywana jest każda *`lista3`*, aż do wystąpienia kodu powrotu =0, po czym wykonywana jest odpowiadająca jej *`lista4`*, 4. Jeżeli wszystkie *`listy3`* zwróciły niezerowy kod powrotu, wykonywana jest *`lista5`*, o ile istnieje. UWAGA: kod powrotu z polecenia `if` jest równy kodowi powrotu ostatniego wykonanego polecenia lub zero, gdy żaden z warunków nie był spełniony.

test	`test` *`wyrażenie`* `[` *`wyrażenie`* `]`
	Polecenie `test` bada `wyrażenie` i zwraca odpowiedni kod powrotu. Operatory: `-a` and `-o` or `!` not Badanie typu pliku: `-s` *`plik`* czy istnieje i jest niezerowy `-f` *`plik`* czy istnieje i jest zwykłym plikiem `-d` *`plik`* czy istnieje i jest katalogiem `-r` *`plik`* czy istnieje, a wywołujący ma prawo czytania `-w` *`plik`* czy istnieje, a wywołujący ma prawo zapisu `-x` *`plik`* czy istnieje, a wywołujący ma prawo wykonania *`plik1`* `–nt` *`plik2`* plik1 jest nowszy niż plik2 *`plik1`* `–ot` *`plik2`* plik1 jest starszy niż plik2 Badanie relacji arytmetycznych: *`w1`* `–eq` *`w1`* (oznacza: w1 = w2) *`w1`* `–ge` *`w2`* (w1 ≥ w2) *`w1`* `–gt` *`w2`* (w1 > w2) *`w1`* `–le` *`w2`* (w1 ≤ w2) *`w1`* `–lt` *`w2`* (w1 < w2) *`w1`* `–ne` *`w2`* (w1 ≠ w2)

	Badanie wyrażeń tekstowych: `t` tekst niezerowy `-z t` tekst ma zerową długość `-n t` tekst ma niezerową długość `t1 = t2` teksty t1 i t2 są identyczne `t1 != t2` teksty t1 i t2 są różne

```
$ WIEK=150
$ if [ $WIEK -gt 50 ]; then echo "Zabytek"; \
> else echo "Nie takie stare. ;fi
Zabytek
$
$ chmod –R 400 /var/log
$ if [ !–w /var/log ]; then echo Brak możliwości zapisu \
do kroniki; fi
Brak możliwości zapisu do kroniki
$
```

Rys. 35. Przykłady użycia konstrukcji sterującej `if`

Ostatnią z ważnych konstrukcji językowych powłoki jest funkcja. Umożliwia ona wykonywanie powtarzalnych sekwencji poleceń, bez potrzeby wielokrotnego ich wprowadzania.

Funkcje	
`function`	`[ function ]` *`nazwa`* `() { ` *`lista;`* `}` Definiowanie funkcji o nazwie *`nazwa`*. Treścią funkcji są polecenia zawarte w liście *`lista`*. Wykorzystywane mogą być parametry pozycyjne. Kod powrotu dla funkcji jest kodem ostatnio wykonanego polecenia.
`(wywołanie funkcji)`	*`nazwa`*

```
$ function listowanie() { ls –la |grep l* | wc –l; }
$ listowanie
   1
$
```

Rys. 36. Przykład definicji i wywołania funkcji

6.5.3 Strumienie, potoki i filtry

Po utworzeniu, proces korzysta z trzech standardowych kanałów komunikacji z otoczeniem, nazywanych *strumieniami* (ang. *streams*):

a) Standardowego strumienia wejściowego (**stdin**) o deskryptorze pliku równym 0,
b) Standardowego strumienia wyjściowego (**stdout**) o deskryptorze pliku równym 1,
c) Standardowego strumienia diagnostycznego (**stderr**) o deskryptorze pliku równym 2.

Strumienie **stdin** i **stdout** służą odpowiednio jako kanały dla danych wejściowych i wyjściowych, strumień **stderr** jest kanałem wyjściowym dla komunikatów o błędach. Jeśli nie ustalono inaczej, wszystkie strumienie związane są z terminalem, w którym dany proces został uruchomiony.

Każdy strumień danego procesu może być, niezależnie od pozostałych, związany z:

- Plikiem zwykłym lub specjalnych (reprezentującym urządzenie),
- Strumieniem innego procesu.

Czynność wiązania strumienia z innym niż terminal źródłem danych lub miejscem ich przeznaczenia nazywamy *przeadresowaniem* lub *przekierowaniem* strumienia.

Przeadresowanie strumienia wejściowego do pliku zwykłego – operator <

```
$ sort <plik –o wynik
$
```

Rys. 37. Przeadresowanie strumienia `stdin` do pliku

Polecenie `sort` powyżej, zamiast z terminala, będzie pobierać dane z pliku o nazwie `plik`. Dla użycia operatora < istotne jest, aby wskazany plik istniał.

Przeadresowanie strumienia wyjściowego do pliku zwykłego – operatory > i >>

Również strumień wyjściowy polecenia można powiązać z plikiem. Operator > zapewnia umieszczenie danych wyjściowych we wskazanym pliku. Jeśli istniał on wcześniej, to jego poprzednia zawartość zostanie usunięta, a dane wyjściowe będą w nim zapisywane od początku pliku. Jeśli plik nie istniał, to zostanie utworzony. Operator >> zapewnia dopisywanie danych na końcu wskazanego pliku (bez usuwania poprzedniej zawartości), jeśli istniał on wcześniej. Jeśli plik nie istniał, to działanie operatora jest takie samo jak operatora >.

```
$ cat >dopisywanie
Aaa
Bbb
Ccc
^D
$ cat >>dopisywanie
Ddd
Eee
^D
$ more dopisywanie
Aaa
Bbb
Ccc
Ddd
Eee
$
```

Rys. 38. Przykłady przeadresowania strumienia `stdout`

Niekiedy zachodzi potrzeba wyłączenia wyjścia komunikatów diagnostycznych na terminal. Przyczyną może być chęć eliminacji komunikatów nieistotnych lub konieczność zachowania ich w pliku do celów późniejszej analizy. Aby przeadresować strumień stderr, używamy operatora 2>, w którym cyfra oznacza deskryptor pliku strumienia stderr, a znak większości – znany już symbol przeadresowania.

```
$ zupa
-bash: zupa: command not found
$ zupa 2>bledy
$ more bledy
-bash: zupa: command not found
$
```

Rys. 39. Przeadresowanie strumienia `stderr`

Wywołując nieistniejące polecenie, sprowokowaliśmy powłokę do sygnalizacji błędu: `-bash: zupa: command not found`. Kolejne wywołanie, z przeadresowaniem strumienia diagnostycznego do pliku, spowodowało, że komunikat o błędzie na terminalu się nie pojawił. Można go jednak było odtworzyć z pliku `bledy`.

Pozostaje do wyjaśnienia, jak powiązać strumienie stdout i stderr z tym samym plikiem tak, aby zachować chronologię operacji wyjściowych. Do tego celu służy operator `&1` wskazujący plik związany ze strumieniem o deskryptorze pliku 1, tj. strumieniem stdout. Jego użycie ma postać: `polecenie > plik_wyjściowy 2>&1`. Istotna jest przy tym kolejność przeadresowywania strumieni.

Przeadresowanie strumienia do pliku specjalnego

W celu powiązania strumienia standardowego z plikiem specjalnym, należy zamiast nazwy pliku wskazać nazwę urządzenia reprezentowanego przez ten plik. Mogą być to np. urządzenia `/dev/tty`xx, gdzie xx jest numerem terminala, `/dev/null`, `/dev/zero` i inne, jak pokazano w przykładach poniżej.

```
$ echo "Ahoj!" >/dev/tty07
$ oblicz 2>/dev/null
$
```

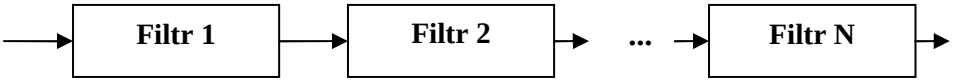
Rys. 40. Przykłady przeadresowania strumieni standardowych do urządzeń

Przeadresowywanie strumieni do innych strumieni

Filtrem (ang. *filter*) nazywamy program (polecenie), który kontaktuje się ze swoim otoczeniem poprzez standardowy strumień wejściowy i wyjściowy. Innymi słowy, filtr pobiera dane wejściowe ze standardowego strumienia wejściowego i wyprowadza wyniki do standardowego strumienia wyjściowego.

Rys. 41. Struktura filtru

Zestawiając wyjście pierwszego filtru z wejściem następnego, otrzymujemy tzw. *potok* (ang. *pipe*), co ogólnie przedstawimy na rysunku:



Rys. 42. Budowa potoku

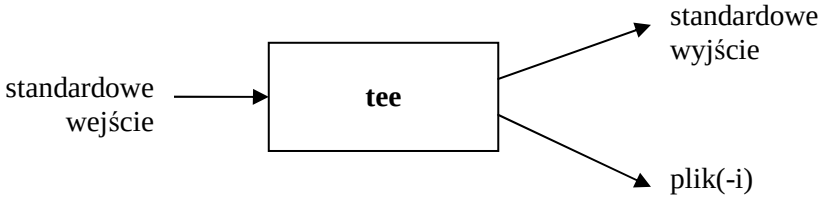
Zapis poleceń tworzących potok przyjmuje postać:
`filtr1 | filtr2 | ... filtrN`. Kreska pionowa | jest tu operatorem łączącym kolejne filtry w polecenia złożone.

```
$ echo Ten katalog zawiera `ls | wc –l ` plików.
Ten katalog zawiera 8 plików.
$ more /etc/passwd | grep root
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
$ more /etc/passwd | sort | more
...
(kolejne ekrany zawierające wiersze pliku /etc/passwd posortowane wg nazw użytkowników)
...
$
```

Rys. 43. Przykłady potoków

Polecenie `echo Ten katalog zawiera `ls | wc –l ` plików.` zawierało wynik wykonania innego polecenia, które z kolei było potokiem złożonym z polecenia listowania katalogu bieżącego i polecenia zliczającego liczbę uzyskanych wierszy. Potok `more /etc/passwd | grep root` wybierał z pliku `/etc/passwd` te wiersze, w których występował ciąg znaków `root`. Ostatni potok wyprowadzał na terminal zawartość pliku `/etc/passwd` posortowaną według nazw użytkowników.

Szczególną postacią filtru jest tzw. *trójnik* (ang. *tee*), który oprócz strumieni stdin i stdout korzysta z pliku (lub plików), do którego (lub których) kopiuje dane wyprowadzane do stdout.



Rys. 44. Działanie trójnika

Definicja polecenia `tee` przedstawiona jest dalej.

tee	tee [-a] [*plik*]...
	Polecenie tee jest filtrem, który czerpie dane wejściowe ze standardowego strumienia wejściowego oraz wyprowadza wyniki do standardowego strumienia wyjściowego oraz kopiuje je do jednego (lub większej liczby) pliku wskazanego poprzez parametry. Opcja –a oznacza, że dane wyjściowe są dopisywane na końcu pliku (-ów).

```
$ cat /etc/passwd | sort | tee /etc/passwd.sorted | more
```
(na terminalu wyświetlone są kolejne ekrany zawierające wiersze pliku `/etc/passwd` *posortowane wg nazw użytkowników; dodatkowo – te same posortowane wiersze zapisane są w pliku* `/etc/passwd.sorted`*)*
```
$
```

Rys. 45. Przykład użycia trójnika

Użyteczną funkcją powłoki jest definiowanie nazw zastępczych, czyli tzw. *aliasów* poleceń. Dzięki temu polecenia, których zapis wymaga podania większej liczby znaków (np. opcji i parametrów) można wywoływać w sposób wygodny.

alias	alias [*nazwa*[=*wartość*]]...
	Polecenie wywołane bez parametru wypisuje istniejące definicje aliasów. Jeżeli podano nazwę i wartość, to tworzony jest nowy alias o wskazanej nazwie i wartości. Jeśli podano tylko nazwę, to wyświetlana jest definicja aliasu o tej nazwie, o ile taki alias istnieje.

Aby zapoznać się ze znanymi powłoce definicjami nazw aliasowych, należy użyć polecenia alias bez parametrów. Aby usunąć zdefiniowany alias, stosujemy polecenie unalias.

unalias	unalias [-a] [*nazwa*]...
	Polecenie to usuwa wskazane nazwy aliasowe z listy aliasów. Podanie opcji –a powoduje usunięcie wszystkich definicji aliasów.

W przykładzie zamieszczonym poniżej zdefiniujemy i sprawdzimy nowy alias, a następnie go usuniemy i ponownie sprawdzimy listę aliasów.

```
[root@vhomer ~]# alias dir='ls -l'
[root@vhomer ~]# alias
alias cp='cp -i'
alias dir='ls -l'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias
--show-dot --show-tilde'
[root@vhomer ~]# unalias dir
[root@vhomer ~]# alias
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias
--show-dot --show-tilde'
[root@vhomer ~]# alias ll
alias ll='ls -l --color=tty'
[root@vhomer ~]#
```

Rys. 46. Zastosowanie poleceń `alias` i `unalias`

6.6 Zadania i ćwiczenia

1. Sprawdź w pliku `/etc/passwd` jaka jest Twoja powłoka domyślna. Wylistuj powłoki dostępne w Twoim systemie, a następnie dokonaj chwilowej zmiany tej powłoki na inną. Przywróć poprzednią powłokę. Teraz dokonaj trwałej zmiany powłoki domyślnej. Sprawdź poprawność tego ustawienia poprzez wylogowanie się i ponowne zalogowanie**.** Uwaga: nie wybieraj programu powłoki o nazwie `nologin`, `noshell` itp.
2. Za pomocą polecenia `echo` i cytowania pojedynczych znaków wyprowadź napisy oznaczone literami `a.` – `e.` (każdy w osobnym wierszu, z zachowaniem podanej pisowni):
   a. `Wykorzystane materiały eksploatacyjne (tonery, kartridże itp.) należy utylizować zgodnie z prawem.`
   b. `Do znaków o specjalnym znaczeniu`

należą m.in. ; & ( ) ^ < > .
c. Na wystawie widniała cena: $11,99.
d. Znak '#' bywa używany zamiast słowa "numer" lub "liczba".
e. Znak "@" oznacza w języku angielskim 'at' czyli przy', 'w miejscu' itp.

3. Każdy napis z poprzedniego zadania wyprowadź z wykorzystaniem cytowania ciągu znaków.
4. Wyprowadź na terminal napis: `Wykonano dnia` *`data_i_czas`*. Wstawiana data i czas winny być ustalone programowo (bez udziału użytkownika), w chwili wykonywania polecenia.
5. Wyprowadź na terminal dwuwierszowy napis:
   `To jest katalog:` *`"bieżący katalog"`*.
   `Ten system nazywa się:` *`"nazwa komputera"`*.
   Uwaga: katalog i nazwa komputera winny być wyświetlone w cudzysłowach. Nazwę bieżącego katalogu poda polecenie `pwd`, a nazwę komputera - `uname –n` .
6. Napisz polecenia z warunkowym wykonaniem, które w bieżącym katalogu mają:
   - usunąć plik `zbedny` (za pomocą polecenia `rm` *`nazwa_pliku`*),
   - w razie powodzenia, wylistować ze szczegółami zawartość katalogu bieżącego (za pomocą polecenia `ls –l` *`nazwa_katalogu`*),
   - w razie niepowodzenia, wypisać komunikat: `"Usunięcie katalogu nie powiodło się"`. Sprawdź poprawność rozwiązania, gdy plik `zbedny` istnieje i gdy nie istnieje.
7. Napisz polecenie warunkowe, które wylistuje na konsoli zawartość pliku `omega`. W razie powodzenia (np. braku takiego pliku), wyprowadź napis „`Udało się (plik został znaleziony"`. W razie niepowodzenia, wylistuj w krótkim formacie nieukryte pliki bieżącego katalogu. Sprawdź to polecenie przy braku i przy istnieniu pliku `omega`.
8. Wyprowadź do pliku `kalendarze` kalendarz na rok poprzedni, bieżący i przyszły. Kalendarze mają być oddzielone od siebie kreską poziomą złożoną z 40 znaków łącznika (minus).
9. Zadanie poprzednie zmodyfikuj tak, aby w trakcie wy-

prowadzania kalendarzy do pliku, były one równocześnie wyświetlane na terminalu, z zatrzymywaniem po każdej stronie.

10. Na końcu pliku `kalendarze` umieść linię poziomą złożoną z 41 znaków równości i dwa puste wiersze, a poniżej nich dopisz dane trzech najbliższych kolokwiów (nazwa przedmiotu, data, godzina rozpoczęcia i numer sali).
11. Zaprojektuj polecenie, które dla każdej nazwy dnia tygodnia wyprowadzi tekst „`Dzień jak co dzień`".
12. Napisz polecenie, które dla każdej nazwy miesiąca wyprowadzi w osobnym wierszu jego nazwę i liczbę dni w danym miesiącu. Zastosuj konstrukcję `case` z minimalną liczbą przypadków. Dla lutego liczbę dni podaj jako `28 lub 29`.
13. Opracuj funkcję, która na podstawie numeru roku obliczy liczbę dni w tym roku. Działanie funkcji sprawdź dla kilku danych, obejmujących lata zwykłe i przestępne.
14. Napisz polecenie, które oblicza wartości kolejnych elementów *ciągu Fibonacciego* i wyświetla na terminalu tylko elementy parzyste z podaniem ich indeksów.
    Uwaga: ciąg Fibonacciego ma tę własność, że dla każdego $i>1$, element następny jest sumą dwóch elementów poprzednich, tj. $a_i = a_{i-2} + a_{i-1}$. Elementy początkowe mają wartości $a_0=1$, $a_1=1$.
15. Napisz funkcję, która dla zadanej wartości zmiennej powłoki `N` obliczy wartość `N!` korzystając z iteracyjnej metody obliczeń. Sprawdź funkcję dla kilku wartości `N`.
16. Zdefiniuj alias o nazwie `dir` zastępujący polecenie `ls -la`. Wypróbuj jego działanie, a następnie zlikwiduj, sprawdzając skuteczność tej likwidacji.

7 Mów do mnie jeszcze... czyli Opis wybranych poleceń

7.1 Klasyfikacja poleceń

Jedna z możliwych klasyfikacji poleceń systemu *N*X bazuje na lokalizacji kodu implementującego dane polecenie; wyróżniamy tu więc:

- *polecenia wbudowane*, których kod stanowi część powłoki,
- *polecenia zewnętrzne*, których kody znajdują się w odrębnych plikach.

Zbiór i sposób wywoływania dostępnych poleceń wbudowanych zależy od używanej powłoki. Nazwa polecenia należącego do drugiej klasy jest równocześnie nazwą pliku przechowującego kod implementujący to polecenie. W szczególności, do drugiej klasy należą programy użytkowe i wszelkie programy wytworzone przez samego użytkownika.

Niezależnie od powyższej klasyfikacji, polecenia systemu *N*X można podzielić na klasy, zależnie od przeznaczenia (realizowanej funkcji). Wyróżniamy tu:

- *polecenia informacyjne*, w tym
  - dostarczające informacji o systemie i użytkownikach,
  - dostarczające informacji o dostępnych poleceniach,
- *polecenia komunikacji między użytkownikami*,
- *polecenia zarządzania urządzeniami systemu*, w tym ich tworzenia, modyfikacji, usuwania i kontroli stanu,
- *polecenia zarządzania systemem plików*, w tym ich tworzenia, modyfikacji, naprawiania, usuwania i kontroli stanu całego systemu oraz zarządzania samymi plikami (tworzenie, usuwanie, zmiana atrybutów, wyszukiwanie itp.),
- *polecenia przetwarzania plików tekstowych*, w tym ich tworzenia, modyfikacji i porządkowania,
- *polecenia zarządzania użytkownikami i ich uprawnieniami*, w tym tworzenia, modyfikacji i usuwania użytkowników i ich grup, zarządzania dostępem do systemu, nadawania im praw własności plików oraz zarządzania dostępem do plików,
- *polecenia zarządzania procesami*, w tym ich tworzenia, wznawiania i usuwania, ustalania pierwszo- lub drugoplanowego trybu wykonywania oraz zarządzania priorytetami,
- *polecenia związane z aplikacjami* z różnych dziedzin zastosowań,
- inne polecenia, w tym związane z bezpieczeństwem systemu i jego zasobów.

Powłoki (z wyjątkiem tej, która jest uruchamiana na początku sesji) również mogą być wywołane przez użytkownika. Nie ma więc powodu, aby ich nie zaliczyć do poleceń.

Szereg poleceń dostępnych w systemach *N*X działa w sposób interaktywny: po ich wywołaniu udostępniany jest wewnętrzny język poleceń, za pomocą którego użytkownik wykonuje kolejne operacje w ramach danego polecenia. Przykładami mogą być tu powłoki (np. `sh`), edytory tekstów (np. `vi`), programy zarządzania partycjami dyskowymi (np. `fdisk`) i wiele innych.

W dalszej części tego rozdziału przedstawiony zostanie przegląd najważniejszych poleceń dostępnych w systemach *N*X, przy czym mało używane funkcje tych poleceń zostaną pominięte. Polecenia zarządzania procesami zostaną przedstawione w rozdziale 9.

Poszczególne polecenia będą przedstawione w postaci tabelarycznej:

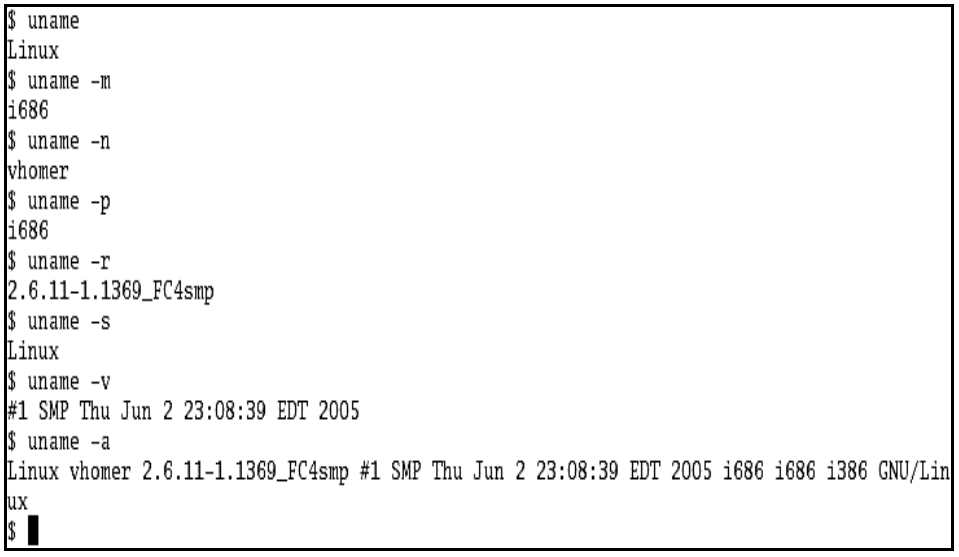
`nazwa polecenia`	`Postać (format) polecenia`
	Opis funkcji polecenia, opcji i parametrów

oraz, odpowiednio do potrzeb, ilustrowane przykładami.

7.2 Polecenia informacyjne

Posługując się poleceniami z tej grupy, użytkownik systemu *N*X może zapoznać się z podstawowymi danymi na temat systemu komputerowego i jego użytkowników oraz zasięgnąć informacji na temat dostępnych poleceń, ich funkcji i sposobu wykorzystania.

`uname`	`uname [ -m | -n | -p | -r | -s | -v | -a ]`
	Wyświetlenie informacji o systemie *N*X. Użycie opcji powoduje odpowiednio: `-m` - podanie typu maszyny, `-n` - podanie nazwy hosta, `-p` - podanie typu procesora, `-r` - podanie danych wydania (ang. *release*) systemu operacyjnego, `-s` - podanie nazwy systemu operacyjnego, `-v` - podanie numeru wersji (ang. *version*) systemu operacyjnego. Opcja `–a` powoduje podanie pełnego zestawu informacji w kolejności odpowiadającej opcjom: `snrvm`.



Rys. 47. Przykłady użycia polecenia `uname`

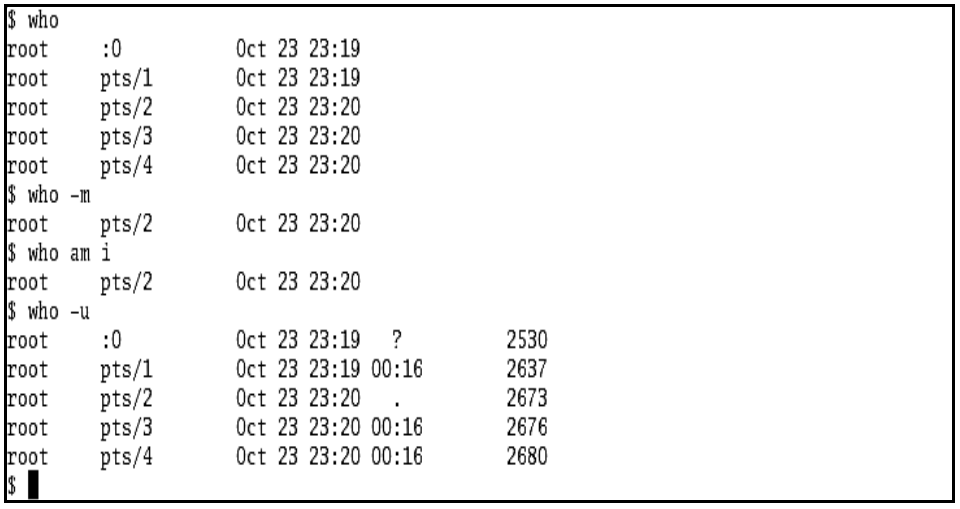
`hostname`	`hostname [`*`nazwa`*`]`
	Polecenie użyte bez parametru wyświetla nazwę hosta. Podane z parametrem ustawia tę nazwę na wartość parametru. Druga postać dostępna jest tylko dla superużytkownika.

W przytoczonym niżej przykładzie użytkownik bieżący czasowo przejmuje uprawnienia superużytkownika, aby móc nadać komputerowi nazwę.

```
$ su - root
Password:
[root@vhomer ~]# hostname vhomer
[root@vhomer ~]# hostname
vhomer
[root@vhomer ~]#
```

Rys. 48. Przykłady użycia polecenia `hostname`

`who`	`who [ -m | -u ] [am i]`
	Wyświetlenie informacji o wszystkich otwartych sesjach. Opcja `-m` powoduje, że polecenie `who` działa tak, jak `who am i`. Opcja `-u` powoduje dodatkowo podanie czasu nieaktywności użytkowników i identyfikatorów procesów powłok. Wywołanie polecenia z argumentami `am i` powoduje podanie tylko danych bieżącej sesji.



Rys. 49. Przykłady użycia polecenia `who`

`logname`	`logname`
	Polecenie wyprowadza nazwę użytkownika, który rozpoczął sesję

```
[root@vhomer ~]# logname
root
[root@vhomer ~]# su - skrypt
$ logname
root
$
```

Rys. 50. Przykład użycia polecenia `logname`

W przykładzie przytoczonym powyżej sesję terminalową otworzył użytkownik `root`. Chwilowe wejście w uprawnienia użytkownika `skrypt` nie zmieniło wyniku wykonania polecenia `logname`.

`w`	`w [-s] [`*`użytkownik`*`]`
	Polecenie użyte bez opcji i parametru wyświetla informacje o użytkownikach systemu, dane logowania i uruchomionych procesów. Opcja –s powoduje wyświetlanie w postaci skróconej, a użycie parametru ogranicza wykaz do użytkownika o podanej nazwie.

```
$ w
 22:33:38 up  1:20,  3 users,  load average: 1,30, 1,22, 0,82
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU
WHAT
root     :0       -                21:17   ?xdm?   5:47   0.55s
/usr/bin/gnome-session
root     pts/1    :0.0             22:27    0.00s  0.09s  0.01s w
root     pts/2    :0.0             21:21   59.00s  0.21s  0.11s
/usr/bin/mc –P /tmp/mc-root/mc
$ w -s
 22:33:40 up  1:20,  3 users,  load average: 1,30, 1,22, 0,82
USER     TTY      FROM               IDLE WHAT
root     :0       -                ?xdm?  /usr/bin/gnome-session
root     pts/1    :0.0              0.00s w -s
root     pts/2    :0.0              1:01  /usr/bin/mc –P /tmp/mc-
root/mc.pwd.7104
$ w les
 22:33:42 up  1:20,  3 users,  load average: 1,28, 1,21, 0,83
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU
WHAT
$
```

Rys. 51. Przykłady użycia polecenia `w`

Kolejne kolumny dla pierwszego wywołania polecenia oznaczają odpowiednio: nazwę użytkownika, oznaczenie terminala i jego okna, czas otwarcia sesji, łączny czas nieaktywności, czas wykorzystany przez wszystkie procesy korzystające z terminala, czas wykorzystany przez proces wskazany w kolumnie WHAT i postać polecenia, które uruchomiło daną sesję.

`date`	`date [MMDDhhmm[[CC]YY].[ss]]`
	Polecenie `date` użyte bez parametru powoduje wyświetlenie daty i czasu systemowego. Podanie parametru umożliwia zmianę daty i czasu systemowego. `MM` oznacza dwie cyfry numeru miesiąca, `DD` – dwie cyfry numeru dnia w miesiącu, `hh` – dwie cyfry godziny, mm – dwie cyfry minuty, `CC` – dwie cyfry numeru stulecia, `YY` – dwie cyfry numeru roku, `ss` – dwie cyfry oznaczające sekundy. Skuteczna zmiana daty lub czasu systemowego wymaga posiadania uprawnień superużytkownika. Polecenie to posiada liczne możliwości formatowania daty i czasu (zobacz `man date`).

```
[root@vhomer ~]# date 101518101997
śro paź 15 18:10:00 CEST 1997
[root@vhomer ~]# date
śro paź 15 18:10:00 CEST 1997
[root@vhomer ~]#
```

Rys. 52. Przykłady użycia polecenia `date`

cal	`cal [-my] [`*`miesiąc`* `[`*`rok`*`]]`
	Polecenie tworzy kalendarz na zadany okres czasu (miesiąc, rok). Opcja `-m` powoduje przyjęcie, że pierwszym dniem tygodnia jest poniedziałek; jej brak oznacza, że jest to niedziela. Opcja `-y` wyświetla kalendarz na rok bieżący. Parametr *`rok`* musi być podany jako liczba czterocyfrowa.

```
$ date
paź 19 20:34:02 CEST 2005
$ cal -y
                                2005

       styczeń                luty                   marzec
ni po wt śr cz pi so   ni po wt śr cz pi so   ni po wt śr cz pi so
                   1            1  2  3  4  5            1  2  3  4  5
 2  3  4  5  6  7  8    6  7  8  9 10 11 12    6  7  8  9 10 11 12
 9 10 11 12 13 14 15   13 14 15 16 17 18 19   13 14 15 16 17 18 19
16 17 18 19 20 21 22   20 21 22 23 24 25 26   20 21 22 23 24 25 26
23 24 25 26 27 28 29   27 28                  27 28 29 30 31
30 31
      kwiecień                 maj                    czerwiec
ni po wt śr cz pi so   ni po wt śr cz pi so   ni po wt śr cz pi so
                1  2    1  2  3  4  5  6  7               1  2  3  4
 3  4  5  6  7  8  9    8  9 10 11 12 13 14    5  6  7  8  9 10 11
10 11 12 13 14 15 16   15 16 17 18 19 20 21   12 13 14 15 16 17 18
17 18 19 20 21 22 23   22 23 24 25 26 27 28   19 20 21 22 23 24 25
24 25 26 27 28 29 30   29 30 31                26 27 28 29 30

       lipiec                 sierpień               wrzesień
ni po wt śr cz pi so   ni po wt śr cz pi so   ni po wt śr cz pi so
                1  2       1  2  3  4  5  6                  1  2  3
 3  4  5  6  7  8  9    7  8  9 10 11 12 13    4  5  6  7  8  9 10
10 11 12 13 14 15 16   14 15 16 17 18 19 20   11 12 13 14 15 16 17
17 18 19 20 21 22 23   21 22 23 24 25 26 27   18 19 20 21 22 23 24
24 25 26 27 28 29 30   28 29 30 31             25 26 27 28 29 30
31
     październik            listopad                grudzień
ni po wt śr cz pi so   ni po wt śr cz pi so   ni po wt śr cz pi so
                   1            1  2  3  4  5               1  2  3
 2  3  4  5  6  7  8    6  7  8  9 10 11 12    4  5  6  7  8  9 10
 9 10 11 12 13 14 15   13 14 15 16 17 18 19   11 12 13 14 15 16 17
16 17 18 19 20 21 22   20 21 22 23 24 25 26   18 19 20 21 22 23 24
23 24 25 26 27 28 29   27 28 29 30            25 26 27 28 29 30 31
30 31

$
```

Rys. 53. Przykład użycia polecenia `cal`

man	man [*lista_rozdziałów*] [*rozdział*] [*nazwa*]
	Systemy *N*X dostarczają opisów poleceń, funkcji i plików systemu. Opisy te (nazywane *stronami podręcznika* – ang. *man pages*) pogrupowane są w tzw. *rozdziały*, inaczej *sekcje*. Polecenie formatuje i wyprowadza stronę wskazaną przez parametr nazwa. Jeśli została podana *lista_rozdziałów* (w postaci ciągu nazw rozdziałów rozdzielonych dwukropkami), to poszukiwanie ogranicza się do rozdziałów wymienionych w tej liście. Podobnie, jeśli został podany parametr *rozdział*. Brak nazwy strony powoduje wyprowadzenie odpowiedniego komunikatu. Polecenie man man dostarcza własnego opisu. Numeracja rozdziałów podręcznika: 1. podstawowe polecenia systemowe, 2. strony przeznaczone dla programistów - wywołania systemowe, 3. strony przeznaczone dla programistów - funkcje biblioteczne, 4. pliki specjalne, 5. pliki konfiguracyjne systemu, 6. gry, 7. dodatkowe strony przeznaczone dla programistów, 8. polecenia zarządzania systemem. Niektóre dystrybucje systemów *N*X posiadają dodatkowe rozdziały podręcznika.

Przykład użycia polecenia man podano w punkcie 5.5.

Pliki źródłowe stron podręcznika wymagają przed wyświetleniem rozpakowania i przeformatowania, co jest dokonywane automatycznie przez polecenie man. W celu uzyskania pliku pozbawionego znaków cofania i podkreśleń należy plik wyjściowy przefiltrować jak pokazano niżej.

```
$ man man | col –v >man.txt
$
```

Rys. 54. Przykład korekty formatu pliku wyjściowego dla polecenia man

Odwołania do podręcznika systemu *N*X

W literaturze dotyczącej systemów *N*X, odwołania do konkretnych stron podręcznika podawane są w postaci np. ls (1), co oznacza stronę o nazwie ls zamieszczoną w rozdziale 1.

7.3 Polecenia do komunikacji między użytkownikami

Polecenia z tej grupy pozwalają na przesyłanie wiadomości między użytkownikami danego systemu, lub - jeśli system ma dostęp do sieci komputerowej - z użytkownikami odległych systemów.

write	`write` *`nazwa_logowania`*
	Polecenie `write` (ang. *write* – pisz) służy do nadania wiadomości do zalogowanego użytkownika. Wiadomość zostanie wyświetlona na jego terminalu, o ile ma on włączone zezwolenie na odbiór takich wiadomości (zob. polecenie `mesg`). Wiadomości wysłane przez superużytkownika będą wyświetlane zawsze.

```
$ write root
write: root is logged in more than once; writing to pts/3
Cześć, administratorku!
Jak się kręcą dyski?
^D
$
```

Rys. 55. Wysyłanie wiadomości przez użytkownika `skrypt` za pomocą polecenia `write`

Koniec wiadomości jest sygnalizowany przez nadawcę przez użycie polecenia klawiszowego ^D (niewidoczne), które na terminalu odbiorcy jest przedstawione jako EOF.

```
[root@vhomer pub]#
Message from root@vhomer (as skrypt) on pts/2 at 17:34 ...
Cześć, administratorku!
Jak się kręcą dyski?

EOF

[root@vhomer pub]#
```

Rys. 56. Wynik wykonania polecenia z Rys. 55

Po wyświetleniu wiadomości na terminalu odbiorczym, należy wprowadzić znak nowej linii, co spowoduje pojawienie się zachęty.

mesg	`mesg [y | n]`
	Polecenie `mesg` (od ang. *message* – wiadomość) wywołane z parametrem `y` zezwala na nadawanie i odbiór przez danego użytkownika wiadomości nadanych przez innych poleceniem `write`. Użycie parametru `n` wyłącza zezwolenie na odbiór tych wiadomości. Postać polecenia bez parametrów służy do sprawdzenia aktualnego stanu zezwolenia. Wiadomości nadane przez superużytkownika będą wyświetlane zawsze.

W przykładzie poniżej użytkownik wykonuje dwie próby wysłania wiadomości za pomocą polecenia `write`.

```
$ mesg
is n
$ # 1. próba wysłania wiadomości
$ write les
write: you have write permission turned off.
$ mesg y
$ # 2. próba wysłania wiadomości
$ write les
write: les has messages disabled
$
```

Rys. 57. Przykłady użycia polecenia `mesg`

Przy pierwszej próbie nadawca miał ustawiony zakaz nadawania wiadomości (komunikat błędu: `you have write permission turned off`), przy drugiej zaś ma ustawione zezwolenie na nadawanie wiadomości, ale okazało się, że adresat `les` miał ustawiony zakaz ich odbierania (komunikat: `les has messages disabled`). Aby skutecznie przesłać wiadomość, odbiorca powinien wcześniej mieć ustawione `mesg y`.

wall	`wall [tekst]…`
	Polecenie `wall` (od ang. *write to all* – pisz do wszystkich) służy do wysłania wiadomości do wszystkich zalogowanych użytkowników. Wiadomość może być podana jako ciąg parametrów tekstowych. Brak parametru oznacza tryb interaktywny, tj. wczytywanie tekstu wiadomości z terminala aż do wystąpienia polecenia klawiszowego ^D. Po odebraniu wiadomości, na terminalu odbiorczym należy wprowadzić znak nowej linii.

Typowym zastosowaniem polecenia `wall` jest podawanie przez administratora systemu ważnych i pilnych komunikatów przeznaczonych dla wszystkich zalogowanych użytkowników. Jeśli użytkownik `root` wykona polecenie: `wall Awaria UPS-a. Zamykam system!`, to na terminalach wszystkich użytkowników pojawi się wiadomość przedstawiona poniżej.

```
[les@vhomer ~]$
Broadcast message from root (pts/5) (Tue Oct 25 19:18:45
2005):

Awaria UPS-a. Zamykam system!

[les@vhomer ~]$
```

Rys. 58. Przykład działania polecenia `wall`

`mail`	`mail [`*`adres_odbiorcy`*`]`
	Polecenie `mail` (ang. *mail* – poczta), tzw. *klient poczty elektronicznej* służy do wysyłania i odbierania wiadomości poczty elektronicznej. Użyte bez parametru sprawdza stan skrzynki pocztowej zalogowanego użytkownika. Użyte z parametrem, którym jest elektroniczny adres pocztowy (tzw. *e-mail*), uruchamia tryb redagowania tekstu wiadomości, której nadawcą jest aktualnie zalogowany użytkownik. `adres_odbiorcy` ma postać `użytkownik@komputer.domena`

W przykładzie zamieszczonym poniżej użytkownik `skrypt` wysyła wiadomość pocztą elektroniczną do użytkownika `les` komputera `vhomer`. W odróżnieniu od poleceń `write` i `wall`, przesyłka zostanie przekazana nawet wtedy, gdy odbiorca nie jest zalogowany – trafi ona do tzw. *skrytki pocztowej* związanej z kontem odbiorcy i będzie oczekiwać na odczytanie. Konto odbiorcy znajduje się na dowolnym komputerze w sieci, z którą komputer nadawcy ma połączenie stałe lub czasowe.

```
$ mail les@vhomer
Subject: przykład nr 1
To jest treść wiadomości nr 1.
Kropka rozpoczynająca następny wiersz jest ogranicznikiem
tekstu.
.
Cc:
$
```

Rys. 59. Przykład użycia polecenia `mail` w celu wysłania przesyłki poczty elektronicznej

Parametr polecenia w postaci `les@vhomer` wskazuje, iż odbiorcą wiadomości jest użytkownik `les` posiadający konto na komputerze `vhomer`. W trakcie konwersacji z programem mail użytkownik określa temat wiadomości (`Subject:`), a następnie podaje jej treść w kolejnych wierszach; kropka na początku wiersza oznacza koniec tekstu. Wiersz rozpoczynający się od `Cc:` (od ang. *carbon copy* – tu: do wiadomości) pozwala na podanie adresów odbiorców, którzy otrzymają kopię treści przesyłki, przy czym ich wykaz będzie jawny. Adresat przesyłki sprawdza zawartość swojej skrytki poleceniem `mail` bez parametrów.

7.4 Polecenia dotyczące urządzeń

Do tej grupy należą polecenia, których zadaniem jest tworzenie, usuwanie oraz kontrola lub zmiana stanu urządzeń systemu *N*X.

`clear`	`clear`
	Polecenie `clear` (ang. *clear* – wyczyść) usuwa zawartość ekranu terminala.

Przykład użycia polecenia `clear` zamieszczono w punkcie 5.5.

`stty`	`stty [-a]`
	Polecenie to, użyte bez parametrów, służy do wyświetlania parametrów terminala; opcja `–a` włącza szczegółowy wykaz parametrów. Objaśnienie znaczenia ważniejszych parametrów terminala i przykłady użycia polecenia podano w punkcie 5.5.

Do operowania nośnikami wymiennymi, takimi jak dyskietki, dyski Iomega ZIP, taśmy magnetyczne, płyty CD-ROM itp. potrzebne są różnorodne polecenia obsługi, w tym formatowania, ustawiania w określonej pozycji, wysuwania oraz logicznego przyłączania i odłączania. Polecenia z tej grupy może wykonać tylko superużytkownik, ponieważ dotyczą one fizycznych zasobów systemu.

`fdformat`	`fdformat [-n]` *`urządzenie`*
	Polecenie `fdformat` (od ang. *floppy disk format* – formatowanie dyskietki) służy do formatowania dyskietki znajdującej się we wskazanym urządzeniu. Opcja `–n` wyłącza kontrolę poprawności sformatowanego nośnika. Jako parametr podaje się najczęściej `/dev/fd0`. Podręcznik (`man fdformat`) podaje nazwy urządzeń, jakie należy wskazać dla uzyskania niestandardowej pojemności.

```
[root@localhost ~]# fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440
kB.
Formatting ... done
Verifying ... bad data in cyl 0
Continuing ... bad data in cyl 1
Continuing ... Read: : Błąd wejścia/wyjścia
Problem reading cylinder 2, expected 18432, read -1
[root@localhost ~]#
```

Rys. 60. Przykład użycia polecenia `format`

Powyższy przykład formatowania dyskietki pokazuje zarazem sposób sygnalizacji błędów wykrytych podczas weryfikacji nośnika.

Kolejne polecenia spełniają ważną rolę, zwłaszcza w odniesieniu do wymiennych nośników pamięciowych, choć polecenia `mount` i `umont` nie ograniczają się tylko do nich.

`mount`	`mount [-r | -w] [-t` *`typ`*`]` *`urządzenie punkt_montowania`*
	Polecenie `mount` (ang. *mount* – montuj) służy do utworzenia logicznego połączenia urządzenia pamięci zewnętrznej o bezpośrednim dostępie z systemem plików. Opcja `–r` powoduje użycie wskazanego urządzenia wyłącznie do odczytu, opcja `–w` zezwala na zapis i odczyt. Pominięcie opcji `-r` i `-w` jest równoważne użyciu `–w`. Parametr *`urządzenie`* wskazuje na partycję dyskową, dyskietkę, urządzenie CD, DVD, pamięć typu *flash* itp. *`punkt_montowania`* jest pustym katalogiem, w którym zostanie odwzorowana zawartość struktury plikowej urządzenia. Parametr *`typ`* wskazuje na sposób organizacji (format) systemu plików na montowanym urządzeniu i przyjmuje wartości m.in. `msdos`, `vfat`, `ntfs`, `ext2fs`, `ext3fs`, `iso9660`. Faktyczne możliwości akceptowania formatów systemu plików zależą od wersji jądra systemu *N*X. Stan logicznego połączenia urządzenia z systemem plików trwa do chwili wykonania polecenia `umount` ze wskazaniem tego samego urządzenia lub punktu montowania jak w poleceniu `mount`, albo do chwili zatrzymania lub restartu systemu. Polecenie `mount` użyte bez parametrów wyświetla listę zamontowanych urządzeń i może być zastosowane przez każdego użytkownika systemu.

umount	umount {*urządzenie* | *punkt_montowania*}
	Polecenie umount (od ang. *unmount* – odmontuj, zdemontuj) likwiduje stan logicznego połączenia pomiędzy urządzeniem pamięciowym o bezpośrednim dostępie a systemem plików. Dla wykonania tej operacji wystarczy wskazać urządzenie lub punkt jego zamontowania.

```
[root@vhomer ~]# mount
/dev/sda1 on / type ext3 (rw)
/dev/proc on /proc type proc (rw)
/dev/sys on /sys type sysfs (rw)
/dev/devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/shm on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
automount(pid1779) on /misc type autofs
(rw,fd=4,pgrp=1779,minproto=2,maxproto=4)
automount(pid1830) on /net type autofs
(rw,fd=4,pgrp=1830,minproto=2,maxproto=4)
/dev/fd0 on /media/floppy type vfat (rw)
[root@p166v ~]# ls /media/floppy
control.hl_   gdi.ex_       ncdw.dl_   setup.inf  user.ex_
winhelp.ex_
cpwin386.cp_  instaluj.com  setup.exe  setup.shh
wfwsetup.dl_  winsetup.ex_
disk1         krnl386.ex_   setup.hl_  setup.tx_  win.cn_
xmsmmgr.exe
[root@vhomer ~]# umount /media/floppy
[root@ vhomer ~]# mount
/dev/sda1 on / type ext3 (rw)
/dev/proc on /proc type proc (rw)
/dev/sys on /sys type sysfs (rw)
/dev/devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/shm on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
automount(pid1779) on /misc type autofs
(rw,fd=4,pgrp=1779,minproto=2,maxproto=4)
automount(pid1830) on /net type autofs
(rw,fd=4,pgrp=1830,minproto=2,maxproto=4)
[root@vhomer ~]#
```

Rys. 61. Przykłady użycia polecenia mount i umount

W przytoczonym przykładzie superużytkownik najpierw sprawdził, które systemy plików są zamontowane. Następnie zamontował dyskietkę w katalogu

`/media/floppy` - typ systemu plików został określony automatycznie. Potem sprawdził jej zawartość, wykonał czynność odmontowania i ponownie sprawdził stan montowania.

Katalog montowania

W systemach *N*X zazwyczaj głównym katalogiem montowania jest `/mnt`. Zawarte w nim podkatalogi, np. `/mnt/cdrom`, `/mnt/floppy` są punktami montowania poszczególnych urządzeń. Nie jest to jednak reguła.

Warto zwrócić uwagę na występujący niekiedy problem niemożności odmontowania urządzenia:

```
[root@p166v ~]# mount /dev/fd0 /media/floppy
[root@p166v ~]# cd /media/floppy
[root@p166v floppy]# ls
control.hl_   gdi.ex_       ncdw.dl_   setup.inf  user.ex_
winhelp.ex_
cpwin386.cp_  instaluj.com  setup.exe  setup.shh
wfwsetup.dl_  winsetup.ex_
disk1         krnl386.ex_   setup.hl_  setup.tx_  win.cn_
xmsmmgr.exe
[root@p166v floppy]# umount /media/floppy
umount: /media/floppy: device is busy
umount: /media/floppy: device is busy
[root@p166v floppy]#
```

Rys. 62. Urządzenie zajęte nie daje się odmontować

Przyczyną odmowy odmontowania może być to, że urządzenie jest zajęte transmisją, niektóre ze znajdujących się tam plików są otwarte przez działające w systemie procesy lub to, że niektóre ze znajdujących się tam katalogów są katalogami bieżącymi otwartych sesji. Aby czynność odmontowania była skuteczna, należy wcześniej upewnić się, że dane urządzenie nie jest w żaden sposób używane (otwarte).

Polecenia `mount` i `umount` zawsze występują parami

Po zamontowaniu nośnika wymiennego nie wolno go usuwać z urządzenia przed skutecznym wykonaniem operacji odmontowania, nawet jeśli jest to technicznie wykonalne, pod groźbą uszkodzenia integralności systemu plików znajdującego się na nośniku i zakłócenia wykonywania zadań. Urządzenia CD i DVD zazwyczaj blokują zamontowany nośnik.

`eject`	`eject [floppy | cdrom]` Polecenie służy do mechanicznego wysunięcia nośnika wymiennego (zwykłe dyskietki, dyskietki Iomega Zip, płyty CD-ROM itp.) z jednostki napędowej. Uwaga: Nie wszystkie urządzenia posiadają mechanizm wysuwania nośnika – w takim przypadku czynność tę należy wykonać ręcznie (np. dyskietki). Domyślnym urządzeniem jest CD-ROM.

Dla wysunięcia nośnika musi on być uprzednio odmontowany. Jeśli nie dokonała tego aplikacja korzystająca z urządzenia, musi to wykonać superużytkownik.

`fdisk`	`fdisk` *`nazwa_woluminu`* Polecenie `fdisk` (of ang. *fixed disk* – tu: dysk twardy) służy do interaktywnego zarządzania partycjami woluminu dyskowego. *`nazwa_woluminu`* wskazuje urządzenie fizyczne, np. dysk twardy, na którym znajduje się lub znajdzie się jedna lub wiele partycji. Po uruchomieniu są udostępniane polecenia wewnętrzne pozwalające na tworzenie partycji, ustalanie ich formatu, kontrolę wykorzystania pojemności urządzenia itp. Zobacz `man fdisk`.

`df`	`df [-k]` Polecenie `df` (od ang. *disk free*) wyświetla raport na temat wykorzystania pojemności poszczególnych zamontowanych systemów plików o niezerowej liczbie bloków, tj. oprócz pseudosystemów. Opcja `-k` powoduje, iż wielkość obszarów dyskowych podawana jest w blokach 1-kilobajtowych zamiast w jednostkach domyślnych. Polecenie to jest dostępne dla wszystkich użytkowników.

```
$ df
System plików      bl.  1K B        użyte dostępne %uż. za-
mont. na
/dev/sda1              3804796   2793576    814828  78% /
/dev/shm                127292         0    127292   0%
/dev/shm
$
```

Rys. 63. Przykład użycia polecenia `df`

7.5 Polecenia do badania komunikacji między komputerami

W punkcie tym opiszemy polecenia służące do badania konfiguracji sieciowej komputera. Należy wspomnieć, że podstawowe parametry identyfikujące komputer w sieci komputerowej ustalane są w trakcie instalowania systemu operacyjnego, a ich modyfikacja jest związana ze zmianą zawartości szeregu plików, o czym mowa szerzej w punkcie 10.2.

`ifconfig`	`ifconfig [-a | interfejs]`
	Polecenie `ifconfig` (od ang. *interface configuration* – konfiguracja interfejsu) użyte bez parametrów wyświetla parametry aktywnych interfejsów sieciowych, w tym (zawsze) wewnętrznego interfejsu sieciowego (ang. *loopback interface*) oraz innych, o ile zostały zainstalowane i skonfigurowane. Użycie opcji `-a` powoduje wyświetlenie parametrów wszystkich interfejsów, w tym nieaktywnych. Podanie parametru *`interfejs`* ogranicza wyświetlanie do wskazanego interfejsu. Parametr ten jest nazwą urządzenia z numerem porządkowym, np. `eth0` jest pierwszym interfejsem sieciowym typu Ethernet. Opis innych możliwości polecenia, w tym konfigurowania i sterowania interfejsem, dostępny jest przez `man ifconfig`.

```
[root@vhomer ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:A4:EA:4B
          inet addr:156.17.73.191  Bcast:156.17.73.255
Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fea4:ea4b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14393 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1854 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1903616 (1.8 MiB)  TX bytes:236412 (230.8 KiB)
          Interrupt:177 Base address:0x1080

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1907 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1907 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3524804 (3.3 MiB)  TX bytes:3524804 (3.3 MiB)

[root@vhomer ~]#
```

Rys. 64. Przykład użycia polecenia `ifconfig`

Czytelnik zechce samodzielnie zinterpretować opis parametrów interfejsów.

route	`route [-n]`
	`route add [-net|-host] cel [netmask maska]`
	`[gw bramka] [interfejs]`
	`route add default gw adres_bramki`
	`route del [-net|-host] cel [netmask maska]`
	`[gw bramka] [interfejs]`
	
	Polecenie `route` (ang. *route* – trasa) użyte bez parametrów wyświetla statyczne tablice trasowania połączeń z poszczególnymi hostami i podsieciami adresowymi IP. Opcja `-n` powoduje wyświetlenie adresów IP zamiast nazw.
	W postaci `route add` polecenie to dodaje nowy wpis do tablicy trasowania połączeń. W szczególności, postać `route add default` służy do zdefiniowania domyślnej trasy połączeń, która zostanie użyta w razie nieznalezienia trasy w pozostałych wpisach tablicy.
	Postać `route del` służy do usuwania podanego wpisu z tej tablicy.
	Znaczenie parametrów jest następujące: `-net` i `-host` wskazują, iż następujący po nich parametr *`cel`* określa odpowiednio adres IP podsieci lub adres IP hosta. *`maska`* oznacza maskę podsieci, *`bramka`* jest adresem IP bramki (ang. *gateway*) przez którą dana podsieć komunikuje się z innymi podsieciami. *`interfejs`* oznacza interfejs sieciowy wykorzystywany do połączeń z daną podsiecią lub hostem.

```
$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric
Ref    Use Iface
156.17.73.0     *               255.255.255.0   U     0
0        0 eth0
169.254.0.0     *               255.255.0.0     U     0
0        0 eth0
default         furtka          0.0.0.0         UG    0
0        0 eth0
$
```

Rys. 65. Przykład użycia polecenia `route`

Dla poprawnego komunikowania się ze światem zewnętrznym szczególnie ważny jest interfejs domyślny, oznaczony jako `default`. Brak wskazania interfejsu domyślnego zazwyczaj powoduje niemożność komunikacji z innymi komputerami.

`ping`	`ping [-c` *`liczba`*`] [-s` *`długość`*`]` *`host`* Polecenie bada połączenie sieciowe ze wskazanym komputerem lub sieciowym urządzeniem aktywnym przez wykorzystanie protokołu ICMP ping, tj. poprzez wysyłanie do niego pakietów danych i raportowanie każdej odpowiedzi. Parametr `-c` *`liczba`* określa liczbę pakietów do wysłania, parametr `-s` *`długość`* – wielkość pojedynczego pakietu w bajtach, *`host`* jest nazwą domenową docelowego komputera lub jego adresem IP. Jeżeli nie podano liczby pakietów, badanie połączenia sieciowego będzie trwać aż do wydania polecenia klawiszowego ^C. Raport z badania zawiera m.in. najkrótszy, średni i najdłuższy czas odpowiedzi oraz liczbę i odsetek utraconych pakietów.

```
$ ping localhost
PING vhomer (127.0.0.1) 56(84) bytes of data.
64 bytes from vhomer (127.0.0.1): icmp_seq=0 ttl=64 time=0.118ms
64 bytes from vhomer (127.0.0.1): icmp_seq=1 ttl=64 time=0.121ms
64 bytes from vhomer (127.0.0.1): icmp_seq=2 ttl=64 time=0.200ms
64 bytes from vhomer (127.0.0.1): icmp_seq=3 ttl=64 time=0.126ms
64 bytes from vhomer (127.0.0.1): icmp_seq=4 ttl=64 time=0.575ms
64 bytes from vhomer (127.0.0.1): icmp_seq=5 ttl=64 time=0.125ms
64 bytes from vhomer (127.0.0.1): icmp_seq=6 ttl=64 time=0.123ms
64 bytes from vhomer (127.0.0.1): icmp_seq=7 ttl=64 time=0.152ms
^C
--- vhomer ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7032ms
rtt min/avg/max/mdev = 0.118/0.192/0.575/0.147 ms, pipe 2
$
```

Rys. 66. Przykład użycia polecenia `ping`

Polecenie `ping` jest zazwyczaj tym, które jako pierwsze zostaje użyte w celu zbadania poprawności skonfigurowania komputera do pracy w sieci komputerowej.

Komputer nie odpowiada na `ping`?

Brak odpowiedzi od docelowego komputera nie oznacza jeszcze, iż interfejs sieciowy nie został prawidłowo skonfigurowany. Niektóre komputery mogą bowiem mieć, ze względów bezpieczeństwa, wprowadzoną blokadę wysyłania odpowiedzi ICMP echo (tzw. *stealth mode* – tryb niewidzialny). W razie wątpliwości należy wykonać polecenie `ping` z innym adresem docelowym.

`traceroute`	`traceroute [-n] host [wielkość_pakietu]`
	Polecenie to raportuje trasę, jaką pokonują w sieci komputerowej pakiety kontrolne wysyłane od nadawcy do wskazanego komputera (urządzenia aktywnego). Opcja przyspiesza wykonywanie polecenia, ponieważ nie są wykonywane odwołania do serwisu nazw (DNS). Raport z badania dostarcza m.in. informacji o długości trasy rozumianej jako liczba przeskoków (tzw. *hops*) przez kolejne urządzenia rutujące oraz stwierdzonych czasów opóźnień. Parametr host jest nazwą domenową docelowego komputera (urządzenia) lub jego adresem IP.

```
$ traceroute www.pwr.wroc.pl
traceroute to www.pwr.wroc.pl (156.17.5.189), 30 hops max, 38 byte packets
 1  gw224 (156.17.73.254)  0.776 ms  1.763 ms  0.688 ms
 2  156.17.18.210 (156.17.18.210)  2.212 ms  1.643 ms  0.716 ms
 3  156.17.18.246 (156.17.18.246)  0.787 ms  1.627 ms  1.998 ms
 4  156.17.18.254 (156.17.18.254)  1.337 ms  0.967 ms  1.179 ms
 5  g12wcss-g12centrum.wask.wroc.pl (156.17.255.117)  1.391 ms  1.849 ms  0.972 ms
 6  g48centrum-g12wcss.wask.wroc.pl (156.17.255.129)  1.016 ms  2.574 ms  1.079 ms
 7  www.pwr.wroc.pl (156.17.5.189)  0.978 ms  1.660 ms  0.844 ms
$
```

Rys. 67. Przykład użycia polecenia `traceroute`

Dla każdego przeskoku (adresu) na trasie połączenia podawany jest czas podróży trzech próbek. Brak odpowiedzi oznaczany jest gwiazdką. Trasa połączenia z docelowym komputerem może zmieniać się w czasie, zależnie od chwilowego obciążenia poszczególnych odcinków połączeń i algorytmów wybierania optymalnych ścieżek przez urządzenia rutujące. Analiza uzyskanego raportu pozwala jednak na wskazanie tzw. *wąskich gardeł* na trasie połączenia.

7.6 Polecenia dotyczące plików

W systemach komputerowych pliki są jednymi z najważniejszych zasobów. Stosownie do tego kształtuje się względna liczba dotyczących ich poleceń.

7.6.1 Nawigacja i pobieranie informacji

`cd`	`cd [`*`katalog`*`]`
	Polecenie `cd` (od ang. *change directory* – zmień katalog) powoduje zmianę katalogu bieżącego sesji na katalog wskazany jako parametr polecenia. Jako parametr mogą występować również symbole `~`, `.` i `..` (zobacz punkt 3.4). Katalogiem domyślnym (przy braku parametru wywołania) jest katalog określony w zmiennej środowiska `$HOME`.

`pwd`	`pwd`
	Polecenie `pwd` (od ang. *print working directory* – pokaż katalog bieżący) wyświetla pełną ścieżkę do katalogu bieżącego dla danej sesji.

Przykłady zastosowania poleceń `cd` i `pwd` zamieszczono poniżej.

```
$ cd /var/log
$ pwd
/var/log
$ cd .
$ pwd
/var/log
$ cd ..
$ pwd
/var
$ cd
$ pwd
/home/skrypt
$
```

Rys. 68. Przykłady użycia poleceń `cd` i `pwd`

`ls`	`ls [-alsR] [`*`katalog`*`]…`
	Polecenie `ls` (od ang. *list* – listuj) wyświetla zawartość wskazanych katalogów. Opcja `-a` powoduje, że nie są pomijane pliki ukryte, opcja `-l` włącza wyświetlanie szczegółowe. Zastosowanie opcji `-R` uruchamia wyświetlanie rekurencyjne, tj. z uwzględnieniem podkatalogów danego katalogu itd.

	Opcja `-s` powoduje dodatkowo wyświetlanie ilości miejsca zajętego przez plik w systemie plików. Wykaz w postaci szczegółowej zawiera m.in. atrybuty pliku, nazwę właściciela i grupy właścicielskiej, datę i czas modyfikacji i lokalną nazwę pliku. Jeśli data i czas modyfikacji wybiegają w przeszłość o więcej niż o 6 miesięcy lub w przyszłość o więcej niż 1 godzinę w stosunku do daty i czasu aktualnego, to zamiast godzin i minut wypisywany jest numer roku.

W przykładach zamieszczonych poniżej polecenie `ls` wykonano z opcją `-a` i bez wskazywania katalogu, co spowodowało wyświetlenie informacji o całej zawartości katalogu bieżącego.

```
$ pwd
/home/skrypt
$ ls -a
.   .bash_history  .bash_profile  .gtkrc  .mc       .zshrc
..  .bash_logout   .bashrc        .kde    .viminfo
$ ls -la
razem 92
drwxr-xr-x  4 skrypt users 4096 paź 26 11:34 .
drwxr-xr-x  4 root   root  4096 paź 26 06:14 ..
-rw-------  1 skrypt users  208 paź 25 13:27 .bash_history
-rw-r--r--  1 skrypt users   24 maj 10 18:15 .bash_logout
-rw-r--r--  1 skrypt users  199 paź 25 12:58 .bash_profile
-rw-r--r--  1 skrypt users  124 maj 10 18:15 .bashrc
-rw-r--r--  1 skrypt users  120 maj 22 23:18 .gtkrc
drwxr-xr-x  3 skrypt users 4096 paź 25 11:41 .kde
drwxr-xr-x  3 skrypt users 4096 paź 25 13:27 .mc
-rw-------  1 skrypt users 5912 paź 26 11:34 .viminfo
-rw-r--r--  1 skrypt users  658 sty 17  2005 .zshrc
$
```

Rys. 69. Przykłady użycia polecenia `ls`

Czytelnik zechce zwrócić uwagę na to, iż przedmiotem działania polecenia `ls` był katalog prywatny użytkownika `skrypt`. Znajdujące się tu pliki ukryte (o nazwach rozpoczynających się kropką) definiują środowisko pracy użytkownika.

`du`	`du [`*`opcja`*`]… [`*`plik`*`]…`
	Polecenie `du` (od ang. *disk used* – wykorzystanie dysku) wyświetla raport na temat wykorzystania pojemności pamięci plików dla przechowania katalogów i plików wskazanych w parametrach wywołania.

	Opcja `-k` wymusza prezentację objętości w jednostkach 1 KB, opcja `-m` - w jednostkach 1 MB. Opcja `-s` powoduje, ze dla każdego parametru obliczane jest tylko podsumowanie objętości, opcja `-c` zaś powoduje obliczenie łącznej objętości wszystkich plików i katalogów wymienionych jako parametry.

```
$ du -s /bin /home
6652    /bin
252     /home
$ du -scm /bin /home
7       /bin
1       /home
7       razem
$ du /home
28      /home/les/.kde/Autostart
36      /home/les/.kde
84      /home/les
28      /home/skrypt/.kde/Autostart
36      /home/skrypt/.kde
8       /home/skrypt/.mc/cedit
48      /home/skrypt/.mc
152     /home/skrypt
252     /home
$
```

Rys. 70. Przykłady użycia polecenia `du`

Należy zauważyć, iż w niektórych sytuacjach wykonanie polecenia nie będzie w pełni możliwe z powodu braku dostępu do katalogu w celu określenia objętości pliku. Wykaz tworzony przez polecenie `du` będzie wtedy zawierać wiersze postaci: `du: `/etc/cups/certs' : Brak dostępu`, a objętość niedostępnych plików nie zostanie uwzględniona w obliczonych sumach.

`find`	`find` *`katalog wzorzec`* [*`operacja`*]
	Polecenie `find` (ang. *find* – znajdź) wyszukuje pliki pasujące do `wzorca` i dla każdego z nich wykonuje operację wskazaną przez trzeci parametr. Parametr `katalog` wskazuje korzeń poddrzewa systemu plików, od którego rozpoczyna się rekurencyjne poszukiwanie. Drugi parametr może mieć np. postać: `-name` *`wzorzec_nazwy_pliku`*. Trzeci parametr może mieć np. postać: `-print`, co oznacza wylistowanie nazw znalezionych plików. Pominięcie trzeciego parametru jest równoznaczne z użyciem `-print`.

W następującym przykładzie w katalogu `/var/log` poszukujemy wszystkich plików o nazwach kończących się na `.log`.

```
$ find /var/log  -name *.log
/var/log/mysqld.log
find: /var/log/audit: Brak dostępu
find: /var/log/samba: Brak dostępu
/var/log/scrollkeeper.log
find: /var/log/httpd: Brak dostępu
find: /var/log/squid: Brak dostępu
/var/log/Xorg.0.log
/var/log/gdm/:0.log
/var/log/boot.log
/var/log/prelink.log
/var/log/anaconda.log
find: /var/log/ppp: Brak dostępu
$
```

Rys. 71. Przykłady użycia polecenia `find`

Jak widać, z powodu braku wystarczających uprawnień niektóre katalogi nie zostały przeszukane.

7.6.2 Zarządzanie plikami

`mkdir`	`mkdir [`*`opcje`*`] `*`katalog…`*
	Polecenie `mkdir` (od ang. *make directory* – utwórz katalog) tworzy nowe katalogi w miejscach systemu plików wskazanych przez parametry polecenia. Właścicielem nowych katalogów jest użytkownik wydający polecenie. Opcja `–m` *`tryb`* pozwala na ustalenie trybu dostępu do tworzonych katalogów zgodnie z wartością *`tryb`* (podaną symbolicznie, jak dla polecenia chmod). W razie braku tej opcji atrybuty dostępu do katalogu zostają ustalone odpowiednio do aktualnej wartości maski użytkownika. Opcja `–p` wymusza tworzenie brakujących katalogów nadrzędnych względem katalogów tworzonych.

W kolejnym przykładzie w katalogu prywatnym użytkownika utworzymy dwa poziomy podkatalogów: katalog `nadrzędny`, a w nim katalog `podrzędny`.

```
$ pwd
/home/skrypt
$ mkdir -p nadrzedny/podrzedny
$ ls -R
.:
nadrzedny

./nadrzedny:
```

```
podrzedny

./nadrzedny/podrzedny:
$
```

Rys. 72. Przykład użycia polecenia `mkdir`

Polecenie `ls -R` posłużyło do sprawdzenia tak utworzonej struktury.

`rmdir`	`rmdir [`*`opcje`*`] `*`katalog`*`...`
	Polecenie `rmdir` (od ang. *remove directory* – usuń katalog) usuwa z systemu plików niepuste katalogi wskazane przez parametry polecenia. Wykonywanie polecenia jest przerywane, gdy usuwany katalog jest niepusty, chyba że została podana opcja `--ignore-fail-on-not-empty`. Opcja `-p` powoduje podjęcie próby skasowania również odpowiednich katalogów nadrzędnych.

```
$ pwd
/home/skrypt
$ ls
nadrzedny
$ ls nadrzedny
podrzedny
$ rmdir -p nadrzedny/podrzedny
$ ls
$
```

Rys. 73. Przykład użycia polecenia `rmdir`

Podany powyżej zapis konwersacji obejmuje sprawdzenie, że w katalogu bieżącym rzeczywiście istnieje podkatalog `nadrzedny` ze swoim podkatalogiem o nazwie `podrzedny`. Polecenie `rmdir` zostało wywołane rekurencyjnie, w wyniku czego wskazana struktura podkatalogów przestała istnieć.

touch	touch [*opcja*]… *plik*…
	Polecenie touch (ang. dotknij) zmienia czas ostatniego dostępu lub czas ostatniej modyfikacji pliku. Jeśli wskazany plik nie istnieje, a nie została użyta opcja –c, to zostanie utworzony plik pusty. Opcja –a zmienia czas ostatniego dostępu, opcja –m zaś czas ostatniej modyfikacji pliku. Czas podawany jest w postaci –t [[CC]YY]MMDDhhmm[.ss], gdzie znaczenie symboli jest takie, jak dla polecenia date. Brak określenia czasu spowoduje oznakowanie plików czasem bieżącym.

Dla zilustrowania działania polecenie touch utworzymy w trybie interaktywnym pusty plik o nazwie gogo, a następnie wpiszemy do niego jakąś treść, sprawdzając za każdym razem dane tego pliku poleceniem ls. Używając touch zmienimy datę i czas ostatniej modyfikacji pliku na 1 lutego 2000, godzina 02:03:00.

```
$ cat >gogo
^D
$ ls -la gogo
-rw-r--r--  1 skrypt users 0 paź 6 13:18 gogo
$ cat >>gogo
abcde
123
^D
$ ls -la gogo
-rw-r--r--  1 skrypt users 10 paź 6 13:18 gogo
$ touch -m -t 200001020304 gogo
$ ls -la gogo
-rw-r--r--  1 skrypt users 10 lut 1  2000 gogo
$
```

Rys. 74. Przykład użycia polecenia touch

Interpretacja faktu wyświetlenia numeru roku zamiast godzin i minut jest zadaniem Czytelnika.

cat	cat [*plik*]…
	Polecenie cat (od ang. *concatenate* – połącz) łączy ze sobą zawartość kolejnych plików stanowiących parametry polecenia. Wynik łączenia wyprowadzany jest standardowo na terminal. Użycie polecenia bez parametru oznacza, iż dane wejściowe będą pobierane z terminala.

Polecenie `cat`, w powiązaniu z mechanizmem przekierowywania standardowych strumieni: wejściowego i wyjściowego, pozwala na wygodne tworzenie, łączenie i wyświetlanie zawartości plików tekstowych. Polecenie to może być użyte w trybie interaktywnym i nieinteraktywnym.

Tryb interaktywny polega na tym, że użytkownik wprowadza kolejne wiersze do pliku wprost z terminala. Koniec wprowadzania sygnalizowany jest przez użytkownika poleceniem klawiszowym ^D.

```
$ cat >telefony
123 4567        Jurek
321 3332        Basia
456 1928        Grześ
^D
$
```

Rys. 75. Interaktywny tryb użycia polecenia `cat`.

Nieinteraktywny tryb działania polecenia `cat` charakteryzuje się użyciem dodatkowego parametru określającego ogranicznik wprowadzanego tekstu. Napotkanie wiersza tekstu identycznego z ogranicznikiem oznacza koniec pliku wejściowego. Sam ogranicznik nie jest wprowadzany do pliku.

```
$ cat >telefony2 <<KONIEC
> 101 2230      Andrzej
> 204 5692      Iwona
> 590 3456      Celina
> KONIEC
$
```

Rys. 76. Nieinteraktywny tryb użycia polecenia `cat`

Zwróćmy uwagę na różnice: przy pracy na terminalu w trybie nieinteraktywnym początek każdego wiersza zawiera zachętę zdefiniowaną przez zmienną środowiska `PS2` (zobacz rozdział 6.). Zachęta nie jest zapamiętywana w pliku. W trybie tym nie występuje też konieczność sygnalizacji końca pliku poleceniem klawiszowym, w związku z czym tryb ten nadaje się do zaprogramowanego wprowadzania danych do plików wewnątrz skryptów powłokowych (zobacz rozdział 8.).

Oprócz wykorzystania poleceń `touch` i `cat` nowe pliki można tworzyć jako kopie plików istniejących. Służą do tego polecenia `cp` i `copy`.

cp	cp [*opcje*]… *plik kopia_pliku* cp [*opcje*]… *plik*… *katalog_docelowy* Polecenie cp (od ang. *copy* – kopiuj) w pierwszej postaci tworzy kopię pliku *plik* w pliku *kopia_pliku*. Druga postać polecenia służy do umieszczenia kopii wielu plików w katalogu wskazanym jako ostatni parametr. Jeśli podano więcej niż dwa parametry, a ostatni z nich nie jest katalogiem, to wywołanie polecenia jest błędne. Standardowo kopiowaniu podlegają tylko pliki a nie katalogi, chyba że została użyta opcja –R. Powoduje ona kopiowanie całych poddrzew katalogów źródłowych. Opcja –f powoduje usuwanie istniejących plików docelowych, opcja –i włącza interaktywny tryb pracy, w którym użytkownik potwierdza zamiar nadpisywania istniejących plików docelowych. Opcja –p powoduje nadanie kopii pliku tych samych atrybutów (właściciel, grupa, prawa dostępu i znaczniki czasowe), jakie miał plik źródłowy. Brak tej opcji powoduje, że prawa dostępu dla kopii są takie jak pliku źródłowego, minus maska użytkownika. Opcja –P powoduje utworzenie brakujących katalogów nadrzędnych dla kopii.

```
[root@vhomer /]# cp -R /home /home.kopia
[root@vhomer /]# ls -l|grep home
drwxr-xr-x    4 root root  4096 paź 24 21:20 home
drwxr-xr-x    4 root root  4096 paź 27 20:38 home.kopia
[root@vhomer /]#
```

Rys. 77. Przykład użycia polecenia cp

mv	mv [*opcje*]… *plik kopia_pliku* mv [*opcje*]… *plik*… *katalog_docelowy* Polecenie mv (od ang. *move* – przenieś) służy do przenoszenia pojedynczych plików (pierwsza postać) i całych grup plików do innych lokalizacji lub do zmiany nazw plików. Opcja –i włącza tryb interaktywny. Opcja –f powoduje usuwanie plików docelowych bez potwierdzania. Opcja –u włącza tryb aktualizacji, w którym plik nie zostanie przeniesiony, jeśli jego docelowe miejsce ma taką samą lub późniejszą datę modyfikacji. Opcja –b powoduje tworzenie zapasowych kopii plików nadpisywanych lub usuwanych.

Poniżej posłużymy się poleceniem mv dla zmiany nazwy pliku telefony na spis.telefonów.

```
$ ls
dead.letter  man.txt  telefony
$ mv telefony spis.telefonów
```

```
$ ls
dead.letter  man.txt  spis.telefonów
$
```

Rys. 78. Przykład użycia polecenia `mv`

`rm`	`rm [`*`opcja`*`]… `*`plik`*`…`
	Polecenie `rm` (od ang. *remove* – usuń) służy do usuwania plików, a z opcją `–r` również katalogów. Opcja `–i` włącza tryb interaktywny, w którym użytkownik jest proszony o potwierdzanie zamiaru kasowania każdego pliku. Aby plik skasować, należy odpowiedzieć y lub Y. Opcja `–f` wymusza nieinteraktywny tryb kasowania, nawet jeśli podano opcję `–i,` co oznacza, że pliki zostaną skasowane bezwarunkowo. Opcja `–r` lub `–R` powoduje rekurencyjne usuwanie katalogów łącznie z całą zawartością.

W przykładzie poniżej, po prezentacji stanu katalogu bieżącego `/home/skrypt`, tworzymy w nim podkatalog `kasownik`, w którym umieszczamy trzy pliki. Z poziomu katalogu usuwamy dopiero co utworzony podkatalog `kasownik` i sprawdzamy poprawność wykonania polecenia `rm`.

```
$ pwd
/home/skrypt
$ ls
dead.letter  man.txt  telefony
$ mkdir kasownik
$ cd kasownik
$ pwd
/home/skrypt/kasownik
$ cat >a
123     qwe
asd
$ touch b
$ cat >c
aaaa
ssss
dddd
$ ls -l
razem 20
-rw-r--r--  1 skrypt users 12 paź 29 10:11 a
-rw-r--r--  1 skrypt users  0 paź 29 10:11 b
-rw-r--r--  1 skrypt users 15 paź 29 10:12 c
$ cd ..
```

```
$ rm -r kasownik
$ ls
dead.letter  man.txt  telefony
```

```
$
```

Rys. 79. Przykład użycia polecenia `rm`

Uwaga, niebezpieczne polecenie `rm -rf !`

Czytelnik zauważy, jak potencjalnie niebezpieczne jest użycie z poleceniem rm opcji `-f`, a szczególnie `-r`. Dla terminalowego trybu pracy brak bowiem możliwości odzyskania skasowanych plików (ani w postaci polecenia typu undelete znanego z MS DOS, ani w postaci kosza, jaki jest dostępny dla graficznego trybu pracy).

Wniosek: nie należy śpieszyć się z potwierdzeniem wprowadzonego polecenia rm przez klawisz <**Enter**>.

Na koniec tej grupy polecenie, które udostępnia pliki (i katalogi) dodatkowo w innych lokalizacjach niż ich faktyczne położenie.

`ln`	`ln [-s] istniejąca_nazwa dodatkowa_nazwa`
	Polecenie `ln` (od ang. *link* – połączenie) tworzy dowiązanie, pozwalające na korzystanie z pliku wskazanego przez pierwszy parametr, dodatkowo również przez użycie nazwy podanej jako drugi parametr. Użycie polecenia bez opcji `-s` tworzy tzw. *twarde dowiązanie* do pliku. Dowiązania twarde nie różnią się od zwykłego opisu pliku w katalogu. Podczas kasowania pozycji pliku w katalogu plik jest fizycznie usuwany dopiero wtedy, gdy kasowaniu podlega ostatnie wskazanie na ten plik. Zastosowanie opcji `-s` tworzy tzw. *dowiązanie miękkie* lub *symboliczne*, które jest plikiem zawierającym nazwę pliku wskazywanego, traktowanym w katalogu w specjalny sposób. Usunięcie dowiązania miękkiego nigdy nie usuwa samego pliku.

W zamieszczonym przykładzie, po sprawdzeniu stanu katalogu bieżącego utworzyliśmy w nim podkatalog `KOPIA`, a następnie w tym podkatalogu umieściliśmy dowiązanie symboliczne do pliku `telefony`, nazwane `telefony.kopia`. Przykład zakończyło sprawdzenie uzyskanych efektów.

```
$ pwd
/home/skrypt
$ ls
dead.letter  man.txt  telefony
$ mkdir KOPIA
$ ln -s telefony KOPIA/telefony.kopia
$ ls KOPIA
telefony.kopia
$ ls -l KOPIA
razem 4
lrwxrwxrwx  1 skrypt users 8 paź 29 10:31 telefony.kopia ->
telefony
$
```

Rys. 80. Przykład użycia polecenia `ln`

Zauważmy, że w miejscu, gdzie polecenie `ls -l` umieszcza lokalną nazwę pliku, dla dowiązania symbolicznego występuje zapis składający się z nazwy lokalnej, znaków tworzących strzałkę i nazwy pliku, do którego dowiązanie zostało utworzone. Ponadto pierwszym znakiem w wierszu jest litera `l` (od ang. *link*), a opis uprawnień dostępu ma postać `lrwxrwxrwx`. Notacja ze strzałką nie dotyczy dowiązań twardych.

7.6.3 Kompresja i archiwizacja plików

`tar`	`tar` *`opcje parametry`*
	Polecenie `tar` (od ang. *tape archive* – archiwum taśmowe) służy do scalania wielu plików w jeden plik albo wykonywania operacji odwrotnej. Opcje mogą być poprzedzone łącznikiem `-` (ale nie muszą): `c` oznacza operację scalania wielu plików w jeden plik (budowanie archiwum plikowego), `x` oznacza operację odwrotną - wydzielanie plików z archiwum, `v` oznacza „gadatliwy” tryb pracy, `f` oznacza, iż za opcjami będzie podane określenie pliku archiwum, `z` oznacza, że plik archiwum jest (z opcją `x`) lub ma być (z opcją `c`) skompresowany programem `gzip`. Parametry oznaczają nazwy plików dodawanych do archiwum lub wydzielanych z niego.

`gzip`	`gzip [-d] [`*`plik`*`]…`
	Polecenie `gzip` służy do bezstratnej (odwracalnej) kompresji plików. Plik skompresowany zawiera pojedynczy plik źródłowy. Uwaga: polecenie `gzip` nie należy do zbioru standardowych poleceń systemów *N*X. Jego pojawienie się w tym miejscu wynika ze względów praktycznych.

Polecenia `tar` i `gzip` często są wykorzystywane łącznie. Przy tworzeniu pliku archiwalnego najpierw zbiór plików scalany jest do pojedynczego pliku, a następnie kompresowany. Powstały w wyniku tych czynności plik ma zazwyczaj nazwę kończącą się na `.tgz` lub `.tar.gz`. Podczas odzyskiwania plików z archiwum kolejność użycia poleceń jest odwrotna. Odpowiednio zmieniony jest też sposób ich wywołania.

Utworzymy teraz skompresowane archiwum zawierające pliki z katalogu bieżącego, tworząc najpierw z wielu plików jeden plik poleceniem `tar`, a potem kompresując ten plik poleceniem `gzip`.

```
[skrypt@vhomer ~]$ ls
dead.letter  man.txt  spis.telefonów
[skrypt@vhomer ~]$ tar cvf archiwum.tar *
dead.letter
man.txt
spis.telefonów
[skrypt@vhomer ~]$ ls
archiwum.tar  dead.letter  man.txt  spis.telefonów
[skrypt@vhomer ~]$ gzip archiwum.tar
[skrypt@vhomer ~]$ ls -la
razem 140
drwx------  4 skrypt users  4096 paź 29 11:27 .
drwxr-xr-x  4 root   root   4096 paź 24 21:20 ..
-rw-r--r--  1 skrypt users 11010 paź 29 11:27 archiwum.tar.gz
-rw-------  1 skrypt users  3537 paź 27 20:42 .bash_history
-rw-r--r--  1 skrypt users    24 maj 10 18:15 .bash_logout
-rw-r--r--  1 skrypt users   199 paź 24 21:21 .bash_profile
-rw-r--r--  1 skrypt users   124 maj 10 18:15 .bashrc
-rw-------  1 skrypt users  5136 paź 29 11:23 dead.letter
-rw-r--r--  1 skrypt users   120 maj 22 23:18 .gtkrc
-rw-r--r--  1 skrypt users 12535 paź 29 09:19 man.txt
drwxr-xr-x  2 skrypt users  4096 paź 29 11:24 .mc
-rw-r--r--  1 skrypt users    40 paź 27 18:50 spis.telefonów
drwx------  2 skrypt users  4096 paź 25 18:32 .ssh
-rw-------  1 skrypt users  6709 paź 29 11:21 .viminfo
[skrypt@vhomer ~]$
```

Rys. 81. Przykład użycia poleceń `tar` i `gzip`

Analogiczny rezultat można uzyskać również w nieco inny sposób, używając polecenia `tar` z opcją `z`:

```
[skrypt@vhomer ~]$ ls
dead.letter  man.txt  spis.telefonów
[skrypt@vhomer ~]$ tar czvf archiwum.tar.gz *
dead.letter
man.txt
spis.telefonów
[skrypt@vhomer ~]$ ls -la
```

```
razem 140
drwx------  4 skrypt users  4096 paź 29 11:31 .
drwxr-xr-x  4 root   root   4096 paź 24 21:20 ..
-rw-r--r--  1 skrypt users 10997 paź 29 11:31 archiwum.tar.gz
-rw-------  1 skrypt users  3537 paź 27 20:42 .bash_history
-rw-r--r--  1 skrypt users    24 maj 10 18:15 .bash_logout
-rw-r--r--  1 skrypt users   199 paź 24 21:21 .bash_profile
-rw-r--r--  1 skrypt users   124 maj 10 18:15 .bashrc
-rw-------  1 skrypt users  5136 paź 29 11:23 dead.letter
-rw-r--r--  1 skrypt users   120 maj 22 23:18 .gtkrc
-rw-r--r--  1 skrypt users 12535 paź 29 09:19 man.txt
drwxr-xr-x  2 skrypt users  4096 paź 29 11:24 .mc
-rw-r--r--  1 skrypt users    40 paź 27 18:50 spis.telefonów
drwx------  2 skrypt users  4096 paź 25 18:32 .ssh
-rw-------  1 skrypt users  6722 paź 29 11:27 .viminfo
[skrypt@vhomer ~]$
```

Rys. 82. Inny sposób tworzenia skompresowanego archiwum

Różnica wielkości uzyskanych archiwów wynika z różnic ich wewnętrznej struktury i nie świadczy o różnej zawartości logicznej (zadanie 6.).

Redukcja objętości pliku wykonywana może być również przy pomocy innych algorytmów, niż realizowany w aplikacji `gzip`, np. poleceniem `compress` (zakończenie nazwy pliku `.Z` – obecnie rzadko stosowane) i bz2 (zakończenie nazwy pliku `.bz2`) .

7.7 Przetwarzanie plików tekstowych

7.7.1 Wyrażenia regularne

Wyrażenia regularne stanowią mechanizm definiowania wzorców tekstów dla niektórych poleceń. Dzięki wzorcowi można wskazać wiele elementów, które mają być użyte przy wykonywaniu danego polecenia.

Najprostsze wyrażenia regularne można budować za pomocą tzw. znaków wieloznacznych `*` i `?`, jak to ma miejsce w odniesieniu do działań na plikach, opisanych w rozdziale 3.

W przykładzie poniżej otrzymujemy wykaz plików z katalogu `/etc` o nazwach rozpoczynających się od „`mo`”.

```
[skrypt@vhomer ~]$ ls /etc/mo*
/etc/modprobe.conf  /etc/modprobe.conf~
/etc/modprobe.conf.dist  /etc/motd
[skrypt@vhomer ~]$
```

Rys. 83. Przykład użycia polecenia z wyrażeniem regularnym

Dokładnie rzecz biorąc, jeśli znak wieloznaczny występuje w argumencie plikowym, powłoka określa najpierw zbiór plików pasujących do wzorca, a następnie wykonuje dane polecenie dla wszystkich plików z tego zbioru. Zastosowanie wyrażeń regularnych nie ogranicza się jednak tylko do nazw plików.

Bardziej skomplikowane wyrażenia regularne tworzymy posługując się następującym zbiorem znaków specjalnych (*metaznaków*):
- znane już: znak zapytania `?` i gwiazdka `*`,
- nawiasy prostokątne `[` i `]` i łącznik `-`,
- wykrzyknik `!`.

Metaznaki `[`, `]` i `-` pozwalają na określanie zbioru znaków odpowiadającego danej pozycji wzorca.

Tabela 19. Metaznaki w wyrażeniach regularnych

Wyrażenie regularne	Znaczenie wyrażenia regularnego jako wzorca
Wzorzec jednego znaku	
`?`	dokładnie jeden znak
`[ab]`	każdy ze znaków wymienionych w nawiasach prostokątnych
`[a-d]`	każdy ze znaków od a do d, wg odpowiadających im kodów ASCII
`[!ab]`	każdy ze znaków, oprócz a i b
`[!a-d]`	każdy ze znaków nienależących do zbioru {a, b, c, d}
Wzorzec ciągu znaków	
`*`	ciąg znaków dowolnej długości, włączając ciąg pusty
`[ab]*`	zero lub więcej znaków ze zbioru określonego przez wyrażenie regularne w nawiasach
`[a-d]*`	
`[!ab]*`	zero lub więcej znaków nienależących do zbioru określonego przez wyrażenie regularne w nawiasach
`[!a-d]*`	

Jak widać, o ile znak zapytania ma tutaj znaczenie analogiczne jak dla systemu MS DOS, o tyle funkcja znaku gwiazdki jest bardziej złożona.

Dla wygody zdefiniowano też tzw. *klasy* (podzbiory) znaków, co upraszcza budowanie wyrażeń regularnych.

Tabela 20. Klasy znaków w wyrażeniach regularnych

Oznaczenie klasy znaków	Oznaczenie równoważne	Opis
`[:alnum:]`	`0-9a-zA-Z`	Znaki alfanumeryczne
`[:alpha:]`	`a-zA-Z`	Znaki alfabetyczne
`[:cntrl:]`		Znaki sterujące
`[:digit:]`	`0-9`	Cyfry dziesiętne
`[:graph:]`		Symbole graficzne
`[:lower:]`	`a-z`	Małe litery

Tabela 20. (cd.)

[:print:]		Znaki widzialne na wydruku
[:punct:]		Znaki przestankowe
[:space:]		Odstępy poziome
[:upper:]	A-Z	Duże litery
[:xdigit:]	0-9A-F	Cyfry szesnastkowe

Nawiasy prostokątne w oznaczeniach klas są ich integralną częścią. Oznacza to, że przy stosowaniu klas w wyrażeniach regularnych potrzebne są dodatkowe nawiasy, co można prześledzić na podanych poniżej przykładach interpretacji wyrażeń regularnych.

Przykłady interpretacji wyrażeń regularnych

rain.[aA]?[12]

wszystkie napisy ***rain.xyz***, gdzie ***x*** ∈ {a,A}, ***y*** = dowolne, ***z*** ∈ {1,2};

x,y,z – pojedyncze znaki

backup[!1-3]

wszystkie napisy ***backupX,***
gdzie ***X*** jest pojedynczym znakiem różnym od cyfr 1, 2 i 3

/etc/[[:alnum:]]*

wszystkie nazwy plików ***/etc/X,*** gdzie ***X*** jest ciągiem znaków alfanumerycznych

/var/log/[[:alpha:]]*.[0-5]

wszystkie nazwy plików postaci ***/var/log/X.C***, gdzie ***X*** jest ciągiem znaków alfanumerycznych, po którym następuje kropka i jedna cyfra dziesiętna ***C*** z zakresu 0..5

/var/www/html/iwo*.[hH][tT][mM][lL]

wszystkie nazwy plików rozpoczynające się od ***/var/www/html/iwo*** i kończące się kropką, po której występuje ciąg "***html***" zapisany za pomocą liter dowolnej wielkości

/[[:lower:]]*

wszystkie nazwy plików zapisane wyłącznie małymi literami

/*[[:digit:]]*

wszystkie nazwy plików zawierające cyfry dziesiętne

Wykonanie polecenia **ls**

/etc/[[:alpha:]]*[[:punct:]] daje wynik:

```
/etc/group-
/etc/gshadow-
/etc/modprobe.conf~
/etc/passwd-
/etc/shadow-
```

tj. wyświetla nazwy plików z katalogu **/etc** o nazwach lokalnych zbudowanych z liter i zakończonych znakiem przestankowym

7.7.2 **Przeglądanie plików**

more	more [-*liczba*] [*plik*]…
	Polecenie more (ang. *more* – więcej) jest rodzajem filtru i służy do wyprowadzania zawartości wskazanych plików na terminalu, z zatrzymywaniem po wyświetleniu każdej porcji tekstu i możliwością sterowania pracą poprzez polecenia interaktywne, wzorowane na edytorze vi. Do poleceń tych należą: h lub ? wyświetlające podsumowanie wszystkich poleceń, spacja powodująca wyświetlenie następnej porcji tekstu (domyślnie – całego ekranu), enter powodujący wyświetlenie następnych wierszy tekstu (domyślnie – jednej), b lub ^B powodujące wyświetlenie poprzedniej porcji tekstu (domyślnie – całego ekranu) i q lub Q powodujące zakończenie działania polecenia more. Użyte polecenia interaktywne nie są prezentowane na terminalu. Parametr –*liczba* oznacza rozmiar ekranu w wierszach. Użycie polecenia bez parametru plikowego oznacza, iż dane do wyświetlania będą pobierane ze standardowego strumienia wejściowego.

W przykładzie zamieszczonym poniżej przedstawiono pierwszą porcję zawartości pliku /etc/passwd, przy wymuszeniu rozmiaru pola listowania równego 15 wierszy.

```
$ more -15 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
--More--(33%)
```

Rys. 84. Wyświetlanie pierwszej porcji pliku poleceniem more

W ostatnim wierszu wyświetlanym na terminalu znajduje się informacja o względnym położeniu wskaźnika końca wyświetlanej porcji pliku. Plik zawie-

rający mniej wierszy niż wynosi rozmiar ekranu jest wyświetlany w całości, bez przechodzenia do trybu interaktywnego.

Polecenie `more` stało się inspiracją do opracowania polecenia `less`[31], które jest o wiele bardziej rozbudowane pod względem funkcjonalnym, choć wymaga mniejszej ilości zasobów komputera. Polecenia interaktywne pozwalają tu na poruszanie się w obrębie pliku w dowolnych kierunkach. Szczegółowe zapoznanie się z działaniem polecenia `less` zostawiamy Czytelnikowi.

W praktyce często nie jest potrzebne przeglądanie całej zawartości pliku, a jedynie skontrolowanie zawartości tylko jego początkowych lub końcowych wierszy. Do tego celu służą polecenia `head` i `tail`, które omówimy poniżej.

`head`	`head [-[n] liczba] [plik]...`
	Polecenie `head` (ang. *head* – tu: przód) jest filtrem, który dla każdego pliku podanego jako parametr wyświetla jego początkowe wiersze w ilości podanej jako *`liczba`*. Domyślną liczbą wierszy jest 10. Użycie polecenia bez parametru plikowego oznacza, iż dane do wyświetlania będą pobierane ze standardowego strumienia wejściowego.

```
$ head -n 3 /etc/passwd /etc/hosts
==> /etc/passwd <==
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin

==> /etc/hosts <==
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1       vhomer  localhost.localdomain   localhost
$
```

Rys. 85. Przykład użycia polecenia `head`

`tail`	`tail [-[n] liczba] [plik]...`
	Polecenie `tail` (ang. *tail* – ogon) jest filtrem, który dla każdego pliku podanego jako parametr wyświetla jego końcowe wiersze w ilości podanej jako *`liczba`*. Domyślną liczbą wierszy jest 10. Użycie polecenia bez parametru plikowego oznacza, iż dane do wyświetlania będą pobierane ze standardowego strumienia wejściowego.

Następujący przykład dotyczy wyświetlenia 8 ostatnich wierszy historii poleceń użytych w pewnej sesji.

[31] Gra słów: ang. more - *więcej* i ang. *less* – mniej.

```
$ tail -8 .bash_history
fdformat /dev/fd0
mount /dev/fd0
mount /dev/fd0 /media/floppy
clear
ssh les@localhost
ssh skrypt@vhomer
$
```

Rys. 86. Przykład użycia polecenia `tail`

Polecenie `tail` jest szczególnie przydatne w odniesieniu do plików używanych w trybie dopisywania na końcu (ang. *append*), w tym wszelkiego rodzaju kronik[32].

`diff`	`diff [`*`opcje`*`] `*`plik1 plik2`*
	Polecenie `diff` (ang. *difference* – różnica) wyszukuje i wyświetla różnice pomiędzy plikami podanymi jako parametry. Opcja `-i` powoduje brak rozróżnienia pomiędzy literami dużymi i małymi. Opcja `-w` powoduje ignorowanie odstępów podczas porównywania. Opcja `-l` powoduje dzielenie wyjścia na strony.

7.7.3 Filtrowanie zawartości plików

Polecenia opisane w tym punkcie stanowią kolejne filtry zawartości plików, rozszerzając listę zapoczątkowaną przez `more`, `head` i `tail`.

`grep`	`grep [`*`opcje`*`] `*`wzorzec`*` [`*`plik`*`]...`
	Polecenie to przeszukuje wskazane pliki i wyświetla wiersze zawierające tekst pasujący do wzorca. Opcja `-v` odwraca znaczenie wzorca – wyszukiwane będą wiersze niepasujące. Opcja `-n` powoduje wyświetlanie numeracji wierszy pasujących do wzorca. Opcja `-c` wyłącza listowanie pasujących wierszy, zamiast tego wyprowadzana jest liczba wierszy pasujących do wzorca. Opcja `-i` wyłącza rozróżnianie małych i dużych liter. Wzorcem może być tzw. *wyrażenie regularne* (zobacz punkt 7.7.1) - w najprostszym przypadku jest to zwykły ciąg znaków. Użycie polecenia bez parametru plikowego oznacza, iż dane do wyświetlania będą pobierane ze standardowego strumienia wejściowego.

Filtr `grep` zastosujemy teraz do zbadania, gdzie w wykazach użytkowników i ich grup występuje tekst „`root`”.

[32] W systemach *N*X większość kronik systemowych tworzona jest w katalogu `/var/log`.

```
$ grep root /etc/passwd /etc/group
/etc/passwd:root:x:0:0:root:/root:/bin/bash
/etc/passwd:operator:x:11:0:operator:/root:/sbin/nologin
/etc/group:root:x:0:root
/etc/group:bin:x:1:root,bin,daemon
/etc/group:daemon:x:2:root,bin,daemon
/etc/group:sys:x:3:root,bin,adm
/etc/group:adm:x:4:root,adm,daemon
/etc/group:disk:x:6:root
/etc/group:wheel:x:10:root
$ grep -c root /etc/passwd /etc/group
/etc/passwd:2
/etc/group:7
$
```

Rys. 87. Przykłady użycia polecenia `grep`

Każdy wiersz wyprowadzany przez `grep` rozpoczyna się nazwą badanego pliku zakończoną dwukropkiem.

`wc`	`wc [-lwcL] [`*`plik`*`]…`
	Polecenie `wc` (od ang. *word count* – liczba słów) zlicza w podanych plikach liczbę bajtów (opcja `-c`), słów (opcja `-w`), wierszy (opcja `-l`) lub określa tylko długość najdłuższego wiersza (opcja `-L`). Użycie polecenia bez parametru plikowego oznacza, że dane do wyświetlania będą pobierane ze standardowego strumienia wejściowego. Brak opcji jest równoważny podaniu `-lwc`.

Przykłady działania polecenia `wc` zamieszczono poniżej.

```
$ wc /etc/passwd
  39   60 1771 /etc/passwd
$ wc -l /etc/passwd
39 /etc/passwd
$ wc -w /etc/passwd
60 /etc/passwd
$ wc -c /etc/passwd
1771 /etc/passwd
$ wc -L /etc/passwd
69 /etc/passwd
$
```

Rys. 88. Przykłady użycia polecenia `wc`

`tr`	`tr [`*`opcje`*`]… `*`wzór1`*` [`*`wzór2`*`]`

	Polecenie `tr` (od ang. *translate* – przetłumacz) jest filtrem, który tłumaczy tekst wg podanych zbiorów znaków lub usuwa wskazane znaki z tekstu. Tekst pierwotny pobierany jest ze standardowego strumienia wejściowego, tekst zmieniony wyprowadzany jest na standardowy strumień wyjściowy. `wzór1` i `wzór2` są zbiorami znaków (wzorcami) definiującymi sposób działania. Tłumaczenie zachodzi wtedy, gdy podano dwa wzorce i nie wystąpiła opcja `-d`. Jeśli drugi wzorzec jest dłuższy niż pierwszy, to dodatkowe znaki na jego końcu są ignorowane. Jeśli drugi wzorzec jest krótszy od pierwszego, to na końcu drugiego wzorca powiela się jego ostatni znak do wymaganej długości. Opcja `-t` powoduje skrócenie zbioru *`wzór1`* do długości zbioru *`wzór2`* przed przystąpieniem do tłumaczenia. W trakcie tłumaczenia tekstu wejściowego każdy jego znak występujący we wzorcu `wzór1` jest zastępowany w tekście wyjściowym odpowiadającym mu (pod względem numeru zajmowanej pozycji) znakiem wzorca `wzór2`. Przy usuwaniu znaków z tekstu drugi parametr nie jest używany, opcja `-d` powoduje usuwanie z wejścia znaków znajdujących się w zbiorze *`wzór1`*. Opcja `-s` powoduje usuwanie powtarzających się znaków wskazanych w zbiorze *`wzór1`*. Podstawowe zasady kodowania znaków w zbiorach: - zapis `znak` oznacza pojedynczy znak, - zapis `znak1-znak2` oznacza wszystkie znaki ze zbioru uporządkowanego wg kodu ASCII od znaku `znak1` do znaku `znak2`, - znaki sterujące kodowane są jak niżej (wybrane przykłady): `\\` - znak '\' `\a` - BELL (dzwonek) `\b` - BSP (cofacz) `\f` - FF (zmiana strony) `\n` - NL (nowa linia) `\r` – RET (powrót karetki) `\t` – TAB (znak tabulacji) - `\NNN` – dowolny znak o kodzie ósemkowym `NNN` (1-3 cyfry ósemkowe) Zbiory znaków we wzorcach można też wskazywać za pomocą tzw. *klas* określonych dla wyrażeń regularnych (zobacz punkt 7.7.1).

W przykładzie zamieszczonym poniżej dokonamy nieco karkołomnego zabiegu tłumaczenia liter małych na duże – "na żywo", to znaczy przy użyciu terminala równocześnie jako źródła tekstu oryginalnego, jak i miejsca prezentacji tekstu przetłumaczonego. Walor ilustracyjny tego przykładu będzie większy, jeśli Czytelnik powtórzy przedstawione czynności na swoim terminalu pamiętając, że nieparzyste wiersze konwersacji stanowią tekst wejściowy, parzyste zaś – wyjściowy. Koniec tekstu, jak zawsze, oznaczamy poleceniem klawiszowym `^D`.

```
$ tr a-z A-Z
```

```
asdfghjkl
ASDFGHJKL
123dfg789
123DFG789
^D
$ tr [:lower:] [:upper:]
asdzxcqwe
ASDZXCQWE
^D
$
```

Rys. 89. Przykłady użycia polecenia `tr` na terminalu

Typowym użyciem filtru takiego jak `tr` jest zastosowanie go w potoku poleceń, o czym była mowa w rozdziale 6.

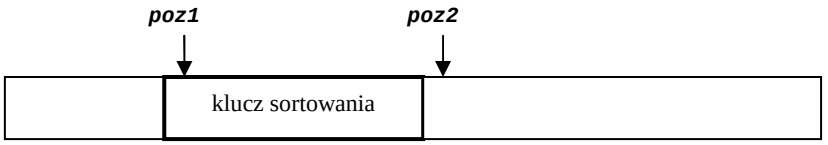
```
$ cat /etc/passwd | tr [a-z] [A-Z]
ROOT:X:0:0:ROOT:/ROOT:/BIN/BASH
...
(dalszy ciąg pominięty)

$
```

Ostatnie przykłady pokazują sposób działania polecenia `tr` dla wzorców różniących się długością.

```
$ cat >aaa <<FIN
> aa
> bbb
> cccc
> KON
$ cat aaa | tr 'a' 'ABC'
AA
bbb
ccc
$ cat aaa | tr 'abc' 'AB'
AA
BBB
BBBB
$
```

`sort`	`sort [`*`opcje`*`] [-o `*`plik_wyjściowy`*`] [+`*`poz1`*`] [-`*`poz2`*`] [`*`plik`*`]...` Polecenie `sort` (ang. *sort* – sortuj) porządkuje zawartość pliku wiersz po wierszu. Opcja `-c` powoduje sprawdzenie, czy wszystkie pliki podane w wykazie są posortowane; jeśli nie, to wypisywany jest komunikat o błędzie. Opcja `-b` powoduje ignorowanie początkowych pustych znaków w wierszu podczas szukania klucza. Opcja `-f` powoduje traktowanie dużych liter jako małe. Opcja `-i` powoduje ignorowanie podczas sortowania znaków spoza zakresu ASCII. Parametr `-o` *`plik_wyjściowy`* oznacza, że wynik sortowania zostanie zapisany do pliku *`plik_wyjściowy`*, zamiast na standardowe wyjście. `+`*`poz1`* i `-`*`poz2`* oznaczają odpowiednio pozycję pierwszego znaku klucza sortowania i pozycję znaku następującego po ostatnim znaku tego klucza. Brak parametru `-`*`poz2`* oznacza, iż klucz kończy się na końcu wiersza.



Rys. 90. Położenie klucza sortowania dla polecenia `sort`

Działanie polecenia sortowania zilustrujemy posługując się fragmentem pliku `/etc/protocols` opisującego protokoły internetowe:

```
ip      0       IP              # internet protocol, pseudo protocol number
#hopopt 0       HOPOPT          # hop-by-hop options for ipv6
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
cbt     7       CBT             # CBT, Tony Ballardie
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway
bbn-rcc 10      BBN-RCC-MON     # BBN RCC Monitoring
nvp     11      NVP-II          # Network Voice Protocol
pup     12      PUP             # PARC universal packet protocol
argus   13      ARGUS           # ARGUS
emcon   14      EMCON           # EMCON
xnet    15      XNET            # Cross Net Debugger
chaos   16      CHAOS           # Chaos
udp     17      UDP             # user datagram protocol
```

Rys. 91. Dane do przykładu sortowania z kluczem

Posortujemy ten plik według nazwy protokołu występującej w trzeciej kolumnie (pozycje znakowe 17.-30.). Wynik umieścimy w pliku `protocols.sort`.

```
[skrypt@vhomer ~]$ sort +17 -31 -o protocols.sort protocols
[skrypt@vhomer ~]$ ls
protocols  protocols.sort
[skrypt@vhomer ~]$ more protocols.sort

argus   13      ARGUS           # ARGUS
cbt     7       CBT             # CBT, Tony Ballardie
egp     8       EGP             # exterior gateway protocol
chaos   16      CHAOS           # Chaos
emcon   14      EMCON           # EMCON
ggp     3       GGP             # gateway-gateway protocol
#hopopt 0       HOPOPT          # hop-by-hop options for ipv6
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
igp     9       IGP             # any private interior gateway
bbn-rcc 10      BBN-RCC-MON     # BBN RCC Monitoring
ip      0       IP              # internet protocol, pseudo protocol number
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
nvp     11      NVP-II          # Network Voice Protocol
pup     12      PUP             # PARC universal packet protocol
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
udp     17      UDP             # user datagram protocol
xnet    15      XNET            # Cross Net Debugger
[skrypt@vhomer ~]$
```

Rys. 92. Wynik przykładowego sortowania z kluczem

`uniq`	`uniq [`*`opcje`*`] [`*`plik_wejściowy`*`] [`*`plik_wyjściowy`*`]`
	Polecenie `uniq` (ang. *unique* – pojedynczy) analizuje posortowany plik wejściowy z punktu widzenia powtarzania się sąsiednich wierszy i wypisuje wynik analizy na wyjściu. Opcja `-c` powoduje wypisywanie liczby powtórzeń danego wiersza razem z jego tekstem. Opcja `-i` powoduje ignorowanie różnic pomiędzy małymi a dużymi literami. Opcja `-d` powoduje wypisywanie tylko linii powtarzających się, opcja `-u` zaś – tylko linii różnych. Aby przy porównywaniu pominąć początkowe znaki wiersza, należy użyć opcji `-liczba`, gdzie `liczba` jest ilością ignorowanych znaków.

Dla zilustrowania działania polecenia `uniq` posłużymy się wcześniej utworzonym plikiem `tekst.uniq`. Na początku sprawdzimy jego zawartość, a następnie posortujemy (czego wymaga polecenie `uniq`) do pliku `tekst.sort`.

```
$ cat tekst.uniq
Ala ma kota. Bill ma nosa.
Ola ma kota. Bill ma nosa.
Jan ma psa. Henry nie ma grypy.
ala ma kota. bill ma nosa.
$ sort tekst.uniq -o tekst.sort
$ cat tekst.sort
```

```
ala ma kota. bill ma nosa.
Ala ma kota. Bill ma nosa.
Jan ma psa. Henry nie ma grypy.
Ola ma kota. Bill ma nosa.
$ uniq tekst.sort
ala ma kota. bill ma nosa.
Ala ma kota. Bill ma nosa.
Jan ma psa. Henry nie ma grypy.
Ola ma kota. Bill ma nosa.
$ uniq -3 tekst.sort
ala ma kota. bill ma nosa.
Ala ma kota. Bill ma nosa.
Jan ma psa. Henry nie ma grypy.
Ola ma kota. Bill ma nosa.
$ uniq -3 -i tekst.sort
ala ma kota. bill ma nosa.
Jan ma psa. Henry nie ma grypy.
Ola ma kota. Bill ma nosa.
$ uniq -3 -ic tekst.sort
      2 ala ma kota. bill ma nosa.
      1 Jan ma psa. Henry nie ma grypy.
      1 Ola ma kota. Bill ma nosa.
$
```

Rys. 93. Przykłady użycia polecenia `uniq`

Wprowadzenie opcji `-3` powoduje ignorowanie różnic w trzech pierwszych pozycjach znakowych każdego wiersza. Pierwsze wywołanie polecenia nie daje jednak na wyjściu żadnego efektu – sąsiadujące ze sobą podobne wiersze nr 1 i 2 różnią się wielkością liter. Drugie wywołanie z dodatkowym ignorowaniem wielkości liter pomija drugi wiersz wejściowy. Trzecie wywołanie polecenia dostarcza statystyki powtórzeń wierszy.

Kolejne dwa filtry umożliwiają wymianę plików tekstowych pomiędzy platformami MS DOS i pochodnymi oraz *N*X. Mówiąc dokładniej, polecenia dokonują korekty sposobu przedstawienia kodu nowego wiersza. Pliki tekstowe utworzone w systemach rodziny MS DOS mają koniec wiersza oznaczony dwoma znakami, o kodach odpowiednio 13_{10} (CR) i 10_{10} (LF), podczas gdy pliki tekstowe w systemach *N*X posiadają w tym miejscu tylko jeden znak, o kodzie 10_{10}.

dos2unix	dos2unix [opcje] [[-o] *plik*]… [-n *plik_wejściowy plik_wyjściowy*]… Filtr ten dokonuje konwersji pliku tekstowego utworzonego w formacie systemu MS DOS na format *N*X. Jeśli nazwa pliku nie jest poprzedzona symbolem opcji lub jest poprzedzona symbolem –o, to przekształcony plik zostanie zapisany w miejscu pliku źródłowego (domyślny tryb działania polecenia). Jeśli nazwa pliku poprzedzona jest symbolem –n to przekształcany jest plik określony jako *plik_wejściowy*, a wynik tej operacji umieszczony w pliku wskazanym przez parametr *plik_wyjściowy*. Opcja –k powoduje zachowanie daty pliku źródłowego.

W celu zademonstrowania działania filtru dos2unix posłużymy się plikiem boot.ini systemu MS Windows XP, który poprzez dyskietkę umieścimy w środowisku *N*X. Montowanie nośnika wymaga skorzystania z uprawnień superużytkownika.

```
$ su - root
Password:
[root@vhomer ~]# mount /dev/fd0 /media/floppy
[root@vhomer ~]# cd /media/floppy
[root@vhomer floppy]# cat boot.ini
[boot loader]
timeout=15
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Profe ↵
ssional" /fastdetect /NoExecute=OptIn
[root@vhomer floppy]# hexdump –c boot.ini
0000000   [   b   o   o   t       l   o   a   d   e   r   ]  \r  \n   t
0000010   i   m   e   o   u   t   =   1   5  \r  \n   d   e   f   a   u
0000020   l   t   =   m   u   l   t   i   (   0   )   d   i   s   k   (
0000030   0   )   r   d   i   s   k   (   0   )   p   a   r   t   i   t
0000040   i   o   n   (   1   )   \   W   I   N   D   O   W   S  \r  \n
0000050   [   o   p   e   r   a   t   i   n   g       s   y   s   t   e
0000060   m   s   ]  \r  \n   m   u   l   t   i   (   0   )   d   i   s
0000070   k   (   0   )   r   d   i   s   k   (   0   )   p   a   r   t
0000080   i   t   i   o   n   (   1   )   \   W   I   N   D   O   W   S
0000090   =   "   M   i   c   r   o   s   o   f   t       W   i   n   d
00000a0   o   w   s       X   P       P   r   o   f   e   s   s   i   o
00000b0   n   a   l   "       /   f   a   s   t   d   e   t   e   c   t
00000c0       /   N   o   E   x   e   c   u   t   e   =   O   p   t   I
00000d0   n  \r  \n
00000d3
[root@vhomer floppy]# dos2unix -n boot.ini ~/boot.iniX
dos2unix: converting file boot.ini to file boot.iniX in UNIX format ...
[root@vhomer floppy]# hexdump -c boot.iniX
0000000   [   b   o   o   t       l   o   a   d   e   r   ]  \n   t   i
0000010   m   e   o   u   t   =   1   5  \n   d   e   f   a   u   l   t
0000020   =   m   u   l   t   i   (   0   )   d   i   s   k   (   0   )
0000030   r   d   i   s   k   (   0   )   p   a   r   t   i   t   i   o
0000040   n   (   1   )   \   W   I   N   D   O   W   S  \n   [   o   p
```

```
0000050   e   r   a   t   i   n   g       s   y   s   t   e   m   s   ]
0000060  \n   m   u   l   t   i   (   0   )   d   i   s   k   (   0   )
0000070   r   d   i   s   k   (   0   )   p   a   r   t   i   t   i   o
0000080   n   (   1   )   \   W   I   N   D   O   W   S   =   "   M   i
0000090   c   r   o   s   o   f   t       W   i   n   d   o   w   s
00000a0   X   P       P   r   o   f   e   s   s   i   o   n   a   l   "
00000b0       /   f   a   s   t   d   e   t   e   c   t       /   N   o
00000c0   E   x   e   c   u   t   e   =   O   p   t   I   n  \n
00000ce
[root@vhomer floppy]# ls -la boot.ini
-rwxr-xr-x  1 root root  211 cze 12 12:37 boot.ini
[root@vhomer floppy]# ls -la ~/boot.iniX
-rwxr-xr-x  1 root root  206 paź 17 19:10 boot.iniX
[root@vhomer floppy]# umount /dev/fd0
umount: /media/floppy: device is busy
umount: /media/floppy: device is busy
[root@vhomer floppy]# cd ~
[root@vhomer ~]# umount /dev/fd0
[root@vhomer ~]# exit
logout

$
```

Rys. 94. Przenoszenie pliku tekstowego pomiędzy środowiskiem MS DOS a *N*X

Na rysunku powyżej podkreślono znaki sterujące `\r` i `\n`. Po zamontowaniu dyskietki i zmianie katalogu bieżącego, wyświetliliśmy zawartość pliku `boot.ini`, najpierw jako tekstu, potem – fizycznego obrazu zawartości pliku (ang. *dump* – zrzut) w postaci znakowej. Podwójny znak zmiany wiersza odnajdujemy jako następujące po sobie znaki kodowane `\r` (CR) i `\n` (LF). Dokonaliśmy konwersji formatu zapisując rezultat do pliku `boot.iniX` w katalogu prywatnym superużytkownika, a następnie wyświetliliśmy jego fizyczny obraz zawartości. Łatwo było stwierdzić, że w pliku wyjściowym znaki `\r` zostały wyeliminowane. Fakt usunięcia znaków dodatkowo sprawdziliśmy, porównując objętość pliku wejściowego i wyjściowego. Różnica 211-206 = 5 wskazała, że pliki zawierały po 5 wierszy. Na koniec odmontowaliśmy nośnik i powróciliśmy do uprawnień zwykłego użytkownika.

`unix2dos`	`unix2dos [opcje] [[-o] plik]…` `[-n plik_wejściowy plik_wyjściowy]…`
	Filtr ten dokonuje konwersji pliku tekstowego utworzonego w formacie systemu *N*X na format MS DOS. Znaczenie opcji i parametrów jest analogiczne jak dla polecenia `dos2unix`.

7.7.4 **Inne polecenia**

Polecenie `echo` umożliwia kontakt zadania uruchomionego jako skrypt z jego użytkownikiem lub operatorem.

`echo`	`echo [-n] [tekst]…`
	Polecenie `echo` wyprowadza kolejno teksty stanowiące parametry wywołania. Użycie opcji `-n` powoduje, iż po ostatnim parametrze nie jest wyprowadzany znak nowej linii. Polecenie użyte bez opcji i parametru wyprowadza pusty wiersz.

```
$ echo

$ echo To jest pierwsza część; echo tekstu.
To jest pierwsza część
tekstu.
$ echo -n To jest pierwsza część; echo " tekstu."
To jest pierwsza część tekstu.
$
```

Rys. 95. Przykłady użycia polecenia `echo`

Możliwość pominięcia znaku nowej linii na końcu wiersza pozwala na łączenie napisów wyprowadzanych przez odrębne polecenia. W przykładzie powyżej, dwa polecenia `echo` zapisano w tym samym wierszu, rozdzielając je średnikiem.

`read`	`read [-a` *`tablica`*`] [-p` *`zachęta`*`] [`*`zmienna`*`]…`
	Polecenie `read` (ang. *read* – czytaj) powoduje wprowadzenie jednej linii tekstu ze standardowego wejścia, wyodrębnienie słów i przypisanie ich do kolejnych zmiennych powłoki. Brak wskazania zmiennych oznacza, iż wprowadzony tekst zostanie przypisany do zastępczej zmiennej `REPLY`. Użycie parametru *`tablica`* powoduje, że kolejne wyodrębnione słowa będą umieszczane w zmiennej tablicowej o tej nazwie, począwszy od indeksu 0. Parametr *`zachęta`* stanowi tekst zaproszenia do wprowadzenia wiersza tekstu i dotyczy wyłącznie terminala. W przypadku, gdy liczba wyodrębnionych słów jest większa niż liczba zmiennych, ostatnia ze zmiennych będzie zawierać wszystkie słowa, które nie zmieściły się w poprzednich zmiennych. W przypadku, gdy liczba zmiennych przekracza liczbę słów, zmienne niewykorzystane będą zawierać wartości puste.

Zamieszczone poniżej przykłady ilustrują podstawowe funkcje polecenia `read`.

```
$ read -p 'Napisz coś: '
Napisz coś: A qq!
$ echo $REPLY
A qq!
$ read -p 'Napisz coś: ' COS
Napisz coś: Znowu?
$ echo $COS
Znowu?
$ read -a TABLICA
Ten tekst trafi do tablicy.
$ set | grep TABLICA
TABLICA=([0]="Ten" [1]="tekst" [2]="trafi" [3]="do"
[4]="tablicy.")
_=TABLICA
$
```

Rys. 96. Przykłady użycia polecenia `read`

Początkowe dwa wywołania polecenia `read` ilustrują sposób użycia domyślnej zmiennej powłokowej `REPLY` oraz zmiennej `COS` wskazanej przez wydającego polecenie. Sprawdzenia rezultatów dokonano poleceniem `echo`. Możliwość wyprowadzenia tekstu zachęty pozwala przy tym na wskazanie użytkownikowi oczekiwanego z jego strony działania. Jest to szczególnie przydatne w skryptach (zobacz rozdział 8.). Czytelnik zwróci uwagę na to, że dla uzyskania odstępu pomiędzy zachętą i odpowiedzią użytkownika, treść zachęty należy umieścić np. w apostrofach.
Wywołanie trzecie dotyczy użycia tablicy do przechowania wielu słów. Sprawdzenie rezultatu - przez użycie polecenia `set` z wybraniem tylko wierszy zawierających tekst `TABLICA`.

7.7.5 Edytor tekstowy vi

Wśród poleceń dotyczących przetwarzania plików tekstowych nie może zabraknąć takiego, które umożliwia tworzenie i modyfikację takich plików. Przedstawimy tu podzbiór funkcji pełnoekranowego edytora `vi`, podstawowego narzędzia superużytkownika pracującego poprzez terminal tekstowy. Należy podkreślić, że podzbiór ten obejmuje znikomą część funkcji klasycznego edytora `vi`. W podręczniku [29] poświęcono mu aż 71 z 431 stron.

`vi`	`vi [-rR]… [`*`plik`*`]…`

	Polecenie (od ang. *visual editor* – edytor wizualny) uruchamia pełnoekranowy edytor tekstowy. Parametry plikowe stanowią nazwy plików kolejno podlegających obróbce. Pliki nieistniejące zostaną utworzone. W przypadku użycia polecenia bez parametru plikowego, zostanie utworzony plik tymczasowy, którego zawartość zostanie zapisana pod nazwą wskazaną przez użytkownika przed zakończeniem pracy edytora. Opcja `-r` uruchamia odzyskiwanie pliku, który poprzednio nie został poprawnie zamknięty przez program `vi`. Opcja `-R` otwiera plik w trybie tylko do odczytu, przez co zabezpiecza go przed przypadkowym uszkodzeniem.

Ostatni (dolny) wiersz terminala ekranowego jest wykorzystywany przez edytor do komunikacji z użytkownikiem jako tzw. *wiersz poleceń*, gdzie wyświetlane są wewnętrzne polecenia edytora wprowadzane przez użytkownika i komunikaty edytora. Pozostałe wiersze ekranu (w liczbie o jeden mniejszej niż aktualna wysokość ekranu liczona w wierszach – standardowo 24) służą do wyświetlania zawartości pliku lub jego fragmentu. Wiersze niewykorzystane oznaczone są tyldą ~. Miejsce w pliku (znak, wiersz), którego dotyczą wydawane polecenia edytora, wskazywane jest kursorem blokowym. Czynności edycyjne dokonywane są w buforze, który powinien zostać zapisany w docelowym pliku przed zakończeniem pracy edytora.

Edytor posiada dwa tryby pracy:
- przeglądania, zwany też trybem poleceń,
- wstawiania, zwany też trybem wejściowym.

7.7.5.1 Tryb przeglądania

W trybie tym, który obowiązuje bezpośrednio po uruchomieniu edytora, możliwe jest przeglądanie zawartości aktualnie edytowanego pliku. Ważniejsze polecenia wewnętrzne dostępne w tym trybie zestawiono poniżej.

Tabela 21. Polecenia edytora `vi` - tryb przeglądania

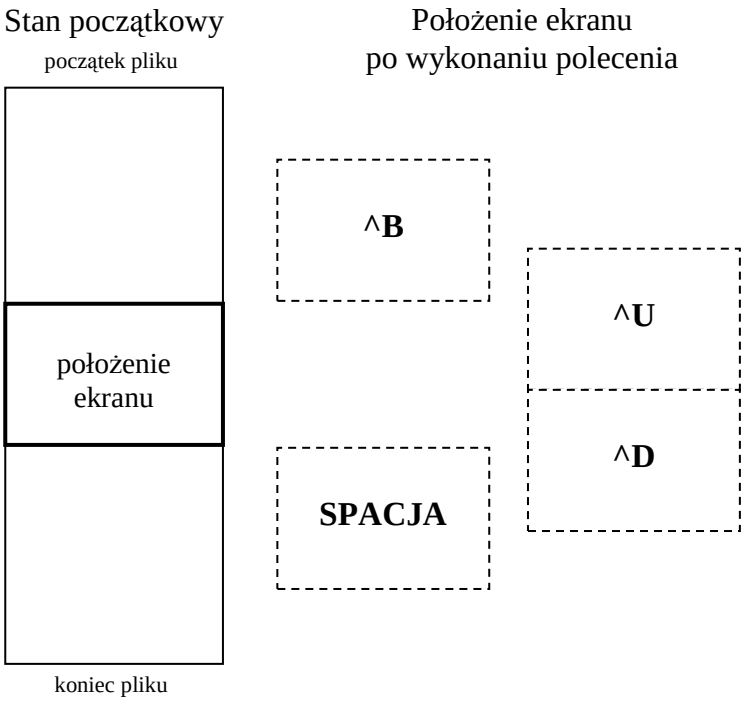
Kod polecenia	Funkcja polecenia
Wprowadzanie zmian w zawartości pliku	
`dd`	usunięcie wiersza wskazywanego przez kursor
`r` *znak*	zastąpienie znaku wskazywanego przez kursor
`^O`	dodanie pustego wiersza poniżej wskazywanego przez kursor i przejście w tryb wstawiania
`^A`	przesunięcie kursora na koniec bieżącego wiersza i przejście w tryb wstawiania
Działania na całym pliku	
`i`	przejście w tryb wstawiania
`:w`	zapis zawartości bufora do pliku
`:w!`	zapis wymuszony, niezależnie od tego czy plik jest chroniony przed zapisem przez użytkownika

:q	zakończenie edycji bez przepisywania bufora do pliku
:x	zapis zawartości bufora do pliku i zakończenie edycji
:x *nazwa*	zapis zawartości bufora do pliku o podanej nazwie i zakończenie edycji

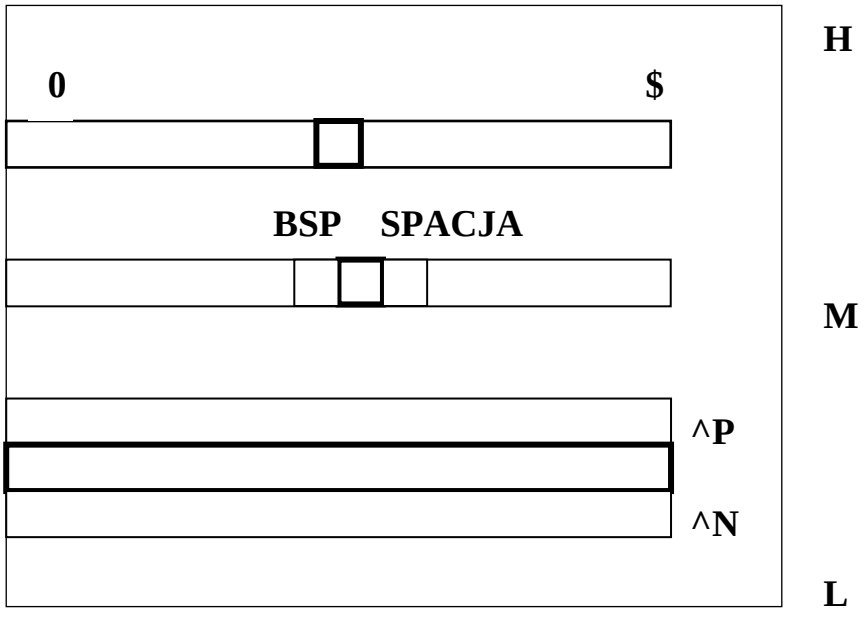
Tabela 21. (cd.)

^G :f	przypomnienie nazwy bieżącego pliku
:f *nowa_nazwa*	zmiana nazwy bieżącego pliku na podaną nazwę
:n	przejście do następnego pliku z listy parametrów plikowych
:p	przejście do poprzedniego pliku z listy parametrów plikowych
!*polecenie_powłoki*	wykonanie polecenia powłoki bez przerywania pracy edytora
Sterowanie kursorem	
H	przesunięcie kursora do górnego wiersza ekranu
M	przesunięcie kursora do środkowego wiersza ekranu
L	przesunięcie kursora do dolnego wiersza ekranu
^P	przesunięcie kursora do poprzedniego wiersza
^N	przesunięcie kursora do następnego wiersza
w	przesunięcie kursora do początku następnego wyrazu
e	przesunięcie kursora do ostatniego znaku następnego wyrazu
b	przesunięcie kursora do początku poprzedniego wyrazu
0 (zero)	przesunięcie kursora do początku wiersza
$	przesunięcie kursora do końca wiersza
SPACJA	przesunięcie kursora o jeden znak w prawo
BSP	przesunięcie kursora o jeden znak w lewo
Wyszukiwanie tekstu według wzorca	
/*wzorzec*	zdefiniowanie wzorca i poszukiwanie pasującego tekstu od aktualnego położenia kursora w kierunku końca pliku („w dół”)
/	powtórzenie wyszukiwania według istniejącego wzorca „w dół”
?*wzorzec*	jak dla polecenia /*wzorzec* lecz w kierunku początku pliku („w górę”)
?	jak dla polecenia / lecz „w górę”
Uzyskiwanie pomocy	
:h	wywołanie podpowiedzi na temat edytora

Kody poleceń nie są wyświetlane na ekranie. Należy przypomnieć, że skutek wprowadzenia podanych kodów poleceń w trybie wstawiania będzie odmienny.



Rys. 97. Graficzna reprezentacja funkcji poleceń przesuwania ekranu



Rys. 98. Graficzna reprezentacja funkcji poleceń sterowania kursorem

Pogrubione ramki na rysunku powyżej wskazują położenie kursora (lub wiersz bieżący) przed wykonaniem polecenia. Położenie kursora (lub wiersz bieżący) po wykonaniu polecenia wskazują kody odpowiednich poleceń.

7.7.5.2 Tryb wstawiania

W trybie tym znaki wprowadzane z klawiatury wstawiane są na pozycji wskazywanej przez kursor. W celu powrotu do trybu przeglądania i zastosowania dostępnych tam funkcji należy użyć klawisza **<Esc>**.

7.7.5.3 Użycie edytora

Po wywołaniu edytora poleceniem `vi plik`, gdzie parametr wskazuje plik nieistniejący, na ekranie terminala pojawia się zawartość przedstawiona poniżej.

W celu rozpoczęcia wprowadzania tekstu należy np. użyć polecenia `i`, które powoduje zmianę trybu pracy na wprowadzanie. W trakcie edycji może wielokrotnie zajść potrzeba zmiany trybu pracy edytora. Po zakończeniu opracowania tekstu należy go zapisać do pliku np. poleceniami **<Esc>** (które przywraca tryb przeglądania) oraz `:wq`.

Zalecane użycie edytora vi

Na ogół edytor `vi` jest wywoływany z jednym parametrem plikowym, co chroni przed niezamierzonymi zmianami w innych plikach. Nie zaleca się też pomijania parametru plikowego, gdyż podjęcie decyzji o nazwie i lokalizacji tworzonego pliku już w trakcie edycji, np. sprawdzenie czy plik o wybranej nazwie istnieje i ewentualne zbadanie jego zawartości, aczkolwiek możliwe, jest utrudnione.

Rys. 99. Ekran roboczy edytora `vi` dla nowego pliku o nazwie `plik`

Oczywiście, istnieją edytory tekstu wygodniejsze niż `vi` i jego wersja ulepszona `vim`. Edytor `vi` jest jednak dostępny praktycznie w każdym systemie *N*X, podczas gdy inne edytory – nie zawsze.

7.8 Polecenia zarządzania użytkownikami i ich uprawnieniami

Definicje pojęć związanych z omawianymi tu funkcjami przedstawiono w punkcie 4.2.1.

7.8.1 Tworzenie, modyfikacja i usuwanie grup i użytkowników

Poniżej przedstawimy polecenia dotyczące tworzenia, zmiany i likwidacji grup użytkowników i ich kont. Prawo do wykonywania tych czynności ma wyłącznie superużytkownik.

`groupadd`	`groupadd [-g` *`gid`*`]` *`nazwa_grupy`*
	Polecenie `groupadd` (od ang. *add group* – dodaj grupę) definiuje nową grupę użytkowników i nadaje jej nazwę *`nazwa_grupy`*. Nazwa ta musi być unikalna w systemie. Parametr *`gid`* pozwala na

	ustalenie wewnętrznego (numerycznego) identyfikatora grupy; jego brak oznacza przydzielenie identyfikatora przez system. Wynikiem działania polecenia jest wiersz dodany na końcu pliku `/etc/group`.

useradd	`useradd [-c` *komentarz*`] [-d` *katalog_prywatny*`] ↵`
	`[-g` *grupa_podstawowa*`] [-G` *grupa_dodatkowa*`...]↵`
	`[-s` *powłoka*`] [-u` *uid* `[-o]]` *nazwa_użytkownika*
	Polecenie `useradd` (od ang. *add user* – dodaj użytkownika) tworzy nowe konto użytkownika systemu *N*X, dla którego nazwa logowania określona będzie jako *`nazwa_użytkownika`*. Nazwa ta musi być unikalna w systemie. Przedrostek `-c` pozwala na wprowadzenie tekstowego opisu użytkownika. Przedrostek `-d` poprzedza niestandardową lokalizację katalogu prywatnego tego użytkownika. Przedrostek `-g` poprzedza niestandardową grupę podstawową, do której będzie należeć nowy użytkownik. Jeśli grupa podstawowa nie została podana, nowy użytkownik będzie członkiem grupy domyślnej. Ewentualna dodatkowa przynależność do innych grup definiowana jest po przedrostku `-G`. Przedrostek `-s` poprzedza ścieżkę do niestandardowej powłoki, jaka będzie uruchamiana po otwarciu sesji przez właściciela konta. Przedrostek `-u` pozwala wymusić przypisanie do nazwy użytkownika podanego unikalnego identyfikatora wewnętrznego. Opcja `-o` pozwala na użycie identyfikatora *`uid`* istniejącego użytkownika.
	Grupy wskazane po przedrostkach `-g` i `-G` muszą wcześniej istnieć. Podobnie musi istnieć powłoka wskazana po przedrostku `-s`.
	Wynikiem działania polecenia jest wiersz dodany na końcu pliku `/etc/passwd`.

```
[root@vhomer ~]# groupadd marketing
[root@vhomer ~]# tail /etc/group
nfsnobody:x:65534:
xfs:x:43:
ntp:x:38:
gdm:x:42:
les:x:500:
apache:x:48:
squid:x:23:
webalizer:x:67:
mysql:x:27:
marketing:x:501:
[root@vhomer ~]#
```

Rys. 100. Przykład tworzenia grupy użytkowników

Wiersz dodany do pliku `/etc/group` zawiera w pierwszym polu nazwę grupy, a w trzecim jej identyfikator. Separatorem pól jest dwukropek.

passwd	passwd [nazwa_użytkownika]
	Polecenie passwd (od ang. *password* – hasło) definiuje lub zmienia hasło służące do uwierzytelniania użytkownika. Postać bezparametrowa może być wykorzystana przez użytkownika aktualnie zalogowanego. Postać z parametrem zaś – tylko przez superużytkownika.

Poznane polecenia useradd i passwd wykorzystamy teraz do utworzenia konta dla użytkownika marketboss, który będzie członkiem domyślnej grupy użytkowników oraz dodatkowo grupy marketing. Konto opiszemy za pomocą komentarza „Kierownik Działu Marketingu". Ustalimy też dla niego początkowe hasło.

```
[root@vhomer ~]# useradd -c "Kierownik Działu Marketingu" -
G marketing ↵
  marketboss
[root@vhomer ~]# tail -5 /etc/passwd
apache:x:48:48:Apache:/var/www:/sbin/nologin
squid:x:23:23::/var/spool/squid:/sbin/nologin
webalizer:x:67:67:Webalizer:/var/www/usage:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
marketboss:x:502:502:Kierownik Działu Marketin-
gu:/home/marketboss:/bin/bash
[root@vhomer ~]# tail -5 /etc/group
squid:x:23:
webalizer:x:67:
mysql:x:27:
marketing:x:501:marketboss
marketboss:x:502:
[root@vhomer ~]#
[root@vhomer ~]# passwd marketboss
Changing password for user marketboss.
New UNIX password:
BAD PASSWORD: it's WAY too short
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@vhomer ~]#
```

Rys. 101. Przykład tworzenia konta użytkownika

Pierwszą czynnością tego przykładu było polecenie useradd. Następnie sprawdziliśmy stan pliku /etc/passwd, w którym został dodany nowy wpis odpowiadający użytkownikowi marketboss oraz pliku /etc/group, gdzie został zarejestrowany fakt, iż użytkownik ten jest członkiem grupy marketing, poza podstawowym członkostwem w grupie domyślnej. Domyślną powłoką jest /bin/bash. Tworzenie konta zakończyło utworzenie odpowiadającego mu hasła – jak zwykle wymagane jest jego

dwukrotne podanie. Czytelnik zechce zauważyć komunikat zwracający uwagę na wykrytą przez polecenie `passwd` niedostateczną moc hasła. Oczywiście, zarówno superużytkownik, jak i właściciel konta mogą je poprawić w dowolnym czasie.

Niekiedy zachodzi konieczność dokonania zmian w definicji grupy lub zmian parametrów konta. Służą do tego polecenia, które przedstawione są poniżej.

`groupmod`	`groupmod [–g` *`gid`* `[-o]] [-n` *`nowa_nazwa_grupy`*`] ` *`nazwa_grupy`*
	Polecenie `groupadd` (od ang. *modify group* – modyfikuj grupę) zmienia parametry grupy użytkowników o nazwie *`nazwa_grupy`*. Parametr *`gid`* pozwala na ustalenie wewnętrznego (numerycznego) identyfikatora grupy, przy czym użycie numeru grupy istniejącej wymaga podania opcji `–o`. Jego brak oznacza przydzielenie identyfikatora przez system. Parametr *`nowa_nazwa_grupy`* określa, jak ma być nazwana dana grupa. Wynikiem działania polecenia jest zmiana danych w pliku `/etc/group`.

`usermod`	`usermod [-c` *`komentarz`*`] [-d` *`katalog_prywatny`* `[-m]]↵`
	`  [-g` *`grupa_podstawowa`*`] [-G` *`grupa_dodatkowa`*`…]↵`
	`  [-l` *`nowa_nazwa_użytkownika`*`] [-s` *`powłoka`*`]↵`
	`  [-u` *`uid`* `[-o]]` *`nazwa_użytkownika`*

	Polecenie `usermod` (od ang. *modify user* – modyfikuj użytkownika) zmienia parametry konta użytkownika systemu *N*X, którego nazwą logowania jest *`nazwa_użytkownika`*. Nowa nazwa użytkownika podawana jest po przedrostku `–l`. Nazwa ta musi być unikalna w systemie. Przedrostek `–c` pozwala na wprowadzenie nowego tekstowego opisu użytkownika. Przedrostek `–d` poprzedza niestandardową lokalizację nowego katalogu prywatnego tego użytkownika. Jeśli zawartość dotychczasowego katalogu prywatnego ma zostać przeniesiona do nowej lokalizacji, należy podać opcję `–m`. Przedrostek `–g` poprzedza nową niestandardową grupę podstawową, do której będzie należeć nowy użytkownik. Ewentualna dodatkowa nowa przynależność do innych grup definiowana jest po przedrostku `–G`. Przedrostek `–s` poprzedza ścieżkę do nowej niestandardowej powłoki, jaka będzie uruchamiana po otwarciu sesji przez właściciela konta. Przedrostek `–u` pozwala wymusić przypisanie do nazwy użytkownika podanego nowego unikalnego identyfikatora wewnętrznego. Opcja `–o` pozwala na użycie identyfikatora *`uid`* istniejącego użytkownika. Grupy wskazane po przedrostkach `–g` i `-G` muszą wcześniej istnieć. Podobnie musi istnieć powłoka wskazana po przedrostku `–s` i identyfikator *`uid`*, jeśli wystąpił z opcją `–o`. Wynikiem działania polecenia jest zmiana danych w pliku `/etc/passwd`.

Polecenie usermod zastosujemy do zmiany nazwy konta użytkownika marketboss na market, z równoczesną zmianą katalogu prywatnego na /home/boss.

```
[root@vhomer ~]# ls /home
marketboss  les  skrypt
[root@vhomer ~]# usermod -d /home/boss -m -l market market-
boss
[root@vhomer ~]# tail /etc/passwd
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
gdm:x:42:42::/var/gdm:/sbin/nologin
les:x:500:500:les:/home/les:/bin/bash
skrypt:x:501:100:skrypt:/home/skrypt:/bin/bash
apache:x:48:48:Apache:/var/www:/sbin/nologin
squid:x:23:23::/var/spool/squid:/sbin/nologin
webalizer:x:67:67:Webalizer:/var/www/usage:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
market:x:502:502:Kierownik Działu Marketin-
gu:/home/boss:/bin/bash
[root@vhomer ~]# ls /home
boss  les  skrypt
[root@vhomer ~]#
```

Rys. 102. Przykład użycia polecenia usermod

Pierwszą czynnością w przykładzie było zbadanie stanu katalogów prywatnych. Po dokonaniu zmian parametrów konta użytkownika sprawdziliśmy dane konta w pliku /etc/passwd oraz ponownie zbadaliśmy stan katalogów prywatnych. Wymagane zmiany rzeczywiście zostały wprowadzone.

Zbiór poleceń do zarządzania grupami i użytkownikami uzupełniają polecenia usuwania.

groupdel	groupdel *nazwa_grupy*
	Polecenie groupdel (od ang. *delete group* – kasuj grupę) usuwa z systemu definicję niepustej grupy użytkowników o nazwie określonej parametrem.

`userdel`	`userdel [-r] nazwa_użytkownika`
	Polecenie `userdel` (od ang. *delete user* – kasuj użytkownika) usuwa z systemu użytkownika o nazwie określonej parametrem. Opcja `-r` powoduje, że po usunięciu plików z katalogu prywatnego danego użytkownika, usunięty będzie również ten katalog.

7.8.2 Pobieranie informacji o grupach i użytkownikach

Kolejna grupa poleceń służy do uzyskiwania informacji na temat grup i użytkowników.

`id`	`id [nazwa_użytkownika]`
	Polecenie `id` (od ang. *identifier* – identyfikator) wyświetla identyfikatory UID oraz GID użytkownika wskazanego przez parametr lub procesu, który uruchomił to polecenie, przy braku parametru.

```
$ id
uid=501(skrypt) gid=100(users) grupy=100(users)↵
  context=user_u:system_r:unconfined_t
$ id root
uid=0(root) gid=0(root)↵
  grupy=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),↵
10(wheel)
$ id les
uid=500(les) gid=500(les) grupy=500(les)
$
```

Rys. 103. Przykłady użycia polecenia `id`

Wyświetlane dane zawierają:
- wewnętrzny identyfikator użytkownika i jego nazwę (pole `uid`),
- wewnętrzny identyfikator podstawowej grupy, do której należy użytkownik i jej nazwę (pole `gid`),
- listę wszystkich wewnętrznych identyfikatorów grup, do których należy użytkownik i ich nazw (pole `grupy`).

`groups`	`groups [nazwa_użytkownika]...`
	Polecenie `groups` (od ang. *groups* – grupy) wyświetla nazwy grup, do których należy użytkownik wskazany parametrem, lub proces, który uruchomił to polecenie, przy braku parametru.

```
[root@vhomer ~]# groups
```

```
root bin daemon sys adm disk wheel
[root@vhomer ~]# groups skrypt les
skrypt : users
les : les
[root@vhomer ~]#
```

Rys. 104. Przykłady użycia polecenia `groups`

`finger`	`finger [`*`nazwa_użytkownika`*`][@`*`nazwa_hosta`*`]`
	Polecenie `finger` (od ang. *finger* – palec) wyświetla dane wskazanego użytkownika. Jeśli podano zarówno nazwę użytkownika jak i nazwę hosta, to poszukiwane są dane wskazanego użytkownika na komputerze o podanej nazwie. Jeśli nazwa użytkownika została podana, ale pominięto parametr określający nazwę komputera, to dane użytkownika poszukiwane są na komputerze, na którym zalogowany jest użytkownik, który użył polecenia (komputer lokalny). Jeśli nie został podany żaden parametr, to wyświetlane są dane logowania użytkowników lokalnych.

```
$ finger root
Login: root                             Name: root
Directory: /root                        Shell: /bin/bash
On since Mon Oct 31 16:19 (CET) on :0 (messages off)
On since Mon Oct 31 16:20 (CET) on pts/1 from :0.0
On since Mon Oct 31 16:20 (CET) on pts/2 from :0.0
   13 seconds idle
On since Mon Oct 31 16:20 (CET) on pts/3 from :0.0
   2 minutes 41 seconds idle
New mail received Sun Oct 30 07:25 2005 (CET)
     Unread since Tue Oct 25 19:41 2005 (CEST)
No Plan.
$ finger les@www.pwsz.nysa.pl
finger: cannot create socket / connect host
$ finger @www.pwsz.nysa.pl
finger: cannot create socket / connect host
$
```

Rys. 105. Przykłady użycia polecenia `finger`

Komunikat `finger: cannot create socket / connect host` informuje, że wykonanie polecenia dotyczącego komputera `www.pwsz.nysa.pl` było niemożliwe. Prawdopodobną przyczyną jest nieaktywność usługi `finger` na zdalnym komputerze (zobacz rozdział 12.).

7.8.3 **Zarządzanie uprawnieniami dostępu do plików**

Dla ustalania, modyfikacji uprawnień dostępu do plików omówionych w punkcie 4.2.2, przewidziano polecenia, które zostaną opisane poniżej.

chown	chown [-R] *nowy_właściciel* [:*nowa_grupa*] *plik*...
	Polecenie chown (od ang. *change owner* – zmień właściciela) zmienia właściciela i opcjonalnie grupę właścicielską wskazanych plików. Właścicielem plików staje się użytkownik wskazany jako *nowy_właściciel*, a grupą właścicielską – podstawowa grupa, do której należy nowy właściciel, chyba że podano parametr *nowa_grupa*. Jeżeli parametr plikowy wskazuje na katalog, to opcja –R powoduje rekurencyjne zmiany we wszystkich podkatalogach i plikach tego katalogu.

W następującym przykładzie zmienimy właściciela i grupę właścicielską pliku /home/skrypt/protocols, sprawdzając stan uprawnień przed i po wykonaniu polecenia chown.

```
[root@vhomer skrypt]# pwd
/home/skrypt
[root@vhomer skrypt]# ls -l
razem 16
-rw-r--r--  1 skrypt users 1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users 1151 paź 29 13:25 protocols.sort
[root@vhomer skrypt]# chown les:les protocols
[root@vhomer skrypt]# ls -l
razem 16
-rw-r--r--  1 les    les   1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users 1151 paź 29 13:25 protocols.sort
[root@vhomer skrypt]#
```

Rys. 106. Przykład użycia polecenia chown

chgrp	chgrp [-R] *nowa_grupa plik*...
	Polecenie chgrp (od ang. *change group* – zmień grupę) działa analogicznie jak chown, lecz dotyczy tylko grupy właścicielskiej plików.

W celu ponownej zmiany grupy właścicielskiej pliku protocols z poprzedniego przykładu na apache, możemy użyć polecenia chgrp, jak pokazano poniżej.

```
[root@vhomer skrypt]# chgrp apache protocols
[root@vhomer skrypt]# ls -l
razem 16
-rw-r--r--  1 les    apache 1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users  1151 paź 29 13:25 protocols.sort
[root@vhomer skrypt]#
```

Rys. 107. Przykład użycia polecenia `chgrp`

`chmod`	`chmod [-R]` *`lista_zmian plik`*`…` `chmod [-R]` *`ósemkowy_tryb_dostępu plik`*`…`
	Polecenie `chmod` (od ang. *change mode* – zmień tryb) modyfikuje tryb dostępu do wskazanych plików według podanych parametrów. *`lista_zmian`* ma postać `[zmiana][,zmiana]…` , gdzie `[zmiana]` jest zdefiniowana jako `[ugoa][+-=][rwxXst]`. Symbole `u`, `g`, `o` i `a` oznaczają odpowiednio prawa dla właściciela, grupy, pozostałych użytkowników i wszystkich użytkowników (`a` jest równoważne `ugo`). Znak `+` oznacza dodanie wskazanych praw, znak `-` - odebranie wskazanych praw, znak `=` zaś - nadanie praw dokładnie takich, jak wskazane. Same prawa określone są symbolami z listy stanowiącej trzeci człon elementu `[zmiana]`. Znaczenie symboli `r`, `w`, `x`, `s` i `t` jest oczywiste. Symbol `X` oznacza prawo wykonania tylko gdy plik jest katalogiem. *`ósemkowy_tryb_dostępu`* określa tryby dostępu do pliku za pomocą 1-4 cyfr ósemkowych, przy czym nieznaczące zera mogą być pomijane. Jeżeli parametr plikowy wskazuje na katalog, to opcja `–R` powoduje rekurencyjne zmiany we wszystkich podkatalogach i plikach tego katalogu.

Poniżej przedstawimy przykłady użycia polecenia z argumentem symbolicznym i ósemkowym.

```
$ ls -l
razem 16
-rw-r--r--  1 skrypt apache 1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users  1151 paź 29 13:25 protocols.sort
$ chmod u+x,g-r,o=rw protocols
$ ls -l
razem 16
-rwx---rw-  1 skrypt apache 1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users  1151 paź 29 13:25 protocols.sort
$ chmod 600 protocols
$ ls -l
razem 16
-rw-------  1 skrypt apache 1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users  1151 paź 29 13:25 protocols.sort
$
```

Rys. 108. Przykłady użycia polecenia `chmod`

umask	umask [–S] [*nowa_maska*]
	Polecenie umask (od ang. *user mask* – maska użytkownika) pozwala na wyświetlenie (wywołanie bez parametru) lub zmianę (wywołanie z parametrem) maski użytkownika wykorzystywanej w czasie tworzenia plików. *nowa_maska* może być podana w postaci symbolicznej lub ósemkowej. Użycie polecenia umask z opcją –S powoduje wyświetlenie trybu dostępu w postaci symbolicznej.

```
$ umask
0022
$ touch plik1
$ ls -l
razem 20
-rw-r--r--  1 skrypt users     0 paź 31 17:02 plik1
-rw-------  1 skrypt apache 1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users  1151 paź 29 13:25 protocols.sort
$ umask 077
$ touch plik2
$ ls -l
razem 24
-rw-r--r--  1 skrypt users     0 paź 31 17:02 plik1
-rw-------  1 skrypt users     0 paź 31 17:02 plik2
-rw-------  1 skrypt apache 1151 paź 29 13:24 protocols
-rw-r--r--  1 skrypt users  1151 paź 29 13:25 protocols.sort
$
```

Rys. 109. Przykład użycia maski użytkownika

Przy standardowej masce użytkownika (022) utworzony plik1 otrzymał atrybuty rw-r--r--. Dla maski 077 plik2 otrzymał atrybuty rw-------.

su	su [–] [-c *polecenie*] [-s *powłoka*] *nazwa_użytkownika*
	Polecenie su (od ang. *switch user* – przełącz użytkownika) jest wbudowanym poleceniem powłoki, które wywołuje nową powłokę interaktywną lub dowolne polecenie, z równoczesną zmianą identyfikatora (UID) użytkownika i jego przynależności grupowej na te, które charakteryzują użytkownika wskazanego w parametrze. *polecenie* jest pojedynczym wierszem zawierającym polecenia do wykonania. Parametr *powłoka* wskazuje powłokę, która ma być wywołana. W razie pominięcia tego parametru wykorzystana zostanie powłoka domyślna użytkownika (określona w pliku /etc/passwd). *nazwa_użytkownika* zaś wskazuje, czyje prawa i zasoby mają być wykorzystane do wykonania wskazanego polecenia lub urucho-

	mienia powłoki. Opcja **–** powoduje aktualizację środowiska wykonawczego, m.in. ustalenie nowej wartości zmiennej `PATH`, ustawienie katalogu bieżącego na katalog prywatny wskazanego użytkownika oraz wykonanie jego plików startowych. Jeżeli polecenie `su` zostało wydane przez użytkownika posiadającego UID różne od zera, zostanie wyprowadzone żądanie podania hasła użytkownika.

Poniżej zamieszczony jest obraz „klasycznej” zmiany roli ze zwykłego użytkownika na superużytkownika.

```
$ su - root
Password:
[root@vhomer ~]#
```

Rys. 110. Wejście w rolę użytkownika `root`

7.9 Inne polecenia

W tym miejscu opiszemy polecenia powłoki, które nie zostały zakwalifikowane do poprzednich grup.

`eval`	`eval [`*`argument`*`]...`
	Polecenie `eval` (od ang. *evaluate* – tu: określ), które jest wbudowanym poleceniem powłoki, składa z kolejnych argumentów zapis polecenia i wykonuje je w powłoce. Kod powrotu polecenia `eval` jest kodem wykonanego polecenia.

`eval` pozwala na wykonanie polecenia, którego zapis powstał w wyniku wykonania operacji na tekstach i został przekazany w postaci argumentu wywołania.

```
$ eval echo Dzisiaj jest  ` date `
Dzisiaj jest pon paź 31 17:10:52 CET 2005
$
```

Rys. 111. Przykład użycia polecenia `eval`

`expr`	`expr` *`wyrażenie`*`...`
	Polecenie `expr` (od ang. *expression* – wyrażenie), oblicza wartości wyrażeń arytmetycznych lub logicznych i wyprowadza wyniki na standardowe urządzenie wyjściowe. Argumenty wyrażeń mogą być liczbami lub ciągami znaków. Operatorami są m.in. symbole działań arytmetycznych: `+`, `-`, `*`, `/` i `%` (obliczanie reszty z dzielenia) lub

	symbole relacji `<`, `<=`, `=`, `=>` i `!=` (różne). Do grupowania wyrażeń można używać nawiasów, wymagane jest jednak ich cytowanie. Bardziej zaawansowane możliwości polecenia opisuje podręcznik systemu *N*X – zobacz `man expr`.

```
$ a=`expr 12 % 5`
$ echo $a
2
$ b=`expr $a + 5`
$ echo $b
7
$ c=`expr 5 = 7`
$ echo $c
0
$
```

Rys. 112. Proste przykłady użycia polecenia `expr`

`exit`	`exit [n]`
	Polecenie `exit` (ang. *exit* – wyjście) jest wbudowanym poleceniem powłoki, które powoduje zakończenie działania aktywnej powłoki z wygenerowaniem zdarzenia EXIT (9.4.3) i ustaleniem kodu powrotu z powłoki na wskazaną wartość. Przy braku parametru `n`, kod powrotu z powłoki jest kodem powrotu ostatnio wykonywanego polecenia.

7.10 Zadania i ćwiczenia

1. Sprawdź:
   a) nazwę bieżącego katalogu,
   b) nazwę zalogowanego użytkownika,
   c) nazwę komputera i bliższe dane systemu operacyjnego.
2. Zmień bieżący katalog na `/var`. Sprawdź jego zawartość w sposób skrócony (bez szczegółów) i pełny (szczegółowy). Nie zmieniając katalogu bieżącego, wylistuj zawartość katalogu `/var/log` w sposób szczegółowy. Dla trzech dowolnie wybranych plików określ ich właściciela i grupę użytkowników.
3. Przejdź do katalogu `/var/spool/cron`. Nie używając żadnych nazw katalogów, przejdź do katalogu `/var`. Jeżeli w Twoim systemie nie ma takiego katalogu, zadanie wykonaj na dowolnym katalogu zagłębionym trzy poziomy poniżej korzenia systemu plików.
4. W katalogu użytkownika utwórz katalog o nazwie `z1`. Za pomocą polecenia `cat` utwórz w nim plik o nazwie `a1` i umieść w nim co najmniej 5 wierszy zawierających nazwi-

ska i imiona znanych pisarzy, tytuły znanych filmów, nazwy ulubionych zespołów muzycznych itp. Sprawdź jego zawartość poleceniem `more`.

5. Za pomocą polecenia `cat` utwórz plik pusty o nazwie `p1`. Sprawdź jego zawartość poleceniem `less`.
6. Za pomocą polecenia `cat` utwórz plik `a2` i umieść w nim co najmniej 8 wierszy zawierających nazwy miejscowości w Polsce lub na świecie. Utwórz plik `a1a2p1` tak, aby zawierał on zawartość plików `a1`, `a2` i `p1` w podanej kolejności; dane winny być pobrane z odpowiednich plików. Sprawdź zawartość pliku `a1a2p1`.
7. Wyświetl początkowe 15 wierszy pliku `a1a2p1` z zadania poprzedniego.
8. Wyświetl statystykę liczby znaków, słów i linii dla pliku z zadania 6.
9. Wyświetl końcowe 10 wierszy pliku `/etc/group`.
10. Korzystając z polecenia `more` wyświetl zawartość dużego pliku tekstowego np. `/etc/passwd`, `/var/log/messages` lub innego i sprawdź działanie poleceń sterujących przeglądaniem. Nie zmieniaj zawartości tego pliku!
11. Napisz polecenia, które co 15 sekund wyświetlą na terminalu aktualną datę i czas. Działanie to powinno bez interwencji użytkownika zakończyć się po 12 cyklach.
12. Przygotuj dyskietkę 1,44 MB sformatowaną dla MS DOS, na której jest minimum kilkanaście KB wolnego miejsca.
    a) Zamontuj dyskietkę poleceniem `mount`, odpowiednio dobierając parametry, w tym typ systemu plików, urządzenie i punkt montowania.
    b) Zmień katalog bieżący na punkt montowania dyskietki. Wyświetl zawartość dyskietki w formacie długim.
    c) Wydaj polecenie odmontowania dyskietki. Jaki widzisz efekt i dlaczego?
    d) Zmień katalog bieżący na dowolny niebędący punktem montowania dyskietki lub jego podkatalogiem i ponownie wydaj polecenie odmontowania. Jaki jest rezultat próby odmontowania?
    e) Na podstawie wykonanych prób sformułuj wniosek dotyczący możliwości odmontowania urządzenia.
13. W systemie DOS lub Windows utwórz na dyskietce tekstowy plik o nazwie `dane.dos` wpisując do niego co najmniej 10 linii tekstu dowolnego wiersza lub piosenki.
    W razie potrzeby usuń rozszerzenie nazwy pliku `.txt`, gdyby zostało dodane przez edytor tekstu (np. `Notatnik`).
    a) Zamontuj dyskietkę w systemie *N*X i przejdź do kata-

logu zawierającego plik `dane.dos`, a następnie utwórz w tym samym katalogu dyskietki tekstowy plik `dane.unx` o dokładnie (!) tej samej treści, co plik `dane.dos`.

b) Porównaj objętości obu plików. Który z nich jest dłuższy?

c) Za pomocą poleceń wymienionych we wstępie tego zadania wykonaj konwersję plików: `dane.dos` → `dane.dos.unx` (pozbądź się dodatkowych znaków końca linii) oraz `dane.unx` → `dane.unx.dos` (wstaw dodatkowe znaki końca linii).

d) Za pomocą polecenia `hexdump –x` *`nazwa_pliku`*, wyświetl szesnastkową zawartość wszystkich plików. Które z nich są identyczne, a które różne? W razie niedostępności polecenia `hexdump` skorzystaj z polecenia `xxd` *`nazwa_pliku`* lub podobnego.

e) Odmontuj dyskietkę i przenieś ją do środowiska DOS/Windows. Wyświetl obydwa pliki za pomocą różnych dostępnych edytorów tekstów lub programów listujących. Czy i jakie widzisz różnice?

14. Zaloguj się jako `root`.
    a) Utwórz grupę użytkowników o nazwie `testy`.
    b) W grupie tej utwórz użytkownika `janek,` podając jako komentarz tekst `Jan Testowy,` ustal początkową powłokę na `/bin/sh.` Dodatkowo włącz tego użytkownika do grupy `users`.
    c) Ustal nowemu użytkownikowi hasło i zapamiętaj je.
    d) Wyświetl końcowe fragmenty (maks. 5 wierszy) plików: `/etc/passwd, /etc/group, /etc/shadow` i zweryfikuj poprawność utworzenia nowego użytkownika.
    e) Nie zamykając sesji użytkownika `root`, na tej samej konsoli zaloguj się jako `janek`. Wylistuj zawartość katalogu macierzystego (prywatnego) w sposób pełny (wszystkie pliki) i szczegółowy.
    f) Wyloguj się jako `janek`.
    g) Jako `root`, wylistuj uprawnienia do plików własnych użytkownika `janek`. Pozbaw wszystkich, poza właściciela, wszelkich możliwości dostępu do tych plików.
15. Nie zamykając sesji użytkownika `root`, zaloguj się jako `janek`. Poleceniem `touch` utwórz plik prywatny o nazwie `specbity`.
    a) Sprawdź uprawnienia do pliku.
    b) Zmień prawa dostępu, dodając `x` dla właściciela.

c) Sprawdź uprawnienia do pliku.
d) Dodaj bit SUID.
e) Sprawdź uprawnienia do pliku.
f) Zmień prawa dostępu, dodając x dla grupy.
g) Sprawdź uprawnienia do pliku.
h) Dodaj bit GUID.
i) Sprawdź uprawnienia do pliku.
j) Zmień prawa dostępu, dodając x dla pozostałych.
k) Sprawdź uprawnienia do pliku.
l) Dodaj bit „sticky”.
m) Sprawdź uprawnienia do pliku.

Zanotuj swoje obserwacje dotyczące zasad prezentacji praw dostępu.

16. Czynności (a-m) zadania poprzedniego umieść w pliku o nazwie `prawa.sh`. Dodaj na końcu komentarz zawierający opis swoich obserwacji dotyczących zasad prezentacji praw dostępu.
17. W edytorze `vi` przepisz nieparzyste linie treści zadania 3 zachowując ten sam układ graficzny. Zapisz zawartość do pliku `zadanie3` i zamknij edytor. Ponownie otwórz edytor `vi` w celu kontynuowania edycji tego pliku i dopisz linie parzyste. Zapisz wynik do pliku bez zamykania edytora. Przejdź na początek pliku i wykonaj korektę. Sprawdź zgodność z oryginałem. Podczas edycji korzystaj z poleceń poznanych w ramach wykładu lub znanych z innych źródeł.
18. W katalogu prywatnym utwórz podkatalog `pliki` oraz podkatalogi `archiwum1` i `archiwum2`. W pierwszym z podkatalogów umieść kilka dowolnych plików. W pozostałych podkatalogach umieść skompresowane archiwa zawierające te pliki, używając odpowiednio poleceń `tar` łącznie z `gzip` oraz `tar` z opcją `z`. Posługując się odpowiednimi poleceniami udowodnij, że niezależnie od sposobu uzyskania, skompresowane archiwa zawierają dokładnie (co do bajta) te same pliki źródłowe.
19. Zrealizuj wyprowadzanie zachęty dla polecenia `read` nie korzystając z jej opcji `-p`.

8 Za panią matką idzie pacierz gładko... czyli Skrypty powłokowe

8.1 Wprowadzenie

Skrypt powłokowy (ang. *shell script*), zwany dalej po prostu *skryptem*, stanowi sekwencję czynności do wykonania przez system komputerowy, zapisaną za pomocą poleceń powłoki i umieszczoną w pliku. Skrypty tworzone są w celu automatyzacji często wykonywanych czynności, a także w celu ich rozpowszechniania.

W systemach *N*X skrypty służą m.in. do instalowania składników oprogramowania i zarządzania usługami systemowymi. Ze względu na znaczną standaryzację systemów tej klasy skrypty mogą być często przenoszone bez zmian pomiędzy różnymi środowiskami. Dobrze napisany skrypt potrafi rozpoznać aktualne środowisko operacyjne i wybrać właściwy sposób realizacji zawartego w sobie algorytmu. Tak działają skrypty przeznaczone do instalowania oprogramowania narzędziowego i użytkowego, dzięki czemu unika się żmudnego przepisywania długich poleceń z podręcznika w celu ich uruchomienia na terminalu.

Załóżmy, że opracowaliśmy skrypt wyświetlający na terminalu zawartość pliku `/etc/resolv.conf`[33]. Treść skryptu umieścimy w katalogu bieżącym w pliku `list`, posługując się poleceniem `cat` w trybie nieinteraktywnym.

```
$ cat >list <<EOF
> # skrypt do wyświetlania zawartości pliku
> # /etc/resolv.conf – wersja 1
> if [ -f /etc/resolv.conf ]
>     then
>           echo „Zawartość pliku /etc/resolv.conf:”
>           echo
>           more /etc/resolv.conf
>     else
>           echo „Niestety. Brak pliku /etc/resolv.conf.”
>           echo „Daj znać administratorowi systemu!”
> fi
> # wiersz następny zawiera ogranicznik pliku
> EOF
$
```

Rys. 113. Listowanie zawartości pliku `/etc/resolv.conf`. Wersja 1.

[33] Znaczenie pliku `/etc/resolv.conf` omówiono w punkcie 10.3.

Teraz nadamy plikowi `list` uprawnienie do jego wykonania przez właściciela i uruchomimy skrypt.

```
$ chmod 700 list
$ ./list
; generated by /sbin/dhclient-script
search localdomain
nameserver 192.168.150.2
$
```

Rys. 114. Uruchomienie skryptu `list`

Czytelnik powinien zwrócić uwagę na sposób wywołania skryptu – jego nazwa została poprzedzona oznaczeniem katalogu bieżącego. Wywołanie bez wskazania katalogu prawdopodobnie doprowadziłoby do komunikatu o błędzie:

```
$ list
-bash: list: command not found
$
```

Rys. 115. Błąd wykonania polecenia - plik nieodnaleziony

Zauważmy teraz, że w treści skryptu `list` kilkakrotnie wskazywaliśmy plik, którego zawartość ma być wyświetlona. Pomijając komentarz, nazwa pliku wystąpiła cztery razy. Można postawić pytanie, czy powtarzania tych przypisań nie można uniknąć? Rozwiązanie jest proste, jeśli skorzystamy z mechanizmu zmiennych powłoki. Wystarczy jednorazowo nadać jakiejś zmiennej wartość będącą nazwą listowanego pliku i posłużyć się tą zmienną tyle razy, ile trzeba. Kolejny rysunek przedstawia zapisywanie drugiej wersji skryptu do pliku `list2`, przygotowanie do jego wywołania i samo wywołanie.

```
$ cat >list2 <<EOF
> # skrypt do wyświetlania zawartości pliku
> # /etc/resolv.conf – wersja 2
> PLIK=/etc/resolv.conf
> if [ -f $PLIK ]
>     then
>           echo „Zawartość pliku $PLIK:”
>           echo
>           more $PLIK
>     else
>           echo „Niestety. Brak pliku $PLIK.”
>           echo „Daj znać administratorowi systemu!”
> fi
> # wiersz następny zawiera ogranicznik pliku
> EOF
```

```
$ chmod 700 list2
$ ./list2
; generated by /sbin/dhclient-script
search localdomain
nameserver 192.168.150.2
$
```

Rys. 116. Listowanie zawartości pliku `/etc/resolv.conf`. Wersja 2.

Wersja druga jest zdecydowanie wygodniejsza od pierwotnej. Aby użyć skryptu dla dowolnego innego pliku, wystarczy zmienić polecenie przypisania w wierszu czwartym. Ale i ta wersja może być ulepszona.

Ostatnim krokiem doskonalenia skryptu będzie całkowite usunięcie nazwy listowanego pliku poza treść skryptu. Nazwa tego pliku jest obecnie podawana jako parametr wywołania skryptu. W razie pominięcia parametru skrypt wyświetli komunikat informacyjny.

```
$ cat >list3 <<EOF
> # skrypt do wyświetlania zawartości pliku
> # podanego jako 1. parametr – wersja 3
> if [ $1 ]
> then
>   if [ -f $1 ]
>     then
>           echo „Zawartość pliku $1:”
>           echo
>           more $1
>     else
>           echo „Niestety. Brak pliku $1.”
>           echo „Daj znać administratorowi systemu!”
> fi
> else
>     echo "Użycie: $0 nazwa_pliku"
> fi
> # wiersz następny zawiera ogranicznik pliku
> EOF
$ chmod 700 list3
$ ./list3 /etc/resolv.conf
; generated by /sbin/dhclient-script
search localdomain
nameserver 192.168.150.2
$ ./list3
Użycie: list3 nazwa_pliku
$
```

Rys. 117. Listowanie zawartości pliku. Wersja 3. (uniwersalna)

Zastosowanie do budowy komunikatu informacyjnego zmiennej standardowej `$0` zamiast umieszczenia tam na stałe nazwy pliku ma tę zaletę, że raz napisany skrypt będzie działać poprawnie nawet po umieszczeniu go pod inną nazwą pliku.

8.2 Przekazywanie parametrów

Jednym ze sposobów przekazania skryptowi informacji o żądanym sposobie jego działania jest mechanizm *parametrów wywołania*, analogiczny jak dla programowania w językach trzeciej generacji. Mechanizm ten składa się z dwóch elementów:

- Zbioru parametrów formalnych występujących w treści skryptu,
- Sposobu przypisywania wartości parametrów wywołania do nazw parametrów formalnych.

Z mechanizmem tym zetknęliśmy się już omawiając trzecią wersję skryptu do listowania zawartości pliku (rys. 117). Parametrami formalnymi są zmienne standardowe *nr*, gdzie *nr* jest liczbą naturalną, zaś sposób przypisania im wartości parametrów wywołania pokazano poniżej.

```
$ polecenie arg1  arg2  arg3  arg4  ... arg9
  ↑         ↑     ↑     ↑     ↑         ↑
  $0        $1    $2    $3    $4  ...   $9
```

Rys. 118. Odpowiedniość pomiędzy wartościami zmiennych standardowych a parametrami wywołania

Zmiennej z numerem *k* odpowiada zatem *k*-ty parametr wywołania, gdzie *k* = 1, 2, …9. W przypadku, kiedy niezbędne jest wywołanie skryptu z liczbą parametrów większą niż 9, stosuje się polecenie `shift`, które przesuwa „okno” udostępniające parametry wywołania.

`shift`	`shift [`*n*`]`
	Polecenie `shift` (ang. *shift* – przesuń) przesuwa ciąg parametrów wywołania (nazywanych też *parametrami pozycyjnymi*) o *n* miejsc w lewo, przy czym opuszczenie *n* oznacza 1 miejsce. W ten sposób dotychczasowy n+1 parametr wywołania staje się pierwszym parametrem i zostanie skojarzony ze zmienną `$1`.

```
$ polecenie arg1  arg2 arg3  arg4  arg5 arg6 ...  arg11 arg12...
  ↑                    ↑     ↑     ↑    ↑         ↑
  $0                   $1    $2    $3   $4   ...  $9
```

Rys. 119. Wynik wykonania polecenia `shift 2`

Wartość zmiennej standardowej `$0` nie ulega zmianie w wyniku użycia polecenia `shift`, zmienia się natomiast wartość zmiennej `$#`.

Istnieje możliwość ponownego przypisania wartości zmiennym standardowym; służy do tego polecenie `set`.

`set`	`set – [arg]...`
	Polecenie `set` (ang. *set* – ustal) przypisuje zmiennym standardowym nowe wartości podane jako kolejne argumenty.

```
$ cat >skrypt5 <<EOF
> echo $1 $2 $3 $4\; liczba argumentów=$#
> set – a b c d
> echo $1 $2 $3 $4\; liczba argumentów=$#
> shift 3
> echo $1 $2 $3 $4\; liczba argumentów=$#
> EOF
$ sh ./skrypt5 1 2 3 4
1 2 3 4; liczba argumentów=4
a b c d; liczba argumentów=4
d; liczba argumentów=1
$
```

Rys. 120. Przykład użycia poleceń `shift` i `set`

Wywołanie skryptu powłokowego spełnia oczywiście ogólne zasady budowy polecenia powłoki. W powłoce `bash` możliwe jest użycie większej liczby parametrów niż 9; następne są oznaczane jako `${10}`, `${11}` itd.

8.3 Uruchamianie skryptów

Skrypty można uruchamiać na kilka sposobów. Warunki wymagane do użycia poszczególnych wariantów wywołania oraz towarzyszące temu okoliczności przedstawiono w tabeli poniżej.

Tabela 22. Możliwości i warunki uruchamiania skryptów

Format wywołania	Powłoka	Wymagane prawa wykonania skryptu	Można uruchamiać pliki binarne	Można podawać argumenty wywołania	Tworzony jest nowy proces
`$ sh skrypt`	potomna	nie	tak	tak	tak
`$ skrypt`	potomna	tak	tak	tak	tak
`$ . skrypt`	bieżąca	nie	nie	nie	nie
`$ exec skrypt`	zastąpiona	tak	tak	tak	nie

8.4 Przykłady skryptów

8.4.1 Wsadowe tworzenie grup i kont użytkowników (wersja 1)

Przedstawimy tu pakiet skryptów o nazwie batchusers, służących do wsadowego zarządzania kontami użytkowników, ułatwiających masowe tworzenie i usuwanie kont. Pakiet zawiera skrypty:

- `bmkgroups` – do wsadowego tworzenia grup użytkowników,
- `bmkusers` – do wsadowego tworzenia kont użytkowników,
- `brmusers` – do wsadowego usuwania kont użytkowników.

Skrypty pobierają dane z tabeli umieszczonej w pliku tekstowym o zawartości przedstawionej poniżej.

```
Login Imię Nazwisko Jedn_Org Grupa  Hasło
```

Rys. 121. Format wiersza tabeli danych dla zarządzania kontami (wersja 1.)

Każdy wiersz pliku tabeli danych zawiera 6 pól:

- `Login` - nazwa użytkownika (konta) w systemie,
- `Imię` - imię użytkownika konta,
- `Nazwisko` - nazwisko użytkownika konta,
- `Jedn_Org` - jednostka organizacyjna,
- `Grupa` - nazwa grupy użytkowników systemu *N*X, do której zostanie dopisane nowe konto,
- `Hasło` - hasło dostępu do konta użytkownika.

Pole wiersza musi stanowić pojedyncze słowo, z punktu widzenia składni. W związku z tym wielowyrazowe napisy dozwolone (tylko) w polu `Jedn_Org` należy ująć w cudzysłowy.

Poniżej podamy treść skryptów tworzących pakiet.

```
#!/bin/sh
# Skrypt bmkgroups z pakietu batchusers
# – wsadowe tworzenie grup użytkowników
if [ ! $1 ]
      then echo Użycie: bmkgroups nazwa_tabeli
else
      echo Wsadowe tworzenie grup wg tabeli $1 ...
      echo [C] Leslaw Sieniawski 2002
      gawk -f mkgroups $1
      echo ...wykonane.
fi
```

Rys. 122. Tekst skryptu `bmkgroups` (wersja 1.)

```
#!/bin/sh
# Skrypt bmkusers z pakietu batchusers
# – wsadowe tworzenie kont użytkowników
if [ ! $1 ]
      then echo Użycie: bmkusers nazwa_tabeli
else
      echo Wsadowe tworzenie kont wg tabeli $1 ...
      echo [C] Leslaw Sieniawski 2002
      gawk -f mkusers $1
      echo ------------------------------------------------
      echo ...wykonane.
fi
```

Rys. 123. Tekst skryptu `bmkusers` (wersja 1.)

```
#!/bin/sh
# Skrypt brmusers z pakietu batchusers
# – wsadowe usuwanie kont użytkowników
if [ ! $1 ]
      then echo Użycie: brmusers nazwa_tabeli
else
      echo Wsadowe usuwanie kont...
      echo [C] Leslaw Sieniawski 2002
      gawk -f rmusers $1
      echo ... wykonane.
fi
```

Rys. 124. Tekst skryptu `brmusers` (wersja 1.)

Jak można się zorientować na podstawie podanych powyżej tekstów, poszczególne skrypty posiadają jeden parametr, którym jest nazwa pliku tabeli danych, i podczas wykonywania wywołują – nieopisane do tej pory – polecenie `gawk`. Stanowi ono dostępną na zasadach licencji GPL wersję programu do przetwarzania plików tekstowych o nazwie `awk`[34]. Program ten przetwarza plik wejściowy czytając jego kolejne wiersze i wykonując na tych wierszach operacje wskazane w pliku sterującym. Skojarzenie nazw skryptów i ich plików sterujących w omawianym pakiecie oraz zawartości tych ostatnich podano poniżej.

Tabela 23. Pliki sterujące dla skryptów pakietu `batchusers`

Skrypt	Plik sterujący
bmkgroups	mkgroups
bmkusers	mkusers
brmusers	rmusers

```
{
if ( $0 != "" )
      {
      system("groupadd "$5" && echo Utworzono grupę: "$5);
      }
}
```

Rys. 125. Zawartość pliku sterującego `mkgroups` (wersja 1.)

```
{
if ( $0 != "" )
      {
      system("useradd -c "$2"_"$3"["$4"] -g "$5" "$1);
      system("echo "$6" | passwd --stdin "$1" >/dev/null");
      system("echo ------------------------------------");
      system("echo Utworzono konto: "$1", [grupa: "$5"]");
      };
}
```

Rys. 126. Zawartość pliku sterującego `mkusers` (wersja 1.)

```
{
if ( $0 != "" ) {
      system("userdel -r "$1" && echo Usunięto konto: "$1);
      }
}
```

Rys. 127. Zawartość pliku sterującego `rmusers` (wersja 1.)

[34] Opis polecenia `awk` i jego rozszerzeń dostępnych w `gawk` zamieszczono jako dodatek do niniejszego podręcznika.

Program `gawk` został zastosowany w celu wyodrębniania poszczególnych pól z kolejnych wierszy tabeli danych i wywoływania (poprzez funkcję wbudowaną `system`) odpowiednich poleceń systemu *N*X.

Wykorzystując plik tabeli danych o nazwie `tabela-1` wywołajmy kolejno skrypty `bmkgroups` i `bmkusers`:

```
[root@vhomer batchusers]# cat tabela-1
leonek  Leon    Karaluch        Stacja_pomp     technicy        12345678
ejacko  Ewa     Jackowska       Ksiegowosc      administracja   726354Kj
[root@vhomer batchusers]# ./bmkgroups tabela-1
Wsadowe tworzenie grup wg tabeli tabela-1 ...
[C] Leslaw Sieniawski 2002
Utworzono grupe: technicy
Utworzono grupe: administracja
...wykonane.
[root@vhomer batchusers]# ./bmkusers tabela-1
Wsadowe tworzenie kont wg tabeli tabela-1 ...
[C] Leslaw Sieniawski 2002
-----------------------------------------------------
Utworzono konto: leonek, [grupa: technicy], haslo: 12345678 .
-----------------------------------------------------
Utworzono konto: ejacko, [grupa: administracja], haslo: 726354Kj .
-----------------------------------------------------
...wykonane.
[root@vhomer batchusers]#
```

Rys. 128. Przykład użycia skryptów z pakietu `batchusers`

W celu usunięcia niektórych kont użytkowników należy skonstruować nowy plik tabeli danych, bądź z pliku użytego do tworzenia grup i kont usunąć wiersze odpowiadające użytkownikom, których konta NIE mają być skasowane. Następnie należy użyć skryptu `brmusers` z parametrem wskazującym na plik opisujący konta do usunięcia.

8.4.2 Wsadowe tworzenie grup i kont użytkowników (wersja 2.)

Przedstawimy tu zmodyfikowaną wersję poprzedniego pakietu, przeznaczoną do zarządzania kontami studenckimi; pakiet został nazwany `batchstud`. Użytkownicy są zaliczani do grup o nazwach odpowiadających kierunkom studiów. Zasadniczą cechą odróżniającą wersję drugą od pierwszej jest automatyczne generowanie losowych haseł dla każdego konta. Raport z wykonania skryptu tworzącego konta jest sformatowany w postaci umożliwiającej przekazanie danych o nazwie konta i haśle odpowiednim studentom.

Tabela danych zawiera wiersze o polach:

```
     ID Imię Nazwisko Kierunek
```

Rys. 129. Format pliku tabeli danych dla zarządzania kontami (wersja 1.)

Każdy wiersz pliku tabeli danych zawiera 4 pola stanowiące pojedyncze słowa:

`ID`	- identyfikator studenta (numer indeksu),
`Imię`	- imię studenta,
`Nazwisko`	- nazwisko studenta,
`Kierunek`	- kierunek studiów.

W dalszej części tego punktu przedstawimy zawartość plików składających się na pakiet `batchstud`, w tym plików skryptów i plików sterujących wykonaniem odpowiednich funkcji `gawk`. Zachowane są tu nazwy skryptów i ich związek z nazwami plików sterujących (tabela 23.).

```
#!/bin/sh
# Skrypt bmkgroups z pakietu batchstud
# – wsadowe tworzenie grup użytkowników
if [ ! $1 ]
      then echo Uzycie: bmkgroups nazwa_tabeli
else
      echo Wsadowe tworzenie grup wg tabeli $1 ...
      echo [C] Lesław Sieniawski 2002
      gawk -f mkgroups $1
      echo ...wykonane.
fi
```

Rys. 130. Tekst skryptu `bmkgroups` (wersja 2.)

```
#!/bin/sh
# Skrypt bmkusers z pakietu batchstud
# – wsadowe tworzenie kont użytkowników
if [ ! $1 ]
      then echo Uzycie: bmkusers nazwa_tabeli
else
      echo Wsadowe tworzenie kont wg tabeli $1 ...
      echo [C] Lesław Sieniawski 2002
      gawk -f mkusers $1
      echo ------------------------------------------------
      echo ...wykonane.
fi
```

Rys. 131. Tekst skryptu `bmkusers` (wersja 2.)

```
#!/bin/sh
# Skrypt brmusers z pakietu batchstud
# – wsadowe usuwanie kont użytkowników
if [ ! $1 ]
      then echo Uzycie: brmusers nazwa_tabeli
else
      echo Wsadowe usuwanie kont...
      echo [C] Lesław Sieniawski 2002
```

```
      gawk -f rmusers $1
      echo ... wykonane.
fi
```

Rys. 132. Tekst skryptu `brmusers` (wersja 2.)

```
{
if ( $0 != "" )
      {
      system("groupadd "$4" && echo Utworzono grupe: "$4);
      }
}
```

Rys. 133. Zawartość pliku sterującego `mkgroups` (wersja 2.)

```
{
srand(19*(1013-NR) + systime()%411);
if ( $0 != "" )
      {
      KONTO=tolower(substr($2,1,1)$3);
      system("useradd -c "$2"_"$3"["$4"] -g "$4" "KONTO);
      do { LICZBA=int(1000000*rand()) }
          while ( length(LICZBA) < 5 );
      HASLO=substr($2,1,1)substr(LICZBA,1,3)substr($3,1,2)↵
        substr(LICZBA,4,2);
      system("echo "HASLO" | passwd --stdin "KONTO"↵
       >/dev/null");
      system("echo ----------------------------------------↵
        -----------------");
      system("echo '**** Informacja POUFNA - nie do↵
        rozpowszechniania!! ****'");
      system("echo Utworzono konto: "KONTO",↵
        [grupa: "$4"], haslo: '"HASLO"'");
      };
}
```

Rys. 134. Zawartość pliku sterującego `mkusers` (wersja 2.)

```
{
if ( $0 != "" ) {

      KONTO=tolower(substr($2,1,1)$3);
      system("userdel -r "KONTO" && echo Usunięto konto:↵
         "KONTO);
      }
}
```

Rys. 135. Zawartość pliku sterującego `rmusers` (wersja 2.)

Dla ilustracji działania pakietu `batchstud` przygotujemy plik tabeli danych o nazwie `tabela` i uruchomimy skrypty tworzące grupy i konta.

```
[root@vhomer batchstud]# cat tabela
1111/z  Leon    Majcher         zootechnika
2222/d  Zenon   Palka           politologia
3333/z  Katia   Melon           mechanika
6482/d  Zora    Minojew         mechanika
3972/z  Pralek  Mechanos        elektryczny
[root@vhomer batchstud]# ./bmkgroups tabela
Wsadowe tworzenie grup wg tabeli tabela ...
Utworzono grupe: zootechnika
Utworzono grupe: politologia
Utworzono grupe: mechanika
Utworzono grupe: elektryczny
...wykonane.
[root@vhomer batchstud]# ./bmkusers tabela
Wsadowe tworzenie kont wg tabeli tabela ...
-------------------------------------------------------
**** Informacja POUFNA - nie do rozpowszechniania!! ****
Utworzono konto: lmajcher, [grupa: zootechnika], haslo: L855Ma82
-------------------------------------------------------
**** Informacja POUFNA - nie do rozpowszechniania!! ****
Utworzono konto: zpalka, [grupa: politologia], haslo: Z623Pa09
-------------------------------------------------------
**** Informacja POUFNA - nie do rozpowszechniania!! ****
Utworzono konto: kmelon, [grupa: mechanika], haslo: K381Me19
-------------------------------------------------------
**** Informacja POUFNA - nie do rozpowszechniania!! ****
Utworzono konto: zminojew, [grupa: mechanika], haslo: Z669Mi93
-------------------------------------------------------
**** Informacja POUFNA - nie do rozpowszechniania!! ****
Utworzono konto: pmechanos, [grupa: elektryczny], haslo: P706Me38
-------------------------------------------------------
...wykonane.
[root@vhomer batchstud]#
```

Rys. 136. Przykład użycia skryptów z pakietu `batchstud`

Analizę algorytmu zastosowanego do generowania nazw kont i haseł pozostawiamy Czytelnikowi, proponując zarazem budowanie własnych odmian pakietu.

8.5 Zalecana struktura skryptu

Skrypt podlega takim samym regułom jak każdy program komputerowy. Powinien on poprawnie i efektywnie realizować określony algorytm postępowania, sprawdzać poprawność wywołania, wykonalność realizujących go poleceń, reagować na ewentualne błędy wykonania oraz dostarczać użytkownikowi niezbędnych informacji na swój temat, zarówno w kwestii przeznaczenia i sposobu użycia, jak i toku działania. Krytyczne czynności o potencjalnie dotkliwych skutkach, jak np. usuwanie plików lub użytkowników oraz przerywanie działających procesów, winny być autoryzowane przez użytkownika uruchamiającego skrypt. Zapis poleceń w pliku skryptu winien być przejrzysty i zachowywać strukturę blokową algorytmu, a wymagające tego fragmenty opatrzone odpowiednimi komentarzami.

Poniżej przedstawiono zalecaną strukturę treści skryptu. Czytelnik powinien zwrócić uwagę na części oznaczone napisami KOMENTARZ, NAGLOWEK, PROLOG, TRESC ZASADNICZA i EPILOG.

```
#!/bin/sh
# KOMENTARZ: funkcje i sposób wywołania skryptu -
#          podać opis wystarczający do jednoznacznego
określenia,
#          do czego skrypt służy i jak należy się nim
posługiwać
#
# NAGLOWEK: podanie informacji o uruchomionym
#          skrypcie, numerze wersji, autorze, itp.
#          (zwłaszcza dla skryptów długo wykonywa
#          nych), np.

echo "Wykonywanie skryptu $0, wersja 1.xy ..."

# PROLOG: badanie, czy wywołanie zawiera wymaganą
#         liczbę argumentów
#         np. (dla 4 wymaganych argumentów)

if [ $4 ]
    then
           # poprawne wywołanie
           # TRESC ZASADNICZA
              #-- badanie poprawności argumentów,
              #-- np. istnienie plików/katalogów, itd.
           # zasadnicza treść skryptu
              #-- polecenia
```

```
else
           # niepoprawne wywołanie
           echo "Wywołanie: $0 arg1 arg2 arg3 arg4"
              #-- objaśnienie znaczenia argumentów
           echo
fi

# EPILOG:  wypisanie komunikatu końcowego,   zwrócenie #
kodu powrotu, np.

if [ (warunek) ]
     then
           echo "... skrypt $0 wykonany poprawnie."
           exit 0
     else
           echo "... nieudane wykonanie skryptu $0."
           exit 1
fi
```

Rys. 137. Zalecana struktura skryptu

W pierwszym wierszu został wskazany typ powłoki, która winna zostać użyta do interpretacji pliku. Wprawdzie wiersz ten ma postać komentarza, lecz wykrzyknik `!` umieszczony bezpośrednio po znaku komentarza `#` powoduje, że treść komentarza ma znaczenie dla powłoki.

Na potrzeby niniejszego opisu, znakami `#--` wskazano miejsca, w których znajdą się polecenia realizujące algorytm skryptu.

Oczywiście, jak zawsze, należy zachować równowagę pomiędzy treścią i formą. Staranność przy opracowaniu skryptu (liczba sprawdzanych warunków wykonalności, objętość komentarzy) winna być proporcjonalna do jego objętości i częstości używania, a także uprawnień użytkowników nim się posługujących.

Nie rozpowszechniaj niesprawdzonych skryptów

Autor skryptu udostępnianego szerszemu gronu użytkowników winien mieć świadomość, że większość z nich podchodzi do cudzych utworów tego typu z dużą ufnością (aby nie powiedzieć – bezkrytycznie).
Nie mając pewności, że skrypt został wszechstronnie sprawdzony, nie należy go rozpowszechniać.

8.6 Typowe błędy popełniane w skryptach

Poniższa, zapewne niekompletna lista powstała na podstawie oceny prac studenckich. Wykaz już zidentyfikowanych rodzajów błędów powinien jednak pomóc w usuwaniu błędów i optymalizacji powstających skryptów.

1. Błędy dotyczące specyfikacji:
   a. Niedokładne zapoznanie się z treścią i wymaganiami, co prowadzi do opracowania skryptów o funkcjach innych, niż zakładane.
2. Błędy dotyczące niewłaściwego użycia poleceń:
   a. Niezamknięty lub niepoprawnie zamknięty zapis polecenia `cat` tworzącego nowy plik w trybie nieinteraktywnym,
   b. Problemy z ustawieniem bitu uprawnienia SGID (omówionego w punkcie 4.2),
   c. Użycie zbędnych poleceń,
   d. Nieznajomość polecenia `tee` i karkołomne próby uzyskania podobnego efektu za pomocą dziwacznych układów poleceń,
   e. Zapominanie o tym, że wiele poleceń UNIX-a wymaga wskazania pliku, katalogu lub ich zbioru, biorących udział w wykonaniu polecenia,
   f. Umieszczanie zbędnego polecenia typu `echo` *`plik`* na końcu potoku, gdzie *`plik`* jest nazwą pliku wyników, utworzonego w ramach tego potoku,
   g. Nieprawidłowe posługiwanie się konstrukcją warunkowego wykonywania poleceń, gdy są określone polecenia do wykonania po poprawnym i po niepoprawnym wykonaniu polecenia głównego.
3. Błędy dotyczące opracowania pliku:
   a. Niepoprawne przenoszenie komentarzy do następnego wiersza,
   b. Brak ukośnika / na początku ścieżki absolutnej (błąd) lub kończenie ścieżki ukośnikiem (przeważnie zbędnym),
   c. Umieszczanie na początku wiersza znaku kontynuacji >,
   d. Brak spacji po nazwie polecenia,
   e. Niestaranny zapis poleceń do pliku (błędy maszynopisania),
   f. Użycie różnych znaków cytowania z obu stron cytowanego tekstu; np. `` `aaa' ``,
   g. Mnogość zbędnych znaków cytowania, niezależnie od tego, czy tak udziwnione polecenie daje poprawny wynik.
4. Błędy dotyczące środowiska wykonania skryptu:
   a. Zakładanie z góry jakiegoś konkretnego katalogu bieżącego,
   b. Zakładanie z góry jakiegoś układu istniejących uprawnień dostępu do pliku,
   c. Zakładanie z góry, że zmienne powłoki mają jakieś konkretne wartości (np. zmienna `PS1`).

5. Inne błędy, np. bezkrytyczne korzystanie z cudzych rozwiązań, w tym rozwiązań wadliwych.

8.7 Korzyści ze stosowania skryptów

Posługiwanie się skryptami dostarcza następujących korzyści:

- redukuje nakład pracy związanej z instalowaniem i konfigurowaniem systemu operacyjnego i oprogramowania aplikacyjnego,
- ułatwia realizację czynności powtarzalnych, w tym zadań samego systemu operacyjnego,
- zmniejsza ryzyko popełnienia błędu,
- zwiększa przenośność oprogramowania,
- ułatwia dokumentację i wymianę procedur postępowania.

Czynność tworzenia skryptów ma wszelkie cechy programowania. Wobec konieczności analizowania różnych sytuacji, jakie mogą pojawić się w czasie ich wykonywania, sprzyja to kształtowaniu nawyku analitycznego podejścia do problemu.

8.8 Zadania i ćwiczenia

1. Napisz skrypt, który na podstawie nazwy pliku podanej jako 1. parametr wywołania wyprowadzi na terminal:
   - nagłówek zawierający co najmniej pełną (bezwzględną) ścieżkę do pliku i liczbę W wierszy w pliku,
   - N początkowych wierszy pliku, gdzie N jest 2. parametrem wywołania
   - napis `"Pominięto K wierszy…"`, gdzie K jest obliczone jako W-N-M (patrz dalej),
   - M końcowych wierszy pliku, gdzie M jest 3. parametrem wywołania.
   - W przypadku pominięcia 2. parametru przyjąć N=10, w przypadku pominięcia 3. parametru przyjąć M=15.
2. Zapoznaj się z poleceniem `du`. Napisz skrypt, który na podstawie nazwy katalogu podanej jako 1. parametr wywołania:
   - - wyprowadzi nagłówek informacyjny skryptu (wg koncepcji autora),
   - - poda liczbę podkatalogów wskazanego katalogu,
   - - obliczy i wyświetli objętości plików zawartych w podkatalogach tego katalogu,
   - - obliczy i wyświetli łączną objętość całego katalogu.

   Wyniki należy wyrazić w KB, a przez podkatalogi rozumie się tylko pierwszy poziom podkatalogów danego ka-

talogu.

3. Napisz skrypt `poszukiwania`, który wyszukuje w katalogu podanym jako 2. parametr wszystkie zwykłe pliki (nie katalogi) o nazwie określonej przez wyrażenie regularne, będące 1. parametrem wywołania i listuje ich nazwy na terminalu, po uprzednim uporządkowaniu wg alfabetu. Trzeci (opcjonalny) parametr w postaci `-r` powinien powodować odwrotne uporządkowanie listy plików.
4. Napisz skrypt, który na życzenie zmienia znaki zachęty (zmienna `PS1`) w ten sposób, że podaje do wyboru 4 możliwości (tak jak są wyświetlane na terminalu) oraz piątą polegającą na pozostawieniu tego ciągu bez zmian. Użytkownik winien podać z klawiatury jedną z cyfr w zakresie 1-5, po czym skrypt winien wyprowadzić na terminal:
   - - symboliczny (parametryczny) zapis znaków zachęty,
   - - postać zachęty, jaka pojawi się na terminalu,
   - - komunikat "*nazwa_użytkownika*, w dniu *data_bieżąca* zmieniłeś znaki zachęty $PS1".
   - Wartości pól *nazwa_użytkownika* i *data_bieżąca* winny być ustalone programowo w treści skryptu, stosownie do bieżącego stanu.

   nazwa_użytkownika oznacza użytkownika, który rozpoczął sesję pracy terminalowej.
5. Napisz skrypt, który:

   Po wywołaniu bez parametrów wyprowadzi napis: `Ten skrypt przechowywany jest w pliku „nazwa pliku"`, a następnie zbada atrybut wykonywalności pliku przez bieżącego użytkownika i wyświetli odpowiednio komunikat: `Plik wykonywalny` albo `Plik niewykonywalny`.

   Po wywołaniu z parametrami poda ich liczbę oraz wyprowadzi wartości kolejnych parametrów w postaci:
   `Parametr nr 1: `*`wartość1`*
   `Parametr nr 2: `*`wartość2`*
   itd. Przyjmij, że skrypt powinien działać poprawnie dla maks. 9 parametrów wywołania, niezależnie od tego, jak zostanie nazwany przechowujący go plik. Dziesiąty i następne parametry powinny być ignorowane.Czy w treści skryptu `list2` (Rys. 116) w poleceniach `echo` można zamiast cudzysłowów użyć apostrofów? Sprawdź swoją odpowiedź praktycznie.
6. Napisz skrypt, którego parametrem wywołania jest kata-

log. Po wywołaniu należy sprawdzić, czy podany katalog istnieje. **Jeśli nie** – wyprowadzić odpowiedni komunikat. **Jeśli tak** – skrypt winien zażądać wprowadzenia nazwy pliku.
Jeśli w danym katalogu taki plik nie istnieje lub użytkownik wywołujący skrypt nie ma prawa czytania z niego, należy wyprowadzić stosowny komunikat.
W przeciwnym razie należy policzyć liczbę bajtów pliku i wyprowadzić ją na konsolę w postaci odpowiedniego napisu.

7. Napisz skrypt, który wczyta z konsoli dwa teksty, porówna je i odpowiednio do wyniku porównania wyprowadzi komunikat: `Teksty: tekst1 i tekst2 są identyczne`, albo: `Teksty: tekst1 i tekst2 różnią się od siebie`.
   Czynności te skrypt ma powtarzać tak długo, aż jako pierwszy tekst zostanie podany ciąg znaków `KONIEC` albo `FINISZ;` drugi parametr nie powinien być wtedy pobierany. Inną możliwością zakończenia działania skryptu jest podanie jako pierwszego tekstu `JUZ` oraz jako drugiego `KONIEC`.
8. Za pomocą skryptu zaprogramuj kalkulator wykorzystujący polecenie `expr`. Konwersacja z użytkownikiem winna odpowiadać następującemu scenariuszowi:

```
$ Prosty kalkulator. Autor: Imię, Nazwisko, wersja: x.y
Podaj pierwszą wartość:
7
Podaj znak działania [ + - * / % : ]
*
Podaj drugą wartość:
12
7 * 12 = 84
Koniec [tn]? n
------------------
Podaj pierwszą wartość:
16
Podaj znak działania [ + - * / % : ]
-
Podaj drugą wartość:
12
16 - 12 = 4
Koniec [tn]? T
$
```

Kalkulator winien wykrywać użycie nieprawidłowego znaku działania oraz akceptować duże i małe litery w odpowiedzi na pytanie o koniec. Przed przystąpieniem do tworzenia skryptu przedstaw algorytm kalkulatora za pomocą schematu blokowego lub pseudokodu.

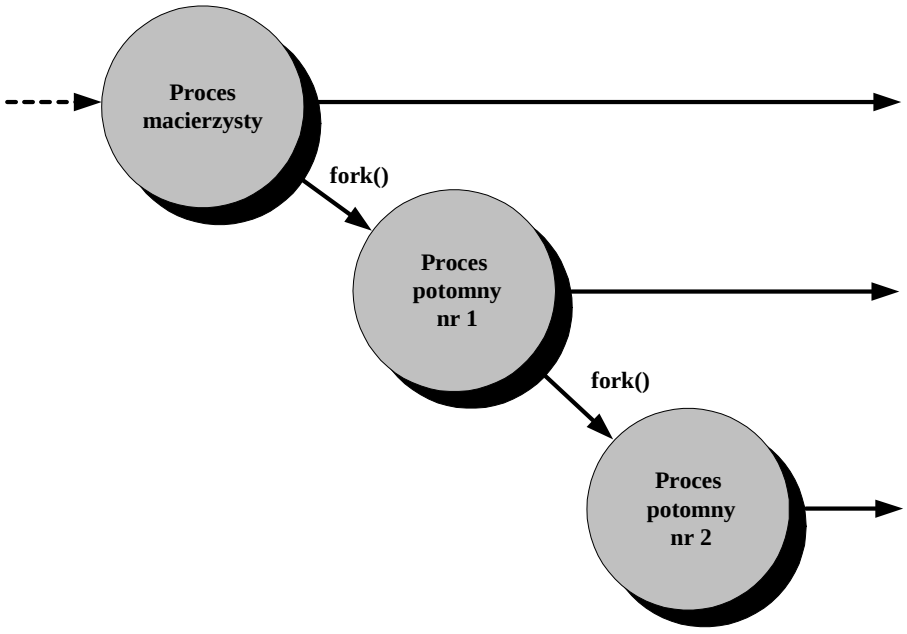
9. Napisz skrypt, który wczyta z konsoli dwa teksty, porówna je, obliczy liczbę znaków i słów w tych tekstach i w zależności od wyniku porównania wyprowadzi komunikaty, odpowiednio:
   `Teksty: tekst1 i tekst2 są identyczne.`
   `Liczba bajtów=b, liczba słów=s.`
   lub
   `Teksty: tekst1 i tekst2 różnią się od siebie.`
   `Pierwszy tekst zawiera b bajtów i s słów.`
   `Drugi tekst zawiera b bajtów i s słów.`
   Czynności te skrypt ma powtarzać tak długo, aż jako pierwszy tekst zostanie podany ciąg znaków `KONIEC` albo `FINISZ`. Jeśli dodatkowo zostanie podany drugi tekst, to powinien zostać wyprowadzony napis: `Drugi parametr zbędny.`
   Uwaga: sposób wprowadzenia tekstów z konsoli ustala autor skryptu.
10. Napisz skrypt kreślący za pomocą znaków kodu ASCII kontur owalu o zadanej wysokości (podanej jako liczba wierszy) i szerokości (podanej jako liczba znaków w wierszu reprezentującym największą szerokość figury). W każdym wierszu skrypt powinien dobierać znak najbardziej odpowiedni dla pochylenia krawędzi idealnego owalu w danym miejscu; w tym celu użyj funkcji.
11. Opracuj skrypt przedstawiający graficznie (za pomocą znaków ASCII) podaną liczbę piramidalną - skrypt winien wyprowadzać obraz płaskiej piramidy "kul" w ilości równej podanej liczbie.
    Uwaga: Liczby piramidalne tworzą ciąg taki, że wartość kolejnej liczby zależy od poprzedniej: $n_1 = 1$; $n_{i+1} = n_i + i + 1$; i = 1, 2,…;

9 Krewni i znajomi królika… czyli Procesy

9.1 Procesy w systemach *N*X

Jak wspomniano w punkcie 3.3, w systemach operacyjnych plik binarny przechowuje kod programu nadający się do bezpośredniego uruchomienia, który nazwiemy tu *programem*. Uruchomienie programu jest najczęściej jednym z elementów tzw. *zadania*, składającego się z szeregu czynności wykonywanych przez komputer. Przed wprowadzeniem kopii pliku programu (lub jego części) do pamięci operacyjnej i przekazaniem mu sterowania, system operacyjny przydziela mu potrzebne zasoby, w tym obszar pamięci operacyjnej oraz wymagane na początku urządzenia i pliki. Po przekazaniu mu sterowania program staje się *procesem* identyfikowalnym przez system operacyjny i działającym w utworzonym dla niego środowisku. Podczas wykonywania rozkazów maszynowych składających się na algorytm programu, proces działa w interakcji z systemem operacyjnym, innymi procesami działającymi w tym samym komputerze lub w komputerach odległych oraz ewentualnie w kontakcie z użytkownikiem. Interakcja z systemem operacyjnym związana jest ze zlecaniem mu do wykonania funkcji usługowych i przyjmowaniem rezultatów tych funkcji. Interakcja z innymi procesami jest związana z ich wzajemnym współdziałaniem. Interakcja z użytkownikiem może wynikać z potrzeby bezpośredniego sterowania pracą programu, wprowadzania danych i prezentacji wyników.

Każdy proces, oprócz procesu `init`, inicjującego działanie systemu, jest tworzony przez inny proces. Proces tworzący nazywany jest *macierzystym* (ang. *parent*), proces utworzony - *potomnym* (ang. *child*), a za tworzenie procesu potomnego odpowiedzialna jest funkcja jądra o nazwie `fork()` (zobacz `man 2 fork`). W wyniku jej wykonania, oprócz istniejącego wcześniej procesu macierzystego, pojawia się nowy proces – potomny względem macierzystego.



Rys. 138. Tworzenie procesów potomnych

Każdy proces dysponuje własnymi zasobami, w tym obszarem pamięci operacyjnej i czasem procesora, o które konkuruje z innymi procesami. Posiada też swój unikalny *identyfikator procesu* - PID (ang. *process identifier*), a miejsce tego procesu w hierarchii określone jest przez *identyfikator procesu macierzystego* - PPID (ang. *parent process identifier*). Proces `init` ma PID=1, PPID=0. Procesy mogą być uruchamiane, zawieszane, wznawiane i usuwane z systemu. Sterowanie procesami jest wykonywane przez funkcje jądra systemu, wywoływane przez inne procesy, w tym obsługę poleceń powłoki, procesy współdziałające i procesy realizujące usługi systemowe.

9.2 Poziomy aktywności systemu

System *N*X może być uruchomiony w różnym zakresie, a mówiąc dokładniej, realizowane przezeń usługi mogą być dostępne w pełni lub tylko częściowo. Wyróżnia się następujące *poziomy aktywności* (ang. *run levels*):

- Poziom 0 – zamykanie systemu,
- Poziom 1 lub S – wprowadzenie systemu w tryb pracy jednego użytkownika (ang. *single user mode*),
- Poziom 2 – wieloużytkownikowy tryb pracy, uruchomione interfejsy sieciowe, lecz bez obsługi sieciowego systemu plików NFS,
- Poziom 3 – jak poziom 2, oraz uruchomiony sieciowy system plików NFS,
- Poziom 4 – (niewykorzystany),

- Poziom 5 - wieloużytkownikowy tryb pracy w środowisku graficznym,
- Poziom 6 – restart systemu.

Definicje poziomów aktywności mogą się nieco różnić w poszczególnych wersjach systemów *N*X.

Zmianę poziomu aktywności realizuje `init` - pierwszy program uruchamiany podczas startu systemu *N*X, odpowiedzialny za uruchamianie i zamykanie innych procesów w systemie. Program `init` korzysta z pliku konfiguracyjnego `/etc/inittab`. W celu przekazania procesowi `init` dyspozycji zmiany, stosuje się polecenie `telinit` *`nr_poziomu`*.

Użycie polecenia `reboot` lub kombinacji klawiszowej <**Ctrl**>+<**Alt**>+<**Del**> jest równoważne poleceniu `telinit 6`. Z kolei polecenia `shutdown –h`, `halt` i `poweroff`, są podobne do polecenia `telinit 0`. Polecenie `reboot` powoduje szybki restart bez zatrzymywania procesów, `halt` – szybkie zatrzymanie systemu, `poweroff` – szybkie zatrzymanie i wyłączenie zasilania. Zmiana poziomu aktywności systemu jest możliwa wyłącznie przez superużytkownika. Systemy Linux najczęściej pracują na poziomie 3 lub 5.

9.3 Badanie stanu procesów

Do sprawdzenia stanu procesów w systemie służy polecenie `ps`.

`ps`	`ps [-eflwH] [`*`pid`*`]…` Polecenie `ps` (od ang. *processes* – procesy) wyświetla dane procesów istniejących w systemie w chwili wykonywania polecenia wskazanych w parametrach *`pid`*, lub wszystkich procesów użytkownika wydającego polecenie, jeśli parametry nie zostały podane. Opcja `-e` powoduje wyświetlenie danych o wszystkich procesach, opcja `–f` włącza format tzw. *drzewiasty*. Opcja `–l` włącza długi format a opcja `-w` włącza szeroki format prezentacji, tj. bez obcinania wierszy dłuższych niż szerokość terminala liczona w znakach. Rezultatem użycia opcji `-H` jest przedstawienie hierarchii procesów.

Wyniki działania polecenia `ps` wyświetlane są w postaci tabelarycznej. Znaczenie nagłówków kolumn tych tabel podano dalej.

Tabela 24. Przegląd nagłówków kolumn wyprowadzanych przez polecenie `ps`

Nazwa kolumny	Zawartość kolumny
CMD	polecenie, które uruchomiło proces
NI	wartość zmiennej *nice* (dodatnia wartość oznacza mniej czasu procesora dla procesu)
PAGEIN	ilość "błędów strony", tj. zapotrzebowań na czytanie strony z dysku
PID	ID procesu
PPID	ID procesu macierzystego
PRI	przypuszczalne kwantowanie procesu w HZ
RSS	rozmiar obszaru rezydentnego w pamięci operacyjnej
SHARE	rozmiar obszaru pamięci dzielonej
SIZE	rozmiar obrazu wirtualnego (kod+program+stos)
S(TAT)	stan procesu: R=runable, S=sleeping, D=uninterruptible sleep, T=trapped, Z=zombie, W=brak stron rezydentnych, N=nice>0
STIME	moment (godzina:minuta) uruchomienia procesu
TIME	czas procesora (CPU) wykorzystany przez proces (godziny:minuty:sekundy)
TRS	wykorzystany obszar urządzenia *swap*
TTY	nazwa terminala, z którego uruchomiono proces lub '?', gdy proces nie korzysta z terminala
UID	ID użytkownika procesu
WCHAN	nazwa funkcji jądra, w której proces śpi

Przykłady działania polecenia `ps` podano poniżej.

```
$ ps
  PID TTY          TIME CMD
 2729 pts/1    00:00:00 bash
 2757 pts/1    00:00:00 ps
$
```

Rys. 139. Przykład użycia polecenia `ps` – procesy własne

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 07:42 ?        00:00:01 init [5]
root         2     1  0 07:42 ?        00:00:00 [migration/0]
root         3     1  0 07:42 ?        00:00:00 [ksoftirqd/0]
root         4     1  0 07:42 ?        00:00:00 [watchdog/0]
root         5     1  0 07:42 ?        00:00:00 [events/0]
root         6     1  0 07:42 ?        00:00:00 [khelper]
root         7     1  0 07:42 ?        00:00:00 [kthread]
root         9     7  0 07:42 ?        00:00:00 [kacpid]
root        62     7  0 07:42 ?        00:00:00 [kblockd/0]
root       113     7  0 07:42 ?        00:00:00 [pdflush]
root       114     7  0 07:42 ?        00:00:00 [pdflush]
root       116     7  0 07:42 ?        00:00:00 [aio/0]
root       115     1  0 07:42 ?        00:00:00 [kswapd0]
root        65     1  0 07:42 ?        00:00:00 [khubd]
root       203     1  0 07:42 ?        00:00:00 [kseriod]
root       363     1  0 07:42 ?        00:00:00 [scsi_eh_0]
root       377     1  0 07:42 ?        00:00:00 [kjournald]
root       889     1  0 07:42 ?        00:00:00 udevd
root       960     1  0 07:42 ?        00:00:00 [kgameportd]
root       989     1  0 07:42 ?        00:00:00 [shpchpd_event]

(wiersze pominięte)

root      2727  2676  0 07:43 pts/1    00:00:00 su - skrypt
skrypt    2729  2727  0 07:43 pts/1    00:00:00 -bash
skrypt    2805  2729  0 07:49 pts/1    00:00:00 ps -ef
$
```

Rys. 140. Przykład użycia polecenia `ps` - wszystkie procesy

Ze względu na znaczną objętość pominięto środkową część wydruku. Procesy są uporządkowane według momentu ich powołania. Zwróćmy uwagę, że w formacie drzewiastym można odczytać zależności pomiędzy procesami, obrazowane wartościami identyfikatorów PID i PPID. Widać np., że proces inicjujący (PID=1) powołał procesy potomne o numerach PID od 2 do 7[35]. Brak ciągłości numeracji PID wynika z tego, iż część procesów zakończyła swój żywot, a zwolnione w ten sposób identyfikatory nigdy nie będą użyte ponownie, aż do restartu systemu. Po restarcie systemu hierarchia procesów będzie tworzona od początku - od procesu `init` z PID=1 i PPID=0.

Wywołanie polecenia `ps` z opcją `-H` dostarcza pseudograficznego obrazu zależności pomiędzy procesami.

[35] Faktycznie, procesów powołanych przez `init` było znacznie więcej. Informacja o nich znajdowała się jednak w pominiętej części wydruku.

```
$ ps -efH

(wiersze pominięte)

root      2657     1  0 07:43 ?        00:00:02   gnome-terminal
--sm-config-preroot   2675  2657  0 07:43 ?        00:00:00
gnome-pty-helper
root      2676  2657  0 07:43 pts/1    00:00:00     bash
root      2727  2676  0 07:43 pts/1    00:00:00       su - skrypt
skrypt    2729  2727  0 07:43 pts/1    00:00:00         -bash
skrypt    2890  2729  0 07:59 pts/1    00:00:00           ps -efH
root      2679  2657  0 07:43 pts/2    00:00:00     bash
root      2680  2657  0 07:43 pts/3    00:00:00     bash

(wiersze pominięte)

$
```

Rys. 141. Fragment obrazu hierarchii procesów

Polecenie, które uruchomiło proces potomny, jest zaznaczane wcięciem tekstu w stosunku do polecenia odpowiadającego procesowi macierzystemu. Proces o PID=2676 jest jednym z potomków procesu o PID=2657 i procesem macierzystym dla innego procesu o PID=2727.

9.4 Polecenia zarządzania procesami

9.4.1 Uruchamianie procesów

Na początku sesji użytkownika uruchamiana jest określona dla niego powłoka domyślna, np. `sh`, `ksh`, `bash`, `csh`, `zsh` lub inna[36]. Powłoka tworzy własne procesy, a jej obecność jest wymagana do interpretacji poleceń podawanych z terminala. Przyjęcie przez system *N*X nowego polecenia, zarówno wbudowanego polecenia powłoki, jak i polecenia realizowanego przez osobny plik, niezwłocznie uruchamia kolejny proces. W ślad za nim, niekiedy pojawia się cała hierarchia procesów potomnych. Nic nie stoi na przeszkodzie, aby polecenie dotyczyło wywołania kolejnej powłoki.

W systemach *N*X istnieje też możliwość wykonania polecenia w innym czasie niż bezpośrednio po wprowadzeniu do systemu. Służą do tego polecenia `at` oraz `crontab`.

[36] Dla przypomnienia: nazwa powłoki domyślnej znajduje się w 7. polu wiersza, w pliku `/etc/passwd`.

at	`at [-f` *`plik_zadania`*`] czas`
	Polecenie `at` (od ang. *at* – tu: w chwili) dopisuje do systemowej kolejki zadań oczekujących zadanie umieszczone w podanym pliku. Wykonywanie zadania rozpocznie się we wskazanym momencie czasu. Jeśli nie został wskazany plik zawierający zadanie, to zostanie ono przeczytane ze standardowego wejścia. *`czas`* może być podany: - w postaci bezwzględnej - jako `HH:MM`, `midnight` (północ), `noon` (południe), `teatime` (16:00), `now` (teraz), - w postaci względnej – jako *`czas_bezwzględny`* + *`liczba`* `[ minutes | hours | days | weeks ]`. Po określeniu czasu może być umieszczona data, w postaci `MMDDYY`, `MM/DD/YY` lub `DD.MM.YY`, `today` (dzisiaj) lub `tomorrow` (jutro).

atq	`atq -q a`
	Polecenie `atq -q a` (od ang. *at queue*– kolejka `at`) wyświetla zawartość systemowej kolejki zadań oczekujących, umieszczonych tam poleceniem `at`.

W przykładzie zamieszczonym poniżej posłużymy się zadaniem, którego treść została wcześniej umieszczona w pliku `wraqq`. Zadanie polega na wysłaniu na terminal użytkownika root wiadomości, której treść jest wkomponowana w plik. Istotne jest, aby wiadomość została dostarczona adresatowi w minutę po wydaniu polecenia ustawiającego zadanie w kolejce. Ponieważ adresat jest zarazem nadawcą, cała procedura odbędzie się na tym samym terminalu.

```
[root@vhomer ~]# more wraqq
write root <<EOF
A qq!
EOF
[root@vhomer ~]# at -f wraqq  now + 1 minutes
job 9 at 2005-11-02 19:10
[root@vhomer ~]# atq -q a
9       2005-11-02 19:10 a root
[root@vhomer ~]#
Message from root@vhomer on <no tty> at 19:10 ...
A qq!
EOF

[root@vhomer ~]#
```

Rys. 142. Planowanie jednorazowej realizacji polecenia

Po wprowadzeniu zadania do kolejki procesów oczekujących otrzymaliśmy informację o nadaniu temu zadaniu numeru 9 i ustaleniu efektywnego czasu realizacji na godzinę 19:10, co dodatkowo sprawdziliśmy poleceniem `atq`. Ostatni fragment Rys. 142 przedstawia efekt wykonania zaplanowanego zadania.

Niektóre zadania realizowane w systemie *N*X wymagają wielokrotnego uruchamiania w zadanych interwałach czasowych. Obsłudze takich potrzeb służy demon `crond`, z którym współpracuje polecenie `crontab`.

`crontab`	`crontab [-u użytkownik] { -e | -l | -r }` Polecenie `crontab` (od ang. *cron table* – tablica crona) zarządza tablicą sterującą cyklicznym uruchamianiem procesu podanego użytkownika. Jeśli nazwa użytkownika nie została podana, to polecenie dotyczy tablicy sterującej użytkownika, który wydał polecenie. Opcja `-e` powoduje uruchomienie edycji tablicy sterującej przy pomocy domyślnego edytora. Opcja `-l` powoduje wyświetlenie zawartości tej tablicy, a opcja `-r` usuwa tę tablicę. Polecenie `crontab` wydane w powłoce uruchomionej przez `su` winno zawsze posiadać parametr `-u`. Tabela crona zawiera: - wiersze komentarza, rozpoczynające się znakiem #, - wiersze puste, - wiersze reguł, opisujące zadania do wykonania i czas ich uruchamiania. Każdy wiersz reguły posiada 6 pól: - pole minuty (dozwolone wartości 0-59), - pole godziny (0-23), - pole dnia miesiąca (0-31), - pole numeru miesiąca (0-12), - pole dnia tygodnia (0-7), przy czym niedzieli odpowiada 0 lub 7, - polecenie zawierające nazwę oraz opcje i argumenty (z pełnymi ścieżkami do plików). W polach 1-5 dozwolone są zapisy w postaci: - gwiazdki *, co oznacza wszystkie wartości dozwolone dla danego pola, - zakres od-do, opcjonalnie uzupełniony o definicję kroku w obrębie podzakresu, - lista wartości oddzielonych przecinkami. Znaczenie reguły jest następujące: polecenie zdefiniowane jako ostatnie, szóste pole reguły jest wykonywane zawsze, gdy minuta, godzina i miesiąc z tej reguły odpowiadają czasowi bieżącemu oraz gdy dzień miesiąca i/lub dzień tygodnia odpowiadają czasowi bieżącemu. Sprawdzanie tabel crona dokonywane jest co minutę.

Dla użytkownika `skrypt` utworzono tablicę crona jak niżej.

```
$ crontab -l
* * * * * /bin/mail -s test skrypt@localhost <tekst

$
```

Rys. 143. Przykładowa tablica crona

Tablica przedstawiona powyżej zawiera pojedynczą regułę, która nakazuje co jedną minutę wysyłać użytkownikowi posiadającemu konto na komputerze lokalnym wiadomość za pomocą poczty elektronicznej. Treść tej wiadomości określona jest w pliku o nazwie `tekst`, a jej tematem jest „`test`”. Po kilku minutach od utworzenia tablicy crona zawartość skrytki pocztowej odbiorcy jest jak poniżej.

```
$ mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/skrypt": 5 messages 2 new 5 unread
 U  1 skrypt@localhost.loc  Wed Nov  2 19:49  17/672   "test"
 U  2 skrypt@localhost.loc  Wed Nov  2 19:50  17/672   "test"
 U  3 skrypt@localhost.loc  Wed Nov  2 19:51  17/672   "test"
>N  4 skrypt@localhost.loc  Wed Nov  2 19:52  16/662   "test"
 N  5 skrypt@localhost.loc  Wed Nov  2 19:53  16/662   "test"
&
```

Można łatwo stwierdzić, że komunikaty faktycznie były wysyłane co jedną minutę. Wykonywanie zaprogramowanego polecenia będzie trwało aż do usunięcia odpowiedniej reguły z tablicy albo usunięcia całej tablicy.

Uwaga, crontab bez opcji!

Użycie polecenia `crontab` bez opcji `-e`, `-l`, `-r` powoduje otwarcie do edycji pustego pliku. Zakończenie edycji przez zapisanie pliku na dysk spowoduje utratę poprzedniej zawartości tablicy crona, jeśli taka istniała.

Dla oceny efektywności implementacji algorytmów przydatna jest możliwość określenia czasu wykonywania określonego zadania. W systemach *N*X przewidziano do tego celu specjalne polecenie:

`time`	`time` *`polecenie`* [*`argumenty`*]
	Polecenie `time` (od ang. *time* – czas) uruchamia wskazane polecenie z jego argumentami, o ile zostały podane, oraz oblicza i wyprowadza dane na temat wykorzystania czasu procesora przez wskazane *`polecenie`* z opcjonalnymi *`argumentami`*.

	Podawane są trzy wartości: `real` – określająca czas zegarowy od uruchomienia do zakończenia, `sys` – określająca efektywny czas procesora w trybie uprzywilejowanym i `user` - efektywny czas procesora w trybie użytkownika. W podręczniku (zobacz `man time`) podane są możliwości uzyskania informacji o wykorzystaniu innych zasobów systemowych, poza czasem procesora.

Pełniejszą definicję składni polecenia `time` znajdzie Czytelnik w punkcie 6.5.2.

Tryby pracy procesora

Procesory uniwersalne posiadają zazwyczaj dwa wyróżnione stany, w których dostępne są inne podzbiory funkcji komputera. W pierwszym ze stanów, nazywanym *trybem uprzywilejowanym* (ang. *priviledged mode*), możliwe jest wykonywanie wszystkich rozkazów maszynowych oraz zapewniony jest pełny dostęp do rejestrów sprzętowych oraz pamięci maszyny. Tryb ten jest wykorzystywany przez jądro systemu operacyjnego, stąd inna nazwa tego stanu - *tryb jądra* (ang. *kernel mode*).
Tryb użytkownika (ang. *user mode*) jest stanem, w którym realizowane są zadania użytkowników i nieuprzywilejowane procesy systemu operacyjnego. Obowiązujące w tym stanie ograniczenia związane z używaniem rozkazów maszynowych, dostępem do określonych obszarów przestrzeni adresowej oraz rejestrów sprzętowych podyktowane są koniecznością wzajemnej izolacji wykonywanych zadań i zapewnienia monopolu jądra w zakresie zarządzania zasobami systemu.

Jako przykład zastosowania polecenia `time` określimy czas realizacji prostej pętli programowej.

```
$ time for ((B=3; B<1234; B++)); do echo B=$B >> echo.lst;
done

real    0m0.063s
user    0m0.032s
sys     0m0.028s
$
```

Rys. 144. Przykład użycia polecenia `time`

9.4.2 Sterowanie wykonywaniem procesów

Istnieją dwa sposoby realizacji procesów w systemach *N*X, różniące się uzależnieniem od interakcji z użytkownikiem.

Pierwszy z nich nosi nazwę *trybu pierwszoplanowego* (ang. *foreground mode*), a proces wykonywany w takim trybie – *procesu pierwszoplanowego* (ang. *foregroud process*). W trybie tym proces posiada bezpośredni dostęp do terminala, za pośrednictwem którego użytkownik steruje zadaniem wydając polecenia, wprowadzając dane wejściowe, zapoznaje się z uzyskiwanymi danymi wyjściowymi oraz poznaje i reaguje na komunikaty wykonywanych programów i samego systemu operacyjnego.

Sposób drugi nazywa się *trybem drugoplanowym* lub *trybem tła* (ang. *background mode*), a proces wykonywany w takim trybie – *procesem drugoplanowym* lub *procesem tła* (ang. *background process*). W trybie tym proces nie korzysta z terminala, a odpowiednie dane wejściowe uzyskuje z plików, potoków poleceń lub łączy międzyprocesowych. Podobne są też miejsca, do których wyprowadzane są dane wyjściowe. Wynika z tego, że proces tła nie reaguje na klawiaturę terminala ani nie wyprowadza na terminal żadnych komunikatów.

Utworzenie procesu przez użytkownika następuje w wyniku skierowania do systemu polecenia do wykonania, tj. wprowadzenia jego nazwy, opcji i parametrów oraz zatwierdzenia klawiszem <**Enter**>. Jeśli bezpośrednio przed użyciem <**Enter**> zostanie umieszczony znak spółki &, to zostanie podjęta próba wykonania tego polecenia jako procesu tła. Próba powiedzie się, jeśli do wykonania polecenia nie będzie potrzebny kontakt z terminalem. Jeśli bezpośrednio przed znakiem <**Enter**> nie będzie znaku spółki, to będzie ono wykonywane jako proces pierwszoplanowy.

Tabela 25. Zapis a sposób realizacji poleceń powłoki

Postać polecenia	Sposób realizacji polecenia
`polecenie` <**Enter**>	proces pierwszoplanowy
`polecenie &` <**Enter**>	proces drugoplanowy

W systemie operacyjnym procesy pierwszo- i drugoplanowe tworzą dwa rozłączne zbiory, z tym że liczba procesów pierwszoplanowych nie może być większa niż jeden. Proces tła może zostać przeniesiony do zbioru procesów pierwszoplanowych, a następnie ponownie skierowany do wykonywania w tle. Podobnie, dłużej trwające zadanie pierwszoplanowe niekorzystające z terminala można przenieść do tła. Do zarządzania procesami tła potrzebny jest jednak mechanizm ich identyfikacji, określony w tabeli dalej.

Identyfikator zadania tła	Znaczenie identyfikatora
%n	Numer zadania o nazwie rozpoczynającej się ciągiem znaków n
%1	Numer pierwszego zadania tła
%%	Numer bieżącego zadania (tj. ostatniego przeniesionego do zbioru zadań tła lub zadania zatrzymanego na pierwszym planie)
%+	
%-	Numer poprzedniego zadania tła

Identyfikator zadania tła jest podawany w nawiasach kwadratowych po utworzeniu zadania. Numeracja ta nie ma nic wspólnego z identyfikatorami PID procesów i jest odrębna dla każdej otwartej sesji.

W kolejnym przykładzie przeszukamy cały system plików w poszukiwaniu plików o nazwach rozpoczynających się od aqq. Wynik poszukiwania i komunikaty diagnostyczne skierujemy do pliku `aqq.lst`.

```
$ find / -name aqq* -print >aqq.lst 2>&1 &
[1] 4636
$
```

Rys. 145. Uruchomienie procesu w tle

Ponieważ zapis polecenia został zakończony znakiem spółki, zostało ono uruchomione w tle. Nadano mu PID = 4636 i numer zadania tła równy 1. Jeśli procesów tła jest więcej niż jeden, to w komunikatach dotyczących ich stanu po zamykającym nawiasie kwadratowym umieszczany jest dodatkowy znak: plus + wskazuje proces bieżący (ostatnio uruchomiony), minusem - są zaś oznaczone wszystkie pozostałe procesy.

Do sterowania zadaniami stosuje się polecenia określone w sposób następujący:

Tabela 26. Polecenia sterowania zadaniami

Polecenie	Funkcja polecenia
%k	Przeniesienie zadania tła numer k na pierwszy plan
fg %k	
%k &	Przeniesienie zadania pierwszoplanowego do zadań tła jako zadanie numer k
bg %1	
^Y	Zawieszenie bieżącego (ostatniego) procesu, gdy będzie on chciał czytać dane z terminala
^Z	Natychmiastowe zawieszenie bieżącego (ostatniego) procesu

Symbole ^Y i ^Z stanowią znane już polecenia klawiszowe terminala.

Poniższy przykład zawiera uruchomienie długo działającej pętli jako zadania tła. Proces ten jest w pewnej chwili przenoszony do zadań pierwszoplanowych, a następnie zatrzymywany (komunikat `Stopped`) i przenoszony do zadań tła. Po pewnym czasie proces zgłasza swoje zakończenie komunikatem `Done`.

```
$ cat >testrap <<EOF
> trap 'echo Odebrano sygnal INT.' exit 1; 1
> for (( i=1; i<1000000; i++ )); do i=$i; done
> EOF
$ . ./testrap &
[1] 1470
$%1
. ./testrap
^Z
[1]+ Stopped      . ./testrap

$ %1 &
[1]+ . ./testrap &

(czekanie ...)

[1]+ Done   . ./testrap
$
```

Rys. 146. Zmiany trybu realizacji procesu

Wykaz procesów tła można uzyskać za pomocą polecenia `jobs`.

`jobs`	`jobs [-lrs]`
	Polecenie `jobs` (od ang. *jobs* – zadania) wyświetla wykaz procesów tła uruchomionych w danej sesji. Opcja `-l` włącza wyświetlanie identyfikatorów procesów. Opcja `-r` ogranicza listę do procesów działających, opcja `-s` zaś – do procesów zawieszonych.

Przykład poniżej ilustruje różnicę w formacie wykazu wyświetlanego przez polecenie `jobs`.

```
[root@vhomer log]# yes >/dev/null &
[root@vhomer log]# jobs
[1]+  Running                 yes >/dev/null &
[root@vhomer log]# jobs -l
[1]+  3811 Running                 yes >/dev/null &
[root@vhomer log]#
```

Rys. 147. Przykłady użycia polecenia `jobs`

Wykorzystano polecenie `yes`, które na standardowe wyjście wypisuje bez końca zadany ciąg znaków (domyślnie „`yes`”). W przykładzie wyjście zostało przekierowane do systemowej „czarnej dziury”.

Do zbioru procesów drugoplanowych należą też tzw. *demony* (ang. *daemons*). Demon jest uaktywniany do wykonania usługi, dla której został opracowany, a po jej zrealizowaniu przechodzi w stan oczekiwania na następne wezwanie. Nazwy demonów z reguły kończą się literą „d”, np. `atd`, `crond`, `ftpd`, `sshd`, `xinetd`.

W sytuacji, gdy liczba wykonywanych procesów jest większa niż liczba dostępnych procesorów (a jest to sytuacja typowa), procesy konkurują pomiędzy sobą o przydział czasu procesora. Zasady tej konkurencji obejmują tzw. priorytety (ang. *priorities*) - proces o wyższym priorytecie ma większą szansę na otrzymanie przydziału czasu procesora. W systemach *N*X priorytety oznaczane są liczbami całkowitymi, przy czym wartość -20 (ujemna) oznacza najwyższy priorytet, a wartość 19 (dodatnia) – najniższy. Wartością średnią jest zero. Za pomocą polecenia `nice` użytkownik może uruchomić proces z priorytetem innym niż domyślny. Priorytet, odziedziczony od procesu macierzystego (w tym `nice`), może być zmieniony poleceniem `renice`.

`nice`	`nice [[-n] `*`przyrost`*`] [`*`polecenie`*` [`*`argument`*`]…]`
	Polecenie `nice` (od ang. *nice* – ładny, miły) uruchamia wskazane polecenie z jego argumentami (o ile zostały podane), nadając utworzonemu w ten sposób procesowi priorytet równy wartości priorytetu domyślnego (tj. priorytetu procesu macierzystego) powiększonej o *`przyrost`*. Wartość parametru *`przyrost`* może być dodatnia (wtedy wynikowy priorytet będzie niższy niż domyślny) lub ujemna (wtedy wynikowy priorytet będzie wyższy niż domyślny). Domyślną wartością przyrostu jest 10. Ujemny *`przyrost`* może podać tylko superużytkownik. Polecenie wydane bez parametrów wyświetla priorytet domyślny.
`renice`	`renice `*`priorytet`*` [[-p] `*`PID`*`…] [[-g] `*`GPID`*`…] [[-u] `*`UID`*`…]`
	Polecenie `renice` ustala nowy priorytet według wartości podanej jako *`priorytet`* dla procesów identyfikowanych przez ich *`PID`*, grup procesów identyfikowanych przez ich *`GPID`* lub użytkowników identyfikowanych przez ich *`UID`*. Zwykły użytkownik może tylko zmniejszyć priorytet procesu.

```
$ nice -5 yes >/dev/null &
[1] 4241
$ jobs -l
[1]+  4241 Running                 nice -5 yes >/dev/null &
$ renice 11 4241
4241: old priority 5, new priority 11
$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S   501  3956  3955  0  76   0 -  1166 wait   pts/2    00:00:00 bash
0 R   501  4241  3956 98  96  11 -   902 -      pts/2    00:00:17 yes
0 R   501  4245  3956  0  78   0 -  1110 -      pts/2    00:00:00 ps
$
```

Jaki priorytet?

1. Zapamiętaj: wyższemu priorytetowi odpowiada mniejsza liczbowa wartość priorytetu.

2. Przy zmianie priorytetów procesów należy zachować daleko idącą ostrożność. Znaczne ich modyfikacje mogą doprowadzić do częściowej lub całkowitej blokady systemu.

Wykonanie następnego polecenia powłoki można wstrzymać o wskazaną liczbę jednostek czasu; służy do tego polecenie `sleep`.

`sleep`	`sleep` *`liczba`*`[smhd]`…
	Polecenie `sleep` (od ang. *sleep* – śpij) wstrzymuje powłokę przed wykonaniem następnego polecenia na czas określony przez sumę wartości parametrów. Litera umieszczona bezpośrednio po liczbie oznacza jednostkę czasu: `s` – sekundy, `m` – minuty, `h` – godziny i `d` – dni.

Przykładowo, polecenie `sleep 1m 10s` oznacza wstrzymanie interpretacji następnego polecenia przez okres 1 minuty i 10 sekund, tj. łącznie przez czas 70 sekund.

9.4.3 Sygnały i usuwanie procesów

Podczas wykonywania, procesy mogą się ze sobą kontaktować na wiele sposobów. Na poziomie powłoki dostępna jest tylko komunikacja poprzez *sygnały*, które są asynchronicznymi komunikatami wysyłanymi przez proces do innego procesu. Z poziomu powłoki sygnały wysyłane są za pomocą poleceń `kill` i `killall`.

`kill`	`kill [-s` *`sygnał`* `| -p] [`*`proces`*`]…` `kill -l`
	Polecenie `kill` (ang. *kill* - zabij) wysyła sygnał określony po `-s` do procesów wskazanych w parametrach. Sygnał może być podany w postaci (skróconego) kodu lub numeru. Brak specyfikacji sygnału oznacza sygnał INT. Procesy mogą być określone przez nazwę lub identyfikator PID. Odbiorca sygnału nie może stwierdzić, który proces jest nadawcą sygnału. Podanie opcji `-p` powoduje tylko wskazanie identyfikatorów procesów, bez nadawania do nich sygnału. W drugiej postaci polecenie listuje dostępne sygnały; ich nazwy są podawane jako: SIGxxx.

W tabeli poniżej zestawiono klasyfikację najważniejszych sygnałów.

Tabela 27. Klasyfikacja sygnałów

Nr sygnału	Kod sygnału	Przyczyna wystąpienia sygnału	Domyślna akcja
1	HUP	zawieszenie procesu; np. odłączenie się terminala, wylogowanie użytkownika, zakończenie procesu macierzystego	A
2	INT	przerwanie procesu, spowodowane przez ^C	A
3	QUIT	wciśnięcie ^\	A
4	ILL	niedozwolona instrukcja	A
5	TRAP	instrukcja śledzenia	
6	ABRT	sygnał spowodowany wywołaniem funkcji *abort*(3)	C
7	BUS	błąd szyny (magistrali)	A
8	FPE	sytuacja wyjątkowa przy realizacji operacji zmiennoprzecinkowej	C
9	KILL	polecenie usunięcia procesu	AEF
10	USR1	sygnał nr 1 użytkownika	A
11	SEGV	naruszenie segmentacji pamięci	C
12	USR2	sygnał nr 2 użytkownika	A
13	PIPE	próba zapisu do potoku bez odbiorcy	A
14	ALRM	sygnał zegara alarmowego	A
15	TERM	programowe polecenie usunięcia procesu wywołaniem funkcji *kill* (1), o ile proces nie przechwytuje tego sygnału	A
i inne			

LEGENDA do tabeli 26: A - zakończenie procesu, C – wykonanie zrzutu pamięci (*dump*), E – brak możliwości przechwycenia sygnału, F – brak możliwości zignorowania sygnału.

Zamieszczone w drugiej kolumnie tabeli napisy są kodami skróconymi; pełny opis kodu ma postać: `"SIG" + kod_z_tabeli`, np. `"SIGHUP"`.

killall	killall [-s *sygnał* | -p] [*proces*]…
	Polecenie killall (od ang. *kill all* – zabij wszystkie) wysyła sygnał określony po –s do procesów wskazanych w parametrach. Sygnał może być podany w postaci skróconego kodu (bez "SIG") lub numeru. Brak specyfikacji sygnału oznacza sygnał INT. Procesy mogą być określone przez nazwę lub identyfikator PID. Odbiorca sygnału nie może stwierdzić, który proces jest nadawcą sygnału. Podanie opcji –p powoduje tylko wskazanie identyfikatorów procesów, bez nadawania do nich sygnału.

Niektóre procesy mogą przechwytywać sygnał INT (1), zamiast go wykonywać. Skutecznym sposobem na przerwanie procesu jest wtedy użycie sygnału KILL (9).

Aby spowodować reakcję polecenia na sygnał podczas wykonywania poleceń powłoki, należy je wykonać pod kontrolą polecenia trap.

trap	trap [*polecenie*] [*nr_sygnału*]…
	Polecenie trap (ang. *trap* - pułapka) jest wbudowanym poleceniem powłoki, które po otrzymaniu jednego z wymienionych sygnałów powoduje wykonanie *polecenia*. Pominięcie *polecenia* oznacza zignorowanie sygnału. Jeżeli nie wskazano żadnego sygnału, to polecenie zostanie wykonane po odebraniu dowolnego sygnału.

Działanie pułapki zilustrujemy następującym przykładem. Najpierw utworzymy plik skryptu o nazwie testrap, którego treścią jest wielokrotne przypisywanie do zmiennej powłoki jej aktualnej wartości. Po uruchomieniu tego skryptu jako zadania tła i zidentyfikowaniu PID utworzonego w ten sposób procesu, wyślemy do niego sygnał HUP.

```
$ cat >testrap <<EOF
> trap 'echo Odebrano sygnal INT.' 1
> for (( i=1; i<1000000; i++ )); do i=$i; done
> EOF
$ sh ./testrap&
[1] 6778
$ kill –HUP 6778
$ Odebrano sygnal INT.

[1]+ Exit 1
$
```

Rys. 148. Przykład użycia pułapki trap

Przedostatni wiersz przykładu informuje nas, że (bieżące) zadanie tła o numerze 1 zostało zakończone.

Niekiedy zachodzi potrzeba takiego uruchomienia zadania, aby było ono niewrażliwe na kierowane do niego sygnały HUP.

nohup	nohup *polecenie* [*argumenty_polecenia*]…
	Polecenie nohup (od ang. *no hangup* – bez zawieszania) uruchamia wskazane polecenie wraz z jego argumentami z priorytetem zwiększonym o 5 i w taki sposób, że nie reaguje ono na skierowane doń sygnały HUP. Standardowe wyjście, zamiast na terminal, zostanie zapisane do pliku nohup.out, a gdy nie jest to możliwe – dopisane do pliku $HOME/nohup.out. W razie, gdy dopisywanie do pliku $HOME/nohup.out nie jest możliwe, polecenie nie jest wykonywane. Utworzony przez nohup proces należy do pierwszego planu, chyba że polecenie zostanie zakończone znakiem spółki.

Użycie nohup jest charakterystyczne dla uruchamiania z terminala długo działających zadań i zamykania sesji przez zakończeniem ich wykonywania.

9.5 Zadania i ćwiczenia

1. Zaktualizuj swoją tabelę procesu cron tak, aby w poniedziałki, środy i piątki o godzinie 10:00 uruchamiane było rekurencyjne listowanie szczegółowego wykazu plików znajdujących się w katalogu głównym i jego podkatalogach do pliku full.list_*czas* w katalogu prywatnym. *czas* wskazuje datę i czas rozpoczęcia listowania. Nazwa pliku nie może zawierać znaków odstępu.
2. Wykorzystując polecenia at i crontab zbuduj skrypt pełniący rolę osobistego asystenta przypominającego za pomocą poczty elektronicznej wyznaczone terminy jednorazowe i powtarzalne.
3. Za pomocą polecenia at zaplanuj minimum 5 zadań. Następnie sprawdź stan kolejki zadań oczekujących na wykonanie.
4. Opracuj skrypt do tworzenia skompresowanych archiwów plikowych. Po uruchomieniu skrypt powinien dla każdego katalogu wymienionego w pliku wskazanym przez 1. parametr wywołania:
   - utworzyć skompresowane archiwum zawierające całą zawartość tego katalogu w formacie tar.gz lub tgz,

	• umieścić to archiwum w katalogu wskazanym przez 2. parametr wywołania, w pliku o nazwie takiej jak nazwa katalogu źrodłowego poszerzona o datę i czas utworzenia pliku archiwalnego; jeżeli plik o takiej nazwie już istnieje, to należy użyć tej nazwy poszerzonej o `.new`. Jeżeli plik z nazwą poszerzoną już istnieje, to należy w nim umieścić nową zawartość. Wypróbuj działanie skryptu umieszczając jego wywołanie w tablicy crona. 5. Opracuj według własnego pomysłu zadanie, którego realizacja wymaga dłuższego czasu (rzędu kilku minut), ale nie jest potrzebny kontakt z terminalem. Uruchom to zadanie jako pierwszoplanowe, po czym je zatrzymaj i przenieś do zadań tła. Sprawdź listę zadań tła. Po pewnej chwili przywróć realizację w pierwszym planie aż do jego ukończenia. Sprawdź, czy zadanie wykonało się poprawnie. 6. Zadanie opracowane powyżej uruchom w trybie drugoplanowym. Po krótkim czasie realizacji usuń z systemu proces uruchomiony dla realizacji tego zadania, posługując się jego identyfikatorem (PID). Sprawdź, czy rzeczywiście proces ten został usunięty. 7. Powtórz zadanie 6., usuwając proces poprzez jego nazwę.

10 Dziel i rządź… czyli Zarządzanie systemem *N*X

10.1 Cykl życia systemu komputerowego

Problematykę zarządzania systemem komputerowym należy rozpatrywać w aspekcie czasu, a dokładniej z punktu widzenia tej fazy cyklu życia systemu komputerowego, którego to zarządzanie dotyczy. W dalszej analizie skoncentrujemy się na zadaniach dotyczących samego systemu operacyjnego, aczkolwiek szereg opisywanych czynności będzie bezpośrednio dotyczyć sprzętu komputerowego.

To, co jest wspólne dla różnych modeli cyklu życia obiektów technicznych (w tym komputerów), to występowanie faz dotyczących:

a. specyfikacji potrzeb,
b. analizy tych potrzeb i opisu istniejących uwarunkowań,
c. projektowania sposobu zaspokojenia potrzeb,
d. realizacji projektu sposobu zaspokojenia potrzeb,
e. wdrożenia uzyskanego rozwiązania do praktyki,
f. eksploatacji rozwiązania,
g. obserwacji sposobu zaspokajania potrzeb przez funkcjonujące rozwiązanie i ewentualnych jego korekt i ulepszeń.
h. wycofania rozwiązania z eksploatacji i rozpoczęcia kolejnego cyklu życia (litera a).

Zauważmy, że niektóre z zadań wykonywanych w cyklu życia systemu komputerowego mają bezpośrednio niewiele wspólnego ze sprzętem lub oprogramowaniem komputerowym jako takim, a dotyczą ogólnych zagadnień funkcjonowania organizacji i zarządzania zespołami ludzkimi. W związku z tym, wymagane umiejętności administratora systemu komputerowego faktycznie wykraczają poza znajomość struktury systemu operacyjnego i poleceń jego powłoki.

10.2 Instalowanie

Tworząc system komputerowy, należy mieć ustalone cele do realizacji i uświadomione warunki, w jakich te cele będą realizowane. Zakładamy więc, że zanim nastąpi zakup sprzętu i oprogramowania, zostaną zrealizowane zadania faz cyklu życia oznaczonych literami a – c powyżej. To znaczy, zostanie ustalone:

a. jakie cele mają być osiągnięte przez system komputerowy oraz jakie potrzeby użytkowe i w jakim zakresie mają zostać spełnione,
b. jakie wymagania i ograniczenia należy spełnić, aby zachować obowiązujące w miejscu zastosowania standardy i zwyczaje branżowe, zapewnić możliwość współpracy z istniejącymi systemami komputerowymi własnymi oraz obsługu-

jącymi współpracowników i partnerów gospodarczych, uzyskać możliwość pewnej, wygodnej i bezpiecznej eksploatacji, np. w warunkach pracy mobilnej,

c. w jaki sposób – jakimi metodami, przy użyciu jakiego sprzętu i oprogramowania oraz zasobów ludzkich - założone cele mają być osiągnięte przy rozpoznanych wymaganiach i ograniczeniach, jaki przewiduje się harmonogram prac i ich budżet, a także uzasadnienie tego, dlaczego zostały wybrane takie, a nie inne rozwiązania szczegółowe.

Istotnym elementem tego procesu jest wybór wersji (lub dystrybucji) systemu operacyjnego. Jak w każdej dziedzinie, tutaj również panują różne stereotypy i mody. Z pewnością należy wybrać wersję taką, która przy istniejących ograniczeniach realizuje wszystkie niezbędne funkcje użytkowe i daje możliwość późniejszego rozwoju, a przede wszystkim może być sprawnie i skutecznie zarządzana przez swojego administratora. Nie od rzeczy jest, aby była to wersja popularna, gdyż jako taka będzie dobrze udokumentowana i łatwiej będzie pozyskiwać wiedzę niezbędną we wszystkich fazach cyklu życia. Problem wyboru takiej czy innej wersji należy więc zawsze sprowadzać do aspektów racjonalnych.

Instalowanie systemu komputerowego rozpoczyna fazę realizacji (litera d). Wskazane w ramach projektu składniki sprzętu i oprogramowania winny być pozyskane i dostarczone do miejsca przeznaczenia. Po obowiązkowej kontroli ilościowej, sprzęt komputerowy winien być zmontowany i przetestowany w sposób przewidziany przez producenta, po czym można przystąpić do instalowania oprogramowania. Nośnikiem, z którego wykonuje się instalowanie systemu operacyjnego zazwyczaj jest samouruchamiająca się (ang. *bootable*) płyta CD-ROM lub DVD.

Brak możliwości startu komputera z czytnika CD-ROM lub DVD

W razie niemożności takiego skonfigurowania BIOS-u komputera, aby program instalujący uruchomił się z płyty CD-ROM lub DVD samoczynnie, należy sprawdzić możliwość przygotowania dyskietek startowych i uruchomienia procesu instalacji z tego właśnie nośnika.

Systemy *N*X często zawierają w odrębnym katalogu nośnika dystrybucyjnego obrazy dyskietek instalacyjnych. W systemie MS Windows należy te obrazy przegrać na nieuszkodzone dyskietki za pomocą programu o nazwie rawrite.exe lub podobnej. Ten sam efekt można uzyskać w środowisku *N*X przy pomocy polecenia dd. Szczegółowego opisu postępowania należy szukać na nośniku dystrybucyjnym.

Współczesne programy instalujące oprogramowanie działają w trybie interaktywnym. Użytkownik odpowiada na kolejno zadawane pytania, a odpo-

wiednie czynności są uruchamiane automatycznie. Ze względu na płynność procesu instalowania systemu operacyjnego, jego administrator powinien uprzednio przygotować niezbędne dane i parametry. Część z nich stanowi elementy projektu, pozostałe zaś są do określenia przez samego administratora komputera, ewentualnie w porozumieniu z administratorem sieci komputerowej (dalej oznaczanym w skrócie przez *ASK*), w której dany komputer będzie działać. W tabeli poniżej podano wykaz podstawowych informacji, które powinny być zgromadzone przed uruchomieniem instalacji systemu operacyjnego.

Tabela 28. Wykaz informacji potrzebnych do instalowania systemu operacyjnego

Nazwa informacji	Źródło informacji
Oznaczenia producentów i typów urządzeń (w razie, gdyby program instalacyjny nie umiał samodzielnie ich rozpoznać)	Specyfikacja dostawy, bezpośrednie sprawdzenie
Nazwa komputera	Projekt uzgodniony z ASK
Nazwa domeny DNS	ASK
Adres IP komputera	ASK
Adres IP bramki podsieci	ASK
Maska podsieci	ASK
Adresy serwerów DNS	ASK
Wybór podstawowego trybu pracy (tekstowy, graficzny)	Projekt, ocena możliwości sprzętu oraz preferencje administratora
Język interfejsu konsoli	Projekt lub preferencje administratora
Kodowanie znaków narodowych	Projekt lub dominujące zwyczaje
Sposób prezentacji daty i czasu	Projekt lub dominujące zwyczaje
Wielkość partycji do utworzenia na dyskach twardych, ich format i sposób ich wykorzystania	Projekt, dokumentacja systemu i wiedza administratora
Wykaz usług systemowych do uruchomienia	Projekt i wiedza administratora, uzgodnienia z ASK
Wykaz standardowych pakietów oprogramowania do zainstalowania	Projekt oraz preferencje administratora
Nazwy grup i kont użytkowników	Projekt lub własne ustalenia administratora
Sposób tworzenia haseł i zasady ich starzenia się	Polityka bezpieczeństwa

Niektóre z tych informacji będą również wykorzystywane w czasie konfigurowania systemu operacyjnego.

10.3 Konfigurowanie usług

Konfigurowanie głównych elementów samego systemu operacyjnego zachodzi w trakcie jego instalowania. Niekiedy jednak zachodzi potrzeba dokonania zmian w parametrach ustalonych pierwotnie. W tabeli poniżej przedstawiono lokalizację ważniejszych plików konfiguracyjnych na przykładzie systemów Linux z rodziny RedHat.

Tabela 29. Ważniejsze dane konfiguracyjne systemów Linux z rodziny RedHat

Opis funkcji	Podstawowe pliki konfiguracyjne
Powiązanie nazw i adresów IP komputerów	`/etc/hosts`, w powiązaniu z plikiem `/etc/nsswitch.conf`
Dane własnej domeny i serwerów DNS	`/etc/resolv.conf`
Dane interfejsów sieciowych	`/etc/sysconfig/network-scripts/ifcfg-*`
Domyślne parametry nowego konta	`/etc/default/*`, `/etc/skel/*`
Dane użytkowników (kont)	`/etc/passwd`
Hasła użytkowników	`/etc/shadow`
Dane grup użytkowników	`/etc/group`
Dane montowanych urządzeń	`/etc/fstab`
Komunikat powitalny po otwarciu sesji	`/etc/motd`
Numeracja protokołów sieciowych	`/etc/protocols`
Numeracja portów dla usług sieciowych	`/etc/services`
Wykaz dozwolonych powłok	`/etc/shells`
Sterowanie kroniką systemową	`/etc/syslog.conf`

W tabeli dostępnej w [20] znajdzie Czytelnik zestawienie lokalizacji ważniejszych plików konfiguracyjnych dla systemów: AIX, FreeBSD, HP-UX, Linux RedHat, Solaris i Tru64, a także porównanie ważniejszych poleceń związanych z zarządzaniem systemem i ograniczeń dotyczących systemu plików w poszczególnych systemach operacyjnych. Ze względów formalnych tabela ta nie mogła być przytoczona w niniejszym podręczniku.

Po zakończeniu ustalania konfiguracji samego komputera należy skonfigurować usługi, stanowiące podstawę realizacji innych funkcji, w tym współdziałania komputera w sieci komputerowej. Należy zwrócić uwagę, że dla niektórych usług istnieją specjalne aplikacje, które same tworzą odpowiednie pliki konfiguracyjne. Ręczna modyfikacja takich plików nie jest zalecana, aczkolwiek nie wszystkie opcje konfiguracyjne są obsługiwane automatycznie. Dotyczy to np. konfigurowania serwera Apache i serwera DNS.

Tabela 30. Konfigurowanie ważniejszych usług w systemach Linux z rodziny RedHat

Nazwa usługi	Opis usługi	Sposób konfigurowania i podstawowe pliki konfiguracyjne
cron	Cykliczne uruchamianie zadań	`crontab –e`
named	Serwer nazw domenowych DNS (opcja)	`/etc/named.conf, /var/named/*`
ftp	Serwer plików vsftpd (opcja)	`/etc/vsftpd/*`
http	Serwer WWW Apache (opcja)	`/etc/httpd/*`
iptables	Zapora sieciowa (opcja)	`/sbin/iptables`
ntp	Serwer czasu (opcja)	`/etc/ntp/*, /etc/ntp.conf`
mail	Serwer poczty elektronicznej (opcja)	`/etc/mail/*`
mysql	Serwer bazodanowy MySQL (opcja)	`/etc/my.cnf`
php	Interpreter języka PHP	`/etc/php.ini`
samba	Serwer plików i drukarek zgodny z MS Windows	`/etc/samba/*`
ssh	Bezpieczna powłoka zdalna	`/etc/ssh/*`
syslog	Systemowy proces kroniki	`/etc/syslog.conf`
xinetd	"superdemon" internetowy	`/etc/xinetd.conf,/etc/xinetd.d/*`

10.4 Tworzenie grup i kont użytkowników

Realizacja użytkowych funkcji systemu komputerowego zawsze dotyczy określonych użytkowników. Są nimi ci, którzy swoimi działaniami powodują i nadzorują realizację określonych procesów przetwarzania danych i ci, którzy z wyników tego przetwarzania korzystają. Struktura użytkowników systemu powinna odpowiadać wewnętrznej organizacji podmiotu, na rzecz którego działa system, przy czym może to być zarówno formalna struktura zarządzania, struktura terytorialna, jak i struktura wynikająca z podobieństwa sprawowanych ról, niezależnie od podporządkowania organizacyjnego.

Podział użytkowników systemu na grupy, sposób tworzenia i parametryzacji ich kont oraz przydzielania uprawnień dostępu do plików i usług systemu powinien odpowiadać rolom poszczególnych osób w danej organizacji. Dlatego, przed przystąpieniem do rutynowego zakładania nowych grup i kont użytkowników, powinna być przeprowadzona analiza ról i wypracowana polityka zarządzania użytkownikami. Paradoksalnie, z reguły im wyższa jest pozycja danej roli w strukturze organizacyjnej, tym mniejsze powinna ona posiadać uprawnienia do tworzenia, modyfikacji i usuwania danych.

Hasła wykorzystywane do uwierzytelniania użytkowników powinny być wystarczająco "mocne", tj. trudne do odgadnięcia, aby uniemożliwiać osobom postronnym niekontrolowane wchodzenie w rolę legalnych użytkowników. Jeśli jest to możliwe, główny administrator systemu (`root`) powinien zastosować oprogramowanie badające moc haseł, w celu uniemożliwienia użytkownikowi zmiany hasła na takie, które jest łatwe do zapamiętania, gdyż z reguły będzie ono równocześnie łatwe do odgadnięcia lub złamania. Hasła nie mogą być jednak zbyt skomplikowane. Zabezpieczenie hasłem można dodatkowo wzmocnić, stosując tzw. *starzenie się haseł* (ang. *password aging*). Dokładniejsza dyskusja problematyki zarządzania hasłami znajduje się w punkcie 12.4.4.

Podobnej rozwagi wymaga nadawanie użytkownikom uprawnień dostępu do plików. Należy przyznać najmniejszy możliwy zakres uprawnień, przy którym rola użytkownika może być skutecznie wypełniana. Tu również bardzo przydatnym okazuje się być mechanizm grup użytkowników, dzięki któremu czynności przydzielania uprawnień dla wielu kont o tych samych rolach można sprowadzić do działań prostych i powtarzalnych.

10.5 Instalacja oprogramowania narzędziowego i aplikacyjnego

Następną czynnością jest zbudowanie najwyższej warstwy systemu komputerowego (rys. 2.), tj. wyposażenie go w oprogramowanie narzędziowe i aplikacyjne. Często wymaga to uprzedniego zapoznania się z procedurą postępowania, zgromadzenia wymaganych danych, uzgodnienia konfiguracji oraz zainstalowania pakietów, modułów oprogramowania i bibliotek funkcji tworzących wymagane środowisko działania. Należy przy tym zadbać o zgodność wersji i właściwą kolejność instalowania poszczególnych składników oprogramowania.

10.6 Szkolenie

Szkolenie użytkowników systemu komputerowego jest często elementem niedocenianym lub traktowanym jako czynność jednorazowa. Poprawnie zorganizowane wdrożenie aplikacji powinno obejmować również aspekty edukacyjne, włączając:

- szkolenie administratorów systemu komputerowego,
- szkolenie administratorów zasobów i usług sieciowych oraz administratorów oprogramowania narzędziowego (np. serwerów baz danych), jeśli takie role zostaną wyodrębnione,
- szkolenie użytkowników aplikacji

i zostać przeprowadzone przed przystąpieniem do eksploatacji systemu komputerowego.

Forma i czas szkolenia winny być dopasowane do potrzeb, w tym stopnia merytorycznego i praktycznego przygotowania słuchaczy. Szkolenie nie powinno jednak ograniczać się tylko do krótkiego instruktażu, a osoby pełniące role szczególnie odpowiedzialne za prawidłowość eksploatacji powinny stale doskonalić swoje kwalifikacje.

Obowiązek stałego doskonalenia

W swojej książce [29], pośród licznych i bardzo trafnych maksym, Peter Norton stwierdził, że UNIX-a nie można przestać się uczyć. Ta sentencja odnosi się w sposób oczywisty nie tylko do całej klasy systemów *N*X, ale sięga znacznie dalej.

10.7 Eksploatacja

10.7.1 Bieżący nadzór nad systemem

Nadzór nad systemem komputerowym obejmuje czynności zapewniające ciągłość działania i świadczenia przezeń usług na rzecz użytkowników. Oprócz wymienionych w poprzednich punktach, są to:

- przeglądanie kronik,
- badanie stanu kolejek,
- badanie integralności plików,
- badanie wykorzystania pojemności dysków,
- badanie funkcjonowania usług sieciowych i usług baz danych,
- wykonywanie kopii bezpieczeństwa istotnych danych.

Kroniki systemowe dostarczają wiedzy na temat sposobu i intensywności wykorzystania danego systemu: przebiegu jego uruchamiania, wykonanych zadań i towarzyszących im błędów, prób logowania, a zwłaszcza korzystania z konta superużytkownika oraz usług sieciowych, takich jak poczta elektroniczna, WWW i przesyłanie plików oraz aktywności serwerów baz danych. Systematyczna analiza kronik pozwala zawczasu wykryć źródła błędnej pracy urządzeń lub potencjalnie groźnych zmian w systemie uprawnień. Podobnie, analiza stanu kolejek pozwala wskazać niedziałające lub źle skonfigurowane demony usług (np. poczty elektronicznej).

Z kolei badanie integralności systemu plików dostarcza wiedzy na temat zmian lub uszkodzeń, spowodowanych przyczynami technicznymi, bądź celową działalnością osób trzecich. Wreszcie, bieżąca kontrola tego, w jakim stopniu został wykorzystany obszar systemu plików, pozwala nie dopuścić do blokady funkcji systemu.

Kopie bezpieczeństwa, zwane też *kopiami awaryjnymi*, wykonywane są dla istotnych fragmentów systemu plików i służą zabezpieczeniu danych na wypadek zdarzeń losowych lub działań osób trzecich (fizyczne uszkodzenie dysków, zmiana ich zawartości przez modyfikację, kasowanie lub dopisanie danych itp.).

Należy przy tym pamiętać, że dane – w różnej postaci - są najważniejszym zasobem systemu komputerowego. O ile uszkodzony sprzęt liczący można wymienić, a oprogramowanie zainstalować ponownie, o tyle odtwarzanie danych wymagać może sięgania do dokumentów źródłowych (które nie zawsze są dostępne) i żmudnych czynności ich ponownej rejestracji. Jest to o wiele bardziej kosztowne niż naprawa lub zakup nowego sprzętu, a niekiedy nawet niewykonalne, co stawia pod znakiem zapytania możliwości normalnego funkcjonowania osób i instytucji korzystających z tych danych. W skrajnym przypadku utrata danych może spowodować upadłość podmiotu gospodarczego.

10.7.2 Aktualizacja oprogramowania

Administrator systemu operacyjnego winien śledzić pojawianie się aktualizacji zainstalowanego systemu operacyjnego oraz użytkowanego pod jego kontrolą oprogramowania.

Jednym z zasadniczych składników systemu *N*X jest jego jądro. Winno być ono odpowiednie dla danej architektury sprzętu komputerowego. Jeżeli istnieje możliwość wyboru wersji jądra, to należy się upewnić, czy jądro preferowane posiada wszystkie funkcje wymagane przez usługi i aplikacje działające w systemie. Uwaga ta dotyczy zwłaszcza jąder rozpowszechnianych na zasadach OpenSource; jądra te bywają modyfikowane dla realizacji szczególnych potrzeb i nie każda pojawiająca się odmiana nadaje się do danego celu.

Numeracja wersji jądra systemu Linux

Nazwa jądra systemu Linux zawiera 3-elementowy numer wersji, w postaci `x.y.z`, ewentualnie rozszerzony o dalsze określenia. Przyjęta konwencja określa, że parzyste numery `y` wskazują tzw. *jądro stabilne*, tj. zatwierdzone do oficjalnej eksploatacji. Nieparzyste wartości `y` oznaczają jądra eksperymentalne, których cykl wytwarzania i akceptacji nie został jeszcze zakończony.
Tekst `smp` w nazwie jądra oznacza, iż może ono obsługiwać również konfiguracje wieloprocesorowe, w tym procesory wielowątkowe (ang. *Hyper Threading*).

W odniesieniu do wszystkich składników oprogramowania systemu komputerowego obowiązuje zasada, aby przy aktualizacji wersji nie postępować automatycznie i bezkrytycznie. Wskazane jest uprzednie sprawdzenie, w warunkach do-

świadczalnych, a nie na eksploatowanym systemie, poprawności współpracy nowszych wersji jądra systemowego, podstawowych pakietów aplikacji i serwerów usług, a dopiero po uzyskaniu pozytywnych wyników przeprowadzenie aktualizacji eksploatowanego systemu.

Aczkolwiek te uwagi wydają się oczywiste, warto o nich wspomnieć, gdyż aktualizacja oprogramowania rzadko jest w pełni odwracalna.

10.7.3 Zarządzanie wykorzystaniem pamięci dyskowej

Wprowadzenie

System komputerowy *N*X, zwłaszcza intensywnie użytkowany, powinien być zabezpieczony przed skutkami ewentualnego niekontrolowanego wzrostu zajętości miejsca w systemie plików, spowodowanego działaniami użytkownika lub wykonywanych w systemie zadań. Jeśli zjawisko takie trwa długo, prowadzi do wykorzystania całej dostępnej pamięci, czego efektem jest niemożność uruchomienia żadnego nowego procesu, w tym otwarcia nowej sesji lub uruchomienia aplikacji, z powodu braku miejsca dla plików roboczych i kolejek systemowych. Pojawiający się na konsoli administratora komunikat `"File system full"` lub podobny jest dowodem na to, iż zaniedbał on okresowe kontrole. Dla potwierdzenia rozpoznania i ustalenia, która partycja dyskowa uległa przepełnieniu, można użyć polecenia `df`.

```
$ df –k
System plików     bl.  1K B      użyte dostępne %uż. zamont. na
/dev/sda1             3804796  3804700        0 100% /
/dev/shm               161880        0    161880   0% /dev/shm
$
```

Rys. 149. Obraz przepełnienia systemu plików `/dev/sda1`

Na rysunku powyżej szczególnie ważna jest kolumna zatytułowana `%uż.` zawierająca odsetek wykorzystanej pojemności dla każdego urządzenia.

System operacyjny sam nie potrafi wyjść ze stanu wyczerpania pojemności pamięci – stan ten musi zmienić użytkownik `root`, kiedy tylko zidentyfikuje awarię. Aby udostępnić wymagane miejsce w systemie plików, można zastosować jedną lub więcej spośród metod wymienionych poniżej:

a) usunąć zbędne pliki z prywatnych katalogów użytkowników,
b) usunąć mniej potrzebne aplikacje,
c) usunąć lub opróżnić zbędne pliki z katalogów systemowych,
d) przenieść pliki lub całe katalogi do innych partycji w systemie plików.

Metody a) i b) w zasadzie wymagają uzgodnień z użytkownikami. Metoda c) wymaga dużej ostrożności i rozwagi; usunięcie ważnego pliku systemowego może w skrajnym przypadku spowodować nawet całkowitą utratę możliwości

działania systemu. Opróżnienie plików uzyskujemy przez usunięcie zawartości, lecz bez usuwania samych plików (patrz zadanie 1.). Metoda d) daje najlepsze efekty, ale daje się zastosować tylko wtedy, gdy w systemie istnieją i posiadają wystarczającą ilość wolnego miejsca inne partycje dyskowe. Po przeniesieniu plików i/lub katalogów, należy w ich poprzednich lokalizacjach umieścić dowiązania wskazujące nowe położenie (patrz zadanie 2.).

Po zapewnieniu wolnego miejsca w pamięci plików system operacyjny *N*X winien kontynuować pracę bez żadnych dodatkowych działań ze strony superużytkownika.

Bieżąca kontrola wykorzystania pamięci plików

Systemy *N*X posiadają mechanizm kontroli wykorzystania pojemności pamięci w systemie plików, którego podstawą są limity (ang. *quota*) przydzielane użytkownikom. Limit przyznany danemu użytkownikowi zawiera dwa ograniczenia wykorzystanej pamięci wyrażone liczbą bloków i dwa ograniczenia dotyczące liczby plików. Dla każdej z par ograniczeń przekroczenie pierwszego - *miękkiego limitu* (ang. *soft limit*) włącza generowanie dla tego użytkownika komunikatów informacyjnych. Przekroczenie zaś drugiego - *twardego limitu*: (ang. *hard limit*) blokuje możliwość zwiększania obszaru wykorzystanego, co praktycznie oznacza brak możliwości uruchamiania zadań. Obydwa limity mogą być ustalane przez superużytkownika dla całych grup łącznie i poszczególnych użytkowników z osobna. Prócz tego istnieje tzw. *okres pobłażliwości*: (ang. *grace period*), to znaczy czas od przekroczenia progu, zanim zacznie działać pierwszy limit (w systemie, w którym jest włączona kontrola limitów). Okresy pobłażliwości dla bloków i plików - jednakowe dla wszystkich - również konfiguruje użytkownik `root`.

Aby uzyskać działający mechanizm kontroli zajętości pamięci dyskowej, należy wykorzystać jądro systemu, które posiada obsługę limitów. Warunek ten jest spełniony dla nowszych wydań jąder systemu Linux. Następnie należy wykonać czynności opisane poniżej (na podstawie [7]).

a. W razie potrzeby trzeba zmodyfikować systemowy skrypt startowy, dodając w nim tekst pokazany poniżej, tuż za poleceniami montowania systemów plików wymienionych w pliku `/etc/fstab`.

```
# badanie wykorzystania limitow i wlaczanie kontroli
if [ -x /sbin/quotacheck ]
    then
        echo "Badanie limitów. To moze chwile potrwac."
        /sbin/quotacheck -avug
        echo "Badanie limitów wykonane."
    fi
if [ -x /sbin/quotaon ]
    then
        echo "Wlaczanie kontroli limitow."
        /sbin/quotaon -avug
    fi
```

Rys. 150. Uzupełnienie systemowego skryptu startowego dla obsługi limitów

b. Zmodyfikować plik `/etc/fstab` - systemy plików, dla których ma być wykonywana kontrola limitów, należy oznaczyć słowem `usrquota`, `grpquota` lub obydwoma, jak pokazano poniżej (pierwszy wiersz poniżej komentarza).

```
$ more /etc/fstab
# This file is edited by fstab-sync - see 'man fstab-sync' for details
LABEL=/                 /                       ext3    defaults,usrquota,grpquota  1 1
/dev/devpts             /dev/pts                devpts  gid=5,mode=620  0 0
/dev/shm                /dev/shm                tmpfs   defaults        0 0
/dev/proc               /proc                   proc    defaults        0 0
/dev/sys                /sys                    sysfs   defaults        0 0
LABEL=SWAP-sda2         swap                    swap    defaults        0 0
/dev/fd0                /media/floppy           auto    pamconsole,exec,noauto,managed 0 0
/dev/hdc                /media/cdrecorder       auto    pamconsole,exec,noauto,managed 0 0
$
```

Rys. 151. Zawartość zmodyfikowanego pliku `/etc/fstab`

c. Ponownie uruchomić system, w wyniku czego powinny zostać utworzone pliki baz danych na potrzeby sterowania limitami.

d. Zaplanować cykliczne uruchamianie polecenia `quotacheck` służącego do sprawdzania stanu limitów, poprzez wywołanie polecenia edycji `crontab –e` i dodanie np. na końcu wyświetlonego pliku wiersza postaci:

```
10 3 * * * /sbin/quotacheck -aug
```

Rys. 152. Programowanie cyklicznego badania limitów

Po zapisaniu pliku, polecenie `quotacheck` pokazane na Rys. 150, będzie uruchamiane codziennie o godzinie 03:10.

e. Przydzielić limity dla poszczególnych użytkowników poleceniem `edquota -u` *`nazwa_użytkownika`*, co spowoduje uruchomienie edytora tekstowego (domyślnie `vi`). Poniżej zilustrowano realizację polecenia `edquota -u skrypt`.

```
Limity dyskowe user skrypt (uid 501):
  System plików              bloki       miękki     twardy   i-węzły    miękki   twardy
  /dev/sda1                    112          700          800         14         0        0
~
~
```

Rys. 153. Edycja limitów użytkownika `skrypt`

Po dokonaniu zmian plik należy zapisać.

f. W celu aktualizacji okresów pobłażliwości należy użyć polecenia `edquota -t` i, podobnie jak w przypadku limitów użytkownika, po dokonaniu zmian plik zapisać.

g. Aby wyświetlić na terminalu raport z wykorzystania limitów, używamy polecenia `repquota -a`.

h. Za pomocą cyklicznie wywoływanego polecenia `warnquota` można zapewnić informowanie użytkowników pocztą elektroniczną o przekroczeniu limitu liczby bloków lub limitu liczby plików. Budowa i treść komunikatu pocztowego określone są przez pliki `/etc/warnquota.conf` i `/etc/quotatab`. Cykliczne wywoływanie polecenia `warnquota` w naturalny dla środowiska *N*X sposób realizuje demon `crond` (zobacz podpunkt d. powyżej).

```
[root@vhomer /]# repquota -a
*** Raport dla limitów user na urządzeniu /dev/sda1
Okres pobłażliwości dla bloków: 7 dni; Okres pobłażliwości dla i-węzłów: 7 dni
                        Limity bloków               Limity plików
Użytkownik      używ.   mięk.   twar. pobł.    używ. mięk. twar. pobł.
---------------------------------------------------------------------
root      -- 3265204       0       0         132401     0     0
daemon    --      20       0       0              3     0     0
lp        --       8       0       0              1     0     0
games     --     336       0       0             84     0     0
nobody    --      16       0       0              2     0     0
vcsa      --      12       0       0              1     0     0
rpm       --   38456       0       0            120     0     0
named     --      80       0       0             10     0     0
netdump   --      16       0       0              2     0     0
smmsp     --      24       0       0              3     0     0
rpcuser   --      48       0       0              6     0     0
apache    --      40       0       0              5     0     0
squid     --      16       0       0              2     0     0
webalizer --      32       0       0              4     0     0
xfs       --       8       0       0              2     0     0
ntp       --      16       0       0              2     0     0
pvm       --       8       0       0              1     0     0
mysql     --      24       0       0              4     0     0
```

Rys. 154. Raport z badania limitów użytkowników

10.8 Rekonfiguracja systemu

Rekonfiguracja systemu komputerowego może polegać na zmianach w zestawie sprzętu komputerowego, zmianach w zainstalowanym oprogramowaniu, zmianach parametrów konfiguracyjnych, kont i haseł użytkowników itp., lub wielu z tych działań równocześnie. Przed rozpoczęciem czynności rekonfiguracyjnych superużytkownik powinien oszacować stopień ryzyka związanego z rekonfiguracją i odpowiednio zabezpieczyć zasoby systemu.

Jak wspomniano wyżej, najcenniejszym zasobem systemu komputerowego są dane. Dlatego działania związane z rekonfiguracją systemu powinny być traktowane jako jedno ze źródeł zagrożenia jego bezpieczeństwa (zobacz rozdział 12.).

10.9 Likwidacja systemu

Likwidacja systemu wynika z zaprzestania jego eksploatacji i wymaga utworzenia archiwalnych kopii istotnych danych i obsługujących je aplikacji oraz złożenia ich w bezpiecznym miejscu, zabezpieczenia znajdujących się w systemie danych i licencjonowanego oprogramowania przed nieupoważnionym użyciem poprzez ich skasowanie z nośników. Wycofany sprzęt komputerowy może zostać zagospodarowany w inny sposób lub skasowany.

Likwidować dopiero po dokonaniu archiwizacji

Błędem jest automatyczne zakładanie, że po likwidacji systemu komputerowego żadne z jego zasobów nigdy nie będą więcej potrzebne. Takie podejście wymaga przemyślenia i uzasadnienia. Odzyskanie skasowanych danych znajdujących się na nośnikach pamięci niekiedy się udaje, ale jest bardzo kosztowne i czasochłonne.

Należy zwrócić uwagę, iż obowiązujący system prawny może nakładać na administratora systemu dodatkowe obowiązki, np. fizycznego niszczenia nośników pamięci i dokumentacji przechowywanej w postaci tradycyjnej.

10.10 Zadania i ćwiczenia

1. Napisz skrypt do usuwania zawartości pliku wskazanego jako pierwszy parametr wywołania, lecz bez usuwania samego pliku. Po wywołaniu skrypt winien wyprowadzić napis informujący o swojej funkcji i dwukrotnie zażądać potwierdzenia zamiaru usunięcia pliku, z podaniem jego nazwy. Dopiero po drugiej pozytywnej odpowiedzi zawartość pliku powinna zostać usunięta. Nazwa pliku do opróżnienia podawana jako parametr powinna być nazwą bezwzględną; nazwa w postaci względnej ma być odrzucona jako nieprawidłowy parametr wywołania.
2. Napisz skrypt do fizycznego przenoszenia poddrzewa systemu plików do innej partycji, z utworzeniem symbolicznego dowiązania wskazującego nowe położenie poddrzewa w jego poprzedniej lokalizacji. Zaprojektuj sposób wywołania skryptu i niezbędne sprawdzenia, w tym tego, czy wolny obszar w partycji docelowej jest wystarczający dla pomieszczenia przenoszonego poddrzewa.
3. Rozbuduj skrypt z zadania poprzedniego tak, aby jako docelowe miejsce lokowania poddrzewa plików mógł również wykorzystać istniejącą, sformatowaną, uprzednio

	zamontowaną lub niezamontowaną partycję dyskową. 4. Opracuj uniwersalny skrypt, który w losowych odstępach czasu będzie tworzył w przypadkowych katalogach pliki o względnie dużej i losowej objętości, aż do uzyskania stanu zapełnienia systemu plików w stopniu (procencie), który jest parametrem skryptu. Nazwy tych plików powinny umożliwiać ich łatwe i jednoznaczne wskazanie oraz usunięcie po zakończeniu eksperymentów. 5. Opracuj plan działania na wypadek zaistnienia sytuacji przepełnienia systemu plików w systemie *N*X, wykorzystując różne metody postępowania, odpowiednie do zaistniałej sytuacji. Zrealizuj odpowiedni skrypt, korzystając z opracowanych wcześniej rozwiązań zadań. Skrypt powinien prowadzić kronikę zdarzeń, obejmującą co najmniej datę i czas wystąpienia zdarzenia, rozpoznanie sytuacji, podjęte czynności naprawcze i uzyskany efekt. 6. Uruchom skrypt z zadania 4. i sprawdź działanie skryptu z zadania 5. Przeanalizuj poprawność podejmowanych decyzji i ich skutek. 7. Rozbuduj tabelę procesu `cron` użytkownika `root` tak, aby w każdy czwartek o godzinie 8:00 superużytkownikowi wysyłany był pełny raport z badania limitów użytkowników, a każdemu użytkownikowi o identyfikatorze UID większym od 99 – odpowiedni wiersz tego raportu.

11 Po co nam to było?... czyli Zastosowania

11.1 Wprowadzenie

O ile systemy *N*X jako stanowiska pracy (ang. *workstation*), w porównaniu z systemami MS Windows stanowią zdecydowaną mniejszość, o tyle ich znaczenie w roli serwerów jest wyraźne i niekwestionowane. Walory systemów Linux zostały dostrzeżone w szczególności przez czołowych producentów oprogramowania narzędziowego i aplikacji, czego efektem jest istnienie dla tej platformy operacyjnej np. komercyjnych wersji rozbudowanych systemów zarządzania bazami danych, jak Oracle, IBM DB/2, Sybase i inne. Systemy Linux doskonale sprawdzają się w zastosowaniach sieciowych i jako serwery usług (np. WWW, FTP, poczta elektroniczna, DNS i inne). Dostępność wielu programów na zasadach licencji GPL (wraz z kodem źródłowym), sprzyja powstawaniu produktów pochodnych o funkcjach rozszerzonych lub ograniczonych w stosunku do pierwowzoru, na ogół przystosowanych do zainstalowania w dowolnym typowym środowisku *N*X. Sam system operacyjny Linux występuje w blisko 200 dystrybucjach, od pełnych, o uniwersalnych funkcjach, do jednodyskietkowych, uruchamianych z CD-ROM-u lub pamięci *USB flash* i wyspecjalizowanych, np. pełniących rolę terminali, ruterów lub zapór sieciowych.

Jeśli chodzi o dystrybucje uniwersalne, to znaczenie praktyczne posiada nie więcej niż kilkanaście wersji, w tym (w kolejności alfabetycznej):

- AIX (IBM Corp.),
- Aurox,
- Debian,
- Gentoo,
- HP-UX (Hewlett Packard),
- Knoppix,
- Mandrake,
- RedHat (Red Hat Inc.) oraz dystrybucje pochodne: Fedora Core i CentOS,
- SCO Unix (Santa Cruz Operation),
- Solaris (Sun Microsystems),
- SuSE (Novell Inc.).

W nawiasach podano nazwy producentów systemów operacyjnych.

Warto zaznaczyć, że w ostatnim czasie producenci tacy jak Apple i Novell, którzy przez długie lata rozwijali własne oryginalne rozwiązania systemowe, w nowszych wersjach swoich produktów zastosowali jądra linuksowe (Mac OS X i Novell Linux Desktop). Z kolei firma Sun Microsystems sponso-

ruje projekt pod nazwą OpenSolaris[37], większość kodu którego jest rozpowszechniana według specjalnej licencji CDDL[38].

Niniejszy rozdział zawiera zarys przykładowych zastosowań systemów *N*X. Celem autora było zaprezentowanie zastosowań typowych oraz tych, które - jego zdaniem - warto upowszechniać. Nie będą omawiane zagadnienia dotyczące instalowania i konfigurowania oprogramowania – przekraczałoby to znacznie ramy tego podręcznika. Pominięto też oprogramowanie ogólnie znane, jak serwer WWW Apache, serwery baz danych MySQL i PostgreSQL, pakiet oprogramowania biurowego OpenOffice.org itp. Czytelnik znajdzie tu krótki opis przeznaczenia oraz uwagi dotyczące sposobu wykorzystania poszczególnych produktów, opracowane w postaci tabelarycznej jak niżej:

Tabela 31. Schemat opisu wybranych produktów programowych

Nazwa produktu	
Producent (autor)	
Platforma systemowa	
Sposób licencjonowania	
Przeznaczenie	
Ogólna charakterystyka	
Strona WWW	
Uwagi	
Inne źródła	

W razie potrzeby należy sięgnąć po szczegółowe instrukcje dla danego produktu, dostarczone razem z oprogramowaniem, lub do literatury opublikowanej przez profesjonalne wydawnictwa.

11.2 Zastosowania systemowe

Przegląd rozpoczniemy od darmowej dystrybucji systemu Linux, będącej pełnym (!) odpowiednikiem komercyjnej wersji Red Hat Enterprise Linux oraz dystrybucji ograniczonych, nadających się do umieszczenia na nośnikach wymiennych i niewymagających instalowania na dysku.

[37] Na bazie systemu OpenSolaris powstał projekt pod nazwą Shillix (http://schillix.berlios.de/).
[38] Ang. Common Development and Distribution License (http://www.sun.com/cddl/).

Nazwa produktu	CentOS
Producent (autor)	
Platforma systemowa	Architektura x86
Sposób licencjonowania	GPL
Przeznaczenie	Serwerowa wersja systemu Linux do zarządzania przedsiębiorstwem
Ogólna charakterystyka	RedHat Enterprise Linux (EL) został zbudowany do eksploatacji aplikacji istotnych dla zarządzania przedsiębiorstwem. Firma RedHat dobrała składniki tego systemu w celu uzyskania stabilności szczególnie ważnej w odpowiedzialnych zastosowaniach. CentOS stanowi odpowiednik wersji komercyjnej, nie posiada jednak składników licencjonowanych. Rozwój systemu CentOS jest wspierany przez firmę RedHat.
Strona WWW	http://www.centos.org
Uwagi	Numer wersji systemu CentOS odpowiada wersji RedHat EL, a jako dokumentacja systemu CentOS jest używana dokumentacja RedHat EL
Inne źródła	[5]

Nazwa produktu	Damn Small Linux
Producent (autor)	
Platforma systemowa	Architektura x86
Sposób licencjonowania	GPL
Przeznaczenie	Uniwersalna minidystrybucja systemu Linux
Ogólna charakterystyka	Damn Small Linux (DSL) zajmuje ok. 50 MB i może być uruchomiony nawet z CD-ROM-u w formacie karty kredytowej jako dystrybucja LiveCD, z pamięci typu USB *pen drive* lub jako aplikacja systemu MS Windows. Może być też zainstalowany i uruchamiany z dysku twardego. Wymaga co najmniej procesora i486DX i pamięci operacyjnej 16 MB. Może działać jako serwer SSH, FTP, HTTP w trybie LiveCD. Udostępnia interfejs graficzny oraz szereg aplikacji sieciowych, internetowych i multimedialnych.
Strona WWW	http://www.damnsmalllinux.org
Uwagi	
Inne źródła	[6]

Nazwa produktu	Slax
Producent (autor)	
Platforma systemowa	x86
Sposób licencjonowania	GPL
Przeznaczenie	Minidystrybucja systemu Linux uruchamiana z płyty CD
Ogólna charakterystyka	Slax oparty jest o dystrybucję Slackware. Celem jego opracowania było udostępnienie możliwie dużej kolekcji użytecznego oprogramowania na małej płycie CD (o średnicy 8 cm)
Strona WWW	http://slax.linux-live.org/?lang=pl
Uwagi	Dostępne są różne wersje dystrybucji Slax, tak z graficznym jak i tekstowym interfejsem użytkownika, przeznaczone do pracy jako serwer albo stanowisko robocze
Inne źródła	

Kolejną grupę stanowią wyspecjalizowane dystrybucje systemu Linux, których funkcje zostały ograniczone do ściśle określonego zbioru.

Nazwa produktu	Coyote Linux Personal Firewall
Producent (autor)	Vortech Consulting, LLC
Platforma systemowa	x86
Sposób licencjonowania	Darmowy dla zastosowań indywidualnych i edukacyjnych
Przeznaczenie	Zapora sieciowa dla domowej lub szkolnej sieci komputerowej
Ogólna charakterystyka	Coyote Linux Personal Firewall korzysta z jądra 2.6 i filtruje pakiety sieciowe korzystając z pakietu iptables. Ma niewielkie wymagania dotyczące konfiguracji sprzętowej
Strona WWW	http://www.coyotelinux.com/index.php
Uwagi	Zarządzanie zaporą poprzez ssh i wbudowany serwer WWW
Inne źródła	

Współcześnie coraz większe znaczenie przypisywane jest oprogramowaniu służącemu do wirtualizacji zasobów systemu komputerowego. Wśród narzędzi tego typu należy wyróżnić tzw. *systemy maszyn wirtualnych*, które, dzieląc zasoby fizycznego komputera, tworzą środowisko do niezależnego funkcjonowania na nim wielu różnych systemów operacyjnych i działających pod ich kontrolą zadań.

Nazwa produktu	VMware
Producent (autor)	VMware, Inc.
Platforma systemowa	MS Windows, popularne dystrybucje systemu Linux
Sposób licencjonowania	komercyjny
Przeznaczenie	Tworzenie wirtualnego środowiska pozwalającego na bezkonfliktowe i równoczesne działanie różnych systemów operacyjnych na jednej maszynie
Ogólna charakterystyka	Gościnne systemy operacyjne: Microsoft Windows, Linux, Free BSD, Novell NetWare, Solaris (x86). Tworzone środowisko komputera gościnnego obejmuje BIOS, pamięć operacyjną, dyski, czytniki CD-ROM, drukarki, interfejsy sieciowe, USB i inne. Wirtualne urządzenia pamięciowe mogą być odwzorowane na urządzenia fizyczne, bądź symulowane w pamięci dyskowej systemu macierzystego. Oprogramowanie VMware działa w środowisku graficznym, współdzieląc fizyczny ekran, klawiaturę i mysz oraz pozwala na konstruowanie różnych konfiguracji sieciowych, w tym wirtualne łączenie maszyn gościnnych.
Strona WWW	http://www.vmware.com
Uwagi	Dostępne są wersje serwerowe GSX i ESX oraz wersje dla stanowisk roboczych

Maszyny wirtualne zbudowane pod kontrolą oprogramowania VMware można następnie wyeksportować i eksploatować w darmowym środowisku programu WMware Player tej samej firmy.

Kolejne systemy: User Mode Linux, Open Virtuozzo i Xen należą do klasy OpenSource i zostały opracowane przy nieco innych założeniach konstrukcyjnych, niż produkty firmy VMware.

Nazwa produktu	User Mode Linux
Producent (autor)	
Platforma systemowa	Linux (jądro 2.4 i późniejsze)
Sposób licencjonowania	GPL
Przeznaczenie	System maszyn wirtualnych dla architektury x86
Ogólna charakterystyka	User Mode Linux (UML) jest specjalnym modułem jądra systemu Linux, które umożliwia równoczesną eksploatację wielu kopii systemu Linux na tym samym komputerze. UML nie wymaga dla siebie środowiska graficznego. Możliwe jest tworzenie różnorodnych sieciowych konfiguracji maszyn gościnnych, którymi są typowe systemy linuksowe. Dostępne jest zdalne zarządzanie systemem UML.
Strona WWW	http://user-mode-linux.sourceforge.net
Uwagi	UML działa jako aplikacja użytkownika. Uprzywilejowane funkcje jądra maszyn wirtualnych realizowane są poprzez odwołania do jądra systemu rzeczywistego.
Inne źródła	[2]

Nazwa produktu	
Producent (autor)	Open Virtuozzo (OpenVZ)
Platforma systemowa	RedHat Enterprise, Fedora Core 4
Sposób licencjonowania	QPL (Q Public License)
Przeznaczenie	System maszyn wirtualnych dla platformy x86
Ogólna charakterystyka	OpenVirtuozzo stanowi darmową wersję komercyjnego produktu Virtuozzo. Pozwala na uruchomienie dużej liczby gościnnych niezależnych maszyn wirtualnych (systemów operacyjnych) wyposażonych w systemy operacyjne RedHat Enterprise Linux, Fedora Core wersja 3 lub 4 i Debian. W wersji darmowej narzędzia do tworzenia i zarządzania maszynami wirtualnymi dostępne są z linii poleceń.
Strona WWW	http://www.openvz.org
Uwagi	
Inne źródła	

Nazwa produktu	Xen
Producent (autor)	University of Cambridge, Computer Laboratory
Platforma systemowa	Linux: RedHat, SuSE, Debian, Mandrake BSD: NetBSD, FreeBSD
Sposób licencjonowania	GPL
Przeznaczenie	Monitor maszyn wirtualnych dla architektury x86
Ogólna charakterystyka	Maszyny wirtualne są zarządzane przez tzw. hypervisor, będący specjalnie przygotowanym jądrem linuksowym. Pod jego kontrolą uruchamiane są maszyny wirtualne, będące odpowiednio przygotowanymi wersjami systemów Linux, NetBSD i FreeBSD. Z poziomu konsoli xena można zarządzać wykorzystaniem zasobów fizycznych przez maszyny wirtualne, zatrzymywać i wznawiać ich pracę itp.
Strona WWW	http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
Uwagi	
Inne źródła	[23]

11.3 Systemy wielomaszynowe

Systemy Linux są szczególnie podatne na budowanie architektur wielomaszynowych. Istnieje wiele rozwiązań pozwalających na łączenie wielu samodzielnych komputerów w większe systemy, w celu uzyskania lepszych właściwości użytkowych. W szczególności, celem takich zabiegów może być:

- uzyskanie znacznej wydajności obliczeniowej (ang. *high performance*; HP) całego zestawu, przy zastosowaniu komputerów o przeciętnych mocach obliczeniowych,
- uzyskanie wysokiej dostępności usług (ang. *high availability*; HA) całego zestawu, przy zastosowaniu komputerów o przeciętnych parametrach niezawodnościowych.

Zestaw wielu maszyn, połączonych dla osiągnięcia jednego lub obydwu z wymienionych wyżej celów, nosi nazwę:

- *klastra* (ang. *cluster*), jeśli połączenia pomiędzy komputerami tworzącymi zestaw mają charakter lokalny, tj. komputery i urządzenia pamięci masowej znajdują się obok siebie i są połączone bezpośrednio lub poprzez przełączniki (koncentratory) sieciowe.
- *systemu rozproszonego* (ang. *distributed system*), jeśli połączenia te mają charakter zdalny, tj. lokalizacje komputerów są oddalone od siebie, a pomiędzy maszynami występują połączenia rutowane.

Nazwa produktu	**openMosix**
Producent (autor)	
Platforma systemowa	x86
Sposób licencjonowania	GPL
Przeznaczenie	Klaster obliczeniowy (HP)
Ogólna charakterystyka	OpenMosix stanowi rozszerzenie jądra systemu Linux, które przekształca zbiór połączonych ze sobą zwykłych komputerów w jeden superkomputer (klaster). Procesy zainicjowane w poszczególnych węzłach klastra migrują pomiędzy węzłami tak, aby wyrównać ich obciążenie zadaniami. Uzyskany w ten sposób układ jest samoadaptujący się i skalowalny, nie wymaga przystosowywania oprogramowania użytkowego do architektury środowiska obliczeniowego.
Strona WWW	http://openmosix.sourceforge.net/
Uwagi	
Inne źródła	

Nazwa produktu	**UltraMonkey**
Producent (autor)	
Platforma systemowa	Linux: RedHat i Debian
Sposób licencjonowania	GPL
Przeznaczenie	Klaster wysokiej dostępności (HA)
Ogólna charakterystyka	Klaster zawiera dwa lub więcej rzeczywistych komputerów. Dostępne konfiguracje zapewniają możliwość równoważenia obciążeń przez przenoszenie zadań pomiędzy komputerami rzeczywistymi i ich wzajemne zastępowanie w razie stwierdzenia, że realizowane usługi przestały być aktywne. UltraMonkey nie zawiera w sobie funkcji replikowania plików ani baz danych, co w razie potrzeby należy zapewnić z pomocą odrębnych narzędzi.
Strona WWW	http://www.ultramonkey.org
Uwagi	Nie wymaga specjalnie spreparowanego jądra systemowego
Inne źródła	

11.4 Symulatory i terminale

Do grupy tej zaliczymy programy imitujące inne platformy systemowe, dla umożliwienia wykonywania w systemie *N*X oprogramowania przeznaczonego dla odmiennego środowiska, a także programy odwzorowujące funkcje terminali różnych systemów operacyjnych.

Nazwa produktu	QEMU
Producent (autor)	
Platforma systemowa	(zob. „Ogólna charakterystyka”)
Sposób licencjonowania	GPL
Przeznaczenie	Uniwersalny emulator procesorów
Ogólna charakterystyka	QEMU emuluje procesory o różnych architekturach, umożliwiając wykonywanie przeznaczonego dla nich oprogramowania. QEMU działa w dwóch trybach: pełnej emulacji systemu (co pozwala na uruchomienie systemu operacyjnego) i w trybie użytkownika (co pozwala na uruchamianie aplikacji przeznaczonych dla określonych procesorów). Emulator w pełni obsługuje procesory: x86, x86_64 i PowerPC. W ograniczonym zakresie obsługuje również: Alpha, Sparc32, Sparc64, ARM, S390, ia64 i m68k.
Strona WWW	http://www.qemu.org
Uwagi	
Inne źródła	[20]

Nazwa produktu	PXES
Producent (autor)	2X Software Ltd.
Platforma systemowa	x86
Sposób licencjonowania	GPL
Przeznaczenie	Tworzenie „cienkiego” klienta lub bezdyskowego stanowiska roboczego
Ogólna charakterystyka	PXES jest mikrodystrybucją systemu Linux, która umożliwia terminalowe połączenie m.in. z dowolnym serwerem Unix/Linux poprzez telnet, ssh oraz X Window (protokół XDM), lub serwerem terminali firmy Microsoft (RDP), serwerami Citrix (ICA), VNC Server (TightVNC) lub NoMachine NX. Dzięki termu można przekształcić przestarzały komputer osobisty w pełnowartościowe stanowisko robocze. Start oprogramowania PXES jest możliwy z różnych nośników, w tym – z serwera sieciowego. W trakcie sesji mogą być udostępnione lokalne urządzenia zewnętrzne. Równocześnie można utrzymywać połączenia terminalowe z wieloma serwerami.
Strona WWW	http://www.2x.com/pxes/
Uwagi	Oprogramowanie PXES jest dostosowywalne do potrzeb użytkownika
Inne źródła	

11.5 Zarządzanie zasobami i bezpieczeństwem systemu

Przedstawimy tu produkty przeznaczone do wspomagania zarządzania zasobami samego systemu operacyjnego *N*X oraz zarządzania sieciami komputerowymi.

Nazwa produktu	openNMS
Producent (autor)	
Platforma systemowa	UNIX: Solaris (Sparc) Linux: CentOS, RedHat, RedHat Enterprise, Fedora Core, Debian, Mandrake, SuSE
Sposób licencjonowania	GPL
Przeznaczenie	Zarządzanie siecią komputerową w skali przedsiębiorstwa
Ogólna charakterystyka	openNMS monitoruje sieć komputerową, raportuje jej poziom sprawności, poprzez protokół SNMP zbiera dane z komputerów i urządzeń aktywnych w celu badania przepustowości sieci, rejestruje i sygnalizuje zdarzenia dotyczące funkcjonowania sieci.
Strona WWW	http://www.opennms.org
Uwagi	Korzysta z serwera bazodanowego PostgreSQL oraz Apache Tomcat i Java SDK
Inne źródła	[1]

Nazwa produktu	Bastille Linux
Producent (autor)	
Platforma systemowa	UNIX: HP-UX i Mac OS X Linux: Red Hat i Fedora Core, SUSE, Debian, Gentoo, MandrakeLinux, TurboLinux
Sposób licencjonowania	GPL
Przeznaczenie	Podstawowe umacnianie bezpieczeństwa systemu Linux
Ogólna charakterystyka	Bastille uruchamia zaporę sieciową, odbiera zwykłym użytkownikom prawa do uruchamiania narzędzi konfiguracyjnych, blokuje lub deaktywuje szereg usług, zabezpiecza hasłem program startowy, włącza starzenie haseł użytkowników, blokuje kombinację klawiszy Ctrl+Alt+Del, blokuje kompilatory, ogranicza możliwość logowania się do konsoli, rozszerza logowanie zdarzeń w systemie, w tym działań użytkowników, zabezpiecza serwer Apache.
Strona WWW	http://www.bastille-linux.org
Uwagi	Proces zabezpieczania odbywa się poprzez udzielanie odpowiedzi na szczegółowo objaśnione pytania. Istnieje możliwość cofnięcia zmian wprowadzonych przez Bastille.
Inne źródła	[8, 9]

Nazwa produktu	OpenSource Tripwire
Producent (autor)	
Platforma systemowa	UNIX, BSD, Linux
Sposób licencjonowania	
Przeznaczenie	Ochrona integralności systemu plików
Ogólna charakterystyka	OpenSource Tripwire bada wskazane pliki pod kątem wystąpienia nieautoryzowanych zmian. Atrybuty plików i katalogów przechowywane są na bezpiecznym nośniku (np. zabezpieczonym przed zapisem) i używane w celu porównania z aktualnymi atrybutami chronionych obiektów. Badanie wykonywane jest cyklicznie pod nadzorem procesu cron, z powiadomieniem superużytkownika o stwierdzonych rozbieżnościach. Poziom tolerancji różnic jest konfigurowalny.
Strona WWW	http://www.tripwire.org, http://sourceforge.net/projects/tripwire
Uwagi	Pod nazwą Tripwire występuje również pakiet komercyjny o podobnym przeznaczeniu. Zbliżone do Tripwire właściwości posiada pakiet AIDE rozpowszechniany na licencji GPL i dostępny dla szerszej klasy platform.
Inne źródła	[10, 14]

11.6 Oprogramowanie użytkowe

W grupie tej przedstawimy dwa ciekawsze opracowania przeznaczone do bezpośredniego wykorzystania przez właściwych użytkowników (ang. *end user*) systemów *N*X.

Nazwa produktu	Scribus
Producent (autor)	
Platforma systemowa	*N*X
Sposób licencjonowania	GPL
Przeznaczenie	System składu tekstu
Ogólna charakterystyka	Profesjonalne studio DTP - funkcjonalny odpowiednik programów Adobe PageMaker, Adobe InDesign lub QuarkXPress – pozwala na skład i przygotowywanie do druku różnorodnych publikacji
Strona WWW	http://www.scribus.org.uk
Uwagi	Dostępna jest wersja polskojęzyczna
Inne źródła	

Nazwa produktu	TinyERP
Producent (autor)	Tiny esprl, Belgia
Platforma systemowa	Linux, MS Windows
Sposób licencjonowania	GPL
Przeznaczenie	Elastyczny, modułowy system informatyczny do zarządzania przedsiębiorstwem klasy ERP
Ogólna charakterystyka	TinyERP integruje większość procesów zachodzących w małych i średnich przedsiębiorstwach. Automatycznie generuje dokumenty. Dostępne moduły: m.in. Kontakty z klientami (CRM), Marketing, Sprzedaż, Zaopatrzenie, Gospodarka magazynowa, Produkcja, Zarządzanie projektami, Środki trwałe, Księgowość, Personel (HR) i Logistyka. Możliwość dostosowania konfiguracji systemu do specyfiki przedsiębiorstwa, w tym modyfikacji formularzy ekranowych, raportów i schematów przepływu pracy. Dobra współpraca z pakietem OpenOffice.org w zakresie importu/eksportu dokumentów.
Strona WWW	http://www.tinyerp.org
Uwagi	Architektura aplikacji typu klient/serwer
Inne źródła	[18]

W Internecie istnieje duża ilość portali publikujących wartościowe rozwiązania różnorodnych problemów eksploatacyjnych dotyczących systemów *N*X. Warto na te zasoby zwracać uwagę, a zwłaszcza na oprogramowanie rozpowszechniane na licencji GNU GPL. Szczególnie wartym polecenia jest archiwum `www.sourceforge.net`, w którym w czasie pisania tego podręcznika było zarejestrowanych ok. 105.700 projektów (w różnym stadium zaawansowania). Znalezione produkty należy jednak zawsze ocenić pod kątem stopnia spełniania stawianych im wymagań oraz dojrzałości technicznej.

12 Nikt nam nie zrobi nic... czyli Bezpieczeństwo systemu *N*X

12.1 Wprowadzenie

Posłużymy się definicją podaną w pracy [11], która za podstawę przyjmuje cel użytkowania komputera.

Bezpieczeństwo systemu komputerowego

Komputer jest bezpieczny, jeśli użytkownik może na nim polegać, a zainstalowane oprogramowanie działa zgodnie z oczekiwaniami.

A więc, przez zaprzeczenie:

Zagrożenie systemu komputerowego

Komputer nie jest bezpieczny, gdy jakakolwiek przyczyna prowadzi do ograniczenia lub utraty zaufania do rezultatów jego działania.

Zwróćmy uwagę na to, że przytoczona definicja dotyczy *bezpieczeństwa biernego* i obejmuje różne rodzaje zagrożeń, a więc nie tylko te, które są wynikiem celowych – często spektakularnych i przez to łatwiej zauważalnych – działań ludzi. Z punktu widzenia użytkownika komputera, przyczyna bowiem niedostępności usług komputera jest mniej istotna niż to, kiedy będzie można ponownie z tych usług skorzystać i czy będzie można na nich polegać.

Dla odróżnienia, *bezpieczeństwo czynne* dotyczy braku możliwości powstania zagrożenia dla osób, mienia lub środowiska naturalnego, spowodowanego działaniem komputera. Ten aspekt nie będzie jednak dalej omawiany.

12.2 Zagrożenia i ich skutki

Przyczyny ograniczenia lub utraty zaufania do komputera można sklasyfikować jak niżej:

a. przyczyny fizykalne

- błędne działanie sprzętu wynikające z wad technicznych samych urządzeń i wad połączeń między nimi,
- niekorzystne warunki pracy urządzeń (zasilanie, lokalizacja, warunki środowiskowe: klimatyczne, mechaniczne, promieniowanie elektromagnetyczne (EM) i inne) ,
- niewłaściwe nośniki i materiały eksploatacyjne,

b. przyczyny eksploatacyjne

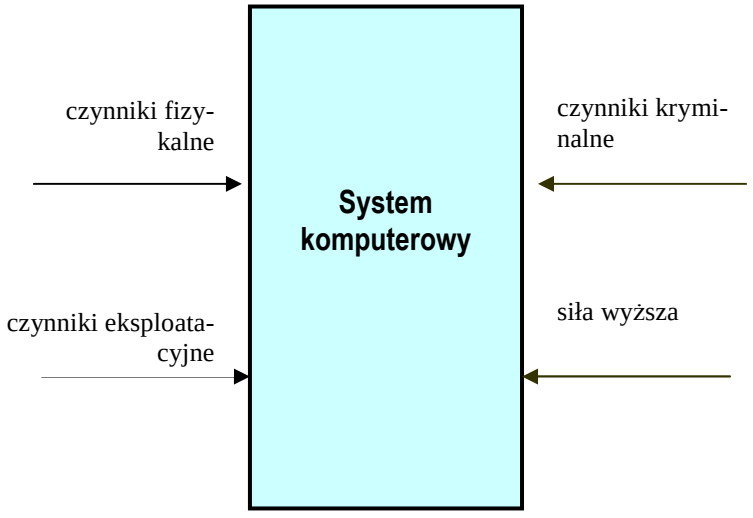
- nieprawidłowe posługiwanie się komputerem przez użytkowników właściwych i administratorów,
- ukryte wady oprogramowania lub jego niedostosowanie do obowiązującego stanu prawnego,
- złośliwe oprogramowanie (np. wirusy, robaki, trojany itp.),
- niepełne lub nieprawdziwe dane wykorzystywane do obliczeń lub nieodpowiednie parametry sterujące pracą samego komputera i jego oprogramowania,

c. przyczyny kryminalne

- nieupoważnione (wrogie) działania osób trzecich, np. włamanie, sabotaż, podsłuch łącza, podsłuch promieniowania EM,

d. przyczyny związane z działaniem tzw. *siły wyższej*

- nagła zmiana klimatu, pożar, powódź, zalanie, katastrofa ekologiczna, rozruchy, działania wojenne itp.



Rys. 155. Czynniki zagrażające systemowi komputerowemu

Wymienione wyżej zagrożenia wywołują skutki bezpośrednie i pośrednie, które wymienimy poniżej.

Skutki bezpośrednie wywoływane przez czynniki zagrażające systemowi komputerowemu:

a. w stosunku do sprzętu komputerowego

- uszkodzenie lub zniszczenie elementów konfiguracji sprzętowej, w tym nośników pamięci zewnętrznej,
- naruszenie integralności połączeń teleinformatycznych,

b. w stosunku do budynków i budowli
   - uszkodzenie lub zniszczenie pomieszczeń wymaganych do eksploatacji komputera i ich wyposażenia (urządzenia i instalacje: zasilające, klimatyzacyjne, przeciwpożarowe, antywłamaniowe itp.),

c. w stosunku do oprogramowania
   - utrata lub naruszenie integralności (zmiana funkcji) oprogramowania,
   - utrata lub naruszenie integralności plików konfiguracyjnych,
   - naruszenie praw autorskich w stosunku do oprogramowania,

d. w stosunku do danych
   - naruszenie poufności (przechwycenie) danych,
   - naruszenie integralności (zniekształcenie, zniszczenie) danych,
   - naruszenie prawnej ochrony w stosunku do baz danych podlegających takiej ochronie,

e. w stosunku do warunków eksploatacji komputera
   - ograniczenie lub uniemożliwienie dostępu do danych lub oprogramowania,
   - ograniczenie wydajności przetwarzania danych przez komputer lub szybkości dystrybucji wyników tego przetwarzania,

f. w stosunku do osób
   - zagrożenie zdrowia i życia,
   - naruszenie dóbr osobistych, w tym prywatności, dobrego imienia itp.

Do wywoływanych skutków pośrednich zaliczymy:

- skutki wymierne - straty majątkowe:
  - koszty odbudowy zniszczeń (pomieszczeń, urządzeń i innego wyposażenia),
  - koszty odtwarzania danych,
  - koszty ponownego zakupu i wdrażania oprogramowania,
  - utrata zysków w okresie ograniczenia lub zawieszenia działalności,
  - koszty wypłacanych odszkodowań i innych świadczeń,
- skutki niewymierne (w krótkim okresie czasu) - utrata wiarygodności instytucji lub utrata pozycji rynkowej firmy.

Jak widać z zamieszczonego przeglądu, skutki oddziaływania na system komputerowych czynników zagrażających mogą być znaczne i rozległe, nie tylko w kategoriach strat materialnych, lecz – znacznie bardziej dotkliwych – utraty możliwości działania, potencjału rozwojowego, pozycji i wiarygodności, czyli tych wartości charakteryzujących dany podmiot, których nie można kupić, lecz trzeba je wypracowywać latami.

Polityka bezpieczeństwa (ang. *security policy*) rozumiana jest jako całokształt działań prawnych, organizacyjnych i technicznych ukierunkowanych na

zapewnienie wymaganego poziomu zaufania do rezultatów działania systemu komputerowego [11]. Jej zadaniem jest utrzymanie:

- Poprawności realizacji funkcji danej organizacji,
- Integralności danych,
- Poufności danych na ustalonym poziomie,

Dostępności danych i usług w czasie i miejscu, gdzie jest to wymagane.

Polityka bezpieczeństwa jest pochodną strategii działania danej organizacji i służy osiąganiu celów wyznaczonych przez tę strategię. Kształt polityki bezpieczeństwa wynika z odpowiedzi na trzy kluczowe pytania:

- Co jest przedmiotem ochrony? (co należy chronić?),
- Jakie są źródła zagrożeń? (przed czym należy chronić?),
- Jaki jest wymagany poziom ochrony? (w jakim zakresie należy chronić?).

Na podstawie tych odpowiedzi dokonuje się określenia i wdrożenia niezbędnych przedsięwzięć w zakresie ochrony.

Dostępność systemu komputerowego

Dostępność systemu komputerowego oznacza pełną gotowość do świadczenia przezeń usług, dla realizacji których został on zbudowany.

Miarą dostępności systemu komputerowego jest stosunek średniego czasu sprawności do łącznego czasu jego eksploatacji. Miara ta bywa nazywana *współczynnikiem gotowości technicznej*.

Współczynnik dostępności (gotowości technicznej)

Współczynnik dostępności a przeciętny czas NIEsprawności [godzin]

Ogólny czas eksploatacji		Współczynnik gotowości technicznej *k*					
[jednostek]	[godzin]	0,9	0,99	0,999	0,9999	0,99999	0,999999
1 doba	24	2,40	0,24	0,024	0,0024	0,00024	0,000024
1 miesiąc	730	73,05	7,30	0,730	0,0730	0,00730	0,000730
1 rok	8 766	876,58	87,66	8,766	0,8766	0,08766	0,008766

Dla k = 0,9999 (tzw. „cztery dziewiątki") średni łączny czas niesprawności systemu wynosi 0,8766 godziny, tj. ok. 53 minuty w roku.

12.3 Obsługa zdarzeń dotyczących bezpieczeństwa

Sprzęt liczący i oprogramowanie komputera, w połączeniu z zewnętrznymi systemami monitorowania, biorą udział w wykrywaniu i obsłudze zdarzeń związanych z zagrożeniem bezpieczeństwa komputera jako całości. Poniżej zestawiono sposoby postępowania ze zdarzeniami stanowiącymi zagrożenie bezpieczeństwa.

Tabela 32. System komputerowy a zagrożenia

Klasa zagrożeń	Zagrożenie	Wykrywanie zdarzeń	Obsługa zdarzeń
Fizykalne	błędy sprzętu i łączy	układy kontrolne sprzętu	**procedury systemowe,** interwencja użytkownika
	niewłaściwe nośniki i materiały eksploatacyjne	układy kontrolne urządzeń zewnętrznych	
	awaria zasilania	układy automatyki zasilania	
Eksploatacyjne	błędy użytkowników	procedury diagnostyczne zawarte w oprogramowaniu	**procedury systemowe** i użytkowe
	wady oprogramowania	procedury diagnostyczne zawarte w oprogramowaniu	**procedury systemowe** i użytkowe
	złośliwe oprogramowanie	oprogramowanie monitorujące	**oprogramowanie monitorujące**
	błędne dane	procedury diagnostyczne zawarte w oprogramowaniu	**procedury systemowe** i użytkowe
Kryminalne	nieupoważnione działania osób trzecich	procedury monitorujące	**procedury systemowe** i użytkowe
Siła wyższa	pożar, zalanie	systemy sygnalizacji i alarmowania	**procedury systemowe,** interwencja użytkownika
	katastrofa ekologiczna, rozruchy, działania wojenne	brak	interwencja użytkownika

Uwaga: drukiem pogrubionym zaznaczono obsługę skutków zdarzeń przez system operacyjny.

W szczególności, system operacyjny, przy udziale odpowiedniego oprogramowania użytkowego, reaguje na wykryte zdarzenia w następujący sposób:

- W razie wystąpienia błędu urządzeń, łączy i nośników danych – dokonywana jest próba powtórnego wykonania nieudanej operacji oraz sygnalizacja i rejestracja zdarzenia, wykorzystanie nadmiarowości w reprezentacji danych, logiczna eliminacja trwale niesprawnych elementów systemu,
- W razie awarii zasilania, zalania lub pożaru – przetwarzanie jest zatrzymywane, uruchamiana sygnalizacja i rejestracja zdarzenia, a następnie system jest zamykany,
- W przypadku działań kryminalnych i wykrycia złośliwego oprogramowania – uruchamiana jest sygnalizacja i rejestracja zdarzenia, uszkodzone pliki są oznaczane, podejmowana jest próba neutralizacji lub usunięcia skutków,
- W razie wykrycia błędu użytkownika, błędu danych lub oprogramowania – uruchamiane są procedury wbudowane w system operacyjny dla korygowania błędu lub jego izolacji, w powiązaniu z uwierzytelnianiem użytkowników i autoryzacją wykonywanych przez nich czynności oraz podejmowana sygnalizacja i rejestracja zdarzeń,
- W przypadku stwierdzenia wykonania krytycznej czynności przez użytkownika, np. zmiany hasła, wrażliwej operacji w bazie danych (np. realizacji przelewu wartości pieniężnych, usunięcia lub zmiany ważnych danych) – zdarzenie jest sygnalizowane i rejestrowane.

Jak widać, sygnalizacja i rejestracja zdarzeń powinny być wykonywane w każdej sytuacji. Pierwsza z nich pozwala na podjęcie natychmiastowych działań ukierunkowanych na neutralizację skutków, druga zaś – na podjęcie szczegółowej analizy *post factum*, dzięki której można określić przyczynę i ewentualnego sprawcę zagrożenia oraz zmodyfikować odpowiednie procedury zabezpieczeń lub całą politykę bezpieczeństwa.

12.4 Ochrona systemów *N*X

Systemy komputerowe klasy *N*X podlegają takim samym rodzajom zagrożeń, jak wszystkie inne. To, co różni pod tym względem egzemplarze komputerów między sobą, to stopień podatności na poszczególne klasy zagrożeń, zależnie od jakości użytego sprzętu i oprogramowania oraz warunków jego eksploatacji, wdrożonych procedur ochrony bezpieczeństwa, a przede wszystkim - stanu świadomości użytkowników. W tym punkcie omówimy aspekty specyficzne dla przedmiotu naszego zainteresowania.

12.4.1 Co należy chronić?

Przedmiotem ochrony w systemach *N*X są:

- Konta użytkowników,
- Dane (wszelkiego rodzaju, włączając pliki konfiguracyjne i sterujące),
- Oprogramowanie.

Jak łatwo zauważyć, ochrona każdego z wymienionych zasobów wymaga ochrony systemu plików, gdzie są one zlokalizowane.

Ochrony wymagają również:

- Sprzęt komputerowy,
- Nośniki danych,
- Łącza teleinformatyczne i połączenia wszelkiego rodzaju,
- Usługi świadczone przez system komputerowy.

12.4.2 Przed czym należy chronić?

Należy zadbać o ochronę systemu komputerowego przed:

- nieświadomymi i celowymi działaniami osób,
- działaniami złośliwego oprogramowania,
- awariami.

12.4.3 W jakim zakresie należy chronić?

O ile odpowiedzi na poprzednie pytania były łatwe i natychmiastowe, o tyle rozstrzygnięcie problemu jakości ochrony nie jest możliwe bez odniesienia się do przyjętej polityki bezpieczeństwa. Z jednej strony oczekuje się, aby skuteczność ochrony była jak największa, z drugiej zaś należy pamiętać o kosztach – mierzalnych i niemierzalnych – wdrożenia i utrzymania restrykcyjnej polityki bezpieczeństwa. Ponieważ wysoki poziom bezpieczeństwa bywa synonimem niewygody w korzystaniu z usług komputera, to po przekroczeniu pewnego progu utrudnień, uzyskane efekty mogą być przeciwne do zamierzonych. Należy więc pamiętać, że rzeczywiście skuteczne są tylko narzędzia i procedury zrozumiałe i akceptowalne przez użytkowników.

12.4.4 Ochrona kont użytkowników

Konto użytkownika jest swoistą legitymacją do korzystania z usług komputera. Posiadanie konta oznacza możliwość wydawania systemowi poleceń, np. na poziomie powłoki lub serwera bazy danych. Aby nie dopuścić do wydawania poleceń przez osoby nieupoważnione lub na ich rzecz, należy te konta chronić przed niekontrolowanym użyciem. Służy temu czynność *uwierzytelniania użytkownika* (ang. *user authentication*) polegająca na sprawdzeniu, czy osoba podająca się za danego właściciela konta jest nim rzeczywiście. Polega ono

na weryfikacji jednego spośród wymienionych nośników poufnych cech charakterystycznych:

- czegoś, co się **zna** (np. hasło, PIN, itp.),
- czegoś, co się **ma** (np. karta z kluczem),
- czegoś, co ktoś trzeci może **potwierdzić** (np. certyfikat cyfrowy),
- czegoś, co można **zmierzyć** (np. odcisk palca, obraz siatkówki oka, próbka głosu).

Każda z metod ma swoje wady i zalety, dlatego do realizacji szczególnie wymagających polityk bezpieczeństwa metody te są łączone.

Mechanizm haseł jest powszechnie używaną, podstawową metodą uwierzytelniania. W systemach *N*X hasła przechowywane są w pliku `/etc/shadow`, jako drugie pole wiersza dotyczącego danego konta, zaszyfrowane tzw. *jednokierunkową funkcją skrótu* (ang. *hash function*). W starszych wersjach systemów używana była funkcja `crypt`, dopuszczająca hasła do 8 znaków długości i tworząca skróty 13-znakowe. Obecnie powszechnie jest używana funkcja MDA5, która przetwarza dłuższe hasła.

```
[root@vhomer ~]# more /etc/shadow
root:$1$QHpYB5WC$6WJnCr/vpeKkPHfwPj9DH0:13080:0:99999:7:::
bin:*:13080:0:99999:7:::
daemon:*:13080:0:99999:7:::

(wiersze pominięte)

[root@vhomer ~]#
```

Rys. 156. Fragment zawartości pliku `/etc/shadow`

Właścicielem pliku `/etc/shadow` jest użytkownik `root`, a tryb dostępu do pliku ma ósemkową wartość 400 (`r--------`).

Obok haseł, systemy *N*X udostępniają mechanizm ich starzenia się (ang. *password aging*), który pozwala na wymuszenie zmiany hasła przez użytkownika i blokowanie kont nieużywanych. Do sterowania starzeniem się haseł służą pola 3.-8. wiersza pliku `/etc/shadow`.

Tabela 33. Przeznaczenie pól rekordów pliku `/etc/shadow`

Nr pola	Zawartość
1	nazwa użytkownika (*logname*)
2	wartość funkcji skrótu MD5 dla hasła
3	ilość dni od 1970-01-01, jakie upłynęły od ostatniej zmiany hasła
4	ilość dni, przez które hasło nie może być zmienione
5	ilość dni, po których hasło musi być zmienione
6	ilość dni przed terminem wygaśnięcia hasła, kiedy użytkownik jest o tym ostrzegany

Tabela 33. (cd.)

7	ilość dni po wygaśnięciu hasła, po upływie których konto jest blokowane
8	ilość dni, jakie upłynęły od 1970-01-01 do blokady konta
9	(pole zarezerwowane)

Zabezpieczenie konta hasłem będzie skuteczne, o ile użytkownicy będą przestrzegać prostych zasad przedstawionych w tabeli poniżej.

Tabela 34. Zasady dotyczące haseł

Należy tworzyć hasła...	Nie należy w haśle umieszczać...	Nie należy też...
składające się minimum z 6 znaków	nazwy użytkownika w żadnej postaci - pisanej wprost, wspak, dwukrotnie, samymi dużymi literami itd.	kartek z zapisanym hasłem umieszczać w dostępnym miejscu (zwłaszcza przyklejać na monitorze komputera, tablicy ogłoszeń, w szufladzie biurka itp.)
zawierające zarówno małe i duże litery	imion ani nazwiska swojego, współmałżonka, krewnych, znajomych lub dzieci w żadnej postaci, również zdrobniałej lub skróconej	czytać hasła na głos albo dyktować go przez telefon, wysyłać pocztą elektroniczną lub w inny sposób przekazywać na odległość
zawierające oprócz liter cyfry i znaki przestankowe	żadnej innej informacji o sobie, która może być łatwo zdobyta, jak: adres, numer telefonu, data urodzenia, numer ubezpieczenia, marka samochodu, komputera itp.	pod żadnym pozorem udostępniać hasła innym osobom niż te, które są upoważnione do dostępu do danych zasobów (nawet na *tzw. chwilę*)
łatwe do zapamiętania, aby nie trzeba ich było zapisywać	słów znajdujących się w jakichkolwiek słownikach, encyklopediach lub innych wykazach terminologicznych	wprowadzać hasła do komputera, gdy niepowołana osoba stoi bezpośrednio przy komputerze i może obserwować klawiaturę
mocniejsze dla dostępu do ważniejszych zasobów	samych liter lub samych cyfr	
	kolejnych znaków klawiatury, np. qwerty, 123456 itp.	

Opracowanie autora na podstawie: "How to Develop a Network Security Policy" oraz CERT Advisories.

Zapasowe konto użytkownika `root`

Możliwość utworzenia dodatkowego konta o uprawnieniach takich, jak konto `root`, pozwala zabezpieczyć się przed skutkami przypadkowej utraty (zagubienia, zniekształcenia) hasła konta `root`.
W sytuacji awaryjnej wystarczy zalogować się za pomocą konta zapasowego i korzystać z pełni uprawnień, w tym możliwości korekty hasła konta `root`.

W przypadku bardziej restrykcyjnych polityk bezpieczeństwa, oprócz starzenia się haseł, należy rozważyć możliwość:

- kontroli mocy hasła (ang. *password strength*),
- wprowadzenia rozwiązań wspierających hasła jednorazowe,
- wprowadzenia uwierzytelniania na zasadzie *wezwanie-odpowiedź* (ang. *challenge-response*),
- zastosowania kryptograficznych urządzeń uwierzytelniających (kart identyfikacyjnych, tokenów kryptograficznych, czytników linii papilarnych, skanerów siatkówki oka, analizatorów mowy itp.).

Kontrola mocy hasła zabezpiecza przed ustanawianiem przed użytkowników haseł zbyt prostych (patrz: tabela 22.). Metoda wezwanie-odpowiedź polega na tym, że logujący się użytkownik otrzymuje od systemu pewien ciąg znaków (tzw. *wezwanie*), ma ten ciąg przekształcić według indywidualnego tajnego algorytmu i przesłać wynik do systemu (tzw. *odpowiedź*). Najczęściej do przeprowadzenia tego przekształcenia służy urządzenie zwane *tokenem*, zabezpieczone za pomocą kodu liczbowego, tzw. *PIN-u*. Do obsługi algorytmów uwierzytelniania w systemie operacyjnym służą tzw. wtyczki - moduły PAM (ang. *Pluggable Authentication Modules*).

Logowanie użytkownika `root`

Superużytkownik nie powinien logować się wprost do konta `root`. Powinien on najpierw zalogować się na zwykłe konto, a następnie wejść w rolę `root`-a poleceniem `su - root`.
Wykonywanie poleceń su jest rejestrowane w pliku `/var/log/messages`.

Zaleca się okresowe sprawdzanie istniejących w systemie kont użytkowników na okoliczność pojawiania się kont nieznanych administratorowi, a zwłaszcza kont ekwiwalentnych użytkownikowi `root` (UID=0, GUD=0).

12.4.5 Ochrona systemu plików

Poszczególne pliki (katalogi) powinny należeć do właścicieli i grup właścicielskich, stosownie do roli, jaką te pliki pełnią w systemie. Tryby dostępu do plików winny być ustalane tak, aby zapewnić minimum uprawnień niezbędnych do ich wykorzystania. Jeżeli podstawowy system kontroli dostępu nie jest wystarczająco mocny, należy rozważyć zastosowanie tzw. *list kontroli dostępu* (ang. *Access Control Lists*; ACL).

Mechanizm list kontroli dostępu dla systemów plikowych ext2 i ext3 udostępnia dodatkowe polecenia `acl`, `chacl`, `getfacl`, `setfacl`) oraz `chattr` i `lsattr`, wprowadzając rozszerzone atrybuty dostępu do plików, wymienione poniżej:

`A` – bez modyfikacji rekordu atime,
`a` – otwarcie pliku w trybie dopisywania,
`c` – automatyczna kompresja/dekompresja pliku,
`d` – bez tworzenia kopii zapasowej przez program `dump`,
`i` – usunięcie możliwości zapisu/zmiany/skasowania pliku,
`j` – wymuszenie zapisu informacji o zmianie pliku do kroniki systemu plików ext3 przed wykonaniem zapisu do pliku,
`s` – po usunięciu pliku zwolnione bloki na dysku są zerowane,
`S` – zapis pliku w trybie synchronicznym,
`t` – dla aplikacji bezpośrednio korzystających z systemu plików (np. LILO),
`u` – po usunięciu pliku będzie możliwe jego odzyskanie.

Dokładniejszy opis pakietu ACL znajdzie Czytelnik w `man acl` oraz np. w [4].

Wśród innych czynności ukierunkowanych na zabezpieczenie systemu plików wymienimy:

- usuwanie nadmiernych uprawnień dostępu, w tym SUID i SGID,
- usuwanie zbędnych (twardych i miękkich) dowiązań do plików,
- tworzenie kopii bezpieczeństwa i kopii archiwalnych na urządzeniach i w systemach składowania (lokalnie, zdalnie i w sposób rozproszony) i porównywanie zawartości lub parametrów plików z wzorcem (np. poleceniami `diff`, `cmp`, `ls -lR`),
- obliczanie i weryfikację sum kontrolnych dla plików (np. programem obliczającym skrót MD5),
- obliczanie i weryfikację sygnatur plików (np. programem `tripwire`),
- szyfrowanie plików i całych woluminów dyskowych.

12.4.6 Ochrona danych

Oprócz ochrony systemu plików jako całości, ochronie podlega zapis wszelkich informacji, niezależnie od właściciela, przeznaczenia, postaci, miejsca i sposobu przechowywania. Ochronie podlegają:

- parametry sterujące systemu operacyjnego,
- parametry sterujące zainstalowanego oprogramowania,
- dane na temat kont użytkowników i ich uprawnień,
- kroniki zdarzeń,
- dane na temat komputerów, adresacji i struktury sieci komputerowej (np. DNS),
- inne informacje.

Skuteczność ochrony danych może być jednak zróżnicowana, stosownie do ich ważności, a zakres ochrony obejmuje:

- ochronę danych przed nieupoważnionym dostępem na poziomie fizycznym, to jest ochronę:
- urządzeń przechowujących dane,
- fragmentów systemu plików,
- ochronę aplikacji (oprogramowania) zapewniających dostęp do danych,
- ochronę kronik dostępu do danych.

Ochrona danych jest realizowana:

- na poziomie aplikacji (poziom logiczny),
    - poprzez zarządzanie uprawnieniami dostępu do aplikacji (hasła do aplikacji, uprawnienia dostępu do tabel bazy danych, formularzy, raportów itp.),
    - poprzez tworzenie kopii bezpieczeństwa i kopii archiwalnych,
    - z wykorzystaniem transakcyjnego modelu dostępu do danych,
- na poziomie systemu plików (poziom fizyczny).

12.4.7 Ochrona oprogramowania

Ochrona oprogramowania obejmuje:

- weryfikację przed jego zainstalowaniem w systemie komputerowym poprzez sprawdzenie źródła pochodzenia,
- poświadczenie autentyczności oprogramowania podpisem cyfrowym (o ile jest dostępne),
- badanie sumy kontrolnej (np. funkcji skrótu MD5) dla pliku lub plików dystrybucyjnych,
- obsługę w czasie eksploatacji,

- sprawdzanie cech plików zawierających poszczególne elementy aplikacji, według zasad ogólnych obowiązujących dla plików (wielkość, data modyfikacji, prawa własności, tryb dostępu),
- wykonywanie testów integralności (sprawdzanie sum kontrolnych za pomocą funkcji skrótu),
- wykonywanie zaleconych przez producenta uaktualnień.

12.4.8 Ochrona nośników

Ochronie podlegają jednostki pamięci dyskowej, kasety z taśmą magnetyczną, płyty z zapisem optycznym, dyski RAM itp. Procedury bezpieczeństwa dotyczące nośników danych i nośników oprogramowania mogą uwzględniać:

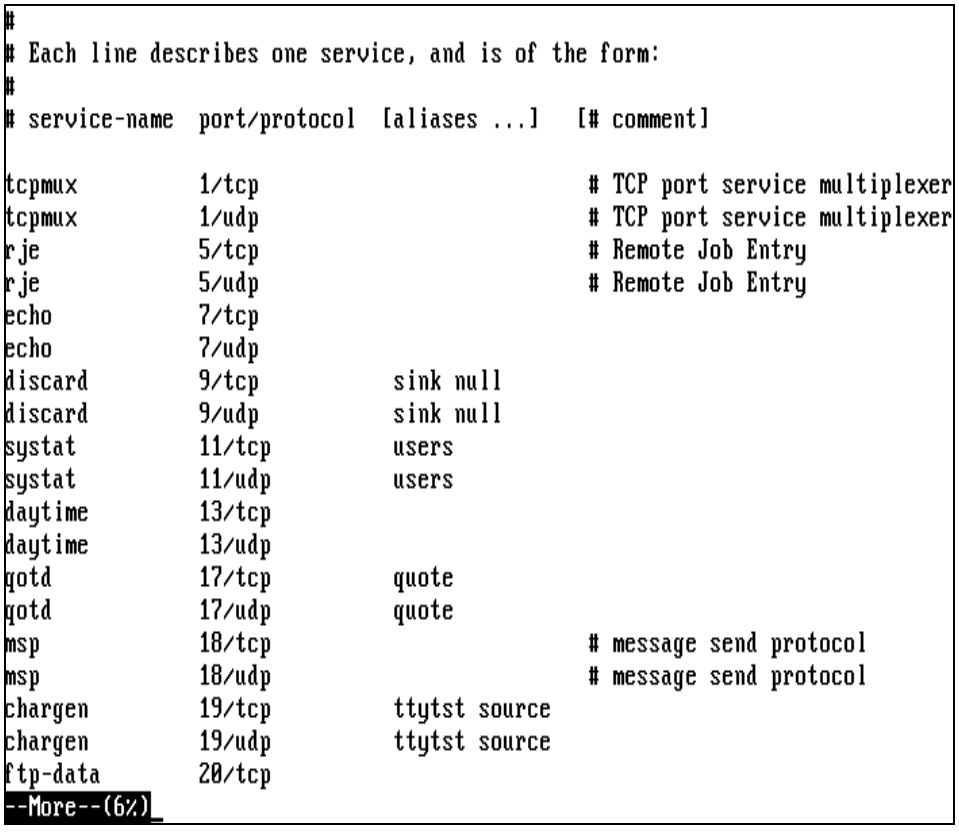
- zastosowanie sejfów ognioodpornych,
- przechowywanie kopii w kilku lokalizacjach oddalonych od centrum przetwarzania danych (tzw. *serwerowni*),
- rotację nośników,
- procedury likwidacji nośników zużytych i uszkodzonych.

12.4.9 Ochrona łączy teleinformatycznych

Ochrona łączy teleinformatycznych dotyczy zapobiegania nasłuchowi i podsłuchowi łączy (kablowych, radiowych, satelitarnych itp.) oraz promieniowania elektromagnetycznego emitowanego przez urządzenia komputerowe i łączące je kable oraz skanowaniu komputerów i ich portów poprzez sieć komputerową.

12.4.10 Ochrona usług

Praktycznie każdy system *N*X realizuje pewien zakres usług sieciowych, nawet jeśli nie jest wyznaczony do roli serwera. Wykaz potencjalnych usług znajduje się w pliku `/etc/services`. Poszczególne wiersze pliku wiążą ze sobą standardową nazwę protokołu z numerem portu danej usługi i nazwą grupy protokołów (TCP, UDP). To, czy dana usługa sieciowa tam opisana jest faktycznie realizowana w danym systemie, zależy od parametrów startowych systemu i poleceń wydanych przez superużytkownika w trakcie eksploatacji komputera.



Rys. 157. Fragment pliku `/etc/services`

W systemach *N*X usługi sieciowe realizowane są na dwa sposoby:

- jako samodzielne demony (np. `named` – serwis nazw domenowych, `crond` – obsługa procesów cyklicznych, `httpd` – serwer WWW, `sshd` – serwer szyfrowanych połączeń terminalowych, i in.),
- jako usługi zarządzane przez nadrzędny demon, tzw. *superserwer internetowy* `inetd` (starszy) lub `xinetd` (nowszy). Programy realizujące poszczególne usługi są wywoływane przez superserwer w miarę potrzeb i po wykonaniu swojej funkcji usuwane.

Demony są procesami rezydentnymi - są uruchamiane podczas startu systemu i usypiane w czasie bezczynności. Za pracę demonów (uruchamianie, zatrzymywanie, restartowanie, kontrolę stanu) odpowiedzialne są skrypty przechowywane w katalogu `/etc/rc.d/`. W jego podkatalogu `init.d` znajdują się skrypty sterujące poszczególnymi usługami. Skrypty te posiadają standardowy sposób wywołania: wymagają podania jednego parametru, którym może być: `start`, `stop`, `restart` lub `status`. Nazwa parametru wyjaśnia jego funkcję.

Konfiguracja usług zarządzanych przez demon `xinetd` opisana jest w plikach umieszczonych w katalogu `/etc/xinetd.d`. Jako przykładu użyjemy pliku konfiguracji usługi `ktalk`, będącej odpowiednikiem UNIX-owego `talk`.

```
$ more /etc/xinetd.d/ktalk
# default: off
# description: KDE version of the talk server (accepting talk requests
#              for chatting with users on other systems).
service ntalk
{
         disable             = yes
         socket_type  = dgram
         wait         = yes
         user         = root
         group        = tty
         serwer              = /usr/bin/ktalkd
}
$
```

Rys. 158. Plik konfiguracyjny usługi `talk`

Z punktu widzenia zarządzania bezpieczeństwem istotna jest wartość parametru `disable` (ang. *zabroń*); `yes` oznacza, że usługa jest nieaktywna.

W następnej tabeli wskazano zalecenia dotyczące udostępniania podstawowych usług sieciowych systemu *N*X.

Usługi konieczne dla realizacji założonych funkcji systemu komputerowego powinny być z kolei zabezpieczone przez zaporę sieciową (ang. *firewall*), w wykonaniu programowym lub sprzętowym. Do zbudowania i skonfigurowania prostej zapory można np. posłużyć się programem Bastille Linux, wspomnianym w punkcie 11.5. Szczegółowe omawianie zasad konfiguracji zapór sieciowych wykracza jednak znacznie poza ramy tej publikacji.

Tabela 35. Zalecenia na temat udostępniania usług sieciowych

Nazwa	Port	Wyłączyć	Warunkowo włączyć	Włączyć	Zastąpić usługą
echo	7	tak			
systat	11	tak			
ftp-data	20		tak		np. sftp/115
ftp	21				
telnet	23	tak			ssh/22
smtp	25		tak		
domain	53		tak		
bootps	67		tak		
bootpc	68				
tftp	69	tak			
gopher	70	tak			
finger	79	tak			
http	80		tak		https/443
pop3	110		tak		pop3s, imaps
nntp	119		tak		
ntp	123		tak		
netbios-ns	137		tak		Uwaga – bardzo nie-bezpieczne!
netbios-dgm	138		tak		
netbios-ssn	139		tak		
snmp	161		tak		
snmptrap	162		tak		
irc	194		tak		ircs/994
ipx	213		tak		
ldap	389		tak		ldaps/636
ldaps	636		tak		
rsync	873		tak		Z ustawieniem: -e ssh
imaps	993		tak		
pop3s	995		tak		
nfs	2049	tak			
(usługi specyficzne dla aplikacji)			tak		

12.5 Reagowanie na incydenty

W razie stwierdzenia faktu dokonania włamania lub innego wrogiego oddziaływania na system komputerowy należy podjąć czynności wymienione poniżej:

- ochronić system i jego zasoby po uszkodzeniu,
- zabezpieczyć dowody (w postaci wiernej kopii zaatakowanego systemu, jeśli jest to możliwe),
- zidentyfikować zakres i źrodła wrogiego oddziaływania,
- zapewnić możliwość przywrócenia zdolności komputera do dalszego działania,
- zabezpieczyć komputer i jego zasoby przed powiększaniem się zakresu szkód,
- zgłosić incydent instytucjom zajmującym się ściganiem przestępczości komputerowej (CERT, policja),
- poinformować zwierzchników i zainteresowanych użytkowników o ewentualnych skutkach włamania,
- wdrożyć rozwiązania zastępcze na czas odtwarzania systemu,
- przywrócić normalne działanie systemu,
- udokumentować zebrane informacje na temat sposobu oddziaływania na komputer, zakresu poczynionych szkód oraz podjętych działań naprawczych,
- wyciągnąć i wdrożyć wnioski na przyszłość.

Zalecenia dotyczące prowadzenia dochodzenia w przypadku włamania do systemu komputerowego

- Wszelkie zmiany w zaatakowanym systemie dokonywane podczas prac związanych z zabezpieczeniem dowodów i przywracaniem sprawności systemu wykonuj na kopiach, a nie na oryginalnym (uszkodzonym systemie).
- Nie ufaj zaatakowanemu systemowi operacyjnemu, wykorzystaj zewnętrzny dysk startowy.
- Dokumentuj wszystkie wykonywane czynności.
- Współpracuj z instytucjami powołanymi do zwalczania przestępczości komputerowej.

Computer Emergency Response Team (CERT)

CERT (http://www.cert.org) jest organizacją powołaną do przeciwdziałania i reagowania na zagrożenia systemów komputerowych o skali ponadlokalnej. W ramach profilaktyki analizuje ona źródła zagrożeń i tzw. *luki w bezpieczeństwie* eksploatowanego oprogramowania i publikuje odpowiednie alerty oraz tzw. *łaty* (ang. *patches*), wskazówki dotyczące postępowania w sytuacjach związanych z zagrożeniami i raporty na temat ogólnego stanu bezpieczeństwa oraz prowadzi działalność edukacyjną. Uczestniczy w eliminacji źródeł zagrożeń.

Witryna polskiej części CERT (http://www.cert.pl) zawiera m.in. dane kontaktowe do zgłaszania incydentów online, pocztą elektroniczną, faksem lub telefonicznie. CERT w razie potrzeby może określić lokalizację sprawcy (w sensie sieciowym i geograficznym) i spowodować zablokowanie generowanego przez niego ruchu sieciowego.

13 Koniec wieńczy dzieło... czyli Podsumowanie

Pisanie podręcznika jest sztuką wyboru:

- po pierwsze – adresata,
- po drugie – zakresu przedstawionego materiału,
- po trzecie – takiego sposobu przedstawienia treści, który uczyni dzieło możliwie przydatnym i atrakcyjnym.

Z różnych też względów powstałe dzieło powinno być inne od wszystkich wcześniej wydanych. Cel to szczytny, choć nie do końca daje się zrealizować, po pierwsze z powodu ustalonego zakresu materiału, który powinien być przedstawiony, po drugie z uwagi na podobnego lub wręcz tego samego adresata.

Jak wspomniano w przedmowie, podręcznik ten jest adresowany do użytkowników umownie nazywanych początkującymi i średnio zaawansowanymi. Intencją jego autora było skupienie się bardziej na tym **jak** używać systemów *N*X, aniżeli opisywać **co** one potrafią, aczkolwiek przedstawienie tego pierwszego nie byłoby możliwe bez encyklopedycznej wiedzy na temat budowy i działania.

Materiały uzupełniające podstawową treść podręcznika miały za cel zwrócenie uwagi Czytelnika na konieczność traktowania tytułowego obszaru z pewnej perspektywy, wyznaczonej przez inne dziedziny wiedzy i umiejętności, z którymi dopiero razem, poprzez synergię, jest ona w stanie zapewnić oczekiwane efekty i korzyści użytkowe. Dominujący we współczesnym świecie system edukacyjny, przewidujący kształcenie adeptów w ramach wielu odrębnych specjalistycznych kursów dotyczących ściśle wyodrębnionych zagadnień lub punktów widzenia, prowadzi do postrzegania wiedzy w postaci zatomizowanej i wyizolowanej. Wraz z pogłębianiem się specjalizacji słabnie poczucie związków, jakie realnie istnieją pomiędzy różnymi aspektami tych samych obiektów i zjawisk.

Byłoby dobrze, aby praktykujący inżynier informatyk pamiętał, że system komputerowy nie kończy się na sprzęcie i oprogramowaniu. Zresztą czy tylko informatyk powinien swój obszar działania traktować całościowo?

Źródła wiedzy… czyli Literatura

Literatura cytowana

1. Tarus Balog, Zarządzanie siecią biznesową z OpenNMS, Linux plus, nr 11/2005
2. Michał Bazewicz, Przynęty wirtualne – UML, P18, Bezpieczeństwo informacji od A do Z, Wiedza i Praktyka, Czerwiec 2002
3. Brian Blackmore, UNIX shell differences and how to change your shell, http://www.faqs.org/faqs/unix-faq/shell/shell-differences/History and Timeline
4. Lisa Carnahan, POSIX Security Interfaces and Mechanisms, http://csrc.nist.gov/publications/nistpubs/800-7/node17.html
5. CentOS 4.1, Linux Magazine, Sierpień/wrzesień 2005
6. Damn Small Linux 1.1, Linux plus, nr 07/2005
7. Ralf van Dooren, Quota mini-HOWTO, v0.5, 2003-08-09, www.linux.org
8. Jean-Pierre Féval, Bastille Linux – narzędzie zwiększające bezpieczeństwo dystrybucji, Linux plus, nr 04/2005
9. Ernest Frankowski, Bastille – utwardzony Linux, B07, Bezpieczeństwo informacji od A do Z, Wiedza i Praktyka, Czerwiec 2004
10. Ernest Frankowski, Badanie integralności Linuksa za pomocą Tripwire, B11, Bezpieczeństwo informacji od A do Z, Wiedza i Praktyka, Październik 2005
11. Simon Garfinkel, Gene Spafford, Bezpieczeństwo w Unixie i Internecie, Wydawnictwo RM, Warszawa 1997
12. Stein Gjoen, HOWTO: Multi Disk System Tuning, v0.33a, May 2002, www.linux.org
13. HOW MUCH INFORMATION 2003?, University of California at Berkeley, http://www.sims.berkeley.edu/research/projects/how-much-info-2003
14. Michał Kopijasz, Czy ktoś grzebał w Twoich plikach?, C02, Bezpieczeństwo informacji od A do Z, Wiedza i Praktyka, Lipiec/sierpień 2002
15. Old Computers. On-Line Museum, http://www.old-computers.com/museum/default.asp
16. Jakub Ósterga ard, Emilio Bueso, The Software-RAID HOWTO, v1.1, 2004-06-03, www.linux.org
17. The OpenGroup, http://www.unix-systems.org/what_is_unix/history_timeline.html
18. Fabien Pinckaers, Zarządzanie przedsiębiorstwem z Tiny ERP, Linux plus, nr 11/2005
19. Podręcznik wbudowany systemu Linux Fedora Core 4 (polecenie `man`)
20. QEMU, Linux plus, nr 01/2005

21. Arnaud Taddei, Shell Choice. A shell comparison, September 28, 1994, CN/DCI/162, http://www.hep.phy.cam.ac.uk/lhcb/LHCbSoftTraining/documents/ShellChoice.pdf
22. UNIXguide.net - http://www.unixguide.net
23. Xen User's Manual, Xen v2.0 for x86, The Xen Team, University of Cambridge, UK, 2002-2004, http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/user.pdf

Podręczniki i monografie

24. Bill Ball, Poznaj Linux, Wydawnictwo MIKOM, Warszawa 1999
25. Lech S. Borkowski, UNIX. Poradnik użytkownika, Wydawnictwo MIKOM, Warszawa 2003
26. Æleen Frish, UNIX. Administracja systemu, Wydawnictwo RM, Warszawa 2003
27. Rickford Grant, Linux. Praktyczny kurs, Helion, Gliwice 2003
28. Zbyszko Królikowski, Michał Sajkowski, System operacyjny UNIX dla początkujących i zaawansowanych, Wydawnictwo Nakom, Poznań 2000
29. Peter Norton, Petera Nortona przewodnik po Unix-ie, Wydawnictwo PLJ, Warszawa 1993
30. Jerry Peek, Grace Todino, John Strang, UNIX. Wprowadzenie, Helion, Gliwice 2002
31. Uresh Vahalia, Jądro systemu UNIX. Nowe horyzonty, Wydawnictwa Naukowo-Techniczne, Warszawa 2001

Załącznik - Tabela kodu ASCII

Dec	Hex	Znak	Dec	Hex	Znak
0	0	NUL	64	40	@
1	1	SOH	65	41	A
2	2	STX	66	42	B
3	3	ETX	67	43	C
4	4	EOT	68	44	D
5	5	ENQ	69	45	E
6	6	ACK	70	46	F
7	7	BEL	71	47	G
8	8	Cofacz	72	48	H
9	9	Tabulator	73	49	I
10	0A	Wiersz odstępu	74	4A	J
11	0B	Home	75	4B	K
12	0C	Koniec strony	76	4C	L
13	0D	Koniec wiersza	77	4D	M
14	0E	SO	78	4E	N
15	0F	SI	79	4F	O
16	10	DLE	80	50	P
17	11	DC1	81	51	Q
18	12	DC2	82	52	R
19	13	DC3	83	53	S
20	14	DC4	84	54	T
21	15	NAK	85	55	U
22	16	SYN	86	56	V
23	17	ETB	87	57	W
24	18	CAN	88	58	X
25	19	EM	89	59	Y
26	1A	SUB	90	5A	Z
27	1B	ESC	91	5B	[
28	1C	Kursor w prawo	92	5C	\
29	1D	Kursor w lewo	93	5D	]
30	1E	Kursor w górę	94	5E	^
31	1F	Kursor w dół	95	5F	_
32	20	Spacja	96	60	`
33	21	!	97	61	a
34	22	"	98	62	b
35	23	#	99	63	c
36	24	$	100	64	d
37	25	%	101	65	e
38	26	&	102	66	f

Tabela kodu ASCII (cd.)

39	27	'	103	67	g
40	28	(	104	68	h
41	29	)	105	69	i
42	2A	*	106	6A	j
43	2B	+	107	6B	k
44	2C	,	108	6C	l
45	2D	-	109	6D	m
46	2E	.	110	6E	n
47	2F	/	111	6F	o
48	30	0	112	70	p
49	31	1	113	71	q
50	32	2	114	72	r
51	33	3	115	73	s
52	34	4	116	74	t
53	35	5	117	75	u
54	36	6	118	76	v
55	37	7	119	77	w
56	38	8	120	78	x
57	39	9	121	79	y
58	3A	:	122	7A	z
59	3B	;	123	7B	{
60	3C	<	124	7C	|
61	3D	=	125	7D	}
62	3E	>	126	7E	~
63	3F	?	127	7F	DEL (kasowanie)

Nazwy kolumn: DEC – dziesiętna wartość kodu znaku, HEX – szesnastkowa wartość kodu znaku

Spis rysunków

Spis tabel

Indeks haseł

R

S

T

U

W

Z