Managementul proceselor Linux

Obiectivele lucrarii

- gestionarea proceselor folosind interfata shell;
- gestionarea proceselor folosind interfata de programare C;
- functii C pentru generarea si tratarea semnalelor;
- sincronizarea proceselor folosind semnale

Atentie: Fiecare student va lucra intr-un director cu numele grupei, urmand a-l sterge la sfarsitul laboratorului:

$ mkdir /home/student/44xy
$ cd /home/student/44xy
>>>>> la sfarsit laborator>>>>>:
$ cd; rm –rf /home/student/44xy; sudo poweroff

1. Gestionarea proceselor folosind interfata shell

Redirectarea intrarii/iesirii unui proces catre un fisier

Exista situatii in care se doreste ca analiza rezultatului unui proces sau a mesajelor de eroare sa se faca prin citire dintr-un fisier arbitrar, chiar daca procesul a fost proiectat pentru a scrie in fisierele **stdout** (are descriptorul 1) si/sau **stderr** (are descriptorul 2).

Redirectarea stdout spre un fisier arbitrar, se face astfel:

```
$ <nume_program>  >  <fisier>
```

Redirectarea stderr spre un fisier arbitrar, se face astfel:

```
$ <nume_program>  2 > <fisier>
```

Redirectarea stderr catre stdout, se face astfel:

```
$ <nume_program>  2>&1 > <fisier>
```

Redirectarea stdout si stderr catre un fisier arbitrar se realizeaza astfel:

```
$ <nume_program>  &> <fisier>
```

Similar, intrarea implicita a unor procese, care este fisierul **stdin** (are descriptorul 0), se doreste sa devina un fisier arbitrar.
Acest lucru este posibil prin redirectarea intrarii procesului, astfel:

```
$ <nume_program>  <   <fisier>
```

Alte fisiere speciale pentru redirectare; de iesire: /dev/null; de intrare: /dev/zero, /dev/random

Determinare identificatorului asociat unui proces

```
$ pidof <nume proces>
```

$ pgrep <nume proces>

Determinarea informatilor de stare pentru un proces

Informatii despre procese care pot fi obtinute cu ajutorul comenzii “ps”.

Nume camp	Semnificatie
USER	Proprietarul procesului
PID	Identificatorul procesului
%CPU	Intervalul de timp total in care un proces s-a aflat in starea RUNNING, exprimat in procente din intervalul de timp de la crearea procesului.
%MEM	Volumul de memorie RSS ocupat de un proces in memoria fizica, exprimat in procente.
VSZ	Volumul de memorie virtuala ocupat de un proces (code+data+stack) , exprimat in kB. Nu include tabelele din kernel asociate procesului.
RSS	Volumul de memorie cu adresare fizica utilizat de un proces care nu a fost transferat in swap, exprimat in kB (en. Resident Set Size)
TTY	Terminalul asociat procesului
STAT	Poate avea valorile: **D** = uninterruptible sleep (usally IO); **R** = running or runnable (on run queue); **S** = interruptible sleep (waiting for event to complete); **T** = stopped (by control signal or tracing proccess); **X** = dead (should never been seen); **Z** = defunct/zombie In plus, mai este utilizat un set de caractere cu semnificatia urmatoare: **<** = procesul are prioritate ridicata; **N** = procesul are prioritate scazuta; **s** = acest proces este leader de sesiune; **+** = procesul este membru al grupului proceselor foreground
START	Momentul de timp la care a fost creat procesul.
TIME	Intervalul de timp total CPU (user+system) acumulat de un proces. Exprimat in formatul “MMM:SS”.

Afiseaza toate procesele:

```
$ps -e
```

Afiseaza toate procesele si relatiile tata-fiu

```
$ps -eH
```

Afiseaza toate procesele inlcuzand PPID

```
$ps -ef
```

```
$ps -efx
$ps -auxI
```

Stari ale proceselor:

Nume stare	Semnaificatie

Ready	procesele in aceasta stare sunt in executie, ceea ce inseamna ca partajeaza resursa CPU.
Sleep	procesele in aceasta stare sunt in asteptarea producerii unui eveniment: acces la resursa, receptionare semnal.
Terminate	procesele in starea terminata sunt procese pentru care se vor elibera resursele asociate de kernel.
Zombie	este un proces fiu terminat a carui informatii de stare nu au fost inca preluate de catre un proces parinte. In aceasta stare un procesul fiu mai ocupa numai intrarea din tabela proceselor. La terminarea procesului fiu, kernelul trebuie sa pastreze informatii cu privire la modul in care s-a terminat. In mod implicit aceste procese vor fi terminate de catre procesul init. Procesele aflate in aceasta stare vor avea atribuita valoarea <defunct>.
Orphan	este un proces fiu a carui parinte s-a terminat. Parintele acestor procese devine procesul init.

Tema: alternativ se poate utiliza comanda: pgrep. Sa se studieze modul de utilizare al acestei comenzi; care sunt avantajele utilizarii ei.

Determinare timpului de executie pentru un proces

```
$time <nume proces>
```

Implicit afiseaza urmatorii parametrii: durata reala de executie a procesului, timp de executie in modul utilizator, timp de executie in modul kernel.

Determinarea resurselor utilizate de un proces

Informatii despre procese care pot fi obtinute cu ajutorul comenzii “top”.

```
top - 19:30:46 up  9:01,  3 users,  load average: 0.16, 0.12, 0.09
Tasks: 121 total,   1 running, 120 sleeping,   0 stopped,   0 zombie
Cpu(s):  7.3%us,  0.0%sy,  0.0%ni, 92.0%id,  0.0%wa,  0.0%hi,  0.7%si,  0.0%st
Mem:    905556k total,   474488k used,   431068k free,    31656k buffers
Swap:  1228964k total,        0k used,  1228964k free,   221032k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3324 rlupu     15   0 88084  33m  14m S  6.0  3.8   1:07.34 totem
  822 root      11  -5     0    0    0 S  0.7  0.0   0:24.38 kjournald
 3322 rlupu     15   0 88084  33m  14m S  0.7  3.8   0:04.52 totem
 2430 root      5  -10 74676  33m  10m S  0.3  3.8  77:12.93 Xorg
 3312 rlupu     16   0  2116 1132  844 S  0.3  0.1   0:01.59 top
    1 root      16   0  1844  624  536 S  0.0  0.1   0:00.14 init
    2 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    4 root      10  -5     0    0    0 S  0.0  0.0   0:00.42 events/0
    5 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
    6 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
    8 root      10  -5     0    0    0 S  0.0  0.0   0:00.38 kblockd/0
    9 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
   95 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khubd
   97 root      10  -5     0    0    0 S  0.0  0.0   0:00.02 kseriod
  158 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
  159 root      15   0     0    0    0 S  0.0  0.0   0:01.88 pdflush
  160 root      25   0     0    0    0 S  0.0  0.0   0:00.00 kswapd0
  161 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 aio/0
  784 root      16   0     0    0    0 S  0.0  0.0   0:00.00 khpsbpkt
```

Rezultatul comenzii top

Nume camp	Semnificatie
PID	Identificatorul procesului
USER	Proprietarul procesului
PR	Prioritatea procesului
NI	Valoarea parametrului nice pentru ajustarea nivelului de prioritate pentru proces. O valoare cat mai negativa inseamna o prioritate cat mai mare.
VIRT	Volumul total al memoriei virtuale utilizate de proces, exprimat in unitati kb (en. Virtual Image). Include: codul si datele procesului, bibliotecile dinamice partajate. VIRT = SWAP+RES;
RES	Volumul de memorie fizica utilizata de proces, care nu a fost transferata in swap, exprimat in unitati kb (en. Resident size). RES=CODE+DATA;
SHR	Volumul de memorie partajata utilizata de proces, exprimata in unitati kb (Shared memory size). Reflecta volumul de memorie care ar putea fi partajat cu alte procese.
S	Starea procesului. Poate avea valorile: **D** = “uninterruptible sleep”; **R** = “running”; **S** = “sleeping”; **T** = “traced or stopped”; **Z** = “zombie”;
%CPU	Intervalul de timp CPU alocat unui proces, exprimat in procente din timpul total CPU masurat de la ultima afisare.
%MEM	Volumul de memorie fizica utilizat de un proces, exprimat in procente.

TIME+	Intervalul de timp CPU total alocat unui proces, exprimat in sutimi de secunda.

Determinarea fisierelor deschise de procese

Toate fisierele deschise:

```
$lsof
```

Toate fisierele socket deschise de un process specificat prin pid:

```
$lsof –i –p 1234
```

Toate fisierele deschise pe un dispozitiv de stocare:

```
$lsof /dev/hda1
```

Toate fisierele deschise de un utilizator specificat prin UID:

```
$lsof –u root
```

Toate fisierele cu descriptorii 1 si 3 deschise de procesul “lsof” lansat de utilizatorul “root”.

```
$lsof –c lsof –a –d 1 –d 3 –u root -r10
```

Determinarea proceselor care utilizeaza un anumit fisier

In exemplul de mai jos se arata cum se poate determina procesul care a deschis un socket daca se cunoaste numarul de port:

```
$ fuser –n tcp 22
```

sau, acelsi lucru poate fi exprimat si astfel:

```
$ fuser ssh/tcp
```

De asemenea, aceasi comanda poate fi utilizata pentru a determina toate procesele care au deschis un anumit fisier sau un fisier oarecare localizat pe un anumit dispozitiv de stocare :

```
$ fuser  /dev/sda7 -v
```

Transmiterea unui semnal catre un proces

Listarea tuturor semnalelor suportate de sistemul de operare:

```
$kill -l
```

Tipuri de semnale	ID semnal	Semnificatie
SIGHUP	**1**	Semnal primit de fiecare proces la deconectarea terminalului sau de control.

SIGABRT	**6**	Semnal generat de functia abort() pentru a arata ca procesul s-a terminat anormal.
SIGKILL	**9**	Semnal transmis pentru terminarea unui process. Nu poate fi tratat sau ignorat. Cu exceptia unor situatii critice, se recomanda folosirea SIGTERM.
SIGUSR1	**10**	Semnal cu semantica definita de utilizator.
SIGUSR2	**12**	Semnal cu semantica definita de utilizator.
SIGALRM	**14**	Semnal generat de functia alarm() catre acelasi proces la expirarea unui temporizator.
SIGTERM	**15**	Semnal de terminare process. Acest semnal poate fi tratat (ex. ptr. stergerea fisierelor temporare).
SIGCONT	**18**	Semnal transmis pentru continuarea executiei unui proces oprit cu SIGSTOP. Daca procesul nu era oprit, va fi ignorat.
SIGSTOP	**19**	Semnal pentru punerea in asteptare a unui proces, cu posibilitate ulterioara de a fi continuat. Nu poate fi ignorat.

Transmiteti un semnal:

```
$kill –s <signal name> <PID>
```

Alternativ, se pot utiliza comanzile pkill sau killall, care au avantajul ca poate primi ca argument, direct numele procesului in locul PID. In consecinta, daca exista mai multe instante ale aceluiasi proces, semnalul va fi transmis catre toate aceste instante. Aceleasi exercitii propuse pentru comanda kill pot fi realizate utilizand aceste comenzi.
De exemplu, pentru terminarea unui process:

```
$pkill –signal <signal name> <nume proces>
```

Modificarea nivelului de prioritate al unui proces (pornirea unui proces cu un nivel de prioritate altul decat cel implicit):

```
$nice <nume proces>
```

Modificarea prioritatii unui proces in curs de rulare:

```
$renice <[-20-19]> <PID>
```

2. Automatizarea operatiilor folosind scripturi

Este posibila automatizarea operatiilor folosind facilitatile de scripting ale inerpretorului de comenzi bash:

In acest scop veti crea un fisier cu secventa de comenzi pe care doriti sa o automatizati; exemplu:

```
$ mcedit script.sh
#! /bin/bash
echo -n "alegeti o culoare (blue, yellow, red ): "
read -e COLOR
```

```
setterm -background $COLOR
echo Ecranul este colorat ☺ $COLOR

In prima linie a scriptului puteti specifica optional interpretorul ce
va rula scriptul.

Pentru rulare, puteti specifica direct shell-ul ce va rula scriptul:
      $ bash script.sh
```

sau puteti rula scriptul ca fisier executabil, dupa ce ati atribuit permisiunile necesare:

$ chmod +x script.sh
$ ./script.sh

Exercitiul 1: Creati un script care lanseaza browserul in background si dupa un interval predefinit de timp in inchide.

Hints**:** rularea in background se face cu &, man sleep

Exercitiul 2: Creati un script care executa a ping catre toate masinile din reteaua locala

Hints: man for, man ping

3. Gestionarea proceselor folosind interfata de programare C

Crearea unui proces: fork()

Procesul care apeleaza functia fork() pentru a crea un fisier se numeste proces parinte, iar procesul creat se mai numeste proces fiu.

Procesele create astfel au segmentul de text identic cu al procesului parinte, insa nu partajeaza segmentul de date, stiva si heap cu acesta. Atunci cand procesul fiu se termina va trimite un semnal SIGCLD catre procesul parinte.

Prototipul functiei fork este urmatorul:

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

Daca variabila **pid** memoreaza valoarea returnata de functia fork, atunci daca:

pid < 0 – semnifica producerea unei erori in executia functiei ;
pid == 0 - este valoarea citita de procesul fiu;
pid > 0 - este valaorea citita de procesul parinte

Exercitiul 1

Sa se scrie un program care utilizeaza apelul fork() pentru a creea un proces. Determinati valoarea intoarsa de functia fork() proceselor parinte si fiu. Incercati sa creati mai multe procese fiu.
Indicatie: utilizati documentatia man fork pentru a obtine prototipul functiei.

Exercitiul 2

Creati un proces fiu care sa stea in asteptare circa 15 secunde. La terminarea procesului parinte, care trebuie sa fie imediata, inspectati starea procesului fiu (ex. ps -aux). Ce observati ? Care este PIDul procesului fiu ?
Indicatie: utilizati functia sleep(); pentru detalii man 3 sleep

Exercitiul 3
Creati un proces fiu a carui parinte este pus sa stea in asteptare circa 15 secunde. La terminarea procesului fiu, care trebuie sa fie imediata, inspectati starea procesului fiu (ex. ps -aux). Ce observati ?
Indicatie: utilizati functia sleep().

Terminarea unui proces: exit()
De regula, terminarea unui proces din proprie initiativa se realizeaza prin apelul functiei exit(). Argumentul functiei exit() este un cod de eroare intors catre procesul parinte pentru o eventuala analiza.

Asteptarea unui proces: wait()
De regula, procesul parinte trebuie sa astepte terminarea proceselor fiu pentru a analiza starea acestora. La terminarea unui proces fiu, procesul parinte receptioneaza un semnal SIGCLD de la kernel impreuna cu un cod de eroare de la procesul fiu.
Prototipul functiei wait este urmatorul:

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

Daca variabila **pid** memoreaza valoarea returnata de functia wait, atunci daca:
pid < 0 - semnifica producerea unei erori in executia functiei ;
pid > 0 - este identificatorul procesului fiu care s-a terminat
Variabila **status** reprezinta informatia de stare a procesului fiu la terminarea acestuia.
Pentru a interpreta valoarea status au fost predefinite urmatoarele macrouri (man wait): **WIFEXITED(status)**, **WIFSIGNALED(status)** si **WIFESTOPPED(status)**.

Exercitiul 4
Propuneti o solutie pentru a evita ca un proces fiu sa ajunga in starea orfan (se va relua exercitiul 2). Apoi, afisati modul in care a fost terminat procesul fiu.
Indicatie: utilizati functia wait() sau waitpid().

Exercitiul 5
Daca un proces parinte poate astepta terminarea unui proces fiu prin executarea functiei wait(). Aratati un mod prin care procesul fiu poate astepta terminarea procesului parinte.
Indicatie: procesul init devine implicit parinte; a se vedea si functia getpid().

Lansarea in executie a unui program: exec(), system()
Presupune inlocuirea procesului care lanseaza un nou proces cu acesta din urma. Mai exact, se inlocuiesc segementele text si de date ale procesului apelant cu cele ale noului proces.
Prototipul functiei execlp este urmatorul:

```
#include <unistd.h>

int execlp (const char *filename, const char *arg1, …, NULL);
```

Daca variabila **cod** memoreaza valoarea returnata de functia execlp, atunci daca:
pid < 0 – semnifica producerea unei erori in executia functiei ;

In caz de succes, nu se revine din functie.

Exercitiul 6
Scrieti un program emulator MSDOS CLI pentru 2-3 comenzi. Acesta va citi comenzile MSDOS de la tastatura si va lansa in executie comenzile corespunzatoare din bash.
Indicatie: utilizati functia execlp().

Exercitiul 7
Scrieti un program care sa afizeze periodic numarul total de pachete transmise pe o interfata de retea.
Indicatie: utilizati functia system() si comenzile shell: ifconfig si grep.

Semnale
Semnalul este un mijloc prin care se poate realiza o intrerupere software a unui proces pentru a fi notificat asupra producerii unui eveniment. Categoriile de evenimente pentru care se poate transmite un semnal sunt: exceptii in executia unui proces, alarme, terminare proces anormala sau pentru comunicatii intre procese.
Semnalele pot avea ca origine un alt proces utilizator sau kernelul sistemului de operare. Procesul care primeste un semnal nu poate sa determine care este originea semnalului.
Exista mai multe tipuri de semnale, fiecare cu semantica specifica evenimentului care il produce (a se vedea tabela de mai sus sau fisierul ...../asm-i386/signal.h din sursele kernelului).
Prototipul functiei signal pentru asignarea unei functii de tratare a unei intreruperi software este urmatorul:

```
#include <signal.h>

int signal(int semnal, void (*handler)(int));
```

Valoarea intoarsa de functie reprezinta un cod de eroare.

Tratarea unei intreruperi: signal()
Exercitiul 8
Sa se scrie un program care sa afiseze tipul semnalului pe care il primeste. Se vor considera semnalele: SIGUSR1, SIGINT, SIGQUIT, SIGKILL si SIGALRM. Pentru validare generati semnale catre proces utilizand comanda kill din bash si functia C alarm() in cazul SIGALRM.

Transmiterea unui semnal: kill()
Exercitiul 9
Sa se scrie doua procese care isi trimit reciproc semnale SIGUSR1 cu perioada de 2 secunde. Utilizati functia pause() pentru a sincroniza cele doua procese (functia pause() determina oprirea din executie a unui proces pana la sosirea unui semnal).

Bibliografie suplimentara:

http://tldp.org/LDP/abs/html/
http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html