Systemy operacyjne 9

Spis treści

- 1 Procesy
  - 1.1 Procesy w systemiach uniksowych
  - 1.2 Polecenie związane z procesami
    - 1.2.1 ps
    - 1.2.2 kill
    - 1.2.3 killall
    - 1.2.4 fuser
    - 1.2.5 top
    - 1.2.6 praca w tle, fg, bg, jobs
    - 1.2.7 nohup
    - 1.2.8 nice, renice
  - 1.3 Zadania
- 2 Strumienie w systemach Linux
  - 2.1 Przekierowania do plików
  - 2.2 Przekierowania z plików
  - 2.3 Przekierowania do aplikacji
  - 2.4 Łączenie strumieni
  - 2.5 Przekierowania a urządzenia
  - 2.6 Zadania
- 3 Filtry strumieniowe
  - 3.1 cat
  - 3.2 head, tail
  - 3.3 more, less
  - 3.4 sort, uniq
  - 3.5 tr
  - 3.6 cut
  - 3.7 grep
  - 3.8 Zadania

Procesy

Procesy w systemiach uniksowych

Z systemami unixowymi związane jest pojęcie procesu. W takim ujęciu, proces, rozumiany jest jako wykonywany w systemie program. Każdy proces charakteryzuję się pewnymi atrybutami: przestrzeń adresowa, licznik programu, stanu rejestrów, deskryptory plików, dane procesu, zależności rodzinne, liczniki statystyczne. Wynikiem obecności w systemie procesów jest to, że jądro systemu może nim sterować tak i może go ustawiać w kilku stanach. Zależnie od źródeł literatury, może być 9 lub 5 stanów. Ostatni przypadek to stan:

Pracujący w trybie użytkownika
: proces znajduje się na procesorze i wykonuje swój kod.

Pracujący w trybie jądra
: jądro wykonuje wywołanie systemowe wykonane przez proces.

Uśpiony
: proces czeka na jakieś zdarzenie, na przykład na odczyt danych z dysku lub otrzymanie danych z sieci.

Gotowy do wykonania
: może być uruchomiony w każdej chwili, jednak nie ma jeszcze przydzielonego procesora.

Zombie
: proces zakończył działanie i czeka na odebranie kodu powrotu przez proces macierzysty.

Wszystkie procesy w Unixie powstają jako procesy potomne procesu głównego init o numerze 1, który tworzony przez jądro podczas uruchamiania systemu. Każdy proces może być zarówno procesem potomnym jak i procesem macierzystym innego procesu. |System wykonuje każdy proces przez określony czas następnie pobiera kolejny proces do wykonania. W tym czasie grupa procesów oczekuje na wykonanie. Sprawne działanie zapewnia szeregowanie z wywłaszczaniem oraz system priorytetów i co pozwalający tak ustawić intensywnie używające procesor procesy tła, aby nie blokowały pracy procesom interakcyjnym.

Polecenie związane z procesami

ps

Podstawowym poleceniem do zarządzania procesami przez użytkownika jest: ps. Polecenie **ps** występuje w systemadch uniksowych w kilku wersjach. Różnią się on między sobą sposobem podawania parametrów i nieznacznie zachowaniem. Wersja dostarczana z systemem Gentoo Linux obsługuje większość opcji podawanych w jednej z trzech konwencji:

1. Opcje w stylu UNIX, które mogą być grupowane i muszą być poprzedzone myślnikiem.
2. Opcje BSD, które mogą być grupowane i nie mogą być użyte z myślnikiem.
3. Długie opcje GNU, które są poprzedzone dwoma myślnikami.

Polecenie:

```
ps [-] [lujsvmaxScewhrnu] [txx] [O[+|-]k1[[+|-]k2...]] [pids]
```

ps uruchomiony bez parametrów wyświetla wszystkie procesy danego użytkownika związane z bieżąca konsolą.

Wybrane opcje w formacie Uniksa:

-?
: wyświetla najważniejsze opcje

-A lub -e
: wyświetla wszystkie procesy

-a
: wyświetla wszystkie procesy posiadające terminal

-l
: długi format

-j
: format prac: pgid, sid

-H
: "forest" (las) - format drzewiasty

Wybrane opcje w formacie BSD:

a
: wyświetla wszystkie procesy posiadające terminal

x
: wyświetla wszystkie procesy posiadające i nie posiadające terminala, należące do bieżącego uzytkownka

l
: długi format

j
: format prac: pgid, sid i inne identyfikatory, dla danego użytkownika

f
: "forest" (las) - format drzewiasty

e
: pokaż środowisko (wszystkie zmienne systemowe) dla każdego polecenia

h
: bez nagłówka

Wybrane opcje o identycznym działaniu:

-u lub u
: podaje nazwy użytkowników i czas startu

-v
: format v

```
knoppix@ttyp0[knoppix]$ ps
  PID TTY          TIME CMD
12369 pts/1    00:00:00 bash
17270 pts/1    00:00:00 ps
```

Wyświetlany jest numer PID, terminal sterujący procesu, całkowity czas, w którym proces zajmował procesor, oraz komenda, za pomocą, której proces został uruchomiony. Wyświetlono jedynie te procesy, które pracują na tym samym terminalu, co użytkownik.

przykład:

```
knoppix@ttyp0[knoppix]$ ps ax
PID       TTY       STAT      TIME      COMMAND
1         ?          S        0:04      init [2]
2         ?         SW        0:19       [keventd]
3         ?         SW        0:00       [kapmd]
4         ?         SWN       0:14       [ksoftirqd_CPU0]
5         ?         SW        2:16       [kswapd]
6         ?         SW        0:00       [bdflush]
7         ?         SW        0:04       [kupdated]
11        ?         SW        2:58      [kjournald]
(...)
```

Wyświetla wszystkie procesy pracujące w systemie. przykład:

```
knoppix@ttyp0[knoppix]$ ps f
PID    TTY      STAT   TIME COMMAND
19376 pts/17    S           0:00 bash
30005 pts/17    R           0:00  \_ ps f
20673 pts/13    S           0:00 bash
32152 pts/13    S           0:00  \_ mc
15158 pts/15    S           0:00      \_ bash -rcfile .bashrc
(...)
```

Wykorzystanie opcji f powoduje wyświetlenie drzewa procesów, uwzględniających zależność proces macierzysty - proces potomny Za pomocą polecenie pstree można wyświetlić drzewo procesów w systemie. przykład:

```
knoppix@ttyp0[knoppix]$pstree
init-+-MailScanner---5*[MailScanner]
     |-Server
     |-aacraid
     |-acpid
     |-arpwatch
     |-atd
     |-aveserver
     |-clamd
     |-crond
     |-dbus-daemon-1
     |-dccm
     |-dovecot-+-dovecot-auth
```

kill

W wielu przypadkach zachodzi potrzeba usunięcia przez użytkownika procesu z systemu Unix. Użytkownik ma takie prawo w stosunku do swoich procesów natomiast użytkownik root do wszystkich. Polecenia do tego służące ma następującą składnię:

```
kill [ -s sygnał | -p ]  [ -a ] pid ...
```

Po wydaniu polecenia kill z właściwym sygnałem, proces przerywa pracę i wykonuje kod obsługi sygnału. Część sygnałów służy do komunikowania procesu o kluczowych wydarzeniach przez jądro. W tabeli znajdują się najczęściej wykorzystywane sygnały,

nazwa	numer	dom. akcja	opis
SIGHUP	1	zakończenie	Wyłączenie terminala sterującego bądź śmierć procesu kontrolującego
SIGINT	2	zakończenie	Przerwanie z klawiatury (CTRL+C)
SIGQUIT	3	zrzut core	Wyjście nakazane z klawiatury
SIGILL	4	zrzut core	Próba wykonania nieprawidłowej instrukcji
SIGABRT	6	zrzut core	Sygnał przerwania pracy procesu wywołany przez abort()
SIGKILL	9	zakończenie	Natychmiastowe usunięcie procesu; niemożliwy do złapania ani zignorowania.

SIGSEGV	11	zrzut core	Nieprawidłowe odwołanie do pamięci wirtualnej
SIGPIPE	13	zakończenie	Zerwany potok: pisanie do potoku, który nie posiada procesu po stronie czytania
SIGALRM	14	zakończenie	Sygnał alarmowy wywołany przez funkcję alarm()
SIGTERM	15	zakończenie	Sygnał zakończenia pracy procesu
SIGCHLD	17	ignorowanie	Zatrzymanie bądź wyłączenie procesu potomnego
SIGCONT	18	start	Kontynuacja zatrzymanego procesu
SIGSTOP	19	zatrzymanie	Zatrzymanie procesu; niemożliwy do złapania ani ignorowania

Przykład:

```
knoppix@ttyp0[knoppix]$ cat /dev/zero > /dev/null &
[1] 8606
knoppix@ttyp0[knoppix]$kill -9 8606
knoppix@ttyp0[knoppix]$
[1]+  Unicestwiony            cat /dev/zero >/dev/null
knoppix@ttyp0[knoppix]$
```

killall

Stosowanie polecenia *kill* jest niezbyt wygodne, gdyż za każdym razem należy sprawdzić PID zatrzymywanego procesu. W systemach linuksowych dostępne jest polecenie *killall*, które odnajduje proces na podstawie nazwy. Najprostsza jego składnia to:

```
killall [-sygnał] nazwa
```

Uwaga: należy pamiętać, że polecenie wysyła sygnał do wszystkich procesów o podanej nazwie.

fuser

Polecenie wyświetle wszystkie procesy używające danegy plik:

```
fuser [-sygnał|-k] plik
```

Opcje pozwalają na wysłanie sygnału (-sygnal) lub zabicie (-k) wszystkich znalezionych procesów. Szczególnie przydatne przed odmontowaniem używanego systemu plików.

top

Top jest programem działającym w czasie rzeczywistym, prezentującym najbardziej absorbujące procesor i pamięć procesy w systemie. Po uruchomieniu, ekran terminala wygląda następująco:

```
20:50:46 up 21 days,  5:22, 35 users,  load average: 0,66, 0,54, 0,43
242 processes: 239 sleeping, 2 running, 1 zombie, 0 stopped
CPU states:   9,3% user,  18,4% system,   0,1% nice,  72,2% idle
Mem:    386248K total,   369808K used,    16440K free,    34032K buffers
Swap:   457844K total,   170436K used,   287408K free,   112924K cached

 PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
9248 gin          19    0  1916   1916  1584           R    12,4       0.4    0:00   top
   1    root          8    0   808     772   752             S     0,0         .1     0:04
init
   2    root          9    0     0           0       0          SW     0,0       0.0     0:19
keventd
   3    root          9    0     0           0       0          SW     0,0       0.0     0:00
kapmd
```

(...)

Standardowo, procesy sortowane są według zużycia procesora. Można jednak przełączyć sortowanie, naciskając jeden z klawiszy:

N
: według numeru PID

A
: według wieku

P
: według użycia procesora

M
: według użycia pamięci

T
: - według czasu pracy

praca w tle, fg, bg, jobs

Polecenia do zarządzania procesami na bieżącej konsoli.

Domyślnie po uruchomieniu procesu jego wyjście kierowane jest na bieżącą konsolę. Jednakże po wciśnięciu klawiszy **Ctrl-Z** konsola zostaje zwolniona a program **zatrzymany i pozostawiony "w tle"** (zakończenie działania powoduje zazwyczaj klawisz *Ctrl-C*!). W tym momencie użytkownik może zdecydować co zrobić z tym procesem.

Innym sposobem uruchomienia programu w tle jest wydanie polecenie zakończonego znakiem *&*:

```
polecenie &
```

Przy pomocy komendy *fg* można ponownie przenieść proces na pierwszy plan. Jednakże wiele procesów może pracować poprawnie w tle. Aby kontynuować pracę procesu w tle należy wydać polecenie *bg*.

Programy *fg* i *bg* uruchomione bez parametrów obsługują ostatnio zatrzymany proces. Istnieje możliwość obsłużenia innego procesu z danej konsoli. W tym celu należy uruchomić powyższe polecenia z parametrem:

```
bg [identyfikator zadania]
fg [identyfikator zadania]
```

W celu pobrania listy działających zadań należy wydać polecenie

```
jobs
```

nohup

Wiele programów nie pozwala na utworzenie procesu nie związanego z konsolą. Zazwyczaj uniemożliwiają tego programy aktywnie komunikaujące się z konsolą. Aby umożliwić tym programom pracę w tle służy polecenie:

```
nohup polecenie [argumenty]
```

Polecenie tworzy plik *nohup.out* do którego przekierowany jest wynik działania programu.

Pozostałe polecenia związane z procesami: nice, bg, fg, jobs, killall

Przykład:

```
nohup mc &
```

nice, renice

Procesy w systemach uniksowych mają określone priorytety, które system dobiera automatycznie na podstawie sposobu działania procesu. Użytkownik ma jednak możliwość wpływania na sposób dobierania priorytetu poprzez określenie wartości *nice* wpływającej na to jaki maksymalny priorytet może proces otrzymać. W systemie Linux liczba nice posiada wartości **ujemne** dla preferowanych zadań i dodatnie dla zadań o niższych priorytetach. "Najwyższą" wartością *nice* jest -20. Wartości ujemne może przypisywać procesom jedynie użytkownik *root*.

Aby uruchomić proces z zadaną wartością *nice* należy wydać polecenie:

```
nice [priorytet] polecenie [argumenty]
```

Aby zmienić wartość *nice* bieżącego procesu należy wydać polecenie:

```
renice [priorytet] PID
```

Zadania

- Uruchom program *top* i sprawdź jakie informacje wyświetla oraz jak zmienić sposób sortowania danych. (pomoc - klawisz *?*)
- Spróbuje zmienić częstotliwość odświeżania (*d*). Czy da się ostawić zerowy interwał.
- Uruchom 3 procesy top (na osobnych konsolach lub oknach xterm). Zmień wybranemu wartość *nice*. Sprawdź zachowanie procesów *top* przy zerowych interwałach.
- Utwórz plik i wyświetl go na 3 konsolach poleceniem *less*. Sprawdź działanie programu *fuser*. Spróbuj przy jego pomocy zakończyć działanie przeglądarek *less*.
- W XWindow wprowadź polecenie ethereal. Czy możesz na tej konsoli wykonywać inne polecenia? Czy znasz sposób pozwalający na wykonywanie inny poleceń na tej konsoli (poza zakończeniem pracy przez aplikację ethereal)?
- Przejdź do konsoli zablokowanej przez ethereal i naciśnij kombinację ctrl+z. Spowoduje to zatrzymanie tego procesu. Co się stało z aplikacją?
- Na konsoli z zatrzymanym ethereal'em wprowadź polecenie bg 1. Co się stało z aplikacją? Jak wytłumaczysz uzyskany efekt?
- Zamknij konsolę na której pracowałeś. Co stało się z aplikacją ethereal?

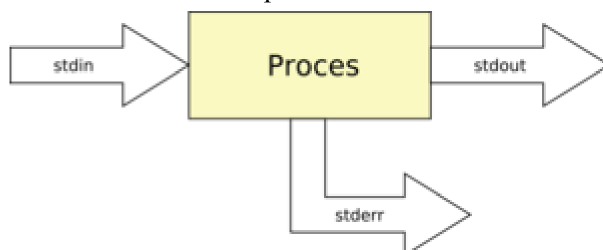
Strumienie w systemach Linux

W Linuxie z każdym procesem związane są tzw. strumienie. Z każdym procesem związane są zwykle trzy strumienie:

stdin
: standardowy strumień wejściowy, zwykle związany z klawiaturą - z niego pobierane są znaki do obróbki przez proces (np. komendy dla powłoki),

stdout
: standardowy strumień wyjściowy, zwykle związany z ekranem - ten strumień reprezentuje wszystkie dane wyprowadzane (wyświetlane) przez program,

stderr
: standardowy strumień błędów, również zwykle związany z ekranem - na ten strumień kierowane są wszystkie komunikaty o błędach. Dzięki zastosowaniu strumieni, poszczególne procesy nie są na stałe związane z klawiaturą czy ekranem, tylko z odpowiednim strumieniem. To powłoka decyduje o tym gdzie kierować dane z poszczególnych strumieni. Dzięki temu łatwo można przekierować standardowe związanie strumieni.



Standardowe strumienie w systemach uniksowych

Przekierowania do plików

Aby przekierować standardowy strumień wyjściowy (**stdout**) np. do pliku, wystarczy po treści komendy użyć znaku ">" i podać nazwę pliku wyjściowego. Najprościej prześledzić to na przykładzie. Komenda:

```
ls -l /etc
```

wyświetli zawartość katalogu /etc na ekran (dokładniej - do stdout, który związany jest domyślnie z ekranem). Zapis:

```
ls -l /etc > /tmp/etc.lst
```

spowoduje zapis w pliku /tmp/etc.lst listy plików w katalogu /etc. Przyjrzyjmy co się stanie w chwili wystąpienie błędów aplikacji z której przekierowujemy dane:

```
ls -l /nie_istniejacy_katalog > /tmp/cos
```

Komenda ta spowoduje utworzenie pustego pliku /tmp/cos i wyświetlenie na ekranie komunikatu:

```
ls /nie_istniejacy_katalog: No such file or directory
```

Komunikat ten trafił na ekran pomimo przekierowania **stdout**, dlatego, że jest komunikatem błędu. Komunikaty takie są wysyłanie do standardowego strumienia błędów, (**stderr**), a nie do stdout. Strumień ten również można przekierować. Dokonuje się tego podobnie jak w przypadku stdout - poprzez dodanie znaków "**2>**" i nazwy pliku po treści polecenia. Zatem:

```
ls -l /nie_istniejacy_katalog 2> /tmp/cos
```

spowoduje wpisanie komunikatu:

```
ls /nie_istniejacy_katalog: No such file or directory
```

do pliku /tmp/cos (czyli zawartość stderr), natomiast na ekran żaden komunikat nie będzie wyprowadzony.

Uwaga:

Wszystkie dotychczasowe przekierowania powodowały utracenie dotychczasowej zawartości plików wynikowych. Stosując "**>>**" zamiast ">" oraz "**2>>**" zamiast "2>" zawartość odpowiedniego strumienia zostanie dopisana do istniejącego pliku, zatem dotychczasowa zawartość pliku zostanie zachowana.

Przekierowania z plików

Czasami przydatne jest przekierowanie standardowego strumienia wejściowego (**stdin**), na przykład przy automatyzacji pracy poleceń czy skryptów. Wówczas proces z przekierowanym **stdin** będzie pobierał znaki wejściowe z pliku, zamiast z klawiatury. Przekierowanie to uzyskuje się poprzez zastosowanie operatora "<" i nazwy pliku wejściowego. Na przykład, jeśli zawartość pliku /tmp/in będzie następująca:

```
ls -l /etc
echo Gotowe!
```

wówczas wywołanie polecenia:

```
bash < /tmp/in
```

spowoduje wylistowanie zawartości katalogu /etc/ oraz wyświetlenie napisu "Gotowe".

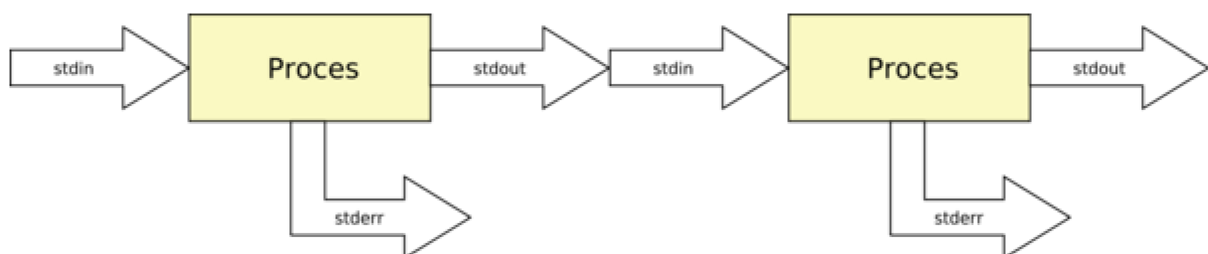
Przekierowania wszystkich trzech strumieni można ze sobą łączyć. Nawiązując do poprzedniego przykładu, poprawne jest wywołanie:

```
bash < /tmp/in >>/tmp/wynik 2>/dev/null
```

Polecenie to uruchomi kopię powłoki bash, wczyta z pliku /tmp/in komendy, wykona je, wynik pracy dołączy do pliku /tmp/wynik, a komunikaty o błędach zostaną zignorowane.

Przekierowania do aplikacji

Istnieje również możliwość przekierowania strumienia wyjściowego jednego procesu na strumień wejściowy procesu drugiego. Operacja ta nazywana jest **potokiem** (pipeline). Możliwość ta jest bardzo często wykorzystywana w codziennej pracy użytkownika systemu Linux. Przekierowanie takie realizowane jest przez podanie znaku "|" na końcu treści polecenia pierwszego (tzn. tego, którego **stdout** ma być przekierowany) oraz wpisanie treści drugiego polecenia (tzn. tego, do którego strumień ma trafić na **stdin**).



Potok w systemach uniksowych

Realizacja praktyczna jest dość prosta. Na przykład:

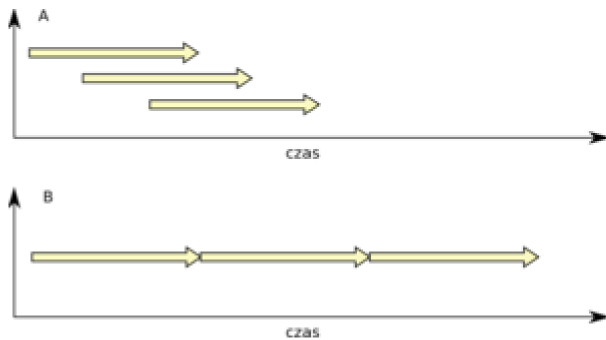
```
ls -l /etc | lpr
```

spowoduje wygenerowanie listy plików z katalogu /etc i przekazanie jej na strumień **stdin** komendy "lpr" (komenda lpr stanowi interfejs linuxowego systemu wydruku). Przekierowania takie można łączyć w dłuższe sekcje, na przykład:

```
ls -R / | sort | uniq | less
```

Posługując się powyższym przykładem można uzyskać podobny efekt zastępując znak „|" średnikiem „;" Nie jest to już potok ale wykonywanie kolejnych poleceń. Należy zwrócić uwagę na dwa aspekty tego przypadku:

- w odróżnieniu od potoków w tym przypadku strumień wyjściowy kierowany jest na kolejny strumień wejściowy procesu po zakończeniu działania procesu poprzedzającego. Tego nie ma w przypadku potoku, gdzie proces zaczyna działać w momencie pojawienia się na jego wejściu strumienia danych,



Potoki (A) a sekwencje wywołań (B)

- nie każdy ciąg poleceń rozdzielonych średnikiem można przekształcić w potok, np.

```
ls -R / ; ps aux ; cat cv.txt
```

nie jest równoważny

```
ls -R / | ps aux | cat cv.txt
```

Łączenie strumieni

Można przekierować obydwa strumienie jednocześnie, łącząc obydwie składnie. Na przykład:

```
ls -l /katalog >/tmp/wynik 2>/tmp/bledy
```

spowoduje utworzenie pliku /tmp/wynik z wynikiem działania powyższej komendy oraz pliku /tmp/bledy z treścią ewentualnych błędów.

W przypadkach, kiedy należy przekierować zarówno **stdout** jak i **stderr** do tego samego pliku, należy posłużyć się operatorem "**&>**", na przykład:

```
ls -l /katalog &>/tmp/wynik
```

lub

```
ls -l /katalog >/tmp/wynik 2>&1
```

Po wykonaniu powyższego polecenia, zawartość **stdout** jak i **stderr** zostanie zapisana w pliku /tmp/wynik, a na ekranie nie pojawią się żadne komunikaty. Składnię tą można wykorzystać do całkowitego tracenia wyniku zastosowanej komendy:

```
komenda &>/dev/null
```

Przekierowania a urządzenia

Ogromne korzyści można czerpać z połączenia mechanizmu strumieni z unixową reprezentacją urządzeń. Ponieważ systemy wywodzące się od Unixa przedstawiają urządzenia jako pliki (zazwyczaj w katalogu /dev) istnieje możliwość przekierowania danych do urządzeń np. (w zalezności od wersji sytemu nazwy urządzeń mogą być różne):

```
cat tekst_do_wydruku > /dev/prn
cat tekst_do_wydruku > /dev/lp0
cat tekst_do_wydruku > /dev/usb/lp0
```

Spowoduje wydruk pliku (bez jakiejkolwiek interpretacji!), zaś:

```
cat plik_z_dzwiękiem > /dev/dsp
cat plik_z_dzwiękiem > /dev/sound/adsp
```

spowoduje jego odegranie (niezależnie czy plik zawierał dane dźwiękowe!)

Czasami przydatne jest przekierowanie **stdout** lub/i **stderr** do tzw. urządzenia pustego (/dev/null).

Wówczas cokolwiek pojawi się w strumieniu wyjściowym nie zostanie wyświetlone ani nigdzie zapisane. Np.:

```
find / -name’costam’ 2> /dev/null
```

wyświetli odnalezione pliki, zaś zignoruje informacje o braku praw dostępu. Jest to przydatne, gdy wynik działania programu jest niepożądany - na przykład w przypadku skryptów automatyzujących działanie serwera, wykonywanych wiele razy na dobę.

Systemu Linux dysponują także urządzeniami logicznymi, które mogą dostarczyć istotnych informacji dla skryptów, np.: /dev/random, /dev/date itp. Istnieje także możliwość bezpośredniego dostępu do „surowych" danych na dyskach oraz w pamięci. Dostęp do nich ograniczony jest wyłącznie do administratora, gdyż możliwość czytania tych danych może być zagrożeniem dla bezpieczeństwa systemu, zaś zapis może spowodować nieodwracalne uszkodzenia systemu.

Zadania

- Utwórz plik o treści „Ala ma kota". Na ile sposobów potrafisz to zrobić?
- Przekieruj strumienie wyjściowe kilku komend do pliku, a następnie przejrzyj ich zawartość
- Uzyskaj w pliku dane na temat błędów popełnionych w wywoływanych komendach
- Dopisz do pliku już uzyskanego **stdout** i **sdterr** na dwa sposoby
- Wykonaj przekierowanie wszystkich 3 strumieni

Filtry strumieniowe

cat

Polecenie **cat** służy do wysłania wybranego pliku (lub kilku plików) na standardowe wyjście. Stanowi dobre narzędzie do rozpoczęcia przetwarzania strumieniowego. Może także służyć do sklejania grupy plików:

```
cat plik1 plik2 plik3 > suma_plikow
```

Ponieważ cat uruchomione bez parametrów pobiera dane ze standardowego wejścia może także służyć do wprowadzania danych z klawiatury np.:

```
cat > nowy_plik
```

spowoduje utworzenie nowego pliku w wypełnieniu go danymi wprowadzonymi z klawiatury.

head, tail

Polecenie **head** i **tail** pozwalają na wyświetlanie części pliku: odpowiednio początku i końca. Np.:

```
head -n 10 plik
```

wyświetli pierwszych 10 linii pliku.

Polecenie tail może spełnia szczególną funkcję po wywołaniu z parametrem "-f". Wyświetla ono wówczas koniec pliku i oczekuje na nowe dane. Może, więc służyć jako „monitor” pliku modyfikowanego przez inną aplikację (pracującą w tle lub na innej konsoli). Np.:

```
wget -t0 -rl5 -oout.txt -Pwp http://www.wp.pl/
tail -f out.txt
```

porządki (**wykonać poniższe instrukcje**):

```
killall wget
rm -r wp
rm out.txt
```

spowoduje pobranie portalu www.wp.pl. Pobieranie będzie realizowane w tle, a dzięki tail w dowolnej chwili można sprawdzić aktualny stan działania programu wget. Działanie tail -f można przerwać kombinacją Ctrl-C nie przerywając działania programu wget.

more, less

Polecenie **more** i **less** pozwalają na łatwiejsze przeglądanie strumienia wyjściowego. W przypadku dużej ilości danych konsola systemu zostaje „przewinięta” i część danych zostaje utraconych. Istnieje wprawdzie możliwość cofnięcia tekstu na konsoli (shift+pgup lub suwak w xterm) jednak bufor danych jest także ograniczony. Aby móc swobodnie czytać dane zwracane przez program wywołujemy polecenie:

```
komenda | more
```

Po każdym zapełnieniu ekranu more zatrzyma wyświetlanie danych i będzie oczekiwał na naciśnięcie dowolnego klawisza. Polecenie less jest wygodniejsze, gdyż pozwala na swobodne przewijanie danych. Pracę z danymi wyświetlonymi przez less możemy zakończyć naciskając klawisz ‘q’.

sort, uniq

Komenda **sort** służy do sortowania (domyślnie: alfabetycznego) linijek tekstu stanowiących dane wejściowe. Gdy się ją wywoła z argumentami będącymi nazwami plików, danymi do sortowania będzie zawartość tychże; w przypadku wywołania bez argumentów (nie będących opcjami, za pomocą, których można zadać bardziej złożone kryteria sortowania), komenda sort oczekuje, że dane do przetworzenia pojawią się w standardowym strumieniu wejściowym. W obu tych przypadkach, wynik sortowania pojawi się na **stdout**. Przykład:

```
cat /etc/passwd | sort
```

zwróci posortowaną listę użytkowników systemu.

Uzupełnieniem komendy sort jest komenda uniqe. Powoduje pominięcie wierszy powtarzających się. Np.:

```
cat /etc/passwd | sort | uniq
```

Nowe wersje polecenia sort mają możliwość usuwania powtarzających się linii, dzięki czemu komenda **uniq** traci na znaczeniu.

tr

Polecenie to służy do usuwania lub zastępowania znaków. Kopiuje znaki ze standardowego wejścia na standardowe wyjście, zastępując po drodze lub usuwając niektóre z nich.

Opcje:

-c
: (ang. complement) zamienia wszystkie znaki, oprócz tych, które występują w pierwszym łańcuchu (dopełnienie zbioru znaków o kodach ASCII 0 - 255)

-d
: kasuje z tekstu wejściowego znaki podane w pierwszym łańcuchu

-s
: jeżeli znak zawarty w drugim łańcuchu wystąpi w tekście wyjściowym kilka razy pod rząd, wielokrotność jest usuwana (wpisywany jest tylko jeden taki znak)

W nawiasach klamrowych można podać zakresy znaków, można też użyć zapisu [a*n], co oznacza n powtórzeń znaku a.

Przykłady:

```
tr ',' '\n' < dane > wynik
```

Polecenie to zastąpi wszystkie przecinki w pliku dane znakami końca linii, a wynik działania polecenia zostanie umieszczony w pliku wynik.

```
echo zapiski | tr asdkpz '.'
```

Polecenie to zastąpi litery a, s, d, k, p, z znakami kropki.

cut

Polecenie **cut** wypisuje wybrane części linii plików podanych w argumentach na standardowe wyjście. W przypadku braku argumentów czyta ze standardowego wejścia. Składnia polecenia:

```
cut [-c] list [file ...]
cut [-f] list [-d char] [-s] [file ...]
```

gdzie opcje:

-c – każdy znak jest polem,
list – oddzielona przecinkami lista numerów wycinanych pól np. 1,4,6-9,12-
    oznacza pole pierwsze, czwarte, od szóstego do dziewiątego oraz od 12 do końca linii
file – lista nazw plików
-f – pole to ciąg znaków oddzielony delimiterem (standardowo tabulatorem),
-d – ustawianie znaku delimitera,
char – dowolny znak lub znaki specjalne i spacja w cudzychsłowiu
-s – opuszczanie linii bez znaku delimitera.

Każdą linię dzielimy na pola za pomocą znaku zwanego separatorem. Domyślnie jest to znak tabulacji. Separator można podawać za pomocą opcji -d. Na przykład, dla linii

grep

Przy przekierowaniach często używa się komendy "**grep**". Komenda ta wyświetla linie pasujące (lub nie) do określonego wzorca. "grep" jest niezwykle rozbudowaną komendą, lecz zwykle używa się kilku jego podstawowych właściwości.

Uproszczona składnia:

```
grep [-v] WZORZEC [PLIK(I)]
```

gdzie:

-v
    oznacza negację wzorca (czyli wzorzec nie może wstąpić)
WZORZEC
    to wzór informacji do wyszukania,
PLIK(I)
    lista plików do kontroli. W przypadku nie podania nazw plików, "grep" pracować będzie na stdin.

Przykłady wykorzystania:

```
ls -l | grep student
```

spowoduje wyświetlenie zawartości tylko tych pozycji katalogu, gdzie znajduje się słowo "student" (czyli np. będących własnością studenta, posiadających słowo "student" w nazwie itp).

```
cat plik.c | grep include
```

Powyższe polecenie wyświetli wszystkie linie pliku plik.c, zawierające ciąg "include"

Wzorzec programu grep stanowi wyrażenie regularne. Wyrażenia regularne są to wyrażenia wzorcowe tworzone za pomocą liter i cyfr w połączeniu ze znakami specjalnymi, które działają podobnie do operatorów. Czynią one łatwiejszymi odnajdowanie i filtrowanie informacji w plikach.

Najważniejsze operatory wyrażeń regularnych

Znak	Opis
.	Dopasuj dowolny znak
$	Dopasuj poprzedzające wyrażenie do końca wiersza
^	Dopasuj występujące po operatorze wyrażenie do początku wiersza
*	Dopasuj zero lub więcej wystąpień znaku poprzedzającego operator
\	Oznacza pominięcie specjalnego znaczenia znaku np.: *
[]	Dopasuj dowolny znak ujęty w nawiasy. np.: [abc]
[-]	Dopasuj dowolny znak z przedziału. np.: [0-9] – wszystkie cyfry; [a-z] – wszystkie małe litery; [0-9a-zA-Z] – wszystkie litery i cyfry
[^]	Dopasuj znak, który nie znajduje się w nawiasach.

Przykłady:

```
grep 'Ala' plik      #znajduje wyraz Ala
grep 'A.a' plik      #znajduje wyrazy takie jak Ala, Aga, Ara, A+a i inne
grep 'A[lg]a' plik   #znajduje TYLKO wyrazy Ala i Aga
grep '^Ala' plik     #znajduje linię ‘Ala ma kota.’ ale odrzuca ‘To jest Ala.’
grep 'Go*gle' plik   #znajduje Gogle, Google, Gooogle itd.
grep '[0-9][0-9]*'   #znajduje dowolny ciąg cyfr
```

Z wyrażeń regularnych korzystają także inne programy, np.: **ed**, **sed**, **awk** i inne.

Cześć programów (np. sed, awk) potrzebuje, aby wyrażenie regularne ujęte było w znaki ‘/’ np.: /abc[0-9]/ inne programy (grep) przyjmuje wyrażenia regularne bez dodatkowych znaków np.: *grep abc[0-9] plik*. Należy jednak pamiętać, że część stosowanych znaków może zostać zinterpretowanych przez shell. Z tego powodu bezpieczniej jest użyć cytowania np.: *grep ‘ala ma .*’* plik.

Zadania

- Wygeneruj listę wszystkich plików w systemie, posortuj a następnie usuń duplikaty, zapisz to wszystko do pliku
- Z listy, którą otrzymałeś w poprzednim ćwiczeniu wygeneruj listę plików nagłówkowych (*.h) i zapisz do pliku
- Sprawdź parametry polecenia sort – czy pozwala ono na sortowanie według innych zasad niż „alfabetycznie”?
- Wyszukaj w /etc/X11/xorg.conf wszystkich sekcji „Input”