Laboratorium 4

4.1 Strumienie w systemach UNIX
4.2 Filtry strumieniowe

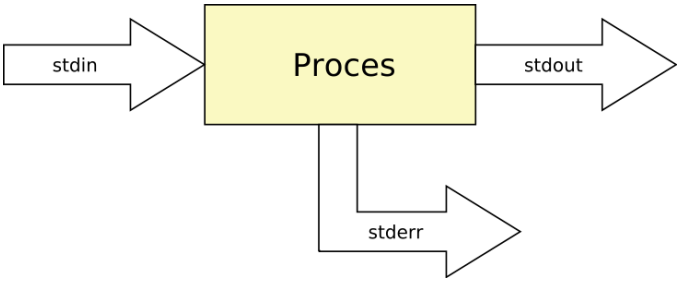
4.1 Strumienie w systemach UNIX

W Linuxie, jaz 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 /etce/ > list.lst
```

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

```
ls: nie ma dostępu do /etce/: 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 /etce/ > list.lst
```

spowoduje wpisanie do pliku **list.lst** komunikatu:

```
ls: nie ma dostępu do /etce/: No such file or directory
```

(czyli zawartość **stderr**), natomiast na ekran żaden komunikat nie będzie wyprowadzony.

Uwaga:
Wszystkie dotychczasowe przekierowania typu **„>”** 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".

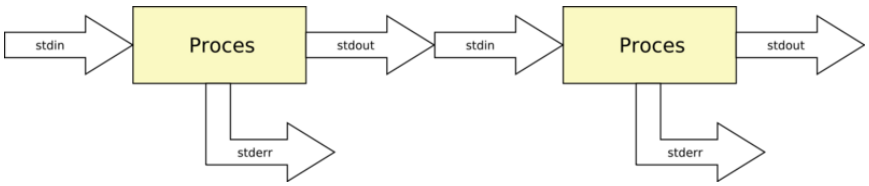
Przekierowania wszystkich trzech strumieni można ze sobą łączyć. Nawiązując do poprzedniego przykładu, poprawne jest wywołanie:

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

Polecenie to uruchomi kopię powłoki bash, wczyta z pliku /tmp/in wszystkie wpisane tam komendy, wykona je, wynik pracy dołączy do pliku /tmp/wynik, a komunikaty o wszystkich błędach, które potencjalnie wystąpią, 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 administratora 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:

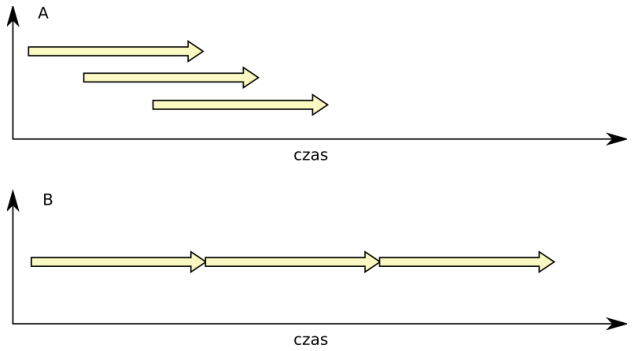
```
ls -l /etc | lpr
```

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 „|” -> „wykres A” średnikiem „;” -> „wykres B”. Nie jest to już wtedy potok ale wykonywanie kolejnych poleceń – sekwencja wywołań. Należy zwrócić uwagę na dwa aspekty tego przypadku:

- w odróżnieniu od potoków w tym przypadku (B) 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 (sekwencję wywołań) można przekształcić w potok, np.

```
ls –R / ; ps aux ; cat cv.txt
```

nie jest równoważny

```
ls –R / | ps aux | cat cv.txt
```

Pytanie do Państwa: dlaczego?

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 jego interpretacji!), zaś:

```
cat plik_z_dzwiękiem > /dev/snd/pcmC0D
```

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

1. Utwórz plik o treści „Ala ma kota". Na ile sposobów potrafisz to zrobić?
2. Przekieruj strumienie wyjściowe 5 komend (ls, who, finger, last, pwd) do pliku, a następnie przejrzyj ich zawartość;
3. Zapisz do pliku dane na temat błędów popełnionych celowo w wywoływanych komendach;
4. Dopisz do pliku z punktu 3 strumienie **stdout** i **sdterr** stosując dwa różne sposoby
5. Wykonaj przekierowanie wszystkich 3 strumieni do pliku jednocześnie. Czy jest to możliwe?

4.2 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.
Zakończenie wpisywanie tekstu -> Enter po ostatniej linii -> Ctrl+D

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
```

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.

Porządki!! (**proszę spróbować wykonać poniższe instrukcje, jeżeli wget nadal pracuje**):

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

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.

```
cat zapiski | tr asdkpz '[.*6]'
```

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

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 administratorzy używają kilka 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
```

Zadania

1. Wygeneruj listę wszystkich plików w systemie, posortuj a następnie usuń duplikaty, zapisz to wszystko do pliku o nazwie pliki_systemowe.out.
2. Z listy, którą otrzymałeś w poprzednim ćwiczeniu wygeneruj listę wszystkich plików nagłówkowych (*.h) i zapisz do pliku.
3. Sprawdź parametry polecenia sort – czy pozwala ono na sortowanie według innych zasad niż „alfabetycznie”?
4. Stosując komendę find znajdź na dysku plik lub pliki zawierające w sobie ciąg znaków „use-ssh-agent”. W celu rozwiązania zadania przejrzyć pomoc dla komendy find połączonej z grep dostępnej na poniższej stronie WWW: http://www.athabascau.ca/html/depts/compserv/webunit/HOWTO/find.htm