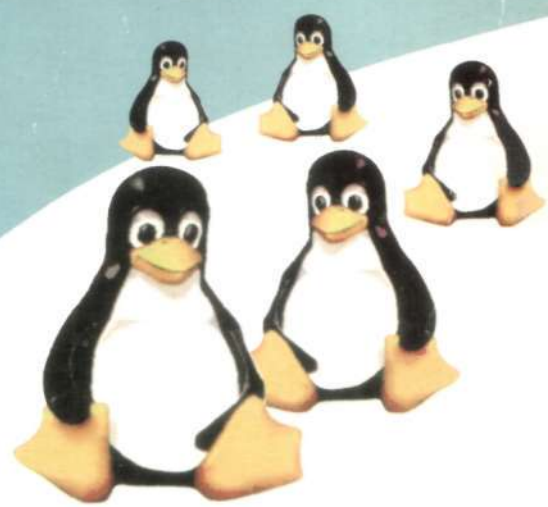
ĆWICZENIA

z systemu

Linux



PODSTAWY OBSŁUGI SYSTEMU

LESZEK MADEJA

MIKOM

... z każdym bitem serce

Wydawnictwo MIKOM

tel./fax: (0 22) 823-70-77, (0 22) 823-94-61

(0 22) 659-28-63

e-mail: mikom@mikom.com.pl
Księgarnia internetowa: http://www.mikom.com.pl

Redakcja: ul. Grójecka 53/57, Warszawa
Telefony redakcji: (0 22) 824-01-70, (0 22) 824-01-72

Sprzedaż hurtowa: ul. Andrzejowska 3, Warszawa-Ochota

Zapraszamy do współpracy autorów

- Wasze materiały dydaktyczne mogą stać się książką
- Wasze doświadczenie może się przydać innym

Czekamy na uwagi i opinie

- które serie są najbardziej interesujące
- jakich książek brakuje w naszej ofercie
- każda Państwa opinia jest dla nas cenna

Piszcie na adres: redakcja@mikom.com.pl

LESZEK MADEJA

ĆWICZENIA Z SYSTEMU LINUX

PODSTAWY OBSŁUGI SYSTEMU

Projekt okładki: **Grzegorz Ławniczak**

Redakcja: **Justyna Domasłowska-Szulc**

Skład komputerowy: **Dorota Świstak**

Linux zdobywa coraz większą popularność dzięki temu, że jest to darmowy (licencja GNU), stabilny, wydajny i uniwersalny system operacyjny, o praktycznie nieograniczonym zakresie zastosowań. Wielu użytkowników komputerów, chociaż często zetknęło się już z Linuksem (a nawet zainstalowało go w swoim komputerze) nie potrafi skorzystać z potęgi tegc systemu, gdyż napotyka barierę odmienności, której samodzielnie nie jest w stanie przekroczyć.

Niektórzy próbują uczyć się Linuksa korzystając z literatury uniksowej, tu jednak czeka na nich kolejna przeszkoda. Literatura uniksowa jest dwubiegunowa. Na jednym biegunie mamy hermetyczne publikacje dla profesjonalnych administratorów i programistów. Na drugim biegunie znajduje się popularne teksty przeznaczone dla zwykłych użytkowników dużych instalacji uniksowych, nie zawierające z definicji zagadnień związanych z administracją systemem. Tymczasem użytkownik Linuksa jest nie tylko zwykłym użytkownikiem, ale jednocześnie administratorem własnego systemu, a przy rozwiązywaniu problemów zdany jest zwykle sam na siebie

Niniejsza książka, wychodzi naprzeciw potrzebom użytkowników Linuksa. Jest bowiem prostym, ale jednocześnie precyzyjnym wprowadzeniem do systemu oraz uczy umiejętności potrzebnych zarówno administratorowi, ja i zwykłemu użytkownikowi. Książka ma pomóc w pokonaniu bariery odmienności, a tym którzy już tę barierę sforsowali w poszerzeniu dokonań wyłomu. Przegląd zagadnień prezentowanych w książce zawarty jest na czwartej stronie okładki.

Wydawca: MIKOM, ul. Andrzejowska 3, 02-312 Warszawa, tel. 823-70-77

Druk: ZWP „HEL", ul. Grenadierów 77, Warszawa, tel. 810-12-71

ISBN 83-7158-199-8

Warszawa, wrzesień 1999

Spis treści

1. Wstęp

Linux (www. 1 i nux. org) jest wielozadaniowym i wielodostępnym systemem operacyjnym klasy UNIX, stworzonym z uwzględnieniem standardu POSIX, dystrybuowanym na podstawie licencji GNU (GNU General Public License) opracowanej przez Free Software Foundation. Licencja ta (www.gnu.org) udostępnia oprogramowanie za darmo, chroniąc jednak prawa autorskie. Cechą charakterystyczną Linuksa jest to, że został napisany od podstaw i nie jest „obciążony" historycznym kodem UNIX-a pochodzącym z początku lat siedemdziesiątych, tak jak większość (a może wszystkie) komercyjnych systemów UNIX.

Pomyślany początkowo jako narzędzie dla hobbystów i profesjonalnych programistów, Linux staje się w coraz większym stopniu systemem operacyjnym dla masowego użytkownika". Dzieje się tak zarówno dzięki merytorycznemu rozwojowi zwłaszcza jądra) systemu, jak i obudowywaniu go (w konkretnych dystrybucjach) coraz bardziej komfortowymi programami instalacyjnymi i konfiguracyjnymi oraz gwałtownemu rozwojowi zaawansowanych interfejsów graficznych. W niniejszej książce unikać będziemy zajmowania się jakże często atrakcyjnymi „wodotryskami". Przede wszystkim: nie korzystamy z interfejsu graficznego. Skoncentrujemy się na samym siermiężnym (konsola znakowa) systemie - objaśniając za pomocą stu dwudziestu ćwiczeń podstawy obsługi i funkcjonowania samego Linuksa, a nie jego interfejsów graficznych (których może być wiele). Sądzę, że dzięki takiemu podejściu niniejsza praca będzie uniwersalna i nabierze trwałego, „referencyjnego" charakteru.

System otrzymujemy na CD-ROM-ie w postaci jednej z tzw. dystrybucji. Oprócz systemu operacyjnego, dystrybucja zawiera ogromną liczbę aplikacji użytkowych, przez co tworzy kompletne środowisko pracy niemal dla każdego. Pełna dystrybucja powinna zawierać oprócz tzw. binariów, czyli gotowych do instalacji systemu i aplikacji, także zgodnie z wymogami licencji GNU) źródła, a więc teksty źródłowe systemu i aplikacji. W chwili obecnej pełna dystrybucja to już co najmniej dwa CD-ROM-y. Teksty źródłowe są potrzebne jedynie wtedy, gdy czytelnik jest zaawansowanym użytkownikiem - programistą.

Dystrybucje tworzone są przez komercyjne firmy bądź organizacje o charakterze niekomercyjnym. Najbardziej znane dystrybucje to Red Hat, Slackware, Debian, SuSE, Caldera. Dystrybucje różnią się od siebie przede wszystkim programem instalacyjnym, organizacją i rozmieszczeniem niektórych plików (np. skryptów startowych), dodatkowymi narzędziami do konfiguracji systemu (programy typu setup), ewentualnie dodawanym użytkowym oprogramowaniem komercyjnym. Łączy je to, co jest

wspólne i najważniejsze, bo dzięki licencji GNU jest dostępne wszystkim: jądro systemu oraz podstawowe oprogramowanie systemowe i użytkowe.

Najtańszym sposobem pozyskania dystrybucji jest zakup czasopisma komputerowego bądź książki z załączonym CD-ROM-em. W tym przypadku bardzo rzadko otrzymujemy (kompletne) źródła. Drugim sposobem jest zakup komercyjnej dystrybucji. Płacimy tutaj za nośniki, papierową dokumentację oraz często załączone do dystrybucji komercyjne, czyli nie objęte licencją GNU oprogramowanie.

Twórcą Linuksa, animatorem i osobą stale czuwającą nad rozwojem jądra systemu jest Linus Torvald, który w 1991 roku, jeszcze jako student informatyki na Uniwersytecie w Helsinkach stworzył (opierając się na dydaktycznym systemie Minb autorstwa profesora Adrew Tannenbauma) na własne potrzeby jądro napisane dl; procesora Intel 386 i udostępnił je społeczności internetowej. Pasjonaci z całeg' świata (przy permanentnym udziale i nadzorze Linusa Torvaldsa) zbudowali na tyr fundamencie potęgę Linuksa i dalej ją pomnażają.

Dzisiaj Linux jest solidnym (stabilnym i wydajnym) systemem operacyjny] śmiało konkurującym w wielu zastosowaniach z komercyjnymi systemami UND owymi pochodzącymi z SCO, HP i Sun. Linux jest chyba jedynym systemem, który obecnie funkcjonuje już niemal na każdej (nie tylko intelowskiej) platformie sprzęt wej. Został przeniesiony na procesory Motoroli, SPARC, stacje Silicon Graphics or przede wszystkim procesory Alpha (konstrukcja Digital Equipment Corporation, produkowana obecnie przez Samsunga). W pełni 64-bitowe procesory Alpha są bard obiecującą platformą (za względnie umiarkowane pieniądze) dla użytkownik Linuksa poszukujących ekstremalnych wydajności. Popularność i zakres zastosov Linuksa systematycznie wzrasta, tak że obecnie żaden z potentatów światu przemysłu komputerowego nie może już ignorować jego istnienia.

Do sukcesu Linuksa w przyczyniło się pięć czynników:

- sieć Internet, jednocząca twórców i użytkowników
- wspaniała idea licencji GNU
- potęga otwartego systemu operacyjnego UNIX
- skorzystanie z istniejącego oprogramowania GNU (m.in. **gcc, bash),** stwc nego na potrzeby projektu HURD realizowanego przez Free Software Foi tion
- sam Linus Torvalds.

Relatywnie wysoka jakość Linuksa i dostępnego dlań darmowego oprogramowania, skłoniła komercyjnych dostawców systemów uniksowych, w szczególności Cruz Operation (SCO Unixware) i Sun (Solaris), do udostępniania swoich systemów operacyjnych do użytku niekomercyjnego (w szczególności edukacyjnego) „z mo" lub „po cenie nośników". Prowadzone jest to w ramach różnego rodzaju marketingowych.

W wielu artykułach publikowanych w prasie komputerowej (i w Internecie) można spotkać się ze śmiałymi opiniami, że obecnie Linux jest jedynym systemem, który może podważyć absolutną dominację Microsoftu na rynku systemów operacyjnych dla komputerów osobistych oraz „małych" serwerów.

Linux odnowił i udostępnił szerokiemu światu UNIX-a oraz wprowadził go pod strzechy. Gdyby nie pojawił się i rozwinął Linux, jest wielce prawdopodobne, że UNIX byłby dzisiaj systemem operacyjnym do wąskich profesjonalnych zastosowań, którym interesowałaby się niezbyt liczna grupa elitarnych specjalistów.

Innym, mniej znanym, darmowym UNIX-em, jest FreeBSD (www. f reebsd. org). O nim jednak nie będziemy tu mówić.

2. Wprowadzenie. Przyjęte w książce konwencje zapisu

Przyjmujemy, że potencjalny czytelnik tej książki ma elementarne obycie z komputerem osobistym, zna podstawy systemu operacyjnego MS-DOS (lub ekwiwalentnego) i podstawową terminologię informatyczną (w szczególności wie, co to jest plik, katalog, ścieżka dostępu itd.). Może być natomiast nowicjuszem w zakresie Linuksa i ogólniej systemu UNIX. Przy pisaniu książki posługiwałem się dystrybucją Red Hat 5.2. Jednak elementarny charakter tematyki niniejszego opracowania, który starałem się zachować (jak i otwarty charakter Linuksa i UNIX-a), powoduje, że jest ono w dużym zakresie uniwersalne i nie zależy od dystrybucji, ani numeru jej wersji. Podstaw Linuksa możemy nauczyć się z niniejszej książki, niezależnie od tego, jaką dystrybucję i jej wersję zainstalujemy. Oczywiście, istnieją między dystrybucjami różnice dotyczące specjalizowanych narzędzi konfiguracyjnych, położenia niektórych plików konfiguracyjnych, konfiguracji skryptów startowych itd. Różnice te nie dotyczą jednak podstawowych poleceń systemowych i zasad działania systemu, na których koncentruje się tematyka niniejszego opracowania.

Zalecam jednak, aby zupełnie początkujący użytkownicy korzystali raczej z dystrybucji Red Hat (niekoniecznie 5.2). Nie jest to wymaganie kłopotliwe, gdyż Red Hat jest (przynajmniej w Polsce) chyba najpopularniejszą dystrybucją Linuksa i jego kolejne wersje są często publikowane w czasopismach komputerowych.

Natomiast po wykonaniu ćwiczeń bardzo zalecam zapoznanie się z innymi dystrybucjami, zwłaszcza Slackware i Debian. Da to nam szersze spojrzenie na świat Linuksa.

W ćwiczeniach nie korzystam ze środowiska graficznego (X-Windows), gdyż zajmuję się podstawami i będąc wierny starej zasadzie Ockhama staram się „nie mnożyć bytów ponad potrzebę". Może się jednak zdarzyć, że będziemy mieli do ćwiczeń tylko „obcą" maszynę, ze skonfigurowanym już systemem, w którym X-Windows startuje automatycznie. Wówczas mamy do dyspozycji trzy rozwiązania. Opiszę je po kolei.

Pierwsze rozwiązanie i najlepsze, to zakończenie pracy X-serwera, co spowoduje przejście do trybu znakowego. Z uwagi na to, że możemy spotkać się z różnymi menedżerami okien, nie można krótko i jednoznacznie opisać, jak to należy wykonać (najprościej chyba zapytać właściciela maszyny). Nie mniej jednak, jeżeli potrafimy zakończyć pracę MS Windows 9X, to jest wielce prawdopodobne, że szybko odkryjemy, jak „wyłączyć X-Windows". W ostateczności możemy użyć kombinacji klawiszy: **[CtrI]+[AIt]+[Backspace].**

Aby ponownie uruchomić X-Windows wydajemy polecenie:

startx

Drugie i z naszego punktu widzenia najgorsze rozwiązanie, to przeprowadzenie ćwiczeń w środowisku X-Windows, w oparciu o okno terminala **(xterm).** Jeśli nie będzie **xterm,** musi być dostępna alternatywna aplikacja typu terminal. Dotyczy to w szczególności nowych, zaawansowanych menedżerów okien, jak GNOME czy KDE. Należy pamiętać, że w systemie możemy uruchomić tylko jedną sesję X-Windows. Wprawdzie w ramach tej sesji możemy otworzyć wiele okien terminala, ale żeby w nich funkcjonować jako różni użytkownicy (co jest wymagane w niektórych ćwiczeniach), musimy korzystać ze swoistej sztuczki, wykorzystując polecenie su (najlepiej z opcją -l) i brać pod uwagę ograniczenia takiego rozwiązania, które, objawić się mogą w niezrozumiały dla nas sposób.

Trzecie rozwiązanie jest kompromisowe. Pozostawiamy uruchomioną sesję X-Windows, ale z niej nie korzystamy. Stosując odpowiednie kombinacje klawiszowe, uzyskujemy dostęp do sześciu konsoli znakowych (tekstowych) zwanych inaczej wirtualnymi terminalami (konsolami). Wszystkie ćwiczenia opisane w książce zostały przeprowadzone na konsolach znakowych.

W środowisku X-Windows (konsola graficzna) uzyskujemy dostęp do sześciu konsoli znakowych, przełączając się do nich (i między nimi) za pomocą kombinacji klawiszowych: **[lewy Ctrl]+[lewy Alt]+[F1]** do **[lewy Ctrl]+[lewj Alt]+[F6].** Z dowolnej konsoli znakowej wracamy na konsolę graficzną (X-Windows) stosując kombinację klawiszy **[lewy Ctrl]+[lewy Alt]+[F7]** albo krótszą wersję [lewy Alt]+[F7J.

W konsoli znakowej (np. po udanym przełączeniu z konsoli graficznej) możemy przełączać się do innych konsoli znakowych także „skróconymi" kombinacjam klawiszy: **[lewy Alt]+[F1]** do **[lewy Alt]+[F6].**

Korzystając ze środowiska X-Windows (czyli stosując drugie rozwiązanie), po czątkujący użytkownik dokłada sobie dodatkowych trudności, nie związanych z za sadniczym tematem ćwiczeń. Nie sprawdzałem opisanych w niniejszej książce ćwi czeń w sesji X-Windows (z **xterm** czy inną graficzną aplikacją typu „terminal") i ni mogę zagwarantować, że w każdym przypadku wszystko będzie przebiegało bez boleśnie, chociaż gwoli prawdy, nie powinno być żadnych poważnych niespodziane (uwzględniając trudności z różnymi użytkownikami, które wymieniłem wyżej).

Dla tych, którzy po zapoznaniu się z materiałem niniejszej książki zapragną pc szaleć w X-Windows, zamieszczam w dodatku E krótką informację na temat konfiguracji i uruchamiania X-Windows.

Książka ta może być pomocą dydaktyczną do pracy w grupie oraz samodzielni nauki. Dla samouków przydatny może być rozdział dotyczący instalacji systemu.

W typowych przypadkach instalacja jest prosta i bezproblemowa: program instalacyjny prowadzi użytkownika za rękę. Nie mniej jednak czasami musimy wykazać się elementarną znajomością sprzętu i funkcjonowania systemu operacyjnego.

W tekście książki przyjmiemy następujące konwencje zapisu:

- pojedynczo wciskany klawisz klawiatury zapisujemy w nawiasach kwadratowych i wyróżniamy pogrubieniem, na przykład [w] albo **[lewy Alt]**
- kombinację klawiszy wciskanych jednocześnie zapisujemy jako **[Alt]+[F1],** co oznacza, że musimy przytrzymać wciśnięty klawisz **[Alt]** i jednocześnie nacisnąć **[FI]**
- sekwencję klawiszy i elementów menu wciskanych (wybieranych) kolejno zapisujemy w listingach monitorowych jako [F9], **<Edit>,** [Esc], [Esc], co oznacza, że najpierw wciskamy klawisz [F9], potem z rozwiniętego (dostępnego) menu programu wybieramy pozycję **<Edit>,** a następnie dwukrotnie wciskamy klawisz [Esc]
- wybierane pozycje menu programu występujące samodzielnie w tekście (nie w sekwencji) podajemy bez nawiasów trójkątnych, na przykład **Edit**

  sekwencję klawiszy i poleceń wprowadzanych kolejno zapisujemy jako **[F9], :w,** [Esc], [a] - co oznacza, że najpierw wciskamy klawisz [F9], potem wprowadzamy z klawiatury dwuznakowy tekst (treść polecenia) :w (ale bez wciskania klawisza **[Enter]),** a następnie wciskamy klawisz [Esc], a dopiero później klawisz [a]
- nazwy użytkowników, katalogów i plików wymienianych w tekście podajemy *italikiem,* na przykład *letclpasswd, leszekl*
- nazwy systemów plików i adresy internetowe oraz niektóre inne obiekty, a także komunikaty wyróżniamy czcionką Couri er, na przykład ext2, www. 1 i nux. org

  piszemy root, mając na myśli katalog główny bądź początek głównego systemu plików (bezwzględnej ścieżki dostępu), natomiast mając na myśli administratora (który jest przecież użytkownikiem systemu) piszemy *root*

  nazwy poleceń i programów wykonywalnych oraz skryptów wymienianych w tekście wyróżniamy pogrubieniem, na przykład ls

  listingi monitorowe zaznaczamy czcionką Couri er

  w listingach monitorowych **pogrubieniem** wyróżniamy polecenia lub dowolne ciągi znaków, **które wprowadza użytkownik** (operator), pozostałe teksty (wyświetlane samoczynnie przez system) zwykłym drukiem. Zasadę „pogrubienia" tych obiektów stosujemy także w tekście komentarza, o ile nie koliduje z przejrzystością tekstu i zdrowym rozsądkiem. Na przykład podanie nazwy użytkownika przy logowaniu do systemu będzie wyglądało tak:

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: leszek
```

- pogrubieniem wyróżniamy nazwy menu, okien itp.

Ponieważ w przeciwieństwie do DOS-a Linux (UNIX) rozróżnia małe i duże litery (a większość programów także różnicuje kody małych i dużych liter), dlatego w wielu sytuacjach nie będzie obojętne, czy wciskając konkretny klawisz wprowadzamy kod małej, czy dużej litery. Przykładowo zapisy [a] i [A] oznaczają fizycznie ten sam klawisz na klawiaturze, ale odpowiadają różnym kodom ASCII. Zapis [a] oznacza (kod ASCII 97) wciśnięcie klawisza, gdy wyłączony jest (wygaszona lampka) **Caps Lock,** natomiast zapis [A] oznacza (kod ASCII 65) wciśnięcie klawisza, gdy włączony jest (zapalona lampka) **Caps Lock.** W szczególności przy wyłączonym **Caps Lock** zapis [A] jest równoważny zapisowi [Shift]+[a],

Na ogół kwestia ta będzie istotna przy naciskaniu pojedynczych klawiszy w konkretnych programach (np. vi). Typowe kombinacje dwuklawiszowe z klawiszem **[Ctrl]** powinny działać niezależnie od ustawienia **Caps Lock,** czyli przykładowo kombinację przerywającą polecenie możemy zgodnie z przyjętą konwencją zapisać jako [Ctrl]+[c] lub [Ctrl]+[C] albo nawet jako [Ctrl]+[Shift]+[c]. Ponieważ w Linuksie posługujemy się głównie małymi literami, czyli pracujemy z wyłączonym **Caps Lock,** preferować będziemy zapis w postaci [Ctrl]+[c].

Większość ćwiczeń dotyczy najróżniejszych poleceń systemu Linux. Polecenie wprowadzamy z klawiatury (podobnie jak w DOS-ie) podając jego nazwę, następnie opcjonalnie poprzedzone najczęściej znakiem minus opcje i ewentualnie argument, którego ono dotyczy. Separatorem poszczególnych składników (nazwy, opcji i argumentu) polecenia jest spacja. Całość kończymy naciśnięciem klawisza **[Enter].** Na przykład:

```
shutdown -r now [Enter]

ls CEnter]
```

To istotne naciśnięcie [Enter], będziemy pomijać (dla większej czytelności), dlatego w dalszej części książki wprowadzenie powyższych poleceń zapiszemy tak:

```
shutdown -r now

ls
```

Najczęściej polecenia będziemy zapisywać wraz ze znakiem gotowości systemu („#" albo „$")• Znak gotowości nie jest wprowadzany przez operatora, lecz automatycznie wypisywany na ekranie monitora przez system operacyjny (a ścisłej biorąc przez tzw. powłokę). Nie należy go zatem wprowadzać z klawiatury. Wprowadzamy tylko treść polecenia. Podanie w listingu znaku gotowości ma uświadomić ćwiczą-

cemu, czy dane polecenie wykonuje zalogowany do systemu jako *root* (superużytkownik - administrator), czy też jako inny (zwykły) użytkownik. Istnieje bowiem konsekwentnie przestrzegana w systemach uniksowych konwencja, w której pojawienie się znaku gotowości „#" (czytaj: hasz) oznacza, że jesteśmy zalogowani jako *root*. Natomiast znak dolara „$" (lub gdy stosujemy powłokę **tcsh** znak procent „%") odpowiada każdemu innemu użytkownikowi systemu.

Zatem jeśli ćwiczący powinien wykonać określone polecenia w trybie użytkownika *root',* wówczas ich zapis będzie wyglądał następująco:

```
# shutdown   -r   now

# ls
```

Jeśli natomiast polecenia te mają (mogą) być wykonane w trybie zwykłego użytkownika, wówczas znak gotowości będzie inny:

```
$ shutdown   -r   now

$ ls
```

W większości przypadków listing monitorowy składa się więcej niż z jednego wiersza. W takim listingu zawsze będzie widoczne pełne zgłoszenie gotowości systemu (np. [root@mono /]# _), zwane też znacznikiem systemowym (ang. *prompt).* W pierwszym wierszu (dla większej czytelności) pełne zgłoszenie systemowe najczęściej będzie pominięte. Wystąpi tu tylko właściwy w danym przypadku znak gotowo- „#" albo „$"). Przykładowo:

```
# useradd   leszekl
[root@mono /root]#  useradd   leszek2
```

Polecenia systemowe mają składniki (parametry) oddzielone od siebie pojedynmi spacjami. W listingach (dla poprawy czytelności) odstęp ten jest często większy niż szerokość jednego znaku. Nie należy się tym sugerować (jedna spacja w zupełnowystarczy). Przykład:

```
# mount   -t   ext2    /dev/fdO    /mnt/floppy
```

Wspomniane już listingi monitorowe są kluczowym elementem książki. Nie zostały tak po prostu „żywcem zdjęte" z ekranu monitora, lecz poddano je wielu świadomym zabiegom:

- zmieniona została czcionka, przez co zmieniły się „wyrównania i proporcje", które mogą być inne niż w oryginale dostępnym na monitorze
- wytłuszczone zostały polecenia (teksty) wprowadzane przez operatora (czyli ćwiczącego)
- dla poprawy czytelności wstawione są w niektórych miejscach dodatkowe spacje

- w ćwiczeniach, których istotą są sekwencje klawiszowe, sekwencje te są wpisane do treści listingu w sposób zgodny z podanymi wyżej konwencjami,
- bardzo długie listingi zostały często skrócone przez usunięcie fragmentów (będących wielokrotnością całych wierszy) i zastąpienie ich wykropkowaniem. Fakt dokonania takiego skrótu (jeśli nie jest on kontekstowo oczywisty) jest odnotowany w komentarzu do danego listingu.

Zakres niniejszego kursu podzielony jest na 15 lekcji, tworzących całości tematyczne. Lekcja jest jednostką logiczną, a nie dydaktyczną. Jednak kolejność lekcji ma charakter dydaktyczny, tzn. w danej lekcji możemy korzystać z umiejętności nabytych w poprzednich lekcjach. Lekcje składają się z ćwiczeń numerowanych kolejno. Tak więc ćwiczenie 11.4 jest czwartym ćwiczeniem jedenastej lekcji.

Opanowanie podstaw obsługi Linuksa, jest w istocie opanowaniem podstaw obsługi dowolnego systemu uniksowego. Rzecz jest więc warta zachodu.

3. Podstawowe informacje o systemie

Linux jest systemem wielodostępnym i wielozadaniowym. Oznacza to, że może z niego korzystać jednocześnie wielu użytkowników, a każdy z nich może uruchamiać wiele własnych programów. System identyfikuje użytkownika na podstawie niepowtarzalnego numeru UID (user identifier). Użytkownik nie musi znać swojego UID, ani nawet wiedzieć o istnieniu takiego mechanizmu identyfikacji. Aby korzystać z Linuksa musi znać swoją nazwę (inaczej nazwę konta użytkownika) oraz skojarzone z nią hasło umożliwiające zalogowanie się do systemu. Każdy z użytkowników będzie dysponował na dysku (w systemie plików) „prywatnym" katalogiem zwanym katalogiem domowym użytkownika. Katalogi domowe są zwyczajowo umieszczone jako podkatalogi katalogu *lhome,* a ich nazwy odpowiadają nazwom (kont) użytkowników. Nie dotyczy to administratora (superużytkownika o z góry zadanej nazwie *root),* którego katalogiem domowym jest */ root.*

Aby rozpocząć pracę, użytkownik musi się zalogować do systemu podając swoją nazwę, a następnie przyporządkowane jej hasło (czasami nie jest ono wymagane). W czasie instalacji systemu tworzone jest automatycznie konto użytkownika o nazwie *root* (identyfikatorze UID=0). Jest to wyjątkowy użytkownik mający uprawnienia do wszystkich zasobów systemu, dlatego jego hasło powinno być pilnie strzeżone i często zmieniane. *Root* (administrator systemu) zakłada następnie konta pozostałych użytkowników podając nazwę i hasło (które użytkownik może później samodzielnie zmienić na inne), tworząc ich katalogi domowe i co najważniejsze może nadawać im zróżnicowane uprawnienia do konkretnych zasobów systemu głównie poprzez tworzenie grup użytkowników i manipulowanie prawami własności i dostępu do plików oraz katalogów.

W przypadku pojedynczego komputera mamy do dyspozycji sześć konsoli znakowych (wirtualnych terminali), między którymi możemy przełączać się za pomocą kombinacji klawiszy [lewy Alt]+[F1] do [lewy Alt]+[F6]. Gdy w systemie uruchomiona jest sesja X-Windows („siódma konsola" - graficzna), będziemy musieli skorzystać z dłuższych kombinacji klawiszowych do przełączania się z konsoli graficznej do wybranej konsoli znakowej: [lewy Ctrl]+[lewy AIt]+[Fl] do [lewy Ctrl]+[lewy Alt]+[F6]. Powracamy na konsolę graficzną przez [lewy Ctrl]+[Iewy Alt]+[F7].

W książce nie korzystamy z X-Windows, zatem będziemy zawsze korzystać tylko z krótszych kombinacji klawiszowych.

Na każdej z konsoli znakowych możemy zalogować się do systemu (jako ten sam lub inny użytkownik) otwierając w ten sposób sesję użytkownika. Podczas każdej z sesji możemy uruchamiać wiele zadań jednocześnie (część z nich w tle). Prawidłowe zakończenie sesji zwykłego użytkownika polega na wylogowaniu się z systemu **(logout).** Po zakończeniu pracy z Linuksem nie możemy tak po prostu wyłączyć zasilania komputera. Wcześniej należy zamknąć system operacyjny. Wykonujemy to poleceniem **shutdown** z odpowiednimi opcjami. Uprawnienia do wykonania tego polecenia posiada administrator *(root).* Znana z DOS-a kombinacja klawiszy **[Ctrl]+ [Alt]+[Del]** powodująca tzw. „gorący" restart komputera, tutaj także działa, z tym że w sposób bardziej cywilizowany: po wydaniu „trójpalcowego salutu" **[Ctrl]+[Alt]+ [Del]** system wykona bezzwłoczny **shutdown,** po czym zresetuje komputer.

Struktura systemu plików Linuksa (UNIX-a) ma charakter drzewa katalogów, tak jak w DOS-ie. Oczywiście, to DOS jest wzorowany na UNIX-ie, a nie na odwrót. Katalogiem głównym jest root (zbieżność nazwy z superużytkownikiem) oznaczany ukośnikiem (ang. *slash)* „/". Pełna, bezwzględna, ścieżka dostępu do pliku będzie zaczynała się od ukośnika, na przykład *lhomelleszekl I dane.*

System plików Linuksa tworzy jedno spójne, nierozdzielne drzewo. Nie znajdziemy tu fizycznych urządzeń znanych z DOS-a, takich jak dysk twardy, a ściślej jedna z jego partycji (C:\), napęd dyskietek (A:\), czytnik CD-ROM (D:\) itd. Aby skorzystać z czytnika czy stacji dyskietek, musimy je zamontować do drzewa systemu plików, przypisując je do już istniejącego, wybranego przez nas katalogu (jest to tzw. punkt montowania). Ściślej rzecz ujmując, montujemy nie puste urządzenie, ale konkretny, fizyczny nośnik (CD-ROM, dyskietkę). Polecenie zamontowania pustego napędu system potraktuje jako błąd i nie zrealizuje go. Przed wyjęciem CD-ROM-u czy dyskietki musimy je zdemontować. W przypadku czytnika CD-ROM nie jest wręcz możliwe wyjęcie płytki bez jej zdemontowania, gdyż po zrealizowaniu polecenia **mount** czytnik zostaje zablokowany i nie będzie reagował na naciskanie przycisku i wysuwu tacki. Powinniśmy o tym pamiętać.

Istotną różnicą między DOS-em i Linuksem jest fakt, że w przeciwieństwie do DOS-a, Linux rozróżnia małe i duże litery. Tak więc w Linuksie pliki: *Moje_Dane* i *moje_dane* są dwoma różnymi plikami.

Przedstawione niżej katalogi znajdujące się w katalogu głównym (/) odpowiadają dystrybucji Red Hat 5.2. Jednakże z uwagi na daleko posuniętą standaryzację będą występować w każdym innym systemie Linuksowym i pełnić te same funkcje. Niżej przedstawiamy co katalogi owe zawierają. W tej chwili nie wszystko w opisie będzie zrozumiałe, ale gdy zdobędziemy nieco wiedzy i umiejętności, możemy wrócić tu raz jeszcze:

Ibin - (binaries) - pliki wykonywalne (w większości binarne), które mogą być wykonywane przez zwykłych użytkowników

Iboot - (bootstrap) - pliki używane przez program ładujący system (zanim jądro nie przystąpi do wykonania **init).** Znajduje się tu także obraz jądra, w na-

szym przypadku jest to **vmlinuz-2.0.36-0.7.** W innych dystrybucjach może on być umieszczony bezpośrednio w katalogu głównym: /

ldev - (devices) - pliki specjalne (reprezentujące urządzenia)

/etc - pliki konfiguracyjne, także tzw. skrypty startowe w katalogu *letclrc.d*

lhome - katalogi domowe zwykłych użytkowników

ilib - (library) - biblioteki wymagane przez programy znajdujące się w *Ibin,* a w katalogu *Iliblmodulesl2.036-0.7*także moduły jądra

llost+found - katalog tworzony automatycznie przy zakładaniu systemu plików ext2 (podstawowego w Linuksie). Jest to kosz dla poleceń **fsck** i **e2fsck,** sprawdzających i naprawiających system plików. Do tego katalogu trafiają pliki czy też ich fragmenty, które straciły dowiązania do swoich i-węzłów

mnt - **(mount)** - standardowy w dystrybucji Red Hat katalog (punkt) montowania urządzeń. Zawiera dwa podkatalogi:

Imntlcdrom - do zamontowania płytki CD-ROM

Imntlfloppy - do zamontowania dyskietki

proc - zawiera wirtualne pliki (tzn. nie istniejące fizycznie na dysku) prezentujące informacje techniczne dotyczące konfiguracji systemu, parametrów jądra i uruchomionych procesów

root - katalog domowy użytkownika *root* (administratora)

sbin - (superuser binaries) - generalnie zawiera pliki wykonywalne (w większości binarne) potrzebne do działań związanych z administracją systemu. Do większości pełne prawo wykonywania ma jedynie administrator *(root)*

tmp - (temporary) - pliki tymczasowe

lusr - (user) - generalnie tutaj zainstalowane są wszystkie aplikacje użytkowe, wymagane przez nie biblioteki i dokumentacja, a w katalogu *lusrlsbin* także wiele programów narzędziowych

Nar - (variable) - zawiera pliki, które często ulegają modyfikacji; zwłaszcza zmienia się ich rozmiar.

Podstawowym elementem systemu jest jądro. Ponieważ jest ono w permanentnym procesie rozwoju, dlatego stosuje się określoną konwencję numerowania kolejnych wersji jądra. Przykładowo dystrybucja Red Hat 5.2 jest oparta na jądrze 2.0.36. Jest to wersja stabilna (druga liczba w oznaczeniu parzysta) nr 36 jądra 2.0. Aktualna' w momencie pisania tekstu wersja rozwojowa (niestabilna) jądra ma oznaczenie 2.1.x (druga liczba jest nieparzysta). Dystrybucje Linuksa opierają się wyłącznie na wersjach stabilnych jądra. Zatem kolejna wersja Red Hata zawierać będzie jeszcze jądro 2.0.x albo już nową generację 2.2.x.

Jądro bezpośrednio współpracuje ze sprzętem z jednej strony, a z drugiej obsługuje poprzez dobrze zdefiniowany interfejs programy użytkowników, wśród których

są między innymi interpretery poleceń systemowych zwane powłokami (ang. *shell)*. Podstawową domyślną powłoką używaną w Linuksie jest **bash** (Bourne Again Shell) opracowany przez Free Software Foundation. Użytkownik może jednak używać innego interpretera. Oprócz **bash-a** istnieją jeszcze trzy inne podstawowe i tradycyjne już powłoki uniksowe (w nawiasie tłustym drukiem podajemy ich odpowiedniki linuksowe):

1. sh - Bourne shell, opracowany przez S.R. Bourne'a **(bash)**
2. ksh - Kom shell, opracowany przez Davida Korna **(pdksh)**
3. csh - C shell, opracowany przez BillaJoy'a **(tcsh).**

Znawcy uważają, że **bash** łączy w sobie zalety powłok Korna i C. Jest w pełni kompatybilny z najstarszą powłoką Bourne'a. Wszystkie ćwiczenia w niniejszej książce zostały wykonane właśnie z jego zastosowaniem. Linux udostępnia ponadto rozszerzoną wersję C shella - **tcsh.** Bourne shell (sh) jest także „oficjalnie dostępny" w naszym systemie linuksowym, ale jak sami niebawem się przekonamy, wprost jako tzw. łącznik symboliczny do bash-a. Korn shell jest obecny w Linuksie w postaci reprezentowanej przez Public Domain Korn Shell **(pdksh).**

Program użytkownika nie ma bezpośredniego kontaktu ze sprzętem. Nie są tu możliwe (jak w DOS-ie) „bezpośrednie odwołania" do sprzętu (BIOS-u), z pominięciem jądra.

Obsługa każdego występującego w komputerze urządzenia (ang. *device)* musi być zaimplementowana w jądrze. Optymalnym rozwiązaniem byłoby zatem skonfigurowanie i skompilowanie indywidualnego jądra do konkretnej konfiguracji sprzętowej komputera. W praktyce komplikuje to jednak proces instalacji i może wymagać od użytkownika odpowiednich kwalifikacji. Dystrybutorzy Linuksa (tworząc programy instalacyjne) wybrnęli z tego problemu stosując z reguły jednocześnie dwie metody. Pierwsza to zestaw dedykowanych do konkretnych konfiguracji sprzętowych, ale w miarę uniwersalnych jąder instalacyjnych wybieranych przez użytkownika lub automatycznie przez program instalacyjny. Druga metoda polega na skorzystaniu z modularnej budowy jądra. Moduły są to fragmenty jądra (odpowiedzialne za obsługę np. kart sieciowych) dołączane w miarę potrzeby do zasadniczej części jądra w czasie ładowania systemu. Mogą być także ładowane i usuwane już w uruchomionym systemie. Jądro dystrybucyjne obsługuje z reguły większą liczbę urządzeń niż występuje w naszym komputerze, przez co jest zbyt duże i siłą rzeczy nie zawsze może być zoptymalizowane pod względem wydajności do konkretnego typu procesora (np. Pentium czy Pentium II), jaki akurat posiadamy. Jeżeli chcemy uzyskać maksymalną wydajność systemu, to indywidualne skonfigurowanie i skompilowanie jądra jest często uzasadnione.

Jądro obsługuje wielozadaniowość, działając w systemie podziału czasu procesora. Każdy proces (program wykonywany w pamięci) wykorzystuje ściśle określony krótki odcinek czasu pracy procesora zwany też często kwantem, po czym jest wy-

właszczany przez inny proces, a ten z kolei po upływie kwantu jest wywłaszczany przez kolejny proces itd. Proces do jego całkowitego wykonania może wymagać wielu kwantów czasu procesora. W ten sposób, mimo istnienia najczęściej tylko jednego procesora w systemie, może on wykonywać wiele programów „jednocześnie".

Sam proces może być procesem użytkownika lub systemowym. W skład procesu wchodzą: wykonywalny program, związane z nim dane oraz kontekst wykonania. Na kontekst wykonania składają się: kontekst procesora (zawartość licznika rozkazów, -;aźnika stosu, rejestru stanu oraz ewentualnie innych używanych rejestrów procesora), identyfikator procesu (tzw. PID - process identifier, jest to liczba całkowita), poziom priorytetu i stan procesu. Użytkownik (identyfikowany przez UID), który uruchomił program, staje się właścicielem procesu.

Proces, który wymknął się spod kontroli (zawiesił się), nie jest groźny dla funkcjonowania systemu, tzn. nie powoduje jego zawieszenia, co najwyżej zablokowana może zostać konkretna konsola wirtualna, lecz inne uruchomione w systemie procesy wykonywane będą nadal. Zablokowaną konsolę odblokowuje się przez zalogowanie z innej konsoli i bezwarunkowe zakończenie wiszącego procesu poleceniem kill. Ta cecha Linuksa (UNIX-a), wynikająca z jego autentycznej wielozadaniowości, daje mu wyraźną przewagę nie tylko nad prostym DOS-em, ale także nad innymi popularnymi emami operacyjnymi, szumnie określanymi jako wielozadaniowe.

Jak już wspomnieliśmy, początkiem (katalogiem głównym) drzewa systemu plików Linuksa jest root oznaczany pojedynczym ukośnikiem: /. Powtórzmy raz jeszże pełna ścieżka dostępu do pliku zawsze będzie zaczynała się od takiego ukośnika (czyli root-a), na przykład *ldevlhdc* podaje pełną (bezwzględną) ścieżkę dostępu do pliku specjalnego *hdc* („sterownika") reprezentującego urządzenie dyskowe (dysk twardy, a w tym miejscu najczęściej czytnik CD-ROM) Master w drugim kanale IDE. Zwróćmy uwagę, że w Linuksie używamy ukośnika (ang. *slash),* a nie odwrotnego ukośnika jak w DOS-ie (ang. *backslash).*

Względna (podana względem bieżącego katalogu) ścieżka dostępu nie może mieć na początku ukośnika „/".

Mówi się, że Linux (UNIX) operuje wyłącznie na plikach, czyli traktuje wszystko jak plik. Wszystkie pliki w Linuksie mają ten sam format fizyczny - ciąg bajtów. Katalog jest plikiem. Urządzenie (ang. *device)* jest również traktowane jako plik. Zapis do urządzenia peryferyjnego jest pisaniem do pliku specjalnego reprezentującego c urządzenie. Mówienie o typach plików w Linuksie (UNIX-ie) jest klasyfikowaniem **ich** według funkcji i przeznaczenia, ponieważ format fizyczny (ciąg bajtów) mają ten **sam.** Powszechnie rozróżnia się co najmniej cztery typy plików:

1. Pliki zwykłe
1 Katalogi (ang. *directory)*

Pliki specjalne (reprezentujące urządzenia) znakowe (ang. *character device)*

Pliki specjalne (reprezentujące urządzenia) blokowe (ang. *błock device).*

Jako cztery dodatkowe typy plików wymienia się często:

5. Gniazda domeny UNIX
6. Łącza nazwane (FIFO)
7. Dowiązania (łączniki) sztywne
8. Dowiązania (łączniki) symboliczne.

Jest kwestią akademicką rozstrzyganie, czy przykładowo dowiązanie sztywne możemy zakwalifikować jako plik, czy nie. Chcemy tu jedynie pokazać sytuację, z jaką możemy się spotkać w literaturze przedmiotu. W niniejszej książce poznamy dowiązania sztywne i symboliczne, natomiast gniazda i łącza nazwane mieszczą się w kategorii bardziej zaawansowanych rozważań.

Wśród plików zwykłych możemy wyróżnić cztery duże grupy:

1. Pliki tekstowe (ang. *ASCII text)*
2. Pliki wykonywalne binarne (ang. *executable)* - programy
3. Pliki wykonywalne tekstowe (ang. *command)* - skrypty powłoki
4. Pliki z danymi aplikacji (ang. *data)*.

Podział plików specjalnych na znakowe i blokowe ma swój początek w zamierzchłych czasach informatyki, gdy istniało ostre rozgraniczenie na urządzenia znakowe i blokowe, wynikające ze sposobu transmisji danych do i z tych urządzeń. Urządzenia znakowe (wolne) generowały przerwania po każdym przesyłanym znaku, kodowanym często tylko sześcioma bitami, obciążając sterowaniem transmisją procesor komputera. Typowe urządzenia znakowe to drukarka wierszowa czy szeregowy terminal znakowy. Urządzenia szybkie (blokowe) umożliwiały przesyłanie dłuższych porcji, czyli bloków danych za jednokrotnym wygenerowaniem przerwania. Rozmiar bloku zależny był od konkretnych rozwiązań sprzętowych. Urządzenia blokowe często wykorzystywały mechanizmy bezpośredniego dostępu do pamięci (DMA). Typowe urządzenia blokowe to dyski i napędy taśmowe. W komputerze osobistym urządzeniami blokowymi są dysk twardy, czytnik CD-ROM i napęd dyskietek.

4. Instalacja dystrybucji Red Hat

Wymagania sprzętowe Linuksa są raczej skromne. Wystarczy dowolny komputer 2 procesorem co najmniej Intel 386 (lub wyższym: 486, Pentium, Pentium Pro, Celeron, Pentium II itd.) i 8 MB pamięci RAM. Potrzebujemy jeszcze oczywiście napędu dyskietek 3,5 cala, czytnika CD-ROM oraz przynajmniej 300 MB wolnego miejsca na dysku twardym. Ponieważ kurs niniejszej książki obejmuje tylko pracę w trybie znakowym, wystarczy dowolna (najtańsza) karta graficzna VGA i monitor monochromatyczny.

Osobiście preferuję procesory Intela i płyty główne z chipsetami Intela, generalnie jednak Linux będzie pracował z innymi procesorami, będącymi odpowiednikami wyroobów Intela. W szczególności będą to procesory firm AMD (np. serii K5 i K6), Cyrix (np. serii 686MX i M2) i IDT (WinChip). Także płyty główne z chipsetami producentów (np. VIA i ALI) powinny pracować bez zarzutu. Zakładamy, że dysk twardy jest typu IDE (EIDE). Oczywiście kontrolery SCSI są obsługiwane, ale konfiguracja wykracza poza zakres niniejszej pracy. Przyjmujemy, że czytnik CD-ROM jest również typu IDE (EIDE). Najstarsze czytniki (jednokrotne i niektóre dwudwukrotnie wykonywane były z tzw. interfejsami firmowymi. Są one obsługiwane przez ile w niektórych konfiguracjach sprzętowych program instalacyjny może automatycznie nie rozpoznać czytnika i wówczas należy podać z konsoli parametry deficzytnik, co początkującemu (i niezbyt biegłemu w subtelnościach sprzętoużytkownikowi sprawić może pewien kłopot.

Do komfortowej pracy dobrze jest mieć komputer wyposażony w co najmniej Pentium 75 i 16 MB RAM. W systemach uniksowych ilość pamięci jest kluczowa, chodzi o wydajność. Jeżeli zatem konfigurujemy komputer specjalnie dla Linuksie kupujmy silniejszego procesora kosztem ilości pamięci RAM, wręcz przeciwnie 32 czy 64 MB RAM to wcale nie za dużo, zwłaszcza gdy po opanowaniu podstaw chcemy popracować w X-Windows. Jeżeli zamierzamy w przyszłości poważnie pracować z Linuksem, to nawet duży (kilka, ... kilkanaście GB) dysk twardy nie będzie przesadą. Jeżeli instalacji dokonujemy w pracowni szkoleniowej, w której komputery nie mają czytników CD-ROM, ale za to spięte są siecią Ethernet, wówczas wystarczy jedna maszyna z czytnikiem, którą wykorzystamy jako serwer, a na pozostałych zainstalujemy system korzystając z dostępu do tego czytnika poprzez NFS Network File System - sieciowy system plików). W dystrybucji Red Hat wykonać to możzemy niemal automatycznie. Taka instalacja opisana jest w ćwiczeniu 9.5.

Linux koegzystować może na jednym dysku z innymi systemami operacyjnymi, z których każdy oczywiście będzie posiadał własną partycję. Możliwa jest też instala-

cja Linuksa na dosowym systemie plików (powstanie wówczas dosowy katalog LINUX), w oparciu o specyficzny system plików umsdos. Jest ona szczególnie prosta w dystrybucji Slackware. Nie jest to rozwiązanie optymalne, jeśli chodzi o wydajność, ale może być atrakcyjne do celów szkoleniowych i zapoznania się z systemem, gdyż nie wymaga rekonfiguracji dysku twardego z już zainstalowanym DOS-em.

Należy podkreślić, że Linux obsługuje praktycznie wszystkie istniejące obecnie systemy plików, zatem w konfiguracjach wielosystemowych pracując w Linuksie będziemy mieli dostęp także do partycji dyskowych wykorzystywanych przez inne systemy.

Na płytce dystrybucyjnej CD-ROM (najczęściej w katalogu *doclHowto)* znajdziemy pliki (z reguły tekstowe) *HOWTO* będące podstawowym źródłem wiedzy dotyczącym instalacji, konfiguracji systemu i rozwiązywania konkretnych problemów instalacyjno-konfiguracyjnych. Na początek zapoznajmy się z zawartością plików *Installation-HOWTO, Hardware-HOWTO* oraz ewentualnie *UMSDOS-HOWTO.*

Większość obecnie dostępnych dystrybucyjnych płyt CD-ROM jest bootowalna, Pozwala to na instalację systemu bezpośrednio z płytki (bez potrzeby wykonywania jednej lub kilku dyskietek instalacyjnych) pod warunkiem, że płyta główna komputera ma opcję startu (ang. *boot)* z CD-ROM-u.

Opisana poniżej procedura instalacyjna dotyczy prostej sytuacji, w której cały komputer (dysk twardy) dedykowany jest dla Linuksa. Dokonana zostanie na komputerze o następującej konfiguracji sprzętowej:

- procesor Pentium 133
- płyta główna z chipsetem VX Intela, 256 kB cache
- napęd dyskietek 3,5 cala
- czytnik CD-ROM NEC IDE 6 x (urządzenie Master w drugim kanale IDE)
- pamięć operacyjna 32 MB EDO RAM
- dysk twardy 540 MB Caviar (urządzenie Master w pierwszym kanale IDE)
- karta graficzna PCI Diamond Stealth II S220 (procesor Rendition Verite 2100) 4 MB SGRAM
- karta sieciowa PCI D-Link DE-5 30CT+ (DEC 21041 chip)
- monitor standard VGA (640x480), monochromatyczny
- mysz szeregowa, trzyprzyciskowa, podłączona do portu COM1.

Powyższa konfiguracja sprzętowa może coś sugerować: jest to komputer, który służył do przygotowania tej książki. Ma wystarczające zasoby do w miarę swobodnej pracy z X-Windows. Ale chcemy jeszcze raz podkreślić, że do przeprowadzenia ćwiczeń opisanych w niniejszej książce (praca w trybie znakowym) w zupełności wystarczy skromniejsza maszyna.

4.1. Instalacja

Wykorzystamy typowy bootowalny CD-ROM z dystrybucją Red Hat 5.2 załą-:zony do czasopisma komputerowego. Najpierw ustawiamy w programie konfiguracyjnym (ang. *setup)* płyty głównej komputera czytnik CD-ROM jako pierwsze urzą-nie startowe (ang. *boot device).* Jeżeli płyta główna wyposażona jest (tak jak w naszym przypadku) w BIOS Award, to po naciśnięciu [Del] w czasie startu kompu-a przechodzimy do BIOS FEATURES SETUP, a tu ustawiamy klawiszami PageUp], [PageDn] parametr Boot Sequence jako: „CDROM,C,A". Teraz sekwencją klawiszy: [Esc], [F10], [y], [Enter] zapisujemy nową zawartość konfiguracji płyty głównej do pamięci nieulotnej, powodując jednocześnie restart komputera i jego próbę uruchomienia się z CD-ROM-u.

Jeżeli w programie konfiguracyjnym płyty głównej (ang. *setup)* nie uda się nam znaleźć sekwencji startowej (ang. *boot sequence),* w której występuje CD-ROM, oznacza to, że płyta nie obsługuje tej funkcji. W przypadku niezbyt starych płyt renomowanych producentów z reguły będzie dostępna nowsza wersja BIOS-u udostępniająca start z CD-ROM-u. Brak tej funkcji nie jest jednak dla nas przeszkodą w zainstalowaniu Linuksa. Musimy w tym przypadku wykonać startową dyskietkę instalacyjną (ang. *bootdisk).*

Do wykonania dyskietki potrzebny będzie dowolny komputer z zainstalowanym ternem DOS (Windows) wyposażony w czytnik CD-ROM-u (D:). W katalogu D:\DOSUTILS CD-ROM-u dystrybucyjnego znajdziemy program rawrite.exe. Służy on do wykonania dyskietek instalacyjnych w oparciu o ich obrazy umieszczone w katalogu DMMAGES. Tworzymy dowolny roboczy katalog i kopiujemy do niego rawrite.exe oraz obraz *boot.img.* Przygotowujemy czystą sformatowaną dyskietkę -4 MB, wkładamy do napędu (A:) i z poziomu naszego katalogu roboczego wydajemy polecenie:

```
rawrlte -d  A:  -f  boot.img  -n
```

Oczywiście, aby uruchomić komputer z dyskietki, należy w programie konfiguracyjnym (ang. *setup)* płyty głównej komputera ustawić (analogicznie jak opisano to wyżej) napęd A: jako pierwsze urządzenie startowe.

Wkładamy dystrybucyjny CD-ROM do czytnika (albo dyskietkę „boot" do napędu dyskietek, jeżeli nie możemy skorzystać ze startu z CD-ROM-u) i resetujemy komputer. Po pojawieniu się znaku gotowości:

boot:

wciskamy [Enter]. Musimy poczekać, aż pojawi się semigraficzne okno powitalne z napisem „Welcome to Red Hat Linux!". Naciskamy [Enter] i dalej będziemy już prowadzeni przez program instalacyjny, który kolejno będzie wyświetlał na ekranie semigraficzne okna. W każdym z nich musimy dokonać wyboru bądź wprowadzić dane z klawiatury. Pojawią się też okna zawierające jedynie określony komunikat.

Między polami w oknie programu instalacyjnego poruszamy się za pomocą klawisza **[Tab]** bądź klawiszy kursora. Przejścia do następnego okna (z zatwierdzeniem dokonanych wyborów dokonujemy przez podświetlenie (naprowadzenie kursora na pole) **OK** i naciśnięcie **[Enter].** Rezygnacja następuje przez podświetlenie **Cancel** i naciśnięcie **[Enter].** Podświetlenie i następujące po nim naciśnięcie **[Enter]** nazywać będziemy dalej wybraniem. Teraz wykonamy pierwsze kroki instalacji:

- wybieramy język dystrybucji **(Choose a Language): English**
- wybieramy typ klawiatury **(Keyboard Type): us**
- wybieramy źródło instalacji **(Installation Method): Local CDROM**
- wkładamy (o ile jej tam jeszcze nie ma) dystrybucyjny CD-ROM do czytnika **(Insert your Red Hat CD into your CD drive now)** i zatwierdzamy za pomocą **OK**
- system spróbuje zamontować CD-ROM i jeśli to się powiedzie, uzyskujemy planszę **Installation Path.** Wybieramy: **Install.**

- W tej chwili mamy już do dyspozycji cztery konsole wirtualne, na których śledzić możemy przebieg instalacji. Przełączamy się pomiędzy nimi sekwencjami klawiszy **[lewy Alt]+[F1]** do **[lewy Alt]+[F4].** Pierwsza konsola **([lewy Alt]+[F1])** to nasz program instalacyjny. Konsola druga ([lewy Alt]+[F2]) udostępnia powłokę **bash** w trybie użytkownika *root,* co umożliwia zaawansowanemu użytkownikowi ingerowanie w proces instalacji. Konsole **[lewy Alt]+[F3]** i **[lewy Alt]+[F4]** zawierają komunikaty generowane w czasie instalacji. Mogą być pomocne w sytuacji, gdy pojawią się problemy w przebiegu procedury instalacyjnej.

Następnie wybieramy rodzaj instalacji **(Installation Class): Custom, a** na pytanie czy mamy kontroler SCSI **(SCSI Configuration)** odpowiadamy przecząco: No.

Teraz program instalacyjny wprowadza nas do ważnego etapu instalacji, czyli wykonania partycji na dysku twardym. Należy pamiętać, że repartycjonowanie dysku spowoduje utratę znajdujących się na nim danych. W naszym przykładzie przyjmujemy, że mamy do dyspozycji cały (540 MB) dysk oraz że wszystkie dane, które do tej pory zostały na nim zapisane, możemy stracić.

Mamy do wyboru **(Disk Setup)** dwa programy partycjonujące: **Disk Druid** (nowszy) i **fdisk** (starszy). Należy pamiętać, że wymieniony tu **fdisk,** nie jest tożsamy z powszechnie znanym dosowym programem o tej samej nazwie. Wybieramy: **Disk Druid.**

Sposób partycjonowania dysku w Linuksie może być sam w sobie tematem obszernych rozważań. W naszym przypadku zastosujemy najprostsze rozwiązanie, co nie znaczy najbardziej eleganckie bądź optymalne, tzn. utworzymy jedną partycję

podstawową i jedną partycję wymiany (ang. *swap).* Partycja wymiany tworzy wirtualną pamięć operacyjną systemu i jest „przedłużeniem" pamięci RAM. Linux obsługuje partycje wymiany o wielkości do 128 MB. Zbyt mała partycja wymiany jest bezużyteczna (wręcz szkodliwa), zbyt duża niepotrzebnie zajmuje powierzchnię dyskową. W naszym przypadku założymy partycję wymiany o rozmiarze 32 MB, a resztę dysku przeznaczymy na partycję podstawową. Wcześniej musimy wszakże usunąć (o ile są) partycje już istniejące na dysku. W naszym przykładzie będzie to jedna *(hdal)* partycja dosowa.

Disk Druid wyposażony jest w czytelny i dobrze opisany semigraficzny interfejs użytkownika. Musimy jedynie pamiętać, że do wyboru partycji i opcji programu dysponujemy klawiszem **[Tab].** Partycje tworzymy następująco:

- usuwamy (wybierając **Delete** albo naciskając [F4]) już istniejącą partycję dosowa
- zakładamy (wybierając **Add** albo naciskając **[FI])** partycję wymiany. Zostanie otwarte okno **Edit New Partition.** Tutaj w polu **Size** wpisujemy: **32.** W menu **Type** (typ partycji) wybieramy: **Linux Swap.** Pól **Mount Point** i **Growable?** nie wypełniamy. Między polami przemieszczamy się za pomocą klawisza **[Tab],** Okno zamykamy przez wybranie OK
- analogicznie jak partycję wymiany, tworzymy partycję podstawową. W polu **Mount Point** wpisujemy ukośnik: /. Ten pojedynczy ukośnik oznacza root, czyli początek naszego systemu plików. W polu Size wpisujemy: **483.** Z menu **Type** wybieramy: **Linux Native.** Pola **Growable?** nie wypełniamy.

- Suma obszarów partycji wymiany (Swap) i podstawowej (Native) nie może być większa od całkowitej dostępnej pojemności dysku. W przeciwnym wypadku **Disk Druid** zgłosi błąd: Not enough free space. Wtedy korzystając z **Edit** (bądź **[F3])** należy dokonać zmniejszenia wielkości partycji podstawowej. W naszym przypadku: 483+32=515 MB. Wprawdzie dysk jest 540 megabajtowy, ale jest to wartość reklamowa (wynikająca z często stosowanej przez producentów dysków sztuczki, w której 1 MB pojemności to dokładnie milion bajtów, a nie 2 , czyli 1048576 bajtów). Rzeczywisty dostępny obszar dla danych to 515 MB. Taką też wartość podaje **Disk Druid** w swoim oknie głównym **(Current Disk Partitions),** w polu **Drive Summaries/Total.** Jeżeli dysk jest wykorzystany całkowicie, to w polu **Drive Summaries/Free** powinna być wartość zero.

Po zakończeniu partycjonowania opuszczamy **Disk Druid** wybierając OK (albo naciskając **[F12]),** a następnie potwierdzając w oknie **Save Changes** zachowanie wprowadzonych zmian za pomocą wyboru Yes.

Program instalacyjny zażyczy sobie restartu komputera, na co wyrażamy zgodę, naciskając klawisz **[Enter].** Niestety po restarcie musimy raz jeszcze przejść całą

procedurę wyboru języka itd., aż do wejścia do programu **Disk Druid.** Tutaj sprawdzamy, czy istnieją założone przez nas partycje. Prawdopodobnie konieczne będzie (przez wybranie **Edit** albo naciśnięcie [F3]) ponowne wpisanie **Mount Point:** / dla partycji podstawowej (Native). Po opuszczeniu **Disk Druida** (przez wybranie OK), program instalacyjny, otwierając okno **Active Swap Space,** przystąpi do rozpoczęcia procedury sformatowania partycji wymiany. Zaznaczamy obowiązkowo gwiazdką „*" w polu wyboru (za pomocą klawisza spacji) partycję *Idevlhda5* Warto też, zwłaszcza przy pierwszym użyciu dysku, zaznaczyć gwiazdką opcję **Check for bad blocks during format.** Wydłuży to czas formatowania, ale będziemy pewni, że nie mamy elementarnych problemów z dyskiem. Po dokonaniu zaznaczeń wybieramy OK w celu sformatowania partycji wymiany

Następnie w oknie **Partitions To Format** postępujemy analogicznie przy formatowaniu partycji podstawowej *{ldevlhdal).* W tym przypadku też zalecamy ustawienie opcji **Check for bad blocks during format.**

Teraz wreszcie w oknie **Components to Install** możemy wybrać z przewijalnej, za pomocą klawiszy kursora, listy odpowiednie zestawy pakietów do zainstalowania. Przy wyborze zakładamy, że system przeznaczony jest do naszych ćwiczeń (nie korzystamy z X-Windows), mamy kartę sieciową oraz będziemy chcieli skompilować jądro. Zaznaczamy gwiazdką następujące zestawy:

```
DOS/Windows Connectivity
File Mangers
Networked Workstation
NFS Server
C Development
Development Libraries
```

Wymaga to łącznie około 220 MB przestrzeni dyskowej. Jeżeli nie posiadamy karty sieciowej ani modemu, to nie interesuje nas praca w sieci i wówczas możemy zrezygnować z Networked Worksation oraz NFS Server. Jeżeli nie mamy zamiaru kompilować jądra, a wszelkie dodatkowe pakiety mamy instalować tylko w postaci RPM, możemy ponadto zrezygnować z C Devel opment i Development Libraries. Taki system wymaga około 180 MB przestrzeni dyskowej. Dodatkowe zmniejszenie zajmowanej powierzchni dysku jest możliwe przez zaznaczanie opcji **(Select individual packages)** indywidualnego wyboru pakietów z poszczególnych zestawów. Ż opcji tej powinni jednak korzystać tylko zaawansowani użytkownicy, którzy wiedzą w miarę dokładnie co robią.

Po zatwierdzeniu (OK), program instalacyjny wyświetli okno z komunikatem **(Install log)** o lokalizacji pliku *ltmplinstall.log* zawierającego listę zainstalowanych **tzw.** pakietów RPM, o których powiemy dopiero w lekcji 13. Po zatwierdzeniu (OK) program instalacyjny sformatuje główną partycję (Linux Native), utworzy na niej system plików ext2 (podstawowy w Linuksie), a następnie zainstaluje wybrane pakiety (co potrwa około **10 minut).**

dzić. czy w programie konfiguracyjnym płyty głównej (ang. *setup)* dysk C jest ustawiony jako (najlepiej pierwsze) urządzenie startowe.

Jeśli wszystko poszło prawidłowo, zobaczymy na ekranie zaproszenie do systemu:

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login:
```

Logujemy się teraz jako *roof.*

```
mono login: root
Password: artemlda
[root@mono /root]# _
```

Wprowadzane hasło nie będzie wyświetlane na ekranie monitora. Poprawne zalogowano do systemu jako *root* da jego zgłoszenie ze znakiem gotowości „#". Teraz zamkniemy system poleceniem **shutdown** i wyłączymy komputer. Zrobimy to następująco:

```
# shutdown -h now
```

Wydanie powyższego polecenia rozpoczyna natychmiastową procedurę zamykaniu. Na ekranie zobaczymy szereg komunikatów. Czekamy na ostatnie dwa:

```
The system is halted
System halted
```

Teraz możemy bezpiecznie wyłączyć komputer lub resetując go ponownie załaduje system.

Instalacja dystrybucji Red Hat w sieci lokalnej z wykorzystaniem CD-ROM-u udostępnianego poprzez NFS jest przedmiotem ćwiczenia 9.5.

4.2. Problemy z instalacją

Podstawowym źródłem wiedzy dotyczącym instalacji systemu i konfiguracji **sprzetu** są pliki *HOWTO.* Będą to najczęściej zwykłe pliki tekstowe (lub *.html).* Szu**kać** należy na CD-ROM-ie dystrybucyjnym w katalogu *doclHowto.* W pierwszej kolejności przejrzyjmy pliki *Installation-HOWTO, Hardware-HOWTO* i jeżeli mamy **kartę** sieciową - *Ethemet-HOWTO.*

Znajomość języka angielskiego wydaje się tu niezbędna. Pojawiła się jednak bardzo cenna inicjatywa tłumaczenia plików *HOWTO* na język polski. Tak przetłumaczone pliki zostały nazwane *JTZ* (Jak To Zrobić), co dobrze oddaje istotę rzeczy. Efekt pracy grupy tłumaczy jest dostępny na stronie internetowej: www .jtz.org.pl.

Dostępny jest już *Install-HOWTO.pl.txt,* czyli polskie tłumaczenie *Installation-*

5. Polecenie, program, skrypt

Polecenie systemowe w Linuksie może być albo poleceniem wewnętrznym (ang. ***build** in)* powłoki (w naszym przypadku bash-a) lub poleceniem zewnętrznym. Polecenie zewnętrzne jest plikiem wykonywalnym przechowywanym w określonym katalogu. Polecenie wewnętrzne jest wbudowane w sam bash, zatem „przechowywane" **jest** w pliku *bash*.

Istnieją dwa rodzaje plików wykonywalnych: skrypty powłoki (teksty z poleceniami wykonywane przez powłokę) i pliki binarne (skompilowane programy w języku maszynowym wykonywane bezpośrednio przez procesor). Polecenia zewnętrzne niczym szczególnym nie odróżniają się od innych plików wykonywalnych, czyli programów i skryptów użytkownika. Zakwalifikowanie pliku wykonywalnego jako polecenia systemowego jest kwestią konwencji, a nie szczególnej struktury wewnętrznej danego pliku lub też sposobu jego uruchamiania. Wszystkie pliki wykonywalne i polecenia wewnętrzne powłoki uruchamia się tak samo: przez podanie nazwy i często opcjonalnych parametrów:

```
nazwa    [parametry]
```

Przez parametry rozumiemy tu opcje oraz argumenty.

Używane w tej książce określenie „polecenie" oznacza w pierwszej kolejności polecenie systemowe (wewnętrzne bądź zewnętrzne), ale ogólniej także dowolny plik wykonywalny.

W innym znaczeniu polecenie to tekst wpisany w linii poleceń powłoki (bash-a) **i** zarwierdzony naciśnięciem klawisza [Enter].

Przez program rozumiemy przede wszystkim wykonywalny plik binarny. Niestety użytkownik nie może na podstawie nazwy odróżnić skryptu od pliku wykonywalnego. Zresztą z funkcjonalnego punktu widzenia nie ma to z reguły dla niego żadnego znaczenia. Dla użytkownika programem jest to, co wykonuje jakąkolwiek akcję. Dlatego też programem nazywać też możemy także aplikację użytkową czy narzędzie **konfiguracyjne** dostarczone z pakietem, które użytkownik tylko uruchamia, nie mając **żadnej** potrzeby wnikania, jakiego rodzaju jest to plik wykonywalny.

W DOS-ie skrypty miały obowiązkowe rozszerzenie .BAT, a programy .COM albo EXE. (zresztą do ich uruchomienia podanie rozszerzenie nazwy nie było wcale potrzebne!). W Linuksie (UNIX-ie) nie tylko nie istnieje pojęcie rozszerzenia nazwy, ale też nie ma żadnej powszechnej umowy dotyczącej nazewnictwa skryptów czy programów przez użycie na przykład określonych przyrostków. System identyfikuje ***jako*** wykonywalny każdy plik, który ma nadane tzw. prawo wykonywania.

Lekcja 1. Elementarne czynności administracyjne

W czasie instalacji Linuksa (Red Hat), użytkownikowi *root* nadaliśmy hasto: artemida. Włączamy komputer. Po załadowaniu się systemu operacyjnego logujemy się **jako** *root:*

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: root
Password: artemIda
[root@mono /root]# _
```

Jesteśmy gotowi do ćwiczeń.

Ćwiczenie 1.1. Podstawy edycji i uruchamiania poleceń

Wiersz poleceń (czyli to co znajduje się za znakiem gotowości „#" **lub** „$") służy do wprowadzania poleceń. Przez wprowadzenie polecenia rozumiemy wypisanie jego **treści** (nazwy, opcji i argumentów) i naciśnięcie klawisza **[Enter].** To wszystko co **robimy** w wierszu poleceń przed naciśnięciem klawisza **[Enter],** nazwiemy edycją polecenia. Łatwo odkryjemy, że klawisze sterujące poziomym ruchem kursora (strzałki) powodują przesuwanie kursora w wierszu poleceń, a klawisz **[Backspace]** kasuje **znaki w** sposób analogiczny jak w DOS-ie. Użyteczną edycyjnie kombinacją jest **[Ctrl]+[u],** którą powoduje usunięcie całego wiersza.

Ogólna składnia polecenia jest następująca:

nazwa_polecen1a opcje argumenty

- Opcje z reguły **poprzedzamy znakiem minus** (np. **-1**). Do wyjątków należą polecenia, które akceptują opcje bez znaku minus (np. ps, **tar),** ale i te działają poprawnie, gdy podamy przed opcjami znak minus. Znak minus piszemy bezpośrednio przed literą opcji. Nie może być między nimi spacji, gdyż spacja jest separatorem składników polecenia.

Opcje (także argumenty) nie zawsze muszą wystąpić, na przykład:

```
# ls /
bin         etc        lost+found root       usr
boot        home       mnt        sbin       var
dev         1 i b      proc       tmp
[root@mono /root]# _
```

Wyświetliliśmy zawartość katalogu głównego /. Nazwą polecenia jest tutaj **Is.** Opcje nie występują. Argumentem jest ukośnik (ang. *slash),* oznaczający katalog główny.

- Już wydane polecenie możemy przerwać naciskając kombinację klawiszy **[Ctrl]+[cj.**

Inne użyteczne kombinacje klawiszowe to:

[CtrI]+[\] - zakończenie wykonywania polecenia (tzw. sygnał 3) z jednoczesnym wykonaniem zrzutu, czyli utworzeniem (w bieżącym katalogu) pliku i zapisaniem w nim obszaru pamięci wykorzystywanego przez dany proces; taki plik nosi zawsze nazwę *core*

[Ctrl]+[s] - przerwanie wyświetlania na ekranie

[Ctrl]+[q] - kontynuowanie wyświetlania na ekranie.

Próba przejścia do nowego wiersza w trakcie edycji (czyli naciśnięcie **[Enter])** spowoduje oczywiście wprowadzenie (zakończenie edycji) polecenia. Wprawdzie najczęściej przejście do nowego wiersza nastąpi automatycznie bez ingerencji z naszej strony, ale dobrze wiedzieć, jak sobie radzić w takiej sytuacji.

W takim przypadku stosujemy odwrotny ukośnik (ang. *backslash),* który maskuje klawisz **[Enter].** Na końcu wiersza, który musimy przedłużyć, wpisujemy: \ i teraz możemy już bezpiecznie nacisnąć **[Enter],** aby przejść do nowego wiersza. Zwróćmy uwagę, że znany nam z DOS-a odwrotny ukośnik jest skierowany w przeciwną stronę, niż ukośnik występujący w Linuksie (UNIX-ie) w zapisie ścieżek dostępu, który samodzielnie oznacza katalog główny (root).

W ogólności odwrotny ukośnik jest stosowany do maskowania pojedynczego znaku specjalnego, który gdy wystąpi bezpośrednio po nim, będzie traktowany jak normalny znak.

Ćwiczenie 1.2. Zakładamy konto użytkownika (useradd, passwd)

Czynnościami administracyjnymi zajmuje się *root,* czyli użytkownik mający nieograniczone uprawnienia w systemie. Jednakże do normalnej pracy (a zwłaszcza do zabawy i ćwiczeń) należy w miarę możliwości korzystać z konta zwykłego użytkownika, gdyż istnieje potencjalne niebezpieczeństwo, że niedoświadczony użytkownik zalogowany jako *root* może narobić wiele szkód w systemie.

Załóżmy teraz dwa nowe konta zwykłych użytkowników:

```
# useradd  leszek1
[root@mono /root]#  useradd  leszek2
```

Założyliśmy dwóch nowych użytkowników: *leszek1* i *leszek2.* Przy okazji zauważmy, że klawiszami pionowymi (strzałkami) kursora możemy przewijać bufor klawiatury, co ułatwia wprowadzanie kolejnych poleceń, a zwłaszcza ich modyfikacje.

Podczas tworzenia konta użytkownika tworzony jest automatycznie jego katalog domowy o nazwie takiej samej jak nazwa użytkownika oraz definiowane są inne parametry, m.in. określana jest powłoka, z jaką użytkownik będzie pracował po zalogowaniu się do systemu. Domyślną powłoką jest bash - standardowa powłoka w Linuksie. Ogólnie rzecz biorąc możemy przypisać użytkownikowi dowolną powłokę. W niniejszej książce nie będziemy jednak eksperymentować z powłokami i zawsze będziemy korzystać z bash-a. Różnice między powłokami są odczuwalne przede :ysfkim przy tworzeniu i wykonywaniu skryptów (skrypt jest odpowiednikiem **.bat* w DOS-ie).

Gdy nauczymy się poruszać po katalogach, możemy sprawdzić, co znajduje się w katalogach domowych *(/home/leszek1* i */homelleszek2)* przed chwilą założonych użytkowników. Nowo założonym użytkownikom musimy nadać hasła:

```
# passwd leszekl
New UNIX password: kasjopea1
Retype new UNIX password: kasjopea1
passwd: all authentication tokens updated successfully
[root@mono /root]#  passwd leszek2
New UNIX password: kasjopea2
Retype new UNIX password: kasjopea2
passwd: all authentication tokens updated successfully
[root@mono /root]#  _
```

Przejdźmy teraz na drugą konsolę wirtualną ([lewy Alt]+[F2]) i zalogujmy się **jako** *leszek1:*

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: leszek1
Password: kasjopea1
[leszek1@mono leszek1]$ _
```

Teraz przejdźmy na trzecią konsolę wirtualną ([lewy Alt]+[F3]) i zalogujmy się **jako** *leszekl:*

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: leszek2
Password: kasjopea2
[leszek2@mono leszek2]$ _
```

Zauważmy, że znak gotowości dla *root-a.* to „#", natomiast dla zwykłych użytkowników *(leszekl* i *leszek2)* to dolar „$".

Ćwiczenie 1.3. Spacerkiem po katalogach i nie tylko

Przejdźmy na pierwszą konsolę wirtualną **([lewy Alt]+[F1])** i wylogujmy się:

```
# logout
```

a następnie zalogujmy się ponownie:

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: root
Password: artemida
Last login: Fri Apr 9 07:03:21 on ttyl
[root@mono /root]# _
```

Jak widzimy, mamy teraz dodatkową informację o ostatnim logowaniu się na to konto. Może to być pomocne, gdy chcemy być pewni, że w międzyczasie nikt obcy nie zalogował się jako *root* i nie buszował po systemie. Mamy także (w nawiasach kwadratowych) informację (root@mono), że na komputerze *mono* zalogował się użytkownik *root,* oraz że znajduje się on obecnie w katalogu */root,* który jest zresztą katalogiem domowym użytkownika *root.* Domyślne miejsce logowania to katalog domowy użytkownika.

Postać zgłoszenia gotowości systemu może być konfigurowana przez użytkownika (poznamy to w ćwiczeniu 14.7).

Przejdźmy teraz do katalogu głównego:

```
# cd /
[root@mono /]# _
```

Widzimy, co zmieniło się w nawiasach kwadratowych. Zagłębimy się teraz w dalszy katalog:

```
# cd /usr/doc/HTML
[root@mono HTML]# _
```

Teraz wiemy, w czym problem. W nawiasach kwadratowych mamy podaną tylko nazwę bieżącego katalogu. Jeżeli nazwa katalogu poprzedzona jest ukośnikiem „/", wiemy, że to katalog znajdujący się w katalogu głównym. Brak ukośnika wskazuje na katalog umieszczony niżej w strukturze drzewa (bardziej zagłębiony). Ponadto musimy pamiętać, że nawet tą skromną informacją nie zawsze możemy dysponować. Aby się dowiedzieć, gdzie się znajdujemy, korzystamy z polecenia **pwd:**

```
# pwd
/usr/doc/HTML
[root@mono HTML]# _
```

Jak widać, **pwd** podaje nam pełną ścieżkę dostępu do katalogu, w którym obecnie się znajdujemy. Natomiast użyte wcześniej polecenie cd (change directory - ana-

logiczne do ciosowego CD) służy do przemieszczania się między katalogami w oparciu zarówno o bezwzględne (zaczynające się od ukośnika), jak i względne (bez ukośnika na początku) ścieżki dostępu oraz o następujące aliasy katalogów:

current (bieżący)

parent (tzw. rodzicielski, czyli bezpośrednio wyższy)

home (domowy użytkownika).

Kropkowe aliasy powinny być znane użytkownikom DOS-a. Nowością będzie tu tylda („~"). Przećwiczymy to sobie:

```
# cd /usr/doc/HTML
[root@mono HTML]# pwd
/usr/doc/HTML
[root@mono HTML]# cd ..
[root@mono doc]# pwd
/usr/doc
[root@mono doc]# cd ~
[root@mono /root]# pwd
/root
[rootOmono /root]# cd / .
[rootOmono /] # cd etc
[root@mono /etc]# pwd
/etc
[rootOmono /etc]# cd ..
[root@mono /]# pwd

[rootOmono /]# _
```

Wgląd do środka katalogu umożliwi nam polecenie Is (list files), z grubsza rzecz btcrąc odpowiednik DIR w DOS-ie:

```
# cd /
[rootOmono /]# Is
bin          etc         lost+found root        usr
boot         home        mnt         sbin       var
dev          l.ib        proc        tmp
[root@mono /]# _
```

Nareszcie mamy coś konkretnego, czyli „zawartość" katalogu głównego. No dobrze. ale czy to są pliki, czy katalogi? Aby nie mieć wątpliwości, posłużymy się opcją

```
# Is -F
bin/         etc/        lost+found/          root/   usr/
boot/        home/       mnt/        sbin/    var/
dev/         lib/        proc/       tmp/
[rootOmono /]# _
```

Teraz wiemy, że mamy tu jedynie katalogi (są one oznaczone ukośnikiem „/"). Linux (UNIX) zapewnia nam jednak porządek, gdyż w katalogu głównym nie plączą się jakieś „autoexec-i", „config-i" itp. (jak to było w DOS-ie).

- Opcja -F polecenia ls wprowadza na ekran własne **oznaczenia typów plików** używając kilku specyficznych znaków. Znaki te występują bezpośrednio (bez oddzielenia spacją) po nazwie pliku i są tylko i wyłącznie dodatkową informacją dla użytkownika, a **nie są częścią nazwy pliku.**

  Wykonajmy polecenie:

  ```
  # ls -F  /bin
  ```

a następnie:

  ```
  # ls /bin
  ```

  Zwróćmy uwagę na różnice: gwiazdka „*" i „@" nie należą do nazwy plików.

  Polecenie ls używa następujących oznaczeń typu pliku:

* - plik wykonywalny (program lub skrypt)
/ - katalog
@ - dowiązanie (łącznik) symboliczne
  - łącze nazwane (FIFO)
= - gniazdo.

Pozostałe pliki (w szczególności zwykłe pliki tekstowe oraz pliki specjalne) nie są dodatkowo oznaczane.

Teraz przygotujmy się na poważniejszy eksperyment i czekajmy spokojnie, aż się skończy, nie bacząc na to, że dysk będzie pracował jak oszalały, a zawartość ekranu będzie się zmieniała kilka razy na sekundę:

```
# cd /
[root@mono /]# ls -R
```

Gdy ekran się już uspokoi, powinniśmy w dolnej jego części zobaczyć:

```
var/tmp:

var/yp:
binding

var/yp/binding:
[root@mono /]# _
```

Cóż takiego się stało? Opcja -R (recursively) oznacza „z podkatalogami" (rekurencyjnie). Zatem niewinnie wyglądające polecenie ls -R, wydane z poziomu katalogu głównego oznacza wypisanie całej zawartości (wszystkie katalogi i wszy-

stkie pliki) drzewa root. Nic dziwnego, że nie zmieści się ona na jednym ekranie. Przykład ten uwidacznia nam siłę opcji -R, która występuje nie tylko w poleceniu ls. Skoro bowiem łatwo możemy „wszystko wyświetlić", równie łatwo możemy „wszystko wykasować". Wprawdzie jeszcze nie wiemy jak kasować, ale już wkrótce nie będzie to dla nas tajemnicą. Teraz przestroga, że konta *root* należy używać tylko w wymagających tego sytuacjach, a konta zwykłych („dolarowych") użytkowników do ćwiczeń i codziennej pracy, wydaje się chyba jasna.

Natomiast problem niemieszczenia się treści na ekranie rozwiązujemy za pomocą more (analogicznie jak w DOS-ie). Przetrenujmy to wydając najpierw polecenie:

```
# ls -F /dev
```

a następnie:

```
# ls -F /dev | more
```

Katalog *ldev* zawiera wystarczająco dużo plików, aby nie zmieściły się one na jednym ekranie (nawet w układzie kolumnowym). Sterowanie tekstem w more odbywa się następująco:

[Enter] - przesunięcie o jeden wiersz

[spacja] - przesunięcie o stronę (ekran) naprzód (następna strona)

[q] albo [Q] - wcześniejsze wyjście (zakończenie pracy).

Na zakończenie tego długiego ćwiczenia zasygnalizujemy jeszcze istnienie polecenia cat:

```
# cat /etc/passwd
root:4SITEB84oQMeg:0:0:root:/root:/:/bin/bash
bin:*:l:l:bin:/bin:


leszek1:cooH.qocRodK.:500:500::/home/leszek1:/bin/bash
leszek2:hsRKI.hYU2c9E:501:501::/home/leszek2:/bin/bash
[rootOtnono /]# _
```

Polecenie cat powoduje w tym konkretnym częstym zastosowaniu wypisanie na ekranie zawartości pliku. W tym przypadku dokonaliśmy tego z bardzo ważnym plikiem konfiguracyjnym *passwd,* zawierającym (jak łatwo się domyślić) informacje dotyczące kont użytkowników systemu. Można zauważyć, że oprócz *root-a.* i założonych przez niego kont *(leszekl* i *leszekl),* istnieją konta użytkowników systemowych (np. *bin).* Są to tak zwani pseudoużytkownicy. Na powyższym listigu monitorowym przekazaliśmy tylko dwa pierwsze i dwa ostatnie wiersze pliku *passwd,* a pozostałe wykropkowanie) dla większej czytelności opuściliśmy. Wszystkie pominięte konta to konta pseudoużytkowników. Konta pseudoużytkowników są dla nas zablokowane gwiazdka w drugim polu), co oznacza, że nie możemy sie na nie zalogować.

Każdy wiersz pliku *passwd* dotyczy informacji o jednym koncie (użytkowniku). Wiersz składa się z pól oddzielonych dwukropkami:

- pole pierwsze to nazwa użytkownika
- pole drugie to zakodowane hasto użytkownika. Gwiazdka w tym polu, a także gwiazdka na początku łańcucha hasła (np.":*KJq2cWdt:") oznacza zablokowanie konta. Puste („::") drugie pole oznacza, że hasło nie jest wymagane
- pole trzecie to numer identyfikacyjny użytkownika (UID). Numery mniejsze od 100 są zarezerwowane dla kont systemowych
- pole czwarte to numer identyfikacyjny grupy (GID), możemy, jeśli mamy ochotę, zajrzeć do pliku *letclgroup*
- pole piąte to ewentualna dodatkowa (tekstowa) informacja o użytkowniku
- pole szóste wskazuje katalog domowy użytkownika
- pole siódme wskazuje powłokę użytkownika (tzw. login shell).

Ćwiczenie 1.4. Podszywamy się (su)

Każdy z nas czasami chciałby być (choć na chwilę) kimś innym. Linux nam to umożliwia. Przejdźmy na drugą konsolę **([lewy Alt]+[F2]),** gdzie jest zalogowany *leszekl.* Marzy on o tym, aby choć na chwilę zostać *root-em.* Nie ma problemu:

```
$ su
Password: artemida
[root@mono leszek1]# _
```

A więc stało się. Oczywiście potrzebna była tu znajomość hasła *root-a.* Zauważmy też zmianę znaku gotowości z „$" na „#". Teraz wracamy do siebie:

```
[root@mono leszek1]# exit
exit
[leszek1@mono leszek1]$ _
```

W ogólnym przypadku polecenie su umożliwia stanie się dowolnym użytkownikiem. Czasami bowiem nawet *root* chciałby zostać „szarym" użytkownikiem, aby zobaczyć, jakie problemy ma ten, którego prawa są ograniczone.

Ćwiczenie 1.5. Zmieniamy hasło (passwd)

Pozostajemy na drugiej konsoli wirtualnej ([lewy AIt]+[F2]), gdzie jest zalogowany użytkownik *leszekl.* Zapragnął on zmienić swoje hasło:

```
$ passwd
Changing password for leszek1
(current) UNIX password: kasjopeal
New UNIX password: konfucjusz
```

```
Retype new UNIX password: konfucjusz
passwd: all authentication tokens updated successfully
[leszek1@mono leszek1]$ _
```

Teraz wylogujmy się i zalogujmy na nowo:

```
$ logout

Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: leszek1
Password: kasjopea1
Login incorrect

login: leszek1
Password: konfucjusz
Last login: Sat Apr 10 08:35:37 on tty2
[leszek1@mono leszek1]$ _
```

Jak widzimy, zwykły użytkownik może swobodnie zmieniać swoje hasło logowania i nie musi w tym celu prosić o to administratora *(root-a)*. Po zmianie hasła logowanie na stare hasło jest nieskuteczne (czego oczywiście należało oczekiwać). Powróćmy teraz do poprzedniego hasła (z uwagi na porządek wymagany w ćwiczeniach niniejszej książki):

```
$ passwd
Changing password for leszek1
(current) UNIX password: konfucjusz
New UNIX password: kasjopea1
Retype new UNIX password: kasjopea1
passwd: all authentication tokens updated successfully
[leszek1@mono leszek1]$ _
```

Ćwiczenie 1.6. Nasze procesy (ps)

Przechodzimy na pierwszą konsolę wirtualną ([lewy Alt]+[F1]):

```
# ps a
PID    TTY  STAT  TIME   COMMAND
319     1    S    0:00   -bash
322     4    S    0:00   /sbin/mingetty tty4
323     5    S    0:00   /sbin/mingetty tty5
324     6    S    0:00   /sbin/mingetty tty6
394     1    R    0:00   ps a
320     2    S    0:00   -bash
321     3    S    0:00   -bash
[root@mono  /root]#_
```

Polecenie ps **a** wypisuje na ekranie procesy aktualnie uruchomione w systemie i dotyczące (opcja a) wszystkich użytkowników. Rozszerzoną informację o procesach uzyskamy przez:

```
# ps au
USER     PID %CPU %MEM SIZE  RSS TTY STAT START TIME  COMMAND
leszek1  320  0.0  2.6 1224  804   2    S   17:26 0:00  -bash
leszek2  321  0.0  2.5 1220  788   3    S   17:26 0:00  -bash
root     319  0.0  2.5 1224  800   1    S   17:26 0:00  -bash
root     322  0.0  0.9  724  296   4    S   17:26 0:00
                                              /sbin/mingetty tty4
root     323  0.0  0.9  724  296   5    S   17:26 0:00
                                              /sbin/mingetty tty5
root     324  0.0  0.9  724  296   6    S   17:26 0:00
                                              /sbin/mingetty tty6
root     396  0.0  1.3  772  428   1    R   17:54 0:00  ps au
[root@mono  /root]# _
```

Każdemu procesowi odpowiada jeden wiersz powyższego wydruku. Poszczególne pola oznaczają:

USER - nazwa użytkownika (właściciela procesu)
PID - identyfikator procesu
%CPU - procent czasu procesora wykorzystywany przez proces
%MEM - procent pamięci fizycznej wykorzystywanej przez proces
SIZE - wirtualna wielkość procesu (wraz z danymi i stosem) w kilobajtach
RSS - wielkość części rezydentnej procesu w kilobajtach
TTY - identyfikator terminala sterującego
STAT - bieżący stan procesu (S - uśpiony, R - wykonywany)
START - moment uruchomienia procesu (godzina: minuty)
TIME - skumulowany czas procesora zużyty przez proces
COMMAND - polecenie.

To oczywiście jeszcze nie wszystkie możliwości ps. Spróbujmy poniższego polecenia:

```
# ps aux | more
```

Opcja **x** umożliwia pokazanie procesów niezwiązanych z żadnym terminalem. Zatem dopiero ps **aux** pozwala na zobaczenie naprawdę wszystkich procesów.

Ćwiczenie 1.7. Montujemy CD-ROM (mount)

Pozostajemy na pierwszej konsoli wirtualnej **([lewy Alt]+[F1]).** Wkładamy CD-ROM do czytnika i wprowadzamy polecenie:

```
# mount -t iso9660 -o ro /dev/hdc /mnt/cdrom
hdc: media changed
ISO9660 Extensions: Microsoft Joliet Level 3
[root@mono /]# _
```

Wszystko się prawdopodobnie udało. Jak to sprawdzić, zaraz zobaczymy. Teraz spróbujmy nacisnąć przycisk wysuwu płyty na czytniku CD-ROM. Tacka się nie wysunie! Płytka jest zamontowana i czytnik zablokowany. Aby wyjąć płytkę, musimy ją zdemontować. Ale jeszcze nie teraz. Pokazane wyżej polecenie mount służy w ogólności do zamontowania dowolnego (obsługiwanego przez Linux) systemu plików. A poszczególne łańcuchy określają:

-t iso9660 - typ systemu plików (tutaj ISO9660)

-o ro - „read only" (tylko do odczytu)

/dev/hdc - pełna ścieżka dostępu do pliku specjalnego urządzenia

/mnt/cdrom - pełna ścieżka dostępu do katalogu, do którego montujemy nasz CD-ROM. Katalog ten musi istnieć przed wydaniem polecenia mount. Katalog *Imnt* jest typowym miejscem montowania. Podczas instalacji systemu (Red Hat) zakładane są w *Imnt* standardowo dwa podkatalogi: *cdrom (/mnt/cdrom) ifloppy (Imntlfloppy).*

W typowym komputerze wyposażonym w dwa kanały IDE - *ldevlhdc* odpowiada urządzeniu Master w drugim kanale IDE i odpowiednio:

/devlhda - Master w pierwszym kanale IDE

/devlhdb - Slave w pierwszym kanale IDE

/devlhdd Slave w drugim kanale IDE.

Dla porządku dodajmy jeszcze, że:

/devlfd0 - to pierwszy (A: w DOS-ie) napęd dyskietek

/devlfd - to drugi (B: w DOS-ie) napęd dyskietek.

Pośrednio stwierdziliśmy zamontowanie CD-ROM-u, ale jak to stwierdzić bardziej naocznie? Korzystając z cd i ls (albo tylko z ls: Is /mnt/cdrom) możemy oczywiście zajrzeć do katalogu */mnt/cdrom.* Jest wszakże jeszcze inny sposób:

```
# df
Filesystem  1024-blocks  Used   Available Capacity Mounted on
/dev/hdal      479096     233127 221225      51%     /
/dev/hdc       666424     666424      0     100%     /mnt/cdrom
[root@mono   /root]# _
```

Polecenie df jest bardzo użyteczne, gdy chcemy sprawdzić, ile jeszcze mamy miejsca na dysku, ale jak widać nie tylko do tego może służyć.

Przejdźmy teraz do katalogu */mnt/cdrom* i spróbujmy zdemontować CD-ROM:

```
# cd /mnt/cdrom
[root@mono cdrom]# umount /mnt/cdrom
umount: /mnt/cdrom: device is busy
[root@mono /root]# _
```

Nie udało nam się (co możemy sprawdzić za pomocą df), ale już wiemy dlaczego. Przechodzimy do katalogu głównego i jeszcze raz demontujemy:

```
# cd /
[root@mono /]# umount /mnt/cdrom
[root@mono /]# _
```

Teraz możemy sprawdzić (df), że płytka została zdemontowana i bez problemów wyjąć ją z czytnika.

Wszystkim przerażonym koniecznością wypisywania tasiemcowego polecenia **mount,** wypada zdradzić skrócone polecenie montowania CD-ROM-u. Jest ono następujące:

```
# mount /mnt/cdrom
```

Taka składnia jest możliwa dzięki temu, że dodatkowe informacje (typ sytemu plików, urządzenie) są określone w pliku *letclfstab,* który został domyślnie skonfigurowany przez autora dystrybucji (Red Hat).

Ćwiczenie 1.8. Korzystamy z dyskietki (fdformat, mke2fs, mkfs)

Jak pamiętamy z MS-DOS-a, dyskietkę przed użyciem należało sformatować. Tak naprawdę polecenie FORMAT (uruchomione bezwarunkowo, tzn. z opcją /U) w MS-DOS-ie wykonuje dwie rzeczy jednocześnie: formatowanie nośnika (na niskim poziomie) oraz założenie systemu plików. W Linuksie obie czynności są rozdzielone. Najpierw dokonujemy formatowania niskopoziomowego (poleceniem **fdformat),** a następnie zakładamy linuksowy (ext2) system plików (poleceniem **mke2fs)** lub inny (poleceniem **mkfs)** z obsługiwanych przez system, w szczególności może to być dosowy system plików (msdos).

Pozostajemy na pierwszej konsoli wirtualnej (użytkownik *moi).* Wkładamy dyskietkę (1,44 MB) do napędu A: i do dzieła:

```
# fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
[root@mono /]# _
```

Polecenie **fdformat** wydaje się bardzo proste, ale niezbędnych jest kilka słów komentarza.

Jak widzimy, dokonywana jest weryfikacja powierzchni nośnika. Jeżeli zależy nam na czasie, a polecenia **fdformat** używamy na przykład tylko do nieodwracalnego

kasowania zawartości dyskietek, wówczas możemy weryfikację wyłączyć używając opcji -n:

```
fdformat -n /dev/fd0H1440
```

Ponadto należy zwrócić uwagę na to, że w Linuksie standardowym plikiem specjalnym reprezentującym napęd dyskietek A: jest *Idevlfd0* (dla napędu B: będzie to *devlfdl)*. Tymczasem polecenie fdformat wyjątkowo używa własnych dedykowanych (plików specjalnych zależnych od typu dyskietki. Wymieńmy dla porządku najważniejsze:

- dyskietka 3,5 " HD (1,44MB) w napędzie A:
- dyskietka 3,5 " HD (1,44MB) w napędzie B:
- dyskietka 5,25"HD (1,2 MB) w napędzie A:
- dyskietka 5,25"HD (1,2 MB) w napędzie B:

Po sformatowaniu niskopoziomowym dyskietki możemy założyć na niej system plików ext2 (podstawowy dla Linuksa):

```
# mke2fs    /dev/fd0
mke2fs 1.12, 9-Jul -98 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label=
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
first data block=l
Block size=1024 (1og=0)
fragment size=1024 (log-0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
  writing superblocks and filesystem accounting information: done
[root@mono /]# _
```

Teraz zamontujemy naszą dyskietkę:

```
# mount -t ext2 /dev/fd0 /mnt/floppy
[root@mono /]# _
```

Sprawdzimy (df), czy tak się rzeczywiście stało. Demontujemy dyskietkę poleceniem o prostszej składni:

```
# unount /mnt/floppy
  [root@mono /]# _
```

i sprawdzamy (df), że rzeczywiście została zdemontowana.

ilogicznie jak dla CD-ROM-u (i dzięki temu samemu mechanizmowi systemowemu) istnieje także skrócone polecenie montownia dyskietki:

```
# mount /mnt/floppy
```

Należy jednak pamiętać, że owe skrótowe montowanie jest zdefiniowane jedynie dla dyskietki z linuksowym systemem plików ext2.

Sformatujemy **(fdformat /dev/fd0H1440)** teraz kolejną dyskietkę i dla odmiany założymy system plików msdos, czyli utworzymy normalną dyskietkę dosową:

```
# mkfs -t msdos /dev/fd0
mkfs.msdos 0.3b (Yggdrasil), 5th May 1995 for MS-DOS FS
[root@mono /]# _
```

Aby skorzystać z naszej dyskietki, musimy ją zamontować:

```
# mount -t msdos /dev/fd0 /mnt/floppy
[root@mono /]# _
```

Zdemontowanie przebiega w znany już sposób:

```
# umount /mnt/floppy
[root@mono /]#
```

Na zakończenie jedna uwaga: jeżeli używamy dyskietki do różnych celów, zmieniając często założony na niej system plików, nie ma potrzeby formatowania jej za każdym razem na niskim poziomie **(fdformat).** Wystarczy to zrobić raz na początku jej użytkowania i później profilaktycznie po pewnym czasie eksploatacji w celu „przemagnesowania" powierzchni oraz sprawdzenia, czy nie pojawiły się jej uszkodzenia.

Jeśli mamy wątpliwości co do aktualnego stanu dyskietki, możemy użyć polecenia zakładające system plików **(mke2fs** albo **mkfs)** z opcją -c (check), sprawdzającą powierzchnię nośnika pod kątem obecności uszkodzonych bloków (ang. *bad blocks).* Uszkodzone bloki zostaną zaznaczone i nie będą wykorzystywane (analogicznie jak w DOS-ie) do przechowywania informacji.

Przykładowo polecenie założenia systemu ext2 będzie wtedy miało postać:

```
# mke2fs -c /dev/fd0
```

a polecenie utworzenia dyskietki dosowej:

```
# mkfs -t msdos -c /dev/fd0
```

Ćwiczenie 1.9. Weryfikacja powierzchni dyskietki i CD-ROM-u (badblocks)

Istnieje polecenie **badblocks,** które możemy wykorzystać do weryfikacji dyskietki, która ma już założony system plików. W tym przypadku dyskietka nie może być zamontowana:

```
# badblocks -s /dev/fd0 1440
```

```
Checking for bad blocks (read-only test): done
[root@mono /] _
```

Opcja -s (show) pokazuje przebieg testowania. Liczba 1440 określa liczbę (kilobajtowych) bloków. Jak można zauważyć test jest typu „read-only" (tylko do odczytu), czyli nie niszczy ewentualnych danych zawartych na nośniku. W zbliżony sposób możemy sprawdzić powierzchnię CD-ROM-u. Sprawdzimy to na przykładzie:

```
# mount /mnt/cdrom
hdc: media changed
ISO9660 Extensions: Microsoft Joliet Level 3
[root@mono /]# df
Filesystem 1024-blocks Used Available Capacity Mounted on
/dev/hdal    479096    233127  221225     51%    /
/dev/hdc     666424    666424       0    100%    /mnt/cdrom
[root@mono /]# umount /mnt/cdrom
[root@mono /]# badblocks -s /dev/hdc 666424
Checking for bad blocks (read-only test): done
[root@mono /] # _
```

Jak widać, CD-ROM został na chwilę zamontowany. Musieliśmy to zrobić, aby za pomocą df uzyskać informację o liczbie (666424) bloków do przetestowania. Liczba ta musi być podana dokładnie. Jeśli będzie zbyt mała, to cały nośnik nie zostanie sprawdzony. Jeżeli będzie zbyt duża, to badblocks będzie testować nieistniejące bloki, zasypując nas komunikatami o błędach. Ogólna składnia polecenia badblocks jest więc następująca:

badblocks -s napęd liczba_bloków

Napęd to */devlfd0* dla napędu dyskietek A: i /devlhdc dla czytnika CD-ROM w naszym przykładowym komputerze (urządzenie Master w drugim kanale IDE). Liczba bloków dla dyskietki (3,5" HD) będzie wynosiła 1440. Natomiast dla CD-ROM-u będzie za każdym razem nieco inna, dlatego musimy ją uzyskać na przykład opisaną wyżej metodą.

Do gruntownego sprawdzenia nowych dyskietek przed ich wprowadzeniem do eksploatacji możemy użyć opcji -w (write - testowanie z zapisem). Jest to test niszczący zarówno dane, jak i system plików dyskietki. Możemy go zatem użyć także do nieodwracalnego kasowania dyskietek z danymi i to używanych poprzednio w dowolnym systemie operacyjnym. Test jest dość długi, ale rzetelny. Polega na kolejnym zapisywaniu nośnika określonymi wzorcami bitowymi (ang. *pattem),* a następnie na ich odczytywaniu. Wkładamy dyskietkę do napędu A: i wydajemy polecenie:

```
# badblocks -sw /dev/fd0 1440
Writing pattern 0xaaaaaaaa: done
Reading and comparing: done          1440
Writing pattern 0x55555555: done
Reading and comparing: done          1440
Writing pattern 0xffffffff: done
```

```
Reading and comparing: done          1440
Writing pattern  0x00000000: done
Reading and comparing: done          1440
[root@mono /]# cd ~
[root@mono /root]# _
```

- Zwróćmy uwagę na sposób zapisu dwóch (w ogólności wielu) opcji polecenia używanych jednocześnie. Stosujemy tylko jeden znak minus, a opcje wypisujemy bezpośrednio jedna za drugą, nie oddzielając ich żadnym znakiem. Jest to właściwe dla większości poleceń.

Jak już zapewne się domyślamy, podstawowym przeznaczeniem polecenia **badblocks** jest sprawdzanie partycji na dysku twardym. Tutaj użyliśmy je do nieco innych celów.

Ćwiczenie 1.10. Korzystamy z dyskietek dosowych (mto*ols)

Często mamy do czynienia z kilkoma dyskietkami dosowymi i chcielibyśmy szybko zobaczyć co na nich jest, a tu zabawa z montowaniem i demontowaniem. Otóż niekoniecznie. Do manipulowania dyskietkami dosowymi istnieje wygodne narzędzie **mtools,** dzięki któremu posługujemy się nimi tak jak w DOS-ie (tzn. bez montowania i późniejszego demontowania).

W dystrybucji Red Hat pakiet **mtools** jest instalowany standardowo i jeśli instalację wykonaliśmy zgodnie z zaleceniami podanymi w rozdziale 4, to powinniśmy mieć ten pakiet na dysku. Aby się o tym przekonać wydamy polecenie:

```
# mtools
Supported commands:
mattrib, mbadblocks, mcd, mcopy, mdel , mdeltree, mdir, mdu
mformat, minfo, mlabel, mmd, mmount, mpartition, mrd, mread
mmove, mren, mshowfat, mtoolstest, mtype, mwrite, mzip
[root@mono /root]# _
```

Wiele z poleceń oferowanych przez **mtools** wygląda dość znajomo. Co więcej, możemy się nimi także w znany z DOS-a sposób posługiwać. W szczególności bardzo prosto możemy (pamiętajmy: bez potrzeby montowania!) obejrzeć dowolną dyskietkę dosową. Wkładamy pierwszą z brzegu dyskietkę do napędu A: i wydajemy polecenie:

```
# mdir a:
 Volume in drive A has no label
 Volume Serial Number is 12D4-1023
Directory for A:/

CHECK      BAT     50  06-19-1993   6:41
KOPIA      BAT    880  08-13-1998  20:26
```

```
WINCHI-1   PDF   8544   12-25-1998   20:00   WinChip2MCL_xls.pdf
ES1948-1   GZ    2626   03-23-1999   23:13   esl948_tar.gz
      4 files              12 100 bytes
                        1 444 352 bytes free

[root@mono /root]# _
```

Zauważmy przy okazji, że mdir obsługuje (właściwe DOS 7.0 występującemu wraz z Windows95) długie nazwy plików, podając z prawej strony pełną (długą) nazwę pliku.

Sprawdźmy, że dysk twardy nie jest dla mtools dostępny, ponieważ nie ma nim żadnej partycji dosowej:

```
# mcd c:
Drive 'C:' not supported
Cannot initialize 'C:'
[root@mono /root]# _
```

Przejdźmy zatem na A: i usuńmy plik *check.bat,* a następnie załóżmy katalog

```
# mcd a:
[rootOmono /root]# mdel  check.bat
[root@mono /root]# mmd  dane
[root@mono /root]# mdir
 Volume 1n drive A has no Tabel
 Volume Serial Number is 12D4-1023
Di rectory for A:/

KOPIA      BAT    880   08-13-1998   20:26
WINCHI-1   PDF   8544   12-25-1998   20:00   WinChip2MCL_xls.pdf
ES1948-1   GZ    2626   03-23-1999   23:13   esl948_tar.gz
dane      <DIR>        04-11-1999   10:46   dane
        4 files              12 050 bytes
                          1 444 352 bytes free

[root@mono /root]# _
```

Możemy również sformatować dyskietkę dla DOS-u:

```
# mformat  -t  80  -h  2  -s 18  a:
[root@mono /root]# mdir a:
 Volume in drive A has no label
 Volume Serial Number is 14DC-555C
Directory for A:/

No files
                       1 457 664 bytes free

[root@mono /root]#_
```

Polecenie **mformat** zakłada jedynie dosowy system plików. Nowa dyskietka powinna zostać przed pierwszym użyciem sformatowana niskopoziomowo za pomocą **fdformat** (ćwiczenie 1.8).

Dobrze jest wykonać dodatkowo weryfikację powierzchni dyskietki, aby ustrzec się niespodzianek. Wykonujemy to bardzo prosto:

```
# mbadblocks  a:
[root@mono  /root]# _
```

Brak komunikatów o Wędach oznacza, że wszystko jest w porządku. Ale żeby mieć pełne zaufanie do tego narzędzia, możemy wyjąć dyskietkę i włożyć inną, o której wiemy, że ma na pewno uszkodzoną powierzchnię:

```
# mformat  -t  80  -h  2  -s 18  a:
[root@mono  /root]#  mbadblocks a:



end_request: 1/0 error, dev 02:00, sector 2088
plain_io: Input/output error
Bad cluster 2091 found
end_request: 1/0 error, dev 02:00, sector 2088
plain_io: Input/output error
Bad cluster 2092 found
[root@mono  /root]#  mdlr a:
 Volume in drive A nas no label
 Volume Serial  Number is 4A5D-F6A1
Di rectory for A:/

No files
                        1 439 232 bytes free

Croot@mono  /root]# _
```

W wyniku działania polecenia **mbadblocks** otrzymujemy serię komunikatów o błędach (pokazujemy tylko dwa ostatnie). Z kolei po wydaniu polecenia **mdir** możemy przekonać się, ile straciliśmy (1439 kB w stosunku do 1457 kB dla nieuszkodzonej dyskietki) miejsca z uwagi na uszkodzoną powierzchnię.

Dla porządku wypada jeszcze dodać kilka uwag dotyczących składni polecenia **mformat.** Występujące w opcjach tego polecenia liczby (80, 2, 18) określają geometrię dyskietki i oznaczają odpowiednio: liczbę ścieżek, liczbę głowic i liczbę sektorów na ścieżce. Są one właściwe dla dyskietki 3,5" HD (1,44 MB). Dla innych (starszych* typów dyskietek liczby te będą następujące:

3,5" DD(720kB) 80,2,9
5,25" DD(360kB) 40,2,9
5,25" HD(1,2MB) 80,2, 15.

Skopiujemy teraz na dyskietkę dosową plik */etc/passwd*. Wkładamy dyskietkę do napędu (pamiętamy, żeby jej nie montować) i wydajemy polecenie:

```
# mcopy /etc/passwd a:
[root@mono /root]# mdlr
 Volume in drive A has no label
 Volume Serial Number is 3720-AE2D
Di rectory for A:/

passwd           632 04-10-1999  20:02  passwd
       1 file                 632 bytes
                        1 456 640 bytes free

[root@mono /root]# _
```

Zwróćmy uwagę, że w linii polecenia mdir pominięto nazwę napędu (a:), a system i tak zareagował prawidłowo. W naszym konkretnym przypadku polecenia pakietu mtools będą się odnosiły do dyskietki A:, gdyż nie ma innych systemów plików msdos w naszym systemie. W ogólności powinniśmy jednak pilnować poprawności składni, zwłaszcza gdy posługujemy się opcjami bądź na przykład usuwamy pliki:

```
# mdel a:\passwd
[root@mono /root]# mdir a:
 Volume in drive A has no label
 Volume Serial Number is 3720-AE2D
Directory for A:/

No files
                      1 457 664 bytes free

[root@mono /root]# _
```

Ćwiczenie 1.11. Oglądamy komunikaty jądra (dmesg)

W czasie ładowania systemu na ekranie monitora widzimy przesuwające się komunikaty jądra dotyczące m.in. wykrywanych urządzeń i ładowanych modułów jądra. Montujemy je w każdej chwili ponownie zobaczyć (już po załadowaniu systemu) korzystając z polecenia dmesg. Ponieważ na pewno nie zmieszczą się na jednym ekranie, dodatkowo zastosujemy polecenie more:

```
# dmesg | more
```

Dodatkowe informacje techniczne o systemie możemy znaleźć w plikach znajdujących się w katalogu */proc*. Ale o tym w ćwiczeniu 12.5.

Ćwiczenie 1.12. Zamykamy system (shutdown, halt, poweroff, reboot)

Jeżeli zapoznaliśmy się z rozdziałem 4, wiemy już, że w celu zamknięcia systemu posługujemy się poleceniem shutdown. Do wykonania tego polecenia jest uprawniony jedynie użytkownik *root*. Na potrzeby niniejszych ćwiczeń ogólna składnia polecenia shutdown jest następująca:

```
shutdown [-r | -c | -h ] czas [wiadomość dla użytkowników]
```

Zapis powyższy oznacza, że obowiązkowym argumentem jest „czas" (musi on zawsze wystąpić). Opcje -r i -h już poznaliśmy. Przypomnijmy, że opcja -h (halted) oznacza zamknięcie systemu z zatrzymaniem komputera, opcja -r (reboot) oznacza zamknięcie systemu i restart komputera, a opcja -c (cancel) umożliwia odwołanie uprzednio wydanego polecenia shutdown. Oczywiście stosowanie odwołania ma sens jedynie wtedy, jeżeli w wydanym poleceniu shutdown, które chcemy odwołać, parametr „czas" nie był określony jako „now" (natychmiast).

Parametr „czas" może być podany,w postaci *bezwzględnej* (hh:mm), na przykład 17:40 (co oznacza: rozpocznij shutdown o godzinie siedemnastej czterdzieści) lub *względnej* (+m), przykładowo +10 (co oznacza: rozpocznij shutdown za dziesięć I minut). Określenie tekstowe „now" odpowiada +0.

Przypomnijmy również, że znana z DOS-a trójklawiszowa kombinacja [Ctrl]+[Alt]+[Del], powodująca tzw. gorący restart komputera, tutaj także funkcjonuje w podobny sposób. Jej konsekwencją jest bowiem wykonanie przez system polecenią:

```
shutdown -r now
```

Tyle teorii, teraz praktyka: przechodzimy na pierwszą ([lewy Alt]+[F1J) konsoli wirtualną i wydajemy polecenie:

```
# shutdown now
```

Przez ekran przewinie się szereg komunikatów, a na końcu uzyskamy:

```
Telling INIT to go to single user mode.
INIT: Going single user
INIT: Sending processes the TERM signal
INIT: Sending processes the KILL signal
bash# _
```

Wydanie polecenia shutdown now bez opcji (-r, -h, -c) powoduje przejście **systemu** do tzw. trybu jednoużytkownikowego (ang. *single user mode)*. Wydajmy **kilka** poleceń, na przykład pwd, ls, aby się przekonać, że system dalej żyje.

Tryb jednoużytkownikowy jest w szczególnych sytuacjach potrzebny administratorowi, gdy musi on wykonać prace, przy których inni użytkownicy mogliby mu **„prze**szkadzać". Sprawdźmy przy okazji, jak wyglądają konsole wirtualne ([lewy Alt]+[F2])

i lewy Alt]+[F3]) zwykłych użytkowników *(leszek1* i *leszek2)*. Stwierdzimy, że użytkownicy nie mogą pracować. Wracamy do normalnej pracy systemu przez exit i logujemy się jako *roof*.

```
bash# exit
```

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: root
Password: artemida
Last login: Sun Apr 11 12:11:44 on ttyl
[root@mono /root]# _
```

Musimy też na nowo zalogować się na drugiej *(leszek1)* i trzeciej *(leszek2)* konsoli wirtualnej.

Wydajmy teraz polecenie shutdown (z pierwszej konsoli wirtualnej) w następującej postaci:

```
shutdown -h +10 Za 10 minut zamykamy

Broadcast message from root (ttyl) Sun Apr 11 14:57:05 1999...

la 10 minut zamykamy
The system is going DOWN for system halt in 10 minutes !!
```

Jak widać, straciliśmy znak gotowości „#". Nie próbujmy na razie go odzyskać. Ponadto widać, że shutdown automatycznie generuje prosty komunikat ostrzegawczy. Jeżeli zatem nie mamy nic ciekawego do powiedzenia użytkownikom, możemy darować sobie własny komunikat. Zajrzyjmy teraz na drugą ([lewy Alt]+[F2]) i trzecią (lewy Alt]+[F3]) konsolę wirtualną. Mamy tu analogiczne komunikaty. Zniknął również znak gotowości „$". Możemy go odzyskać naciskając [Enter]. Wróćmy na pierwszą konsolę ([lewy Alt]+[F1]). Naciskanie [Enter] nie ma tu sensu (co możemy sprawdzić). Jedyną sensowną akcję jest wciśnięcie kombinacji [Ctrl]+[c], co też uczynimy. Otrzymamy następującą odpowiedź:

```
Shutdown cancelled.
[root@mono /root]# _
```

Udało nam się zatem przerwać (odwołać) shutdown, ale niekoniecznie to było naszym zamiarem. Jeżeli po wydaniu polecenia shutdown, chcemy dalej pracować w systemie jako *root,* musimy się do niego zalogować (jako *root* oczywiście) z innej konsoli (np. w naszym przypadku z czwartej: [lewy Alt]+[F4]).

Sprawdźmy czas systemowy:

```
# date
Sun Apr 11 16:05:57 CEST 1999
[rootOmono /root]# _
```

i teraz:

```
# shutdown -r 16:10
Broadcast message from root (ttyl) Sun Apr 11 16:06:20 1999...

The system is going DOWN for reboot in 4 minutes !!
```

Sprawdźmy, czy na konsolach drugiej i trzeciej pojawiły się również stosowne komunikaty i spokojnie poczekajmy, aż system zostanie zamknięty, a komputer zresetowany.

Teraz zalogujemy się jako zwykły użytkownik *(łeszekl)* i korzystając z polecenia su zamkniemy system. Musimy pamiętać, że w tym przypadku należy podać pełną ścieżkę dostępu do polecenia shutdown:

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: leszek1
Password: kasjopea1
Last login: Sun Apr 11 13:25:08 on tty2
[leszek1@mono leszek1]$ su
Password: artemida
[root@mono leszek1]# shutdown -h now
bash: shutdown: command not found
[rootOmono leszek1]# /sbln/shutdown -h now
```

Obserwujemy pojawiające się na ekranie komunikaty, czekając aż zostaną wyświetlone dwa ostatnie:

```
The system is halted
System halted
```

Teraz możemy wyłączyć komputer albo zrestartować go za pomocą przycisk obudowie (lub kombinacją klawiszy [Ctrl]+[Alt]+[Del]).

Przy następnym zamykaniu systemu w trybie zwykłego użytkownika możemy próbować polecenie su z opcją -1:

i sprawdzić, czy w tej sytuacji również musimy podać pełną ścieżkę dostępu do cenis shutdown.

Najczęściej spotykaną w praktyce sytuacją jest chęć natychmiastowego zamknięcia systemu oraz wyłączenia komputera albo zamknięcia systemu i zrestartowania **komputera.** W takich przypadkach wygodne jest użycie jednego z trzech poleceń: **halt. reboot** albo **poweroff.**

Polecenia:

```
# halt
```

oraz

```
# poweroff
```

realizują to samo, a funkcjonalnie odpowiadają **shutdown -h now.** Natomiast polecenie:

```
# reboot
```

funkcjonalnie odpowiada **shutdown -r now.**

Z lekcji pierwszej powinniśmy nie tylko wynieść umiejętność posługiwania się poleceniami, które w jej ramach ćwiczyliśmy, ale również zapamiętać, że w katalogu /dev będziemy szukać plików specjalnych do urządzeń, a w katalogu */etc* różnej maści plików konfiguracyjnych.

Lekcja 2. Katalogi i pliki

Przypomnijmy sobie poznaną z poprzedniej lekcji ogólną składnię polecenia systemowego. Dobrym przykładem jest ls:

```
# ls -aF /root
./              .bash_history       .bashrc      .dosemu/
../             .bash_logout        .cedit/      .mc/
.Xdefaults      .bash_profile       .cshrc       .tcshrc
Croot@mono /root]# _
```

Polecenie składa się z nazwy (ls), opcji (-aF) i argumentów **(/root).** Nazwa, opcje (ang. *options, flags)* i argumenty oddzielone są od siebie spacją. Opcje (tutaj dwie: -a i -F) poprzedzone są (prawie) zawsze znakiem minus (jeśli jest więcej opcji, to możemy zapisać je razem stosując tylko jeden wspólny minus dla wszystkich). Jeżeli argumentów będzie kilka (nie w każdym poleceniu jest to sensowne), wówczas oddzielone będą od siebie spacją. Jak już wiemy z poprzedniej lekcji, obowiązkowo (co jest oczywiste) musi wystąpić nazwa.

Dodatkowo poznaliśmy teraz opcję -a polecenia ls. Umożliwia ona pokazanie plików (katalogów) ukrytych, których nazwa zaczyna się od kropki.

Na dowolnej konsoli wirtualnej logujemy się jako *leszekl:*

```
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i586

mono login: leszek1
Password: kasjopea1
Last login: Fri Apr 9 10:41:16 on tty2
[leszekl@mono leszek1]$ pwd
/home/leszek1
[leszek@mono leszek1]$ _
```

Znajdujemy się w katalogu */home/leszekl,* który jest katalogiem domowym użytkownika *leszekl.* Wszystkie ćwiczenia tej lekcji przeprowadzimy właśnie w tym katalogu.

Ćwiczenie 2.1. Tworzymy katalogi (mkdir)

Najpierw zobaczymy co mamy w katalogu */home/leszekl*:

```
$ ls -aF
./          .Xdefaults          .bash_logout        .bashrc
```

```
../          .bash_history        .bash_profile
[leszekl@mono leszek1]$ _
```

Są tu tylko pliki ukryte, czyli można powiedzieć, że dla normalnego użytkownika jest to katalog pusty, o czym przekonamy się wydając polecenie:

```
$ ls -F
[leszekl@mono leszek1]$ _
```

Założymy teraz dwa katalogi: *katl* i *kat2:*

```
$ mkdlr  katl  kat2
[leszekl@mono leszekl]$ ls
katl   kat2
[leszekl@mono leszekl]$ _
```

Za pomocą przed chwilą poznanego polecenia mkdir (make directory) w katalogu *katl* założymy dodatkowo dwa katalogi (podkatalogi): *pkt1 \pkt2:*

```
$ mkdlr  kat2/pkt1  kat2/pkt2
[leszek1@mono leszek1]$ ls  -F  kat2
pktl/    pkt2/
[leszek1@mono leszek1]$ ls  -FR
katl/    kat2/

kat1:

kat2:

pktl/  pkt2/

kat2/pkt1:

kat2/pkt2:
[leszek1@mono leszek1]$ _
```

Przy okazji udało nam się sensownie wykorzystać opcję -R polecenia ls. Rezultat wygląda na pierwszy rzut oka nieco dziwacznie, ale analizując go wiersz po wierszu widzimy, że ma sens. Otóż powyższy rezultat polecenia ls -FR, wyjaśnimy tak: w katalogu bieżącym *(/home/leszek1)* mamy dwa katalogi: *katl* i *katl.* Katalog *katl* nie zawiera już dalszych podkatalogów. Katalog *kat2* ma dwa podkatalogi *(pktl* i *pkt2).* Katalog *pktl* nie ma już dalszych podkatalogów i także katalog *pkt2* nie ma dalszych podkatalogów.

Tworzenie pliku (katalogu) wiąże się z nadaniem mu nazwy. Jest więc okazja, by rzec parę słów z dziedziny nazewnictwa. Wiemy już, że system rozpoznaje duże i małe litery. W ogólności nazwa pliku (katalogu) może składać się z dowolnego ciągu znaków ASCII niezawierającego znaku spacji.

- W nazwie pliku może występować kropka (jedna lub wiele), lecz jest to znak tak jak każdy inny i nie dzieli on nazwy na „część właściwą" i „rozszerzenie", jak ma to miejsce w DOS-ie. Wszelkie wyróżnienia w nazwie nie są cechą systemu.

lecz tylko lokalną umową, która może dotyczyć konkretnego programu. Pamiętajmy, że katalog też jest plikiem, zatem podawane reguły dotyczą także katalogów.

Linux obsługuje długie nazwy plików (nie ma tu ograniczenia jak w DOS-ie). W podstawowym dla Linuksa systemie plików ext2 (Extended File System Version 2), nazwa pliku może mieć maksymalnie 255 znaków. Dobrym zwyczajem jest używanie (w miarę możliwości) maksymalnie 14 znakowych nazw, gdyż jest to tradycyjne ograniczenie, które można spotkać w starszych systemach plików (np. minix, który czasami występuje na linuksowych dyskietkach).

W systemie UNIX istnieje ograniczenie na długos'ć ścieżki dostępu (nie więcej niż 1023 znaki). Ograniczenie to możemy obejść korzystając z polecenia cd. Nie przeprowadzałem żadnych eksperymentów, jak Linux zachowa się w takiej sytuacji, ale warto o tej możliwości pamiętać.

Dla własnej wygody podczas wprowadzania z klawiatury powinniśmy używać jak najkrótszych nazw. Aby uniknąć niespodzianek, należy ponadto stosować się jeszcze do dodatkowych reguł (przede wszystkim w doborze znaków):

- nazwa nie powinna zaczynać się od kropki (kropką zaczynają się nazwy plików ukrytych), chyba że chcemy utworzyć plik (katalog) ukryty
- nie należy stosować w nazwie następujących znaków: / * ? [ ] < > & ; ' $! % ( ) C) @ \, gdyż mogą powodować różne „interakcje", dając w rezultacie niezamierzone efekty. Są to znaki specjalne (ang. *metacharacterś)* mające dla powłoki inne znaczenie niż znak literowy. Z czasem poznamy, który z nich i w jakich okolicznościach może być „niebezpieczny", gdy spróbujemy użyć go w nazwie
- nie należy również używać znaku minus na początku nazwy (jest to bowiem wskazanie na opcję polecenia)
- nie należy stosować nazwy *test,* jest to bowiem wewnętrzne polecenie bash-a

Jak już wiemy, system nie wyróżnia (znanego z DOS-u) tzw. rozszerzenia nazwy, lecz niektóre programy usługowe rozpoznają jednoliterowe (lub bardziej złożone) przyrostki oddzielone kropką od „właściwej" nazwy pliku. Pamiętajmy, że w nazwie pliku kropka może wystąpić wielokrotnie (a nie tylko raz, jak w DOS-ie).

Ćwiczenie 2.2. Usuwamy katalogi (rmdir, rm)

```
$ rmdir  kat1  kat2
rmdir: kat2: Directory not empty
[leszek1@mono  leszek1]$  ls
kat2
[leszekl@mono  leszek1]$  _
```

Wiemy już, że polecenie **rmdir** (remove directory) usuwa tylko puste katalogi. Czy można usunąć katalog z całą jego zawartością? Oczywiście:

```
$ rm -r kat2
[leszek1@mono leszek1]$ ls
[leszek1@mono leszek1]$ _
```

Użyliśmy tu polecenia **rm** (przeznaczonego do usuwania plików) z „niebezpiecznym" parametrem -r (recursively). Pozwoliło to usunąć *kat2* wraz z całą zawartością.

Ćwiczenie 2.3. Trenujemy edycję poleceń ([Tab], elear)

Zanim zaczniemy intensywniej ćwiczyć polecenia, powinniśmy rozszerzyć nasze umiejętności edycyjne. Wiemy już, że klawiszem **[Backspace]** kasować możemy znaki, klawiszami pionowymi kursora przewijać bufor klawiatury, a poziomymi przesuwać kursor.

Ale nie wiemy zapewne, że wiersz edycji poleceń może pracować w dwóch trybach: trybie emaes (ustawionym domyślnie) lub trybie vi. Nie będziemy zmieniać ustawień domyślnych, w których mamy następujące udogodnienia:

[Ctrl]+[e] - skok na koniec wiersza

[Ctrl]+[a] - skok na początek wiersza

[Ctrl]+[k] - usunięcie tekstu od położenia kursora do końca wiersza

[Ctrl]+[y] - dołączenie uprzednio usuniętego tekstu od miejsca położenia kursora

[Ctrl]+[u] - usunięcie całego wiersza

[Tab] - uzupełnia nazwę pliku (katalogu) do podanego początkowego fragmentu. Jeżeli nie można jednoznacznie uzupełnić, podawany jest sygnał dźwiękowy. Powtórne naciśnięcie **[Tab]** wyświetli listę alternatywnych możliwości.

Udogodnienie związane z użyciem klawisza **[Tab]** jest prawdziwym skarbem. Aby się o tym przekonać, przećwiczymy to na przykładzie. Przejdźmy do katalogu */etc:*

```
$ cd /etc
[1eszekl@mono /etc]$ _
```

Teraz wykonajmy sekwencję: **ls, [spacja], c, [Tab].** Usłyszymy sygnał, a na ekranie zobaczymy:

```
[leszekl@mono /etc]$ ls c
```

Naciśnijmy klawisz **[Tab]** jeszcze raz, a otrzymamy:

```
[ leszek1@mono /etc]$ ls c
conf.linuxconf    cron.daily    cron.monthly    crontab
```

```
conf.moduł es     cron.hourly   cron.weekly    csh.cshrc
[leszekl@mono /etc]$ ls c_
```

Teraz wprowadzimy sekwencję: r, [Tab], usłyszymy sygnał dźwiękowy i otrzymamy:

```
[leszek1@mono /etc]$ ls cron_
```

Jak widzimy, wprowadziliśmy tu tylko jeden znak r, a końcówka „on" została automatycznie uzupełniona (bo jest jednoznaczna!), ale dalej znowu mamy kilka możliwości. Teraz naciśnięcie klawisza [Tab] spowoduje jedynie sygnał dźwiękowy. Dopiero kolejne naciśnięcie [Tab] wyświetli nam to, czego możemy oczekiwać:

```
[leszekl@mono /etc]$ ls cron
cron.daily  cron.hourly  cron.monthly  cron.weekly  crontab
[1eszekl@mono /etc]$ ls cron_
```

Teraz wprowadzamy sekwencję: **tab, [Enter]** , czyli „kończymy zabawę":

```
[leszeklOmono /etc]$ ls crontab
crontab
[leszekl@mono /etc]$ cd ~
[leszeklOmono leszekl]$ _
```

Przy okazji zobaczyliśmy, jaki rezultat daje użycie polecenia ls (bez żadnych rpcji) w stosunku do pliku (a nie do katalogu).

Każdy kto używał DOS-a zetknął się na pewno z użytecznym poleceniem **CLS,** uczącym ekran. W Linuksie poleceniem czyszczącym ekran jest c**lear.** Wykonane **elear** spowoduje wyczyszczenie ekranu i ustawienie w pierwszym wierszu ekranu zgłoszenia gotowości systemu. Łatwo możemy to sprawdzić:

```
$ clear
```

Ćwiczenie 2.4. Kopiujemy, przesuwamy, zmieniamy nazwę i usuwamy plik (cp, mv, rm)

Utwórzmy ponownie katalogi *katl* i *katl:*

```
$ mkdir katl kat2
[leszekl@mono leszekl]$ ls
katl   kat2
[1eszekl@mono leszekl]$ _
```

Teraz skopiujemy znany nam już plik *letclpasswd* do katalogu *katl* i zmienimy jego nazwę na *naszplik*. Następnie *naszplik* przesuniemy (przeniesiemy) do katalogu *katl:*

```
$ cp  /etc/passwd  katl
[leszekl@mono leszekl]$ ls  katl
passwd
```

```
[leszek1@mono leszekl]$ mv kat1/passwd kat1/naszplik
[leszek1@mono leszekl]$ ls kat1
naszplik
[leszekl@mono leszekl]$ mv kat1/naszplik kat2
[leszekl@mono leszekl]$ ls kat1
[leszekl@mono leszekl]$ ls kat2
naszpli k
[leszekl@mono leszekl]$ _
```

Do kopiowania plików służy polecenie cp (copy). Jego pierwszy argument określa „co", a drugi „dokąd". Polecenie mv (move) służy do przesuwania (przenoszenia) plików między katalogami. Jego pierwszy argument określa „co", a drugi „dokąd". Jeżeli ścieżki dostępu pliku źródłowego i przeznaczenia są identyczne, polecenie mv powoduje zmianę nazwy pliku. Zmianę nazwy skopiowanego pliku możemy również uzyskać poleceniem cp. Tak naprawdę powyższe zabiegi miały jedynie na celu pokazanie funkcjonowania polecenia mv, gdyż efekt docelowy uzyskać mogliśmy od razu jedną linijką z poleceniem cp.

Usuńmy teraz plik *naszplik* i zrealizujmy założone zadanie jeszcze raz w ten właśnie sposób:

```
$ rm -1 kat2/naszplik
rm: remove 'kat2/naszplik'? n
[leszekl@mono leszekl]$ ls kat2
naszplik
[leszekl@mono leszekl]$ rm kat2/naszplik
[leszekl@mono leszekl]$ ls kat2
Ueszekl@mono leszekl]$ cp /etc/passwd kat2/naszpl1k
[leszekl@mono leszekl]$ ls kat2
naszplik
[leszekl@mono leszekl]$ _
```

Przy okazji poznaliśmy niezwykle ważną opcję -i polecenia rm, powodującą wystosowanie komunikatu ostrzegającego przed usunięciem pliku. Opcja ta występuje również w poleceniach cp i mv. Jednak tutaj jej użycie jest może ważniejsze, ponieważ jeżeli w miejscu docelowym kopiowania (przesuwania) pliku istnieje już plik o takiej samej nazwie, zostanie on nadpisany bez żadnego ostrzeżenia, o ile polecenie cp zostało wydane bez opcji -i. Zobaczmy to na przykładzie:

```
$ cat kat2/naszplik
root:4SITEB84oQMeg:0:0:root:/root:/:/bin/bash
bin:*:l:l:bin:/bin:


leszekl:cooH.qocRodK.:500:500::/home/leszekl:/bin/bash
leszek2:hsRKl.hYU2c9E:501:501::/home/leszek2:/bin/bash
[leszekl@mono leszekl]$ cp /etc/group kat2/naszpl1k
[leszekl@mono leszekl]$ cat kat2/naszplik | more
root::0:root
bin: :1: root, bin.daemon
```

```
leszekl:x:500:
leszek2:x:501:
[1eszek1@mono leszekl]$ cp -1  /etc/passwd  kat2/naszplik
cp:  overwrite 'kat2/naszplik' ? y
[1eszek1@mono leszekl]$ cat  kat2/naszpl1k
root:4SITEB84oQMeg:0:0:root:/root:/:/bin/bash
bin:*:l:l:bin:/bin:


leszek1:cooH.qocRodK.:500:500::/home/leszek1:/bin/bash
leszek2:hsRKI.hYU2c9E:501:501::/home/leszek2:/bin/bash
[leszek1@mono leszek1]$ _
```

Polecenie cat służyło nam tu do naocznego porównywania zawartości plików. Dla zaoszczędzenia miejsca, na listingu monitorowym pokazaliśmy tylko dwie pierwsze i dwie ostatnie linie oglądanych poleceniem cat plików. Przypomnijmy sobie sposób sterowania more opisany w ćwiczeniu 1.3.

®° Musimy pamiętać, że polecenie cp powinno być użyte przed każdą modyfikacją kluczowego pliku konfiguracyjnego w celu sporządzenia jego kopii bezpieczeństwa.

Ćwiczenie 2.5. Używamy znaków uogólniających i wzorca

Z systemu MS-DOS znamy zapewne tzw. znaki uogólniające (ang. *wildcards):* „*" oraz „?".

Z ich pomocą tworzyło się wzorzec pasujący do grupy (większej liczby) plików, na przykład **.bat, dane??'98.dbf* itd. W Linuksie te znaki grupowe także występują, ale dodatkowo używamy jeszcze nawiasów kwadratowych i klamrowych, co rozszerza nasze możliwości tworzenia wzorca. Zarówno „*" jak i „?" mają tutaj takie samo znaczenie jak w DOS-ie: gwiazdka oznacza dowolnej (także zerowej!) długości łańcuch znaków, a pytajnik dowolny pojedynczy znak.

Sposób wykorzystania wzorca do generowania „grupowych nazw" plików jest właściwy konkretnej powłoce. W naszym przypadku wszystko, co podamy, dotyczy bash-a. Powłoki mogą się w dziedzinie tzw. rozwijania nazw plików między sobą nieco różnić. W szczególności opisane dalej rozwijanie nawiasów jest unikatową cechą bash-a, której nie znajdziemy w innych powłokach.

Znaki grupowe (ang. *wildcards)* są podzbiorem poznanych w ćwiczeniu 2.2 znaków specjalnych (ang. *metacharacters).*

Na początek utwórzmy kilka dodatkowych plików w katalogu *kat2* co umożliwi nam później ćwiczenie wzorców:

```
$ cd kat2
[leszek1@mono  kat2]$  cp  naszpUk   naszplik1
[leszek1Omono  kat2]$  cp  naszpUk   naszpl1k2
[leszek1@mono  kat2]$  cp  naszpUk   Itennasz
[leszek1Omono  kat2]$  ls
itennasz    naszplik   naszplik1   naszplik2
[leszeklOmono  kat23$ _
```

Skopiujmy (różnymi sposobami) pliki *naszplikl* i *naszplikl* do katalogu *katl:*

```
$ cp   naszplikl   naszpl1k2   ../ka1
[leszek1@mono  kat2]$  ls    ../ka1
naszplikl   naszplik2
[leszek1@mono  kat2]$ rm   ../kat1/*
[leszek1@mono  kat2]$  ls   ../kat1
[leszek1@mono  kat2]$ cp  naszpUk*   ../kat1
[leszek1@mono  kat2]$  ls   ../kat1
naszplik    naszplikl   naszplik2
[leszek1@mono  kat2]$ rm   ../kat1/*
[leszek1@mono  kat2]$ cp  naszpUk?   ../kat1
[leszek1@mono  kat2]$  ls   ../kat1
naszplikl   naszplik2
[leszek1@mono  kat2]$  rm  ../kat1/*
[leszek1@mono  kat23$  ls   ../kat1
[leszek1@mono  kat2]$  cp   naszplik[12]   ../kat1
[leszek1@mono  kat2]$  ls   ../kat1
naszplik1   naszplik2
[leszek1@mono  kat2]$ rm   ../kat1/*
[leszek1@mono  kat2]$  ls   ../kat1
[leszek1@mono  kat2]$  cp   naszplik[1-2]   ../kat1
[leszek1@mono  kat2]$  ls   ../kat1
naszplik1   naszplik2
[leszek1@mono  kat2]$ rm   ../kat1/*
[leszek1@mono  kat2]$  ls   ../kat1
[leszek1@mono  kat2]$ _
```

W pierwszym przypadku użyliśmy polecenia cp z jawnie wyspecyfikowanymi plikami. W pozostałych przypadkach posłużyliśmy się wzorcami. Użycie gwiazdki okazało się tu nieprawidłowe, gdyż uwzględnia ona także zerowej długości łańcuch (dlatego skopiowany został także *naszplik).* Użycie pytajnika tylko przypadkowo dało prawidłowy rezultat (ponieważ katalog *kat2* nie zawierał np. pliku *naszplikl).* Jedyne prawidłowe wzorce to te z użyciem nawiasów kwadratowych. Zapis [12] oznaczał: kończący się cyfrą „1" albo cyfrą „2". Zapis [1-2] oznaczał: kończący się dowolną cyfrą z zakresu od „1" do „2" włącznie.

Teraz skopiujmy do *katl* wszystkie pliki, których nazwa zaczyna się na „i", „c", albo „n":

```
$ cp naszplik plik
[leszek1@mono  kat2]$  cp   naszplik   costutaj
[leszek1@mono  kat2]$  ls
```

```
costutaj  itennasz  naszplik  naszplik1  naszplik2  plik
[leszek1Omono kat2]$ cp [inc]* ../katl
[leszek1@mono kat2]$ ls ../katl
costutaj  itennasz  naszplik  naszplikl  naszplik2
[leszek1@mono kat2]$ rm ../kat1/*
[leszek1@mono kat2]$ ls ../kat1
[leszek1@mono kat2]$ _
```

Dla zapewnienia możliwości weryfikacji naszego eksperymentu utworzyliśmy w katalogu *katl* dodatkowo dwa pliki: *plik* (nie pasujący do wzorca) oraz *costutaj* (pasujący do wzorca).

Bash oferuje ponadto bardzo użyteczny mechanizm rozwijania nawiasów. Do jego zapisu używamy nawiasów klamrowych { }:

```
$ cp *Csz,cos3* ../katl
[leszek1@mono kat2]$ ls ../kat1
costutaj  itennasz  naszplik  naszplikl  naszplik2
[leszek1@mono kat2]$ rm ../kat1/*
[leszek1@mono kat2]$ ls ../kat1
[leszek1@mono kat2]$ cp *(1.2) ../kat1
[leszek1@mono kat2]$ ls ../kat1
naszplik1  naszplik2
[leszek1@mono kat2]$ rm ../kat1/*
[leszek1@mono kat2]$ ls ../kat1
[leszek1@mono kat2]$ cp *(c.e,k)* ../kat1
[leszek1@mono kat2]$ ls ../kat1
costutaj  itennasz  naszplik  naszplik1  naszplik2  plik
[leszek1@mono kat2]$ rm ../kat1/*
[leszek1@mono kat2]$ ls ../kat1
[leszek1@mono kat2]$ _
```

Zastosowanie mechanizmu rozwijania nawiasów jest w powyższym przykładzie banalnie proste, ale myślę, że czujemy już jego siłę i użyteczność. Zapis ,,*{sz,cos}*" oznacza: wszystkie pliki mające w nazwie łańcuch „sz" lub „cos". Zapis ,,*{1,2}" oznacza: wszystkie pliki, których nazwa kończy się cyfrą „1" lub cyfrą „2". Wreszcie zapis ,,*{c,e,k}*" oznacza: wszystkie pliki, które mają w nazwie literę „c", „e" lub „k".

Na zakończenie zapoznajmy się z użyciem wykrzyknika „!", jako negacji:

```
$ cp *C!1] ../katl
[leszek1@mono kat2]$ ls ../kat1
costutaj  itennasz  naszplik  naszplik2  plik
[leszek1@mono kat2]$ rm ../kat1/*
[leszek1@mono kat2]$ ls ../kat1
[leszek1@mono kat2]$ cp naszplik[!1] ../kat1
[leszek1@mono kat2]$ ls ../kat1
naszplik2
[leszek1@mono kat2]$ rm ../kat1/*
```

```
[leszek1@mono kat2]$ ls  ../kat1
[leszek1@mono  kat2]$
```

Zapis ,,*[!1]" oznacza tutaj wszystkie pliki za wyjątkiem tych, których ostatnim znakiem nazwy jest cyfra „1". Natomiast zapis ,,naszplik[!1]" oznacza pliki, których nazwa składa się z łańcucha „naszplik" plus jeszcze jednego (nie pustego) znaku. Tym ostatnim znakiem nie może być jednak cyfra „1".

Ćwiczenie 2.6. Kopiujemy i przenosimy katalogi (cp, mv)

Kopiowanie i przenoszenie katalogów odbywa się nieco inaczej niż w przypadku plików, chociaż używamy tych samych poleceń. Zadania realizowane w praktyce dotyczą często całego drzewa, czyli katalogu zawierającego podkatalogi i umieszczone w nich pliki:

```
$ cd ~
[leszek1@mono leszek1]$ cp  kat2  kattest
cp:  kat2: omitting directory
[leszek1@mono leszek1]$ cp -R  kat2  kattest
[leszeklOmono leszekl]$ ls  -F
kat1/   kat2/  kattest/
[leszek1@mono leszek1]$ ls  kattest
costutaj  itennasz  naszplik  naszplik1  naszplik2  plik
[leszek1@mono leszek1]$ cp  -R  kat2   kattest/
[leszek1@mono leszekl]$ ls  -F  kattest
costutaj itennasz kat2/ naszplik naszplikl naszplik2 plik
[leszek1@mono leszek1]$ ls  -F  kattest/kat2
costutaj  itennasz  naszplik  naszplikl  naszplik2  plik
[leszekl@mono leszek1]$ mv  kattest  kat1
[leszekl@mono leszekl]$ ls  -F
kat1/   kat2/
[leszek1@mono leszek1]$ ls -F  kat1
kattest/
[leszek1@mono leszek1]$ ls  -F  kat1/kattest
costutaj itennasz kat2/ naszplik naszplik1 naszplik2 plik
[leszek1@mono leszek1]$ mv   kat1/kattest   kat1/katinny
[leszek1@mono leszek1]$ ls  -F  kat1
katinny/
[leszekl@mono leszek1]$ ls  -F  kat1/katinny
costutaj itennasz kat2/ naszplik naszplik1 naszplik2 plik
[leszek1@mono leszek1]$ cp -R  kat1  kat1/katlnny
[leszek1@mono leszek1]$ ls  -F  kat1
katinny/
[leszeklOmono leszek1]$ ls  -F  kat1/katinny
costutaj       kat1/       naszplik     naszplik2
itennasz       kat2/       naszplik1    plik
[leszek1@mono leszek1]$ rm  -r   kat1/*
[1eszek1@mono leszek1]$ ls  -F  katl
[leszek1@mono leszek1]$ _
```

Skomentujmy teraz to, co zostało zrobione. Katalogi możemy kopiować za pomocą polecenia cp z opcją -R (recursively). Zwróćmy jednak uwagę na specyfikę zastosowania poleceń cp i mv do katalogów. Próba skopiowania (cp) lub przesunięcia (mv) katalogu na istniejący katalog nie powoduje jego „nadpisania" (jak miałoby to miejsce w przypadku pliku zwykłego), lecz umieszczenie kopiowanego (przesuwanego) katalogu wewnątrz wskazanego katalogu docelowego. W wyniku polecenia cp lub mv nie możemy „stracić" katalogu. Gdyby bowiem takie nadpisanie mogło mieć miejsce, zachwiana zostałaby spójność systemu plików, gdyż podkatalogi i pliki znajdujące się w „nadpisanym" katalogu straciłyby „punkt dowiązania" (przestałaby istnieć ścieżka dostępu do nich).

Na koniec, zwróćmy jeszcze uwagę na postać polecenia (rm -r kat/*), usuwającego (rekurencyjnie) z katalogu *katl* wszystkie pliki i katalogi.

Ćwiczenie 2.7. Tworzymy pusty plik (touch)

Aby utworzyć nowy pusty plik, używamy polecenia **touch,** na przykład:

```
$ cd ~
[leszek1@mono leszek1]$ touch nowy
[leszek1@mono leszek1]$ ls
kat1        kat2      nowy
[leszek1@mono leszek1]$ cat nowy
[leszek1@mono leszek1]$ _
```

Polecenie **touch** w istocie służy do zmiany czasu i daty (ang. *timestamps)* utworzenia (modyfikacji) pliku. Jeżeli użyjemy je tak jak powyżej, ale w odniesieniu do istniejącego już pliku, jego czas i data zostaną zmienione na aktualną datę i czas systemowy.

Sprawdzimy te parametry pliku *nowy:*

```
$ ls -1 nowy
-rw-rw-r--  1 leszekl  leszekl     0   Apr 16   09:17  nowy
[leszek1@mono leszek1]$ _
```

Plik nowy został utworzony 16 kwietnia (Apr 16) o godzinie dziewiątej siedemnaście (09:17). Poczekajmy cierpliwie kilka minut i ponownie wydajmy polecenie **touch,** a następnie sprawdźmy czas utworzenia pliku:

```
$ touch nowy
[leszek1@mono leszekl]$ ls -1 nowy
-rw-rw-r--  1   leszek1   leszek1   0   Apr 16   09:21  nowy
[leszek1Omono leszekl]$ rm nowy
[1eszek1@mono leszekl]$ _
```

Ćwiczenie 2.8. Określamy typ pliku (file)

Jak pamiętamy z rozdziału trzeciego, wszystkie pliki w Linuksie mają ten sam format fizyczny, a klasyfikowanie ich odbywa się na podstawie przeznaczenia. Gdy spotkamy się z nieznanym plikiem, możemy spróbować sklasyfikować go poleceniem **file:**

```
$ file /dev/hdc /etc/passwd /dev/tty3 /proc/memlnfo /sbln/shutdown
/dev/hdc:                 błock special
/etc/passwd:              ASCII text
/dev/tty3:                character special
/proc/meminfo             empty
/sbin/shutdown:           ELF 32-bit LSB executable, Intel 80386,
                          version 1, dynamically linked, stripped
[leszekl@mono  leszekl]$'file  /usr/sb1n/makewhat1s
/usr/sbin/makewhatis:          Bourne shell script text
CleszeklOmono  leszekl]$ ls -1  /proc/memlnfo
-r--i—r--   i ro ot   root   0 Apr 15 21:10 /proc/meinfo
Cleszekl@mono  leszekl]$ cat  /proc/memlnfo
        total:      used:      free:    shared:  buffers:  cached:
Mem:  31563776  15929344  15634432   9076736   987136  11030520
Swap: 34058240         0  34058240
MemTotal:       30824 kB
MemFree:        15268 kB
MemShared:       8864 kB
Buffers:          964 kB
Cached:         10772 kB
SwapTotal:      33260 kB
SwapFree:       33260 kB
[1eszekl@mono  leszekl]$ _
```

Polecenie **file** klasyfikuje pliki w oparciu o ich cechy charakterystyczne, w przypadkach typowych dając poprawne wyniki. W naszym przykładzie występują pliki zwykłe tekstowe (ang. *ASCII text),* plik specjalny znakowy (ang. *character special),* specjalny blokowy (ang. *błock special),* plik wykonywalny binarny (ang. *ELF,..executable)* oraz skrypt powłoki (ang. *Bourne shell script text).*

W przypadkach nietypowych do odpowiedzi udzielanych przez **file** musimy mieć ograniczone zaufanie. Przykładem jest tu plik */proc/meminfo* określony przez **file** jako pusty (ang. *empty).* Pustego pliku polecenie **file** nie jest w stanie sklasyfikować, gdyż rozpoznaje ono pliki na podstawie ich zawartości (albowiem format fizyczny mają przecież ten sam) w oparciu o reguły zawarte w pliku *letc/share/magic.* Również polecenie ls podaje zerową długość pliku */proc/meminfo.* Tymczasem poleceniem **cat** możemy bez problemów obejrzeć sobie ten „pusty" plik, zawierający bardzo ciekawą informację dotyczącą wykorzystania pamięci systemu. Pliki, które możemy znaleźć **w** katalogu */proc,* są rzeczywiście wyjątkowe. Są to bowiem pliki nie tyle fizyczne, co wirtualne. Więcej o katalogu *proc* powiemy w ćwiczeniu 12.5.

Ćwiczenie 2.9. Porównujemy pliki (cmp)

Często przy robieniu porządków na dysku spotykamy się z koniecznością sprawdzenia, czy dwa pliki o różnych nazwach nie są aby identyczne, co do zawartości. Pozwoliłoby to usunąć jeden z nich. Prostym poleceniem służącym do tego celu jest **cmp.** Przećwiczymy to na przykładzie:

```
$ cp /etc/passwd plikcmp
[leszekl@mono leszek1]$ cmp /etc/passwd plikcmp
[leszekl@mono leszek1]$ cp /etc/fstab plikcmp
[leszekl@mono leszek1]$ cmp /etc/passwd plikcmp
/etc/passwd plikcmp differ: char 1, line 1
[leszekl@mono leszek1]$ cmp -c /etc/passwd plikcmp
/etc/passwd plikcmp differ: char 1, line 1 is 162 r 57 /
Cl eszekl@mono leszek1]$ cmp -cl /etc/passwd plikcmp


379 157 o 60 0
380 162 r 12 ^J
cmp: EOF on plikcmp
[leszek1@mono leszek1]$ rm plikcmp
[leszek1@mono leszek1]$ cmp -c /sbln/shutown /b1n/ls
/sbin/shutown /bin/ls differ: char 25, linę 1 is 274 M-< 220 M-^P
[leszek1@mono leszek1]$ _
```

Skopiowaliśmy tu *letclpaswwd* pod nazwą *plikcmp* do naszego katalogu domowego i następnie dokonaliśmy porównania **(cmp)** */etc/passwd* z *plikcmp*. Ponieważ pliki są identyczne, nie otrzymaliśmy żadnego komunikatu o błędzie (rezultacie porównania). Następnie pod nazwą *plikcmp* skopiowaliśmy plik */etc/fstab* i w wyniku kolejnego porównania otrzymaliśmy komunikat o błędzie, czyli informację o tym, że pliki się różnią.

Polecenie **cmp** porównuje kolejno znak po znaku (bajt po bajcie) zawartość plików i zatrzymuje się, gdy napotka pierwszą różnicę. W naszym przypadku nastąpiło to już przy pierwszym znaku (ang. *char 1)* znajdującym się oczywiście w pierwszej linii (ang. *line 1).*

Używając opcji -c (character) otrzymujemy jawne wyspecyfikowanie, o jakie znaki ASCII chodzi (podawany jest kod ASCII i odpowiadający mu wydruk znaku np. 162 i „r"). Możemy sprawdzić **(cat /etc/passwd),** że pierwszy znak pliku *passwd* to ,,r", a także **(cat /etc/fstab),** że pierwszy znak pliku *fstab* to „/".

Używając opcji -l (list) powodujemy, że **cmp** nie zatrzyma się przy pierwszej napotkanej różnicy, lecz doprowadzi porównanie do końca (krótszego z plików oczywiście), z wylistowaniem wszystkich napotkanych różnic. W naszym przypadku krótszy *jest fstab,* a komunikat „EOF on plikcmp" oznacza napotkanie znaku końca pliku (EOF - End Of File) w pliku *plikcmp.*

Polecenie **cmp** służy do prostego porównywania plików, zwłaszcza tekstowych. Możemy go również używać do porównywania (w granicach rozsądku) dowolnych typów plików, w szczególności wykonywalnych (tutaj *shutdown* i ls*)*. Nie próbujmy tego jednak z plikami specjalnymi (np. */dev/fd0),* gdyż reprezentują one urządzenia i otrzymamy niespodziewane efekty.

Polecenie **cmp** jest prostym narzędziem do prostych zadań. Zaawansowane porównywanie plików dokonywane jest przez **diff,** ale o nim powiemy w ćwiczeniu 7.4.

Ćwiczenie 2.10. Poszukujemy pliku (find)

W tym ćwiczeniu omówimy **find** (szukaj), które jest klasycznym (standardowym) i solidnym poleceniem między innymi do wyszukiwania plików i zasługuje na znacznie szersze potraktowanie. Tutaj ograniczymy się tylko do zademonstrowania ułamka jego możliwości.

Polecenie **find** prowadzi wyszukiwanie według zadanych kryteriów od wskazanego katalogu „w dół", uwzględniając wszystkie podkatalogi (rekurencyjnie). W szczególności jeżeli wskazanym katalogiem jest root (/), wówczas przeszukiwany jest cały system plików.

Przeprowadźmy kilka przeszukiwań w katalogu domowym użytkownika *leszekl.* Musimy pamiętać, że tylda „~" jest synonimem katalogu domowego użytkownika:

```
$ find ~ -name passwd
[leszekl@mono leszek1]$ find ~ -name "p*"
/home/leszek1/kat2/plik
[leszekl@mono leszek1]$ find ~ -name itennasz
/home/1eszek1/kat2/itennasz
[leszekl@mono leszek1]$ _
```

W powyższym zapisie tylda „~" oznacza katalog domowy użytkownika (czyli punkt rozpoczęcia poszukiwań), **-name** - to opcja określająca wyszukiwanie według nazwy. Pliku *passwd* nie znaleziono. Znaleziono natomiast jeden plik *(plik),* którego nazwa zaczyna się na „p" (wzorzec p*) i dokładnie jeden plik o nazwie *itennasz.*

Jeżeli nie podamy katalogu rozpoczęcia poszukiwań, **find** przyjmie domyślnie katalog bieżący.

Jeśli korzystamy ze wzorca z użyciem znaków grupowych (wykorzystywanych przez powłokę do rozwijania nazw plików), musimy go zacytować, czyli umieścić w cudzysłowie lub pomiędzy znakami apostrofu (pojedynczego cudzysłowu).

Polecenie **find** jest bardzo użyteczne w rękach administratora. By się o tym przekonać, przejdziemy na inną (np. **[lewy Alt]+[F4])** konsolę wirtualną, zalogujemy się jako *root* i będziemy kontynuować poszukiwania:

```
# find / -name passwd
/etc/passwd
/etc/pam.d/passwd
```

```
/usr/bin/passwd
[root@mono /root]# find /etc -name passwd
/etc/passwd
/etc/pam.d/passwd
[root@mono /root]# _
```

Przeszukaliśmy najpierw cały system plików (/), znajdując trzy pliki o nazwie *passwd*. W kolejnym poleceniu zawęziliśmy poszukiwania do katalogu */etc,* znajdując (co było do przewidzenia) tylko dwa pliki. Ale **flnd** może dać odpowiedź na bardziej skomplikowane pytania:

```
# find / -user leszek1 | more
```

W wyniku wykonania powyższego polecenia zostanie przeszukany cały system plików (/), a na ekran zostaną wyprowadzone pełne ścieżki dostępu do tych plików, których właścicielem jest użytkownik *leszekl* **(-user leszek1).** Ponieważ lista nie zmieści się na jednym ekranie, zastosowaliśmy dodatkowo **more.**

Wyprzedzając nieco wypadki, należy w tym miejscu stwierdzić, że każdy plik w Linuksie ma swojego właściciela. Domyślnie jest nim użytkownik, który utworzył plik. Prawo własności pliku może zostać przeniesione na innego użytkownika (patrz lekcja 5). W ramach lekcji 5 zobaczymy, że tak jak w świecie rzeczywistym, z prawem własności wiążą się ściśle określone możliwości dysponowania swoją własnością, które mogą niestety być poważnie ograniczone. Zwróćmy uwagę, jak (nadspodziewanie) wielu plików właścicielem jest *leszekl.* Wiele z tych plików to pliki ukryte (z nazwą zaczynającą się od kropki).

Ćwiczenie 2.11. Łączymy pliki (cat)

Używaliśmy już polecenia **cat** do oglądania zawartości pliku tekstowego, ale tak naprawdę zostało ono stworzone do łączenia (concatenate) plików.

Powróćmy teraz na konsolę użytkownika *leszek1* i rozpocznijmy ćwiczenie:

```
$ cp kat2/* kat1
[leszekl@mono leszekl]$ Is katl
costutaj   itennasz   naszplik   naszplikl   naszplik2   plik
[leszekl@mono leszekl]$ _
```

To było tylko przygotowanie plików do zabawy w katalogu *kat 1.* Teraz przeprowadzimy właściwe eksperymenty. Najpierw obejrzyjmy pliki *naszplikl* i *naszplikl,* a następnie spróbujemy je połączyć:

```
$ cd kat1
[leszek1@mono katl]$ cat naszplikl


[leszek1@mono katl]$ cat naszpl1k2
```

```
[leszek1@mono kat1]$ cmp naszplik1 naszplik2
[leszek1@mono kat1]$ cat naszplik1 naszplik2 | more


[q]
[leszek1@mono kat1]$ _
```

Listing zawartości plików pominęliśmy tutaj, gdyż nie on jest istotny, a zajmuje sporo miejsca. Łatwo zauważyliśmy, że pliki *naszplik1* i *naszplik1* są identyczne. Potwierdza to dodatkowo rezultat ich porównania poleceniem **cmp**. Są one bowiem wykonanymi przez nas w poprzednich ćwiczeniach kopiami pliku */etc/passwd.* Plik powstały z połączenia mogliśmy obejrzeć tylko na ekranie. Wynika to z faktu, że polecenie cat przekazuje rezultat swojego działania na standardowe wyjście (ekran). Wciskamy klawisz [q] w celu opuszczenia more.

Aby utworzyć na dysku plik będący połączeniem plików wejściowych, musimy skorzystać z operatora przełączającego standardowe wyjście do pliku. Nasze polecenie będzie wyglądało tak:

```
$ cat naszplik1 naszplik2 > suma
[leszek1@mono kat1]$ ls
costutaj itennasz naszplik naszplik1 naszplik2 plik suma
[leszek1@mono kat1]$ cmp naszplik1 suma
cmp: EOF on naszplik1
[leszek1@mono kat1]$ cat suma | more


[q]
[leszek1@mono kat1]$_
```

Powstał plik *suma,* który jest „sumą" plików *naszplikl* i *naszplikl* (w istocie podwojonym plikiem */etc/passwd).* Zwróćmy uwagę na rezultat polecenia cmp. Ponieważ *suma* to inaczej podwojony plik *naszplik1*, zatem cmp prowadziło porównanie do napotkania końca (EOF) krótszego pliku i zakończyło pracę (nie znalazłszy po drodze różnic).

Spróbujmy teraz połączyć pliki używając jako nazwy pliku wynikowego:

```
$ cat naszplik1 naszplik2 > naszplik1
cat: naszplik1: input file is output file
[leszek1@mono kat1]$ cat naszplik1 naszplik2 > costutaj
[leszek1@mono kat1]$ cmp suma costutaj
[leszek1@mono kat1]$ rm *
[leszek1@mono kat1]$ cd ~
[leszek1@mono leszek1]$ _
```

Komunikat o błędzie: „plik wejściowy jest plikiem wynikowym" (ang. *input file is output file)* jest istotny. Zrozumiemy to, gdy poznamy za chwilę mechanizm łącze-

nia. Natomiast wykorzystanie innego pliku wynikowego (już istniejącego wcześniej *costutaj)* odbyło się bez problemów. Został on po prostu nadpisany.

Wyjaśnijmy teraz przyczynę komunikatu o błędzie. Ogólnie polecenie połączenia możemy zapisać tak:

```
cat plik1 plik2 > pliksuma
```

Jeżeli plik *pliksuma* nie istnieje, polecenie **cat** utworzy go i umieści w nim „sumę": *plik1* + *plik2.* Jeżeli zaś *pliksuma* już istnieje, zostanie on najpierw opróżniony (wyczyszczony) i do dopiero wtedy będzie umieszczona w nim „suma": *pliki* + *pliki.*

W naszym konkretnym przypadku (plikiem wynikowym miał być *naszplik1)* oznaczałoby to, że najpierw plik *naszplik1* stałby się plikiem pustym, a dopiero potem zostałby połączony z plikiem *naszplik2.* Operacja taka nie tylko nie ma sensu, (gdyż rezultatem zawsze będzie zawartość *naszplik1),* ale w jej wyniku stracilibyśmy bezpowrotnie zawartość pliku *naszplik1.*

Ćwiczenie 2.12. Oglądamy pliki (more, less)

Do oglądania zawartości dużych (tj. nie mieszczących się na jednym ekranie) plików tekstowych używamy najczęściej poleceń **more** i **less.** Stosowane są one także często do stronicowania „wydruków monitorowych" uzyskiwanych z innych poleceń (np. ls). Jeżeli korzystaliśmy z MORE w DOS-ie, to tutaj też nie będziemy mieli problemów. Polecenie **less,** którego nazwa w specyficznie żartobliwy sposób nawiązuje do **more,** jest funkcjonalnym rozwinięciem **more.** Osobiście preferuję less.

Podstawowe operacje klawiszowe w obu poleceniach są takie same:

[spacja] - następny ekran (przewinięcie tekstu o stronę)

[Enter] - następny wiersz (przewinięcie tekstu o jeden wiersz)

[q] - zakończenie pracy

[h] - ekran „pomocy" z opisem komend wewnętrznych polecenia (ang. *help)*

[b] - poprzedni ekran (przewinięcie tekstu o stronę wstecz)

[d] - pół ekranu do przodu (przewinięcie tekstu o pół strony).

Ponadto **less** umożliwia płynne przewijanie tekstu do przodu i wstecz za pomocą klawiszy kursora (góra/dół) oraz wykorzystuje klawisze **[Page Up]** i **[Page Down].**

Utwórzmy długi plik:

```
$ cd kat2
[leszek1@mono  kat2]$ cat  plik  plik  plik > suma
[leszek1@mono  kat2]$ _
```

Plik ten będziemy mogli wykorzystać do samodzielnego przetrenowania operacji klawiszowych poleceń **less** i **more:**

```
$ more suma


[q]
[1eszek1@mono  kat2]$ less suma


[q]
[leszek1@mono  kat2]$ _
```

Naciskamy klawisz [**q**] (quit), aby opuścić **less (more).** Wyświetlając opis pomocy (przez naciśnięcie klawisza [**h**]) po uruchomieniu **more** bądź **less,** zauważymy, że oba polecenia mają znacznie więcej możliwości niż przedstawiamy tutaj. Polecenie **more** jest klasycznym poleceniem występującym we wszystkich znanych systemach uniksowych, lecz w Linuksie zrealizowane jest w skromnym zakresie funkcjonalnym. Za to bardzo rozbudowane są możliwości **less** i dlatego warto się mu przyjrzeć bliżej. Więcej na temat **less** powiemy w ćwiczeniu 8.3.

Oba polecenia możemy użyć w roli filtru do stronicowania rezultatów innych poleceń. W tym przypadku **more** nie daje możliwości przewijania do tyłu (klawisz [**b**] „nie działa"), natomiast **less** zachowuje swoje wszystkie walory. Możemy to sprawdzić:

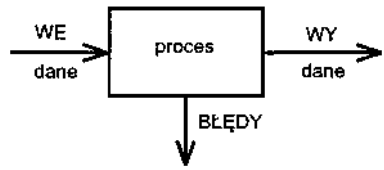
```
$ cat  suma | more


[d], [d], [b]
[q]
[leszek1@mono  kat2]$ cat suma | less


[d], [d], [b]
Cq]
[leszek1@mono   kat2]$ rm suma ; cd ~
[leszek1@mono   leszek1]$
```

Na końcu pokazaliśmy przykład wprowadzenia dwóch poleceń **(rm** i cd) w jednej linii. Oddzielone są od siebie średnikiem. Jest to separator poleceń. Może ich być w jednym wierszu więcej niż dwa.

Lekcja 3. Strumienie, potoki, filtry i sygnały

Tytuł niniejszej lekcji mógłby pojawić się w publikacji na temat hydrologii, hydrauliki, ochrony środowiska czy szeroko rozumianego zdrowia (jedynie sygnały nie bardzo tu pasują). Bez wody nie ma życia, a umiejętność optymalnego korzystania z niej jest niezwykle cenna. Także i w przypadku Linuksa, dobre opanowanie jego .,hydrologii i hydrauliki" pozwoli na mistrzowskie wykorzystywanie możliwości systemu.

Każdemu procesowi w Linuksie są przypisane trzy strumienie danych: wejściowy, wyjściowy i błędów (komunikatów o błędach). Każdy proces (rys. 1) w momencie uruchomienia musi mieć określone miejsca, z których pobiera dane wejściowe, do których przekazuje dane będące efektem działania procesu i do których wysyła komunikaty o błędach (czyli ogólnie mówiąc komunikaty „techniczne" dotyczące wykonywania samego procesu).



Rysunek 1. Proces i jego strumienie

Przykładowo proces: **ls /var** pobiera dane z katalogu */var,* dane wyjściowe przekazuje na ekran monitora, a komunikaty o błędach też na ekran monitora.

Jako rezultat zalogowania się użytkownika w systemie zostaje uruchomiony przypisany mu interpreter poleceń systemowych (ang. *shell),* czyli powłoka. W naszym przypadku jest to **bash.** Domyślnym strumieniem danych wejściowych dla **bash-a** jest klawiatura. Natomiast dane wyjściowe i komunikaty o błędach są przekazywane na ekran. Polecenia uruchamiane z linii poleceń **bash**-a dziedziczą po nim te domyślne przypisania (chociaż nie każde polecenie z nich musi korzystać).

Jak wiemy Linux „zawsze i wszędzie" operuje na plikach. Zatem strumień wejściowy danych uzyskujemy z otwartego określonego pliku na wejściu procesu, strumień wyjściowy przez pisanie do określonego pliku otwartego na wyjściu procesu i wreszcie trzeci plik musimy otworzyć w celu wpisywania tam komunikatów o błędach.

Każdy uruchomiony w systemie proces może otworzyć jednocześnie na swoje potrzeby do dwudziestu plików. Otwarty plik proces identyfikuje poprzez przyporządkowaną mu liczbę całkowitą zwaną deskryptorem pliku.

Trzy liczby (0, 1, 2) są zarezerwowane i odpowiadają trzem plikom, które są zawsze otwierane w momencie utworzenia procesu:

deskryptor 0 - *stdin* (standard input) - wejście standardowe

deskryptor 1 - *stdout* (standard output) - wyjście standardowe

deskryptor 2 - *stderr* (standard error) -- standardowe wyjście błędów (diagnostyczne).

Dla wszystkich poleceń (programów) uruchamianych w ramach powłoki (czyli u nas bash-a) domyślne przypisania tych plików są następujące:

stdin - strumień znaków z klawiatury terminala

stdout - strumień znaków wysyłany na ekran terminala (monitora)

stderr - także strumień znaków wysyłany na ekran terminala (monitora).

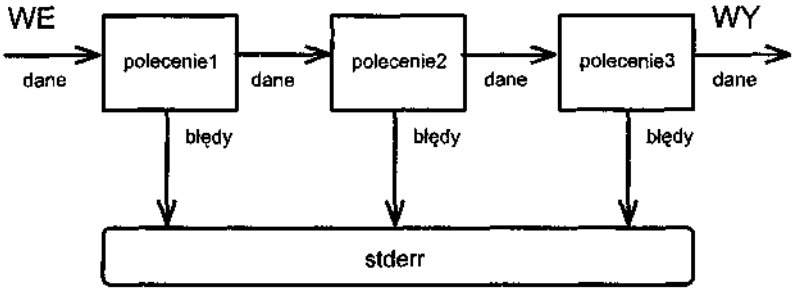
Stosownymi operatorami możemy deskryptory **0, 1, 2** przypisać innym plikom (a nie tylko reprezentującym klawiaturę i ekran), uzyskując w ten sposób tzw. przełączenie wejścia (wyjścia) standardowego, ale o tym powiemy za chwilę.

Polecenie nie musi korzystać z wejść i wyjść standardowych, na przykład ls nigdy nie pobiera danych z klawiatury (lecz zawsze z jakiegoś katalogu, choćby bieżącego).

Natomiast polecenie cat wydane bez żadnego argumentu (i opcji) będzie pobierało dane z klawiatury (standardowe wejście) i natychmiast wypisywało je na ekran (standardowe wyjście). Aby zakończyć tę, z reguły niespodziewaną, sytuację musimy nacisnąć **[Ctrl]+[c]** (czyli przerwać cat, co jest dość barbarzyńskim, ale skutecznym działaniem) lub nacisnąć **[Ctrl]+[d]** (czyli zakończyć działanie cat, co jest bardziej eleganckim rozwiązaniem). **[Ctrl]+[d]** wprowadza znak końca pliku (tzw. EOF - niewidoczny na ekranie). Polecenie **cat** pobiera dane z pliku *stdin* (czyli klawiatury) i wysyła je na ekran (czyli do pliku *stdout),* a gdy „napotka" znak końca pliku to kończy działanie, gdyż oznacza to, że już cały plik *stdin* został przetworzony.

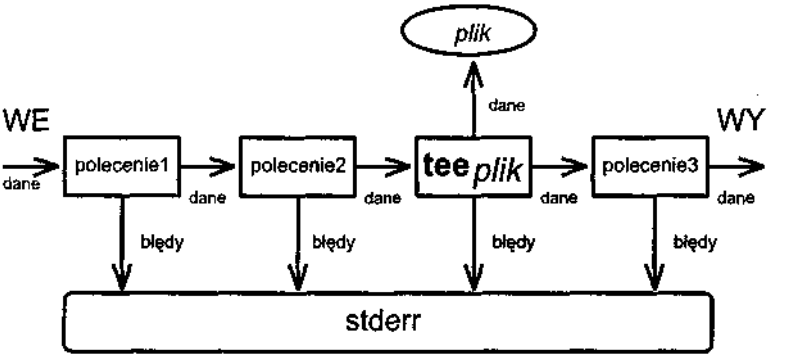
Jeśli polecenie „nie wie", z jakiego wejścia/wyjścia ma korzystać, jest wysoce prawdopodobne, że będzie próbowało ze standardowego. Zachowanie **cat** jest dobrym tego przykładem.

Łatwo wyobrazić sobie konstrukcję (rys. 2) złożoną z wielu poleceń, w której strumień wyjściowy pierwszego polecenia jest dołączony do strumienia wejściowego drugiego polecenia, z kolei strumień wyjściowy drugiego polecenia dołączony jest do strumienia wejściowego trzeciego polecenia itd. Konstrukcję taką nazywamy potokiem (ang. *pipe).*



Rysunek 2. Potok

Często oprócz rezultatu końcowego potoku, interesują nas rezultaty (dane) pośrednie, które na przykład chcielibyśmy zachować w pliku do późniejszego wykorzystania. Wymaga to w wybranym przez nas miejscu potoku rozdzielenia strumienia danych. Służy do tego polecenie tee (rys. 3).



Rysunek 3. Rozdzielenie strumienia danych w potoku

Pojęcie filtru łatwiejsze jest do intuicyjnego uchwycenia niż do precyzyjnego (i prostego zarazem) zdefiniowania, gdyż częściej służy do kontekstowego określenia funkcji danego polecenia w naszym konkretnym działaniu, niż do akademickiego szufladkowania poleceń na filtry i inne.

Jako filtry traktować będziemy polecenia, które:

- są w pełni podatne na działanie mechanizmu przełączania wejścia/wyjścia standardowego
- pobierają dane zawsze z wejścia standardowego *(stdin),* wysyłają je po przetworzeniu na wyjście standardowe *(stdout),* a komunikaty o błędach do standardowego wyjścia błędów *(stderr)*
- oczekują na wejściu prostego strumienia bajtów (znaków) i takiż strumień (po własnym przetworzeniu) wysyłają na wyjście.

Dzięki powyższym cechom, filtry łatwo jest organizować w potoki.

Zgodnie z powyższą (roboczą i „intuicyjną") definicją, polecenie **cat** jest filtrem, natomiast polecenie ls nie. Polecenie ls ma bowiem ograniczoną podatność na działanie mechanizmu przełączania wejścia/wyjścia, gdyż dane pobiera zawsze z katalogu (bieżącego lub wskazanego). Ponadto, w strumieniu wejściowym „doszukuje" się odpowiedniego formatu odpowiadającego katalogowi i według tego oczekiwanego formatu będzie usiłowało interpretować i przetwarzać strumień wejściowy.

Generalnie, filtry przeznaczone są przede wszystkim do operowania na plikach tekstowych.

Właściwe uchwycenie istoty filtru ułatwi nam później tworzenie prawidłowo działających potoków oraz posługiwanie się filtrami, które w ręku sprawnego użytkownika są nieprawdopodobnie uniwersalnym i wydajnym narzędziem.

Tyle wstępu. Zalogujmy się w systemie jako *leszekl* i przystąpmy do ćwiczeń.

Ćwiczenie 3.1. Manipulujemy wyjściami i wejściem standardowym

Przełączając wyjście standardowe polecenia **cat** do pliku, możemy łatwo utworzyć krótki plik tekstowy o zadanej nazwie:

```
$ ls -F
kat1/    kat2/
[leszek1@mono  leszekl]$ cat  >  zapis
to jest zapis testowy, [Enter]
[Ctrl]+[d]
[leszek1@mono  leszek1]$  ls  -F
katl/    kat2/  zapis
[leszek1@mono  leszek1]$  cat  zapis
to jest zapis testowy
[leszek1@mono  leszek1]$  _
```

Operator „>" oznacza przełączenie wyjścia standardowego do wskazanego pliku, w tym przypadku do pliku *zapis,* który zamierzamy utworzyć. W bieżącym katalogu zostaje otworzony plik *zapis,* a polecenie **cat** czeka na dane wprowadzane z klawiatury. Po wprowadzeniu tekstu „to jest zapis tekstowy", naciskamy **[Enter],** co powoduje jedynie przejście do nowej linii w nowo tworzonym pliku. Aby zakończyć akcję (zamknąć plik *zapis),* musimy wcisnąć kombinację klawiszy **[Ctrl]+[d],** która powoduje wprowadzenie znaku końca pliku (EOF).

Idźmy dalej:

```
$ cat zapis > nowy
[leszek1@mono  leszek1]$  ls -F
```

```
kat1/    kat2/  zapis  nowy
[leszekl@mono  leszek1]$ cat  nowy
to jest zapis testowy
[leszekl@mono  leszek1]$ rm zapis nowy
[leszekl@mono  leszek1]$ _
```

W tym przypadku polecenie **cat zapis > nowy** jest równoważne ze skopiowaniem pliku *zapis* do pliku pod nazwą *nowy*.

Ujawnijmy teraz podstawowe operatory służące do przełączania standardowych wyjść i wejścia:

> *plik* - wyjście standardowe jest kierowane do pliku *plik* (zamiast na ekran). Innymi słowy, plikowi *plik* nadawany jest deskryptor nr 1. Jeżeli plik wcześniej nie istniał, to zostanie utworzony. Jeżeli istniał jego poprzednia zawartość zostanie nadpisana (zniszczona).

>> *plik* - analogicznie jak dla „>", z tym że jeśli *plik* już istniał, to jego dotychczasowa zawartość zostanie zachowana, gdyż dane strumienia standardowego wyjścia będą dopisywane na końcu *pliku*.

< *plik* - jako wejście standardowe (zamiast klawiatury) zostanie otwarty plik *plik*. Innymi słowy, deskryptor nr 0 zostanie przypisany plikowi *plik*.

2> *plik* - standardowe wyjście błędów (diagnostyczne) jest kierowane do pliku *plik* (zamiast na ekran). Innymi słowy, plikowi *plik* zostanie przypisany deskryptor nr 2. Jeżeli plik wcześniej nie istniał, to zostanie utworzony. Jeżeli istniał, jego poprzednia zawartość zostanie nadpisana (zniszczona)

2» *plik* - analogicznie jak dla „2>", z tym że jeśli *plik* już istniał, to jego dotychczasowa zawartość zostanie zachowana, gdyż dane strumienia standardowego wyjścia błędów będą dopisywane na końcu *pliku*.

2>&1 - przekierowuje standardowe wyjście błędów (deskryptor nr 2) w to samo miejsce, gdzie jest skierowane wyjście standardowe (deskryptor nr 1). W ogólności znak ampersand ,,&" umożliwia odwołanie do któregoś ze standardowych strumieni. Zatem zapis:

1>&2 - będzie oznaczał przekierowanie wyjścia standardowego (deskryptor nr 1) w to samo miejsce, gdzie jest skierowane standardowe wyjście błędów (deskryptor nr 2).

Pamiętajmy, że w zapisach typu „2>", „2>>", „1>&2" znaki powinny następować bezpośrednio jeden po drugim (tzn. nie może być pomiędzy nimi spacji). Obecność spacji powodować będzie niezamierzone przez nas interpretacje polecenia.

Wykonajmy jeszcze jeden przykład:

```
$ ls -F 2> diag 1>&2
[leszek1@mono leszek1]$ cat diag
diag
kat1/
kat2/
[leszek1@mono leszek1]$ rm diag
[leszek1@mono leszek1]$ _
```

Tutaj standardowe wyjście błędów polecenia ls -F skierowaliśmy (2>) do pliku *diag*. Jednocześnie (1>&2) przełączyliśmy standardowe wyjście (deskryptor 1) w to samo miejsce, gdzie standardowe wyjście diagnostyczne (deskryptor 2), czyli też do pliku *diag*. Z tego powodu rezultat polecenia **ls** nie był widoczny na ekranie, lecz cały znalazł się w pliku *diag*.

Ćwiczenie 3.2. Tworzymy potoki

Już posługiwaliśmy się potokiem, jeszcze go tak jawnie nie nazywając. Było to po raz pierwszy w ćwiczeniu 1.6, gdy skorzystaliśmy z **more.** Potok tworzymy operatorem „I", według poniższej składni:

```
polecenie1 | polecenie2
```

Operator „I" łączy standardowe wyjście polecenial ze standardowym wejściem polecenia2. Oczywiście potok może być dłuższy i zawierać więcej poleceń niż tylko dwa, na przykład:

```
polecenie_początkowe|polecenie1|polecenie2|...|polecenieN|polecenie_końcowe
```

W takim przypadku aby można było utworzyć potok, polecenia „środkowe" (od **polecenie1** do **polecenieN)** muszą być filtrami. Natomiast polecenia skrajne nie, aczkolwiek muszą spełniać określone warunki. Otóż **polecenie_początkowe** powinno wysyłać strumień wyjściowy do standardowego wyjścia, a **polecenie_końcowe** pobierać dane ze standardowego wejścia.

Wykonajmy następujące polecenie (potok):

```
$ rpm -qa | sort -f | less
```

```
[q]
[leszek1@mono leszek1]$ _
```

Polecenie **rpm** -qa wyprowadza na wyjście standardowe listę wszystkich tzw. pakietów RPM zainstalowanych w systemie. Szczegółowo temat RPM omówiony jest w lekcji 13. Teraz wystarczy nam, że mamy listę do posortowania, którą porządkujemy alfabetycznie poleceniem **sort -f.** Polecenie **less** jest już nam znane. Naciskamy

klawisz [q], aby opuścić **less.** W bardzo prosty sposób listę taką zapisać możemy do pliku, korzystając z przekierowania wyjścia standardowego polecenia **sort:**

```
$ rpm -qa | sort -f > Usta
[1 eszekl@mono leszekl]$ ls
katl   kat2   lista
[leszekl@mono leszekl]$ less Usta


[ q ]
[leszekl@mono leszekl]$ _
```

Listę zapisaliśmy do pliku *lista,* który następnie obejrzeliśmy za pomocą **less.**

Ćwiczenie 3.3. Rozgałęziamy potok (tee)

Kontynuujemy przykład z poprzedniego ćwiczenia, nieco rozszerzając zadania naszego potoku. Otóż chcielibyśmy uzyskać na ekranie liczbę zainstalowanych pakietów, ale jednocześnie chcemy posiadać także plik tekstowy (o nazwie *lista),* z uporządkowaną imienną listą pakietów. Posłużymy się w tym celu tzw. rozgałęzieniem potoku:

```
$ rpm -qa | sort -f | tee Usta | we -1
      257
[leszek1@mono leszek1]$ wc Usta
      257      257     4106   lista
[leszek1@mono leszek1]$ _
```

Polecenie o niezbyt szczęśliwej nazwie we -I, zlicza liczbę wierszy (lines) w zadanym pliku (w naszym przypadku będzie ona równa liczbie pakietów RPM). Wydane bez opcji, zlicza wszystko co potrafi, zatem kolejno: liczbę wierszy, liczbę słów i liczbę znaków (bajtów). Wymieńmy pozostałe opcje tego polecenia:

-w - (words) zlicza tylko liczbę słów

-c - (chars) zlicza tylko liczbę znaków (bajtów).

Nas interesuje bardziej polecenie **tee,** które zapewnia rozgałęzienie potoku. Polecenie **tee** pobiera dane ze standardowego wejścia, a na wyjściu tworzy dwa strumienie. Jeden jest podłączony do standardowego wyjścia, drugi do wskazanego pliku (w naszym przykładzie *lista).* Co ciekawe, w szczególnych przypadkach owym wskazanym plikiem może być ekran. Najprostszym zastosowaniem **tee** jest sytuacja, w której chcemy, aby rezultaty polecenia były wyświetlane na ekranie, ale jednocześnie zapamiętane w konkretnym pliku:

```
polecenie | tee plik
```

Plik *plik,* o ile nie istnieje, zostanie utworzony. Jeżeli istnieje, zostanie nadpisany (jego dotychczasowa zawartość będzie zniszczona). Wydając polecenie **tee** z opcją -a

(append), powodujemy, że jeżeli plik już istniał, to dane będą dopisywane na jego końcu, czyli jego uprzednia zawartość zostanie zachowana:

```
$ rpm -qa | sort -f | tee -a lista | wc -l
      257
[leszek1@mono leszek1]$ wc lista
     514     514    8212 lista
[leszek1@mono leszek1]$ rm lista
[leszek1@mono leszek1]$ _
```

Powyższy przykład dobrze ilustruje działanie tee. Nawet mimo wielokrotnego uruchamiania potoku polecenie **wc -l** występujące w potoku zawsze poda prawidłową liczbę pakietów, gdyż pobiera dane bezpośrednio ze strumienia wyjściowego polecenia **sort -f**. Natomiast plik *lista* będzie (z uwagi na opcję -a polecenia tee) sukcesywnie pęczniał.

Wykonajmy jeszcze, być może mało użyteczny, ale wiele mówiący przykład:

```
$ rpm -qa | sort -f | tee 'tty' | we


zlib-devel-1.1.3-2
      257     257    4106
[leszek1@mono leszek1]$ _
```

Zapis 'tty' wskazuje poleceniu tee nie fizyczny plik na dysku, lecz ekran do wyprowadzenia strumienia danych, zatem na ekranie widzimy nie tylko wynik zliczania polecenia wc, ale również nazwy pakietów RPM. Bardzo ważne są tu znaki cytowania, w które ujęty jest napis tty. Są to tak zwane znaki akcentu (odwrotne apostrofy). Zapis bez tych znaków spowodowałby po prostu utworzenie w bieżącym katalogu pliku o nazwie *tty* i wpisanie do niego listy pakietów. Używamy tutaj znaku akcentu *„'",* który znajduje się na jednym klawiszu razem z tyldą „~" - w prawym, górnym rogu klawiatury (o układzie typu amerykańskiego - US). Jak dowiemy się w ćwiczeniu 14.9, znaki akcentu są wykorzystywane do cytowania poleceń.

Ćwiczenie 3.4. Wyrażenia regularne

Celem wyrażeń regularnych (ang. *regular expressions)* jest tworzenie wzorców podawanych jako argumenty poleceń. Wzorce te tworzy się wykorzystując znaki specjalne (poznane już w ćwiczeniu 2.1, przy okazji omawiania zasad tworzenia nazw plików) jako operatory wyrażeń regularnych. Najbardziej chyba znany znak specjalny, to gwiazdka „*".

Z uwagi na historyczny rozwój sytemu UNIX (dziedziczony siłą rzeczy przez Linuksa), zakres implementacji i reguły tworzenia wyrażeń regularnych w poszczególnych poleceniach (które z nich korzystają) mogą się nieco różnić. Dlatego też czę-

sto ważny jest kontekst użycia wyrażenia regularnego. Bardzo obszerną implementacją wyrażeń regularnych posługują się tradycyjne filtry: **grep, awk, ed, sed,** także nieomawiany w tej książce edytor **emacs**.

Wyrażenie regularne definiowane jest często jako wzór opisujący zbiór łańcuchów. Wzorce tworzone za pomocą wyrażeń regularnych są stosowane przede wszystkim w filtrach.

> Mechanizm tworzenia wzorców nazw plików dla powłoki (jaki poznaliśmy w lekcji 2, podobny do tego z DOS-a), zwany inaczej rozwijaniem nazw plików, nie mieści się w kategorii wyrażeń regularnych. Są to dwa różne światy. Przykładowo, wspomniana wyżej gwiazdka „*" oznacza co innego w wyrażeniu regularnym, a co innego we wzorcu nazwy pliku dla powłoki.

Wyrażenie regularne składa się ze znaków zwykłych (reprezentujących samych siebie, np. A a 1 5 itd.) i znaków specjalnych (np. . $ \ ^), będących operatorami wyrażeń regularnych. W szczególności, najprostsze wyrażenie regularne nie musi wcale zawierać znaków specjalnych, wystarczy jeden znak zwykły, na przykład a.

Poniżej przedstawimy podstawowe znaki specjalne używane jako operatory w wyrażeniach regularnych.

. (kropka) - dopasuj dowolny pojedynczy znak (działa analogicznie jak znak „?" w generowaniu nazw plików dla powłoki).

$ - dopasuj poprzedzający operator łańcuch do końca wiersza; w ogólności ów łańcuch może być wyrażeniem regularnym.

^ - dopasuj poprzedzający operator łańcuch do początku wiersza; w ogólności ów łańcuch może być wyrażeniem regularnym.

* - dopasuj **zero** lub więcej wystąpień poprzedzającego operator pojedynczego znaku. W szczególności ów znak może być wyznaczany przez wyrażenie regularne. Zwróćmy uwagę na fakt, iż dotyczy to wielokrotnych wystąpień *tego samego* znaku. Analogia do użycia gwiazdki w generowaniu nazw plików dla powłoki jest tu chybiona, gdyż w przypadku powłoki gwiazdka oznacza dowolnie długi (także pusty) ciąg *dowolnych,* czyli *różnych* znaków. W wyrażeniu regularnym dowolny łańcuch dowolnych znaków zapiszemy jako .*.

\ - odwrotny ukośnik (ang. *backslash)* maskuje znak specjalny. Użycie „\" bezpośrednio po znaku specjalnym powoduje, że ten znak specjalny, reprezentuje tylko siebie i nie będzie interpretowany w sposób szczególny. Odwrotny ukośnik użyty po zwykłym znaku reprezentuje sam siebie i nic w tym przypadku nie maskuje.

[ ] - dopasuj dowolny pojedynczy znak z zestawu między nawiasami. Analogia do użycia nawiasów kwadratowych przy generowaniu nazw plików powłoki jest w tym przypadku właściwa. W nawiasach możemy podać zakres znaków (np. [1-9]) bądź listę ([aStwl23]). Wewnątrz nawiasów kwadratowych wszystkie znaki (za wyjątkiem „^" na pierwszej pozycji i „-" w środku) reprezentują tylko same siebie i tracą ewentualne specjalne znaczenie.

[^] - dopasuj dowolny znak spośród tych, które nie znajdują się w nawiasach. Odpowiada użyciu wykrzyknika wewnątrz nawiasów przy generowaniu nazw plików dla powłoki.

Nie są to wszystkie z zaliczanych do podstawowych operatory wyrażeń regularnych, ale na potrzeby ćwiczeń prowadzonych w tej książce wystarczą. Ponadto, mamy jeszcze drugą grupę operatorów (tzw. rozszerzenia), stosowanych w programach **egrep** (rozszerzony **grep)** i **awk.** Polecenie **awk,** które jest w naszym systemie tak naprawdę dowiązaniem symbolicznym do */bin/gawk),* uruchamia interpreter języka AWK przeznaczonego do przeszukiwania i przetwarzania tekstów. Opis tego języka przekracza ramy przyjęte dla niniejszej książki. W Linuksie stosowana jest wersja GNU awk, czyli **gawk,** posiadająca własne, specyficzne rozszerzenia.

Powróćmy do głównego nurtu naszych rozważań - rozszerzoną składnię tworzą następujące znaki specjalne:

\+ - plus działa analogicznie jak gwiazdka, z tym że znak musi wystąpić co najmniej raz

? - poprzedzający operator znak może wystąpić dokładnie raz albo wcale (zero razy)

|- operator OR (LUB) - dopasuj wzorzec umieszczony z lewej lub prawej strony

( ) - grupowanie zapewnia zachowanie jednoznaczności w złożonych wyrażeniach regularnych.

Teraz zrealizujmy kilka przykładów ilustrujących wykorzystanie operatorów wyrażeń regularnych do budowy wzorców. Pamiętajmy, że wzorce takie tworzy się głównie na potrzeby przeszukiwania plików tekstowych, pod kątem zawartości określonych łańcuchów.

Przykład 1. Wyrażenie regularne: a. om

pasują słowa: atom, arom, abom, atomowy

nie pasują słowa: as kom

Przykład 2. Wyrażenie regularne: **rze$**

pasują wiersze:	Jest dobrze
	Jest bardzo dobrze
	Wcale nie wiadomo, czy jest bardzo dobrze
	Ale może chociaż jest dobrze
	Chyba jednak nie jest dobrze
	Czy widzisz morze
nie pasują wiersze:	Jest dobrze, ale nie wiadomo czy na pewno
	Nasze morze jest mokre

Przykład 3. Wyrażenie regularne: ^Ban

pasują wiersze:	Banany to smaczne owoce
	Bandyci zrabowali banany
	Bank jest po drugiej stronie ulicy
nie pasują wiersze:	banany i mandarynki
	banki
	bankowy fundusz
	bananowy sok

Przykład 4. Wyrażenie regularne: **zoo***

pasują słowa:	zoo, zootechnika, powozownia, zorganizowany, zoooo, pokazowy
nie pasują słowa:	koza, zmykaj, zmarzlina, kowal

Wyrażenie regularne dotyczy łańcucha, który może wystąpić w „dowolnym miejscu" słowa (wiersza). Nie należy sugerować się tu przyzwyczajeniami z DOS-a (czy nawet z Linuksa, dotyczącymi generowania nazw plików dla powłoki). Podkreślmy to jeszcze raz - zapis „zoo*" nie oznacza, że szukamy słowa zaczynającego się na „zo", lecz szukamy tak zaczynającego się łańcucha, który może znajdować się wewnątrz słowa.

Przykład 5. Wyrażenie regularne: **zo***

pasują słowa:	zoo, zobacz, zmykaj, zamysł, zroszony, pokazowy, raz
nie pasują słowa:	albowiem, tylko, kret

Przykład 6. Wyrażenie regularne: **[Rr]ozkaz_nr[12389]**

pasują słowa:	rozkaz_nrl, Rozkaz_nr9, rozkaz_nr3, rozkaz_nrl2, NaszRozkaz_nr345
nie pasują słowa:	Rozkaz_nr7, Polecenie_nrl

Przykład 7. Wyrażenie regularne: li **sta[1-7] [12]**

pasują słowa: l i sta11, lista 32, lista51, lista72, Talista7110

nie pasują słowa: li sta1, l i sta53, lista77, lista91, Lista12

Przykład 8. Wyrażenie regularne: **[Ww]ykaz[^45]**

pasują słowa: Wykazy, wykazy, wykazl, Wykaz7, wykazl5, Naszwykazl2567

nie pasują słowa: wykaz4, Wykaz5, wykaz45, 1 ista6

Przykład 9. Wyrażenie regularne: **ko+**

pasują słowa: koza, koordynator, kooperacja, koooo, rozkosz, rozkopany

nie pasują słowa: kminek, kumotr, klasa, wola, zoo

Przykład 10. Wyrażenie regularne: bar?

pasują słowa: bar, barokowy, bank, bankier, barr, barakabra, abak, tombak

nie pasują słowa: zegar

Zwróćmy uwagę, że zapis „bar?" oznacza, że poszukujemy łańcuchów „ba" i „bar". Zatem każde słowo zawierające przynajmniej jeden z tych łańcuchów będzie spełniało warunki wyrażenia regularnego.

Przykład 11. Wyrażenie regularne: **bomba[1-5]** |dziura

pasują słowa: bomba1, dziura, bomba4, dziural23, Bigbombal77, bombal5

nie pasują słowa: ciura, bomba, Bomba3

W tym przykładzie skorzystaliśmy z rozszerzonego operatora „I".

Możemy później samodzielnie przekonać się jak działają wyrażenia regularne, tworząc plik tekstowy (za pomocą edytora vi, który poznamy na lekcji 4), a następnie przeszukując go za pomocą filtra **grep (egrep),** tak jak w następnym ćwiczeniu.

Ćwiczenie 3.5. Stosujemy filtr (grep)

Wykorzystamy teraz w konkretnych zastosowaniach poznane reguły tworzenia wyrażeń regularnych. Najbardziej chyba znanym linuksowym (uniksowym) filtrem jest grep (global look for a regular expression and print). Wyprowadza on na wyjście standardowe wiersze pliku, zawierające łańcuchy pasujące do podanego wzorca. Wejściem polecenia **grep** jest konkretnie wskazany plik (lub pliki), a jeżeli nie jest or wskazany - wykorzystywane jest wejście standardowe.

Jeżeli polecenie przeszukuje kilka plików, to (o ile nie została podana opcja **-h**) na początku każdego wiersza wypisywana jest nazwa odpowiedniego pliku.

```
$ ls / | grep [r]...
root
[leszek1@mono leszek1]$ ls / | grep t$
boot
mnt
root
[leszek1@mono leszek1]$ ls / | grep [rl.*
proc
root
usr
var
[leszek1@mono leszek1]$ ls / | grep r.*
proc
root
usr
var
[leszek1@mono leszekl]$ ls / | grep ^r.*
root
[leszek1@mono leszekl]$ ls / | grep ^roo*
root
[leszek1@mono leszekl]$ ls / | grep .ooo*
boot
root
[leszek1@mono leszek1]$ ls / | grep .oo*
boot
home
lost+found
proc
root
[leszek1@mono 1eszekl]$ ls / | grep .ooo*|n$
bash: n$: command not found
[1eszek1@mono leszekl]$ ls / | grep ".ooo*|n$"
[leszek1@mono leszekl]$ ls / | egrep ".ooo*jn$"
bin
boot
root
sbin
[leszekl@mono leszekl]$ ls / | egrep '.ooo*|n$'
bin
boot
root
sbin
Cleszekl@mono 1eszekl]$ _
```

Skorzystaliśmy tu z prostego potoku zaczynającego się od polecenia ls /. Pamiętajmy, że chociaż po wydaniu polecenia ls widzimy na ekranie zawartość katalogu

w układzie kolumnowym, to gdy wyprowadzana jest ona na wyjście standardowe, ma postać pliku, w którym każdy wiersz zawiera tylko jedną nazwę pliku (katalogu).

Zwróćmy uwagę na fakt, że tylko proste wzorce dla **grep (egrep)** możemy używać bez znaków cytowania. Bardziej złożone należy ująć w znaki cytowania, czyli cudzysłów bądź apostrofy (pojedyncze cudzysłowy). Obydwa wymienione znaki cytowania znajdują się na tym samym klawiszu typowej (czyli o układzie amerykańskim) klawiatury. Użycie nie ujętego w cudzysłów wzorca zawierającego znak „I" spowodowało, że powłoka zinterpretowała ten znak jako tworzący potok. Pojawił się komunikat o błędzie, gdyż powłoka nie mogła znaleźć „ostatniego polecenia" tworzącego potok, czyli **n$**.

Dobrym zwyczajem jest stosowanie cudzysłowu bądź apostrofu do każdego wzorca, zwłaszcza takiego, który zawiera znaki specjalne. Unikniemy w ten sposób wszelkich niezamierzonych niespodzianek. Więcej na temat zasad cytowania dowiemy się w ćwiczeniu 14.9. Pamiętajmy również, że rozszerzoną składnię (m.in. znak „I") wykorzystuje **egrep.** Ogólna składnia polecenia **grep (egrep)** jest następująca:

```
grep wzorzec lista_plików
```

Wykonajmy na koniec przykład z listą plików:

```
$ grep "mono" /etc/hosts /etc/HOSTNAME
/etc/hosts:192.168.100.1        tnono.mojnet.org.pl        mono
/etc/HOSTNAME:mono.mojnet.org.pl
[leszek1@mono leszek1]$ _
```

Możemy obejrzeć bezpośrednio pliki */etc/hosts* i */etc/HOSTNAME* (korzystając np. z **cat),** aby sprawdzić, czy **grep** wybrał właściwe wiersze.

Ćwiczenie 3.6. Uruchamiamy zadania w tle (&, jobs, fg, bg)

Powłoka (ang. *shell)* otrzymując od użytkownika polecenie uruchamia je, czekając na jego zakończenie. Mówimy, że jest to zadanie pierwszoplanowe. Uruchamianie poleceń (programów) jako pierwszoplanowych jest domyślne. Niestety terminal jest zablokowany do momentu zakończenia zadania. Jest to zwłaszcza kłopotliwe, gdy wykonanie zajmuje dużo czasu. Możemy radzić sobie w ten sposób, że zalogujemy się na innej konsoli wirtualnej i tam będziemy uruchamiać kolejne zadania.

Uważny i dociekliwy czytelnik zauważy zapewne, że cichaczem wprowadzamy tu nowy termin: zadanie, nie usiłując go sprecyzować. Znamy już polecenie, program, proces. Mają one konkretne znaczenie techniczne i są precyzyjnie osadzone w kontekście budowy i funkcjonowania systemu operacyjnego. Termin *zadanie* ma wymiar bardziej „ludzki" i osadzony jest w kontekście użytkownika. Na przykład uruchomiony edytor tekstowy będzie konkretnym procesem w systemie, ale też jednocześnie zadaniem użytkownika, który realizuje swój cel - napisanie tekstu. Zadanie jest zlece-

niem systemowi przez użytkownika określonego działania. Użytkownik patrzy na system operacyjny (wraz z aplikacjami użytkowymi) przez pryzmat możliwych do wykonania zadań. Zadanie jest z reguły utożsamiane z konkretnym pojedynczym procesem systemowym, ale ogólnie na potrzeby wykonania zadania użytkownika system może uruchomić jeszcze dodatkowe („pomocnicze") procesy.

Termin *zadanie* nie posiada jednak ściśle zarezerwowanego pola zastosowań, dlatego może być używany także w nieco innych kontekstach.

Innym sposobem postępowania z „długotrwałymi" zadaniami jest uruchamianie ich jako drugoplanowych, czyli w tle. Zadanie pierwszoplanowe możemy uruchomić (w danej chwili na konkretnej konsoli wirtualnej) tylko jedno, natomiast zadań (procesów) drugoplanowych wiele. Zadanie uruchamiamy jako drugoplanowe wpisując na końcu linii poleceń znak ampersand &. Odpowiednimi poleceniami możemy również przenosić zadania z planu pierwszego na drugi i odwrotnie.

Spróbujemy wykonać kilka przykładów. Z ćwiczenia 1.3 wiemy już, że polecenie Is -R / (powodujące wylistowanie całej zawartości drzewa plików) zajmie nam sporo czasu. Spróbujemy zatem uruchomić je w tle, rezultat umieszczając w pliku *lista*. Ale żeby nic nam nie pojawiało się na ekranie, przeadresujemy także wyjście błędów do specjalnie w tym celu utworzonego pliku *wykbl:*

```
$ ls -R / > Usta 2> wykbl &
[1] 365
[leszek1@mono leszek1]$ jobs
[1]+ Running          ls -R / > lista 2>wykbl &
[leszek1@mono leszek1]$ jobs
[1]+ Exit 1           ls -R / > lista 2>wykbl
[leszek1@mono leszek1]$ ls -F
kat1/   kat/2  lista  wykbl
[leszekl@mono leszekl]$ less wykbl
ls: /home/leszek2: Permission denied


[q]
[leszek1@mono leszek1]$ less lista
bin


[q]
[leszek1@mono leszek1]$ rm wykbl Usta
[leszek1@mono leszek1]$ _
```

Liczba w nawiasie kwadratowym (tutaj [1]) oznacza numer zadania użytkownika. Korzystając z tego numeru użytkownik może odwoływać się do zadania w poleceniach **bg, fg** i **kill,** o których za chwilę. Liczba 365 oznacza identyfikator procesu (PID) w systemie. PID jest najczęściej wykorzystywany w poleceniu **kill** (patrz następne ćwiczenie). Polecenie **jobs** pokazuje zadania użytkownika wykonywane w tle oraz zadania zatrzymane (ang. *stopped).* Jeżeli szybko wydamy polecenie **jobs,** zdą-

żymy jeszcze zobaczyć, że nasze zadanie jest wykonywane (ang. *running)*. Ponowne wydanie jobs chwilę później pokazuje nam, że wprawdzie zadanie zostało zakończone, ale nie wykonane w całości poprawnie (Exit 1). W katalogu domowym użytkownika *leszek1* możemy znaleźć i obejrzeć dwa pliki: *lista* i *wykbl*. Wiemy co zawiera plik *lista*. Natomiast plik *wykbl* zawiera listę błędów, która wskazuje na przyczynę problemów z wykonaniem zadania. Łatwo zauważyć, że jako *leszek1* nie mamy praw dostępu („Permission denied") do wielu katalogów w drzewie plików i z tego powodu nie możemy wylistować ich zawartości. Prawom dostępu poświęcona jest lekcja piąta.

Zalogujmy się na innej konsoli wirtualnej (np. [lewy Alt]+[F3]) jako *root* i wykonajmy to samo zadanie:

```
# ls -R / > Usta 2> wykbl &
[1] 426
[root@mono /root] # jobs
[1]+ Running ls -R / >lista 2>wykbl &
[root@mono /root] # jobs
[1]+ Done ls -R / >lista 2>wykbl
[root@mono /root] # ls
lista wykbl
[root@mono /root] # less lista
bin


[q]
Croot@mono /root] # less wykbl
wykbl (END)
[q]
[root@mono /root] # rm lista wykbl
rm: remove 'lista' ? y
rm: remove 'wykbl' ? y
[root@mono /root] # _
```

Tutaj proces zakończył się normalnie (Done), a plik *wykbl* zawierający listę błędów jest pusty.

Powróćmy na konsolę użytkownika *leszekl*. Teraz uruchomimy dwukrotnie polecenie **ls -R /** i natychmiast je zatrzymamy kombinacją klawiszy **[Ctrl]+[z]**. Zatrzymany proces może być wznowiony poleceniem **fg** bądź **bg**:

```
$ ls -R /


[Ctrl]+[z]
[1]+ Stopped ls -R /
[leszek1@mono leszek1]$ ls -R /


[Ctrl]+[z]
[2]+ Stopped ls -R /
```

```
[leszek1@mono  leszek1]$ jobs
[1]-          Stopped            ls            -R /
[2]+          Stopped            ls            -R /
[leszek 1@mono leszek1]$ jobs  -1
[1]-     411  Stopped                      ls  -R /
[2]+     415  Stopped                      ls  -R /
[leszek1@mono  leszek1]$ fg


/var/yp/binding:
[leszekl@mono  leszekl]$ jobs -1
[1]+     411  Stopped                       ls -R /
Cleszekl@mono  leszekl]$ fg  -x  411


/var/yp/binding:
[1eszeklOmono  leszekl]$ jobs
Cleszekl@mono  leszekl]$ _
```

Poznaliśmy tu ważną opcję **-l** polecenia **jobs**, pozwalającą na wyświetlenie, oprócz względnych numerów zadań użytkownika podawanych w nawiasach kwadratowych, także odpowiadających im bezwzględnych identyfikatorów procesów PID (tutaj liczby 411 i 415).

Proces zatrzymany (ang. *stopped)* kombinacją klawiszy **[Ctr]+[z]** może być wznowiony (kontynuowany).

Do ponownego uruchomienia zatrzymanego zadania na pierwszym planie używamy polecenia fg. Do ponownego uruchomienia zatrzymanego zadania na drugim planie (w tle) używamy polecenia **bg**. Przez ponowne uruchomienie rozumiemy sytuację, w której proces kontynuuje pracę „od miejsca, w którym został zatrzymany" (a nie jeszcze raz od początku). Sposób wskazania z listy zadań zatrzymanych zadania do wznowienia jest taki sam dla obu poleceń:

- przez podanie PID (konieczna opcja -x), na przykład **fg -x 411** (gdzie 411 to PID procesu, który chcemy wznowić)
- przez podanie numeru zadania (z nawiasu kwadratowego), na przykład **fg %2** (gdzie 2 to numer zadania, a % wskazuje, że o numer zadnia właśnie nam chodzi)
- wydanie polecenia bez żadnych opcji spowoduje uruchomienie ostatnio zatrzymanego zadania.

Wykonajmy jeszcze jeden przykład:

```
$ ls  -R  /


[Ctrl]+[z]
[1]+       Stopped                          ls  -R /
```

```
[leszek1@mono leszek1]$ ls -R /

[Ctrl]+[z]
[2]+  Stopped                 ls -R /
[leszek1@mono leszek1]$ jobs
[1]-  Stopped                 ls -R /
[2]+  Stopped                 ls -R /
[leszek1@mono leszek1]$ fg %1

/var/yp/binding:
[leszek1@mono leszek1]$ jobs
[2]+  Stopped                 ls -R /
[leszek1@mono leszek1]$ fg %2

/var/yp/bi nding:
[leszek1Omono leszek1]$ _
```

Ćwiczenie 3.7. Wysyłamy sygnały (kill)

Komunikować się z uruchomionymi procesami możemy między innymi przez wysyłanie do nich sygnałów. Sygnały są zestandaryzowane. Mają przyporządkowane numery i nazwy pisane dużymi literami. Do procesów pierwszoplanowych możemy niektóre sygnały wysyłać za pomocą stosownych kombinacji klawiszowych. Polecenie **kill** umożliwia wysyłanie różnych sygnałów do procesów zarówno pierwszo-, jak i drugoplanowych. Rodzaj sygnału jest określany poprzez jego numer lub nazwę. Proces, do którego wysyłamy sygnał, jest wskazywany względnie poprzez numer zadania użytkownika (uzyskany poleceniem **jobs**) bądź bezwzględnie przez jego systemowy identyfikator procesu PID (uzyskiwany poleceniem **ps**, ale także **jobs -1**). Lista sygnałów, które może wysyłać **kill** znajduje się w pliku */usr/include/linux/signal.h*. Listę tę możemy uzyskać na ekranie przez wydanie polecenia:

```
kill -l
```

Najczęściej interesują nas sygnały związane z przerywaniem, zatrzymywaniem i usuwaniem procesów:

Sygnał 2 (SIGINT) - przerwanie wykonania procesu nakazane z klawiatury (terminala)

Sygnał 3 **(SIGQUIT)** - zakończenie (wyjście) wykonywania procesu nakazane z klawiatury. Jednocześnie wykonywane jest tzw. „zrzucenie core", czyli utworzenie w bieżącym katalogu pliku o nazwie *core* zawierającego obraz pamięci procesu

- **Sygnał 9 (SIGKILL)** - unicestwienie (zabicie) procesu. Bezwarunkowo kończy działanie procesu. Nadawca tego sygnału musi być właścicielem procesu-adresata, bądź administratorem *(root)* systemu; należy stosować rozważnie
- **Sygnał 15 (SIGTERM)** - domyślny sygnał polecenia **kill** (o ile nie podano innego sygnału). Programowe (miękkie) zakończenie procesu. Sygnał może być wysłany także przez funkcję systemową. Nie wszystkie procesy możemy zakończyć za pomocą tego sygnału. W niektórych wypadkach proces będzie reagował tylko na sygnał 9 (SIGKILL)
- **Sygnał 19 (SIGSTOP)** - zatrzymanie wykonywania procesu, z możliwością późniejszego wznowienia jego wykonywania. Po wydaniu polecenia **jobs** zatrzymany proces zostanie wypisany jako „stopped". Wznowienie pracy procesu uzyskujemy poleceniami **fg** bądź **bg.**

Z funkcjonalnego punktu widzenia zwykłego użytkownika sygnały 2, 3, 9, 15 są równoważne - prowadzą bowiem do nieodwracalnego zakończenia pracy procesu ,,przed terminem" jego naturalnego zakończenia bądź po prostu usunięcia go, jeżeli się zawiesił. Sygnał 9 (KILL) jest najbardziej brutalnym zakończeniem funkcjonowania procesu i zaleca się jego bardzo ostrożne używanie: jedynie wtedy, gdy nie udało się zakończyć procesu bardziej łagodnie, czyli sygnałami 2, 3 lub 15.

Uporządkujmy teraz kombinacje klawiszy i związane z nimi sygnały. Pamiętajmy, że kombinacje klawiszowe wysyłają sygnał tylko do bieżącego procesu uruchomionego na pierwszym planie. Jest to jak najbardziej sensowne, gdyż w tle możemy mieć uruchomionych wiele zadań. Wysyłanie sygnału do zadania w tle musi być jednoznaczne (musi określać, dla którego konkretnie zadania sygnał jest przeznaczony). Dwie pierwsze kombinacje klawiszowe znaliśmy wcześniej, ostatnią poznaliśmy dopiero w poprzednim ćwiczeniu:

[Ctrl]+[c] - wysyła sygnał 2 (SIGINT)

[Ctrl]+[\] - wysyła sygnał 3 (SIGQUIT); na ekranie otrzymamy komunikat: Quit (core dumped)

[Ctrl]+[z] - wysyła sygnał 19 (SIGSTOP); na ekranie otrzymamy komunikat w rodzaju: [1]+ Stopped, przy czym liczba w nawiasach kwadratowych może być inna, w zależności od aktualnej liczby zadań użytkownika.

Wykonamy teraz kilka ćwiczeń. Najpierw utworzymy zbiór pięciu zatrzymanych zadań: pięć razy uruchamiając polecenie **ls -R /** i natychmiast je zatrzymując za pomocą kombinacji klawiszowej **[Ctrl]+[z]:**

```
$ Is -R /

[Ctrl]+[z]
```

```
[1]+    Stopped                    ls -R /
[leszek1@mono leszek1]$
```

Powyższe czynności powtarzamy pięć razy, aż do uzyskania takiego rezultatu:

```
$ jobs
[1]     Stopped                    ls -R /
[2]     Stopped                    ls -R /
[3]     Stopped                    ls -R /
[4]-    Stopped                    ls -R /
[5]+    Stopped                    ls -R /
[leszek1@monoleszek1]$_
```

Aby usunąć (sygnał nr 15) zadanie nr 1, wydajemy polecenie:

```
$ kill %1
[1] Terminated                     ls -R /
[leszek1@mono leszek1]$ jobs
[2]     Stopped                    ls -R /
[3]     Stopped                    ls -R /
[4]-    Stopped                    ls -R /
[5]+    Stopped                    ls -R /
[leszek1@mono leszek1]$ _
```

Zwróćmy uwagę na składnię polecenia **kill.** Domyślnym sygnałem jest SIGTERM (15), zatem nie podajemy tu żadnej opcji. Natomiast argumentem jest **%1**, gdzie znak procent wskazuje, że podajemy numer zadania użytkownika (widziany w nawiasach kwadratowych w poleceniu **jobs),** podczas gdy jedynka oznacza zadanie 1.

Zakończmy teraz bezwarunkowo (sygnał 9) zadanie 2:

```
$ kill -9 %2
[2]   Killed                       ls -R /
[leszek1@mono leszek1]$ jobs
[3]     Stopped                    ls -R /
[4]-    Stopped                    ls -R /
[5]+    Stopped                    ls -R /
[leszek1@mono leszek1]$
```

Wystąpiła tutaj opcja (**-9**), w której podaliśmy numer sygnału do wysłania. Rodzaj sygnału możemy również określić wprost przez jego nazwę, natomiast proces wskazać możemy w sposób bezwzględny poprzez jego PID. Usuńmy bezwarunkowo zadanie 3:

```
$ kill -SIGKILL %3
[3]   Killed                       ls -R /
[leszek1@mono leszek1]$ jobs
[4]-    Stopped                    ls -R /
[5]+    Stopped                    ls -R /
[leszek1@mono leszek1]$ _
```

Zostały nam dwa zatrzymane zadania.

```
$ fg   %4
```

```
/var/yp/binding
[leszek1@mono  leszek1]$  jobs
[5]+    Stopped                          ls -R /
[leszek1@mono  leszek1]$  _
```

Zostało ostatnie zadanie. Usuniemy je korzystając z bezwzględnej identyfikacji procesu (PID):

```
$ ps
PID    TTY   STAT   TIME  COMMAND
318     1     S     0:00   -bash
406     1    T     0:01   ls -R /
407     1    R     0:00   ps
[leszek1@mono   leszek1]$ kill  -9   406
[leszek1@mono   leszek1]$ jobs
[5]+    Killed                           ls -R /
[leszek1@mono   leszek1]$ jobs
[leszek1@mono   leszek1]$
```

Aby znaleźć numer PID, wykonaliśmy polecenie ps. Zwróćmy uwagę, że pierwsze polecenie **jobs** wywołane bezpośrednio po usunięciu procesu dało nam informację o zmianie, która zaszła. Następne polecenie **jobs** już nic nie pokazało (tzn. pokazało, że nie ma żadnych zadań użytkownika).

Polecenie **kill** w postaci:

```
kill  -9   PID_procesu
```

może być używane przez administratora do usuwania procesów użytkowników nad którymi stracono kontrolę (zawiesiły się). Należy jedynie pamiętać, że polecenie **ps** wydane bez opcji wyświetla jedynie procesy tego użytkownika, który je wywołał. Aby uzyskać procesy wszystkich użytkowników, należy posłużyć się opcją a (all), czyli wydać polecenie:

```
ps a
```

Lekcja 4. Edytor vi

Nazwa edytora vi (czytaj: wijaj) pochodzi od „visual". Dziś w dobie interfejsów graficznych ów „visual" ma posmak nieco humorystyczny, jednak musimy pamiętać, że vi jest edytorem, który ma długą historię.

Edytor vi to narzędzie nie tyle uniwersalne, co wszechobecne. Jest to standardowy edytor, który znajdziemy nie tylko w każdej dystrybucji Linuksa, lecz w każdym systemie uniksowym. Chcąc dokonać edycji jakiegokolwiek pliku konfiguracyjnego, będziemy mogli to zrobić właśnie za pomocą vi, gdyż on zawsze będzie pod ręką. Natomiast nasz ulubiony (inny niż vi) edytor może nie być w danym systemie zainstalowany i dostępny.

Na pewno wcześniej używaliśmy jakiegoś edytora tekstowego. Teraz lepiej zapomnijmy o tym: wszelkie umiejętności i przyzwyczajenia okażą się nie tylko bezużyteczne, ale mogą być przyczyną niepotrzebnych emocji.

Edytor vi jest wystarczająco złożony, aby móc się w nim zagubić. Często istnieje kilka różnych ścieżek postępowania w celu uzyskania tego samego rezultatu. Nawet jeśli nie zamierzamy opanować całego edytora (w tej lekcji nawet nie będziemy tego próbować), powinnyśmy przetrenować kilka własnych elementarnych ścieżek postępowania. Te umiejętności na pewno się kiedyś przydadzą.

Edytor uruchamiamy najczęściej poleceniem:

```
vi
```

albo poleceniem:

```
v1 nazwa_pliku
```

gdzie: *nazwa_pliku* (ścieżka dostępu) dotyczy istniejącego już pliku (który zamierzamy zmodyfikować) bądź nie istniejącego jeszcze pliku, który dopiero zamierzamy utworzyć.

Uruchomiony edytor może znajdować się w danym momencie w jednym z trzech stanów:

- ⇨ stanie edycji tekstu (1)
- ⇨ stanie poleceń klawiszowych (2)
- ⇨ stanie edycji własnego wiersza poleceń (3).

Należy pamiętać, że stan edycji wiersza poleceń (3) dotyczy wewnętrznego wiersza poleceń edytora, a nie na przykład wiersza poleceń systemowych powłoki.