VSO – Laboratorijske vaje 2

Zaščita datotek
Uporabniki, skupine
Cevovodi in filtri
Procesi in opravila

Zaščita datotek

- Vsaka datoteka ima lastnika in lastniško skupino. Pravice (read, write, execute) lahko določimo posebej za lastnika (user), lastniško skupino (group) in vse ostale (others).

```
$ ls –l
-rw-rw-r--     1 vso studenti 0 nov 23 15:44 a.txt
$ chmod o-r a.txt
$ chmod ug+x a.txt
$ ls –l
-rwxrwx---     1 vso studenti 0 nov 23 15:44 a.txt
```

- Lastnika in lastniško skupino je mogoče zamenjati (root):

```
# chown janez a.txt
# chgrp izredni a.txt
# ls –l
-rw-rw-r--     1 janez izredni 0 nov 23 15:44 a.txt
```

Uporabniki, skupine

- Uporabnika ustvarimo in mu določimo geslo:

```
# useradd janez
# passwd janez
# cat /etc/passwd
```

- Ustvarimo skupino in vanjo dodamo uporabnika:

```
# groupadd izredni
# gpasswd –a janez izredni
```

- Če je uporabnik član več skupin, lahko skupino zamenja:

```
$ newgrp studenti
$ id
$ uid=506(janez) gid=505(studenti) skupine=505(studenti),507(janez),512(izredni)
```

Uporabniki, skupine

- Uporabnika odstranimo:

```
# userdel janez
# cat /etc/passwd

#useradd janez
#userdel –r janez
```

- Uporabnika odstranimo iz skupine, nato odstranimo skupino:

```
# gpasswd –d janez izredni
# groupdel izredni
```

Preusmerjanje

- Preusmerjanje - redirection omogoča procesom dostop do datotek, ki nadomestijo tipkovnico kot standardni vhod in terminal kot standardni izhod.
- Program, ki bere s tipkovnice lahko poženemo tako, da bo bral iz datoteke:

  bash-2.05$ prog <infile
- In izpisuje izhod namesto v terminal v datoteko:

  bash-2.05$ prog>outfile
- Kombiniramo lahko oboje:

  bash-2.05$ prog < infile >outfile
- Podobno lahko preusmerimo še standardne napake:

  bash-2.05$ prog < infile 2>errfile
- Ali preusmerimo standarden napake v standardni izhod

  bash-2.05$ prog 2>&1

Preusmerjanje in cevovodi

- Izhod nekega procesa lahko usmerimo v vhod drugega procesa (cev). Tako dobimo cevovod:

  bash-2.05$ ps aux | grep root

- Cevovodi so bistvo Unixa: uporabljati manjše osnovne gradnike, ki jih glede na želen rezultat povezujemo v cevovode
- Obstaja vrsta programov – filtrov, ki so temu prilagojeni
- grep [*options*] '*pattern*' [*file ...*]
- Iskanje določene besede (vzorca) v datoteki ali skupini datotek je zelo pogosto opravilo.
- grep pomeni “global regular expression print”, uporablja regularne izraze

Regularni izrazi

Znak	**Pomen**
^	Začetek vrstice.
$	Konec vrstice.
[*...*]	Eden izmed znakov.
[^*...*]	Katerikoli znak, ki *ni* naveden med oglatima oklepajema.
[*n-m*]	Območje znakov *n-m*.
.	Katerikoli znak, razen *newline*.
*c**	Večkrat (tudi ničkrat) ponovljen znak, ***c***.
.*	Nič ali več ponovitev katerega koli znaka.
\{*n*\}	Ustreza natančno *n* ponovitvam predhodnega znaka ali regularnega podizraza.
\{*n*,\}	Ustreza vsaj *n* ponovitvam predhodnega znaka ali regularnega podizraza.
\{*n*,*m*\}	Ustreza med *n* in *m* ponovitvam predhodnega znaka ali regularnega podizraza.
\	Ubežni znak.

Primeri

- grep ‘hello’ myfile
  Išče podniz ‘hello’ v datoteki ‘myfile’

- grep ‘hello’ *
  Išče hello v vseh datotekah v trenutnem direktoriju

- grep –l ‘hello’ *
  Izpiše imena vseh datotek v trenutnem direktoriju, ki vsebujejo besedo hello

Primeri

- grep ‘he*lo’ myfile
- grep ‘he.lo’ myfile
- grep ‘he.*lo’ myfile
- grep ‘^#include’ *.c
- grep ‘last\.$’ *.c
- grep ‘[uU]rgent’ myfile
- grep ‘s[aei]t’ myfile
- grep ‘[A-Z]’ myfile
- grep ‘^[A-Za-z]’ myfile
- grep ‘b[^i]d’ myfile
- grep ‘he\{1,2\}llo’
- grep ‘([0-9]\{2\}) [0-9]\{4\}-[0-9]\{3\}’ myfile

Filtri

- tr [options] SET1 [SET2]
  - tr ABC abc

    Zamenja A z a, B z b, C s c.
  - tr –d W

    Izbriše vse črke W
  - tr ‘ ‘ ‘\n’

    Presledke nadomesti z znakom za novo vrstico

Filtri

- sed [*options*]‘*commands*’ *filenames* …
- Ukazi (commands)
  - s/*re1*/*re2*/ zamenjaj (substitute) regularni izraz *r1* z *r2*, prva pojavitev v vsaki vrstici
  - s/*re1*/*re2*/g zamenjaj (substitute) regularni izraz *r1* z *r2*, vse pojavitve v vsaki vrstici
  - /*re1*/q izpisuje vrstice do tiste, ki ustreza *re1*, potem se konča (quit)
  - /*re1*/s/*re2*/*re3*/ zamenjaj regularni izraz *re2* z *re3*, prva pojavitev v vsaki vrstici, ki ustreza regularnemu izrazu *re1*

Delo z besedilom

- Štetje znakov, besed, vrstic
  - ps ax | wc -l
- Podvojene vrstice
  - uniq
- Urejanje
  - sort
- Prvih nekaj, zadnjih nekaj vrstic
  - head, tail
- Iskanje polj v datotekah
  - cut /etc/passwd –d: -f1
- Izpis nazaj
  - tac myfile

Iskanje datotek

- find path [expression]
  - Iskanje datotek v poddrevesu path. Datoteke lahko tudi procesira.
  - Določimo lahko:
    - Kje iskati (path)
    - Kaj iščema; direktorije, datoteke, *povezave* (-type: d, f, l)
    - Kaj narediti z najdenimi datotekami (-exec: nad najdenimi datotekami poženemo proces)
    - Ime datotek (-name)
- Primeri:
  - Iskanje datotek z določenim imenom

    find /etc -name “smb*" -print

    find /etc -name “smb*” -exec cp {} . \; -print

    find /etc -regexp “smb.*” -exec cp {} . \; -print

Iskanje datotek

- PATH (pot do izvršljivih datotek)

  echo $PATH

  PATH=$PATH:~/bin
- Iskanje datotek v poti (PATH)
  - which cat

    /bin/cat
  - which –a cat

    /bin/cat
- locate smb.conf
- updatedb

Procesi v Linuxu

- Ko se proces ustvari, se mu dodeli tudi naslednje datotečne deskriptorje (file pointers):
  - STDIN: standardni vhod (tipično je to tipkovnica)
  - STDOUT: standardni izhod (tipično je to terminal, ekran)
  - STDERR: standardne napake (tipično je to terminal, ekran)
- Standardni proces v Linuxu dobiva vhodne podatke s STDIN, izpisuje rezultate na STDOUT in izpisuje sporočila o napakah na STDERR.

Procesi v Linuxu

- Kaj se dogaja s STDIN in STDOUT, ko v lupini poženemo ukaz?
- Ko poženemo ukaz, lupina ustvari nov proces, otroka in STDIN in STDOUT (in STDERR) poveže na svoje STDIN, STDOUT in STDERR (tipkovnico in terminal). Sedaj lahko proces bere STDIN in izpiše rezultate na STDOUT namesto lupine. Lupina počaka, da se otroški proces konča in izpiše najavko na svoj STDOUT.

Procesi v Linuxu

- Proces je program v teku – program je pognan, running.
- Ko poženemo aplikacijo ali ukaz, ustvarimo nov proces; v sistem dodamo proces (npr. večkrat poženemo kosole)
- V Linuxu hkrati teče več procesov, ki pa niso nujno aktivni. Izpišemo jih lahko z ukazom ps (process status)
- Procesi so oštevilčeni, vsak ima svoj PID Process IDentification number; PID se procesu dodeli v takrat, ko proces ustvarimo.

Stanja procesov, signali

- Proces je lahko v različnih stanjih, vključno z: running, stopped (ali suspended) in terminated.
- Mehanizem za spreminjanje stanja procesom so signali.
- Signali so osnovna oblika asinhrone medprocesne komunikacije (IPC), uporabljamo jih s pomočjo ukaza kill.
- Prenaša se zelo malo podatkov, samo številka signala.
- Tipično nastopata dva procesa: pošiljatelj in prejemnik signala. Pošiljatelj mora pripadati istemu uporabniku (UID) kot prejemnik, ali pa superuserju.

Stanja procesov, signali

- Ko prejemnik sprejme signal, lahko naredi naslednje:
  - Ga ignorira
  - Proces se lahko uniči (terminate)
  - Izvrši se posebna rutina - handler.
- PRIMERI SIGNALOV

  SIGKILL 9

  SIGINT 2 (CTRL+C)

  SIGSTOP 20 (CTRL+Z)
- Pošiljanje signalov iz ukazne vrstice:

  bash-2.05$ kill -9 pid

Multiprogramiranje

- V linuxu hkrati teče več procesov, procesom se dodeljuje (scheduler) časovne rezine - time slicing.
- Procesom lahko spreminjamo prioriteto; ukaza nice in renice.
- Procese izpišemo z ukazom ps (process status):

  bash-2.05$ ps –a

  - PID: Process IDentification
  - TT: Terminal, ko se požene lupina, jo jedro poveže s terminalom
  - TIME: Čas, ki ga je proces dobil do sedaj
  - CMD: Ime ukaza, ki je ustvaril proces

ps

Izpiše uporabnikove procese
ps –u janez

Izpiše procese, ki so bili pognani z določenim ukazom
ps –C xclock

Izpis brez glave (uporabno v skriptih)
ps –C xclock --no-heading

Oblikovanje izpisa
ps –C xclock –o cmd,pid --no-heading

man ps…

Opravila

- Vsako opravilo, ki ga požene lupina, dobi številko opravila (job id).
- PID je številka, ki jo procesu dodeli jedro. Številko opravila dodeli procesu lupina. Jedro te številke ne pozna. Zakaj je potrebna?
  - Nadzor izvajanja procesa (pošiljanje signalov procesom v trenutni lupini z ukazi kot so fg, bg)

Opravila

- Vrste opravil:
  - Opravilo v ospredju ima dostop do tipkovnice in terminala (interaktivne aplikacije, ukaz fg)
  - Opravilo v ozadju nima dostopa do tipkovnice, ima pa dostop do terminala za izpisovanje rezultatov (ukaz bg)
  - Ustavljeno (stopped) opravilo je opravilo, ki je bilo začasno ustavljeno – običajno s signalom SIGTSTOP. Opravilu v ospredju lahko ta signal pošljemo s ctrl+z

Ukazi za delo z opravili

- Izpis opravil:

  bash-2.05$ jobs

  izpiše vsa opravila in njihovo stanje, ki pripadajo trenutni lupini.

- Opravilo lahko iz ospredja postavimo v ozadje:

  1. Najprej ga ustavimo s ctrl+z in
  2. Poženemo ukaz bg, ki ustavljeno opravilo požene v ozadju. (ukaz bg lahko podamo številko opravila (job ID), sicer se privzame upravilo, ki smo ga nazadnje ustavili.)

     bash-2.05$ bg %2

Ukazi za delo z opravili

- Opravilo iz ozadja postavimo v ospredje (povežemo s tipkovnico) z ukazom fg. Tudi fg jahko podamo številko opravila (job ID), sicer privzame nazadnje ustavljeno opravilo.

  bash-2.05$ fg

- Ko v lupini poženemo ukaz, jedro običajno požene nov proces (opravilo) v ospredju. To lahko preprečimo in zahtevamo, da se proces požene v ozadju tako, da za ukazom napišemo znak ‘&’ (npr. xclock &). Tako lupina ostane v ospredju in lahko sprejema nove ukaze, medtem ko proces teče v ozadju…