Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu
Zavod za elektroničke sustave i obradbu informacija
Laboratorij za sustave i signale

Programska podrška mjernih i procesnih sustava

vježba br. 2: **PROCESI**

NAPOMENE

- Za pojašnjenja naredbi u komandnoj liniji upisati naredbu:
  `ime_naredbe --help` (ne radi za sve naredbe)
  MAN(unal) stranice daju više informacija o pojedinoj naredbi i mogu se pogledati ukoliko se u komandnoj liniji upiše:
  `man ime_naredbe`

ZADACI ZA VJEŽBU:

1. MOUNTANJE

Linux omogućava postupak *mount*-anja drugog file systema. Na taj način novi file system postaje dio dosadašnjeg i nalazi se unutar njegovog datotečnog stabla. Dakle i dalje postoji samo jedan root (/) direktorij. Za usporedbu, MS Windows svaki novi file system (disketa, usb-stick, dijeljeni mrežni direktorij) tretira kao novi uređaj (a:, z:) i svaki ima svoj root direktorij (a:\, z:\).

Nama će to trebati kako bi mogli pristupati dijeljenom mrežnom direktoriju "lab-d138" na računalu Diana (to je z: disk pod Windowsima).

U tu svrhu potrebno je u lokalnom matičnom direktoriju `/home/knoppix` napraviti poddirektorij `d138`. Nakon toga može se primijeniti slijedeća naredba:

```
# mount -t smbfs //diana/lab-d138 d138
```

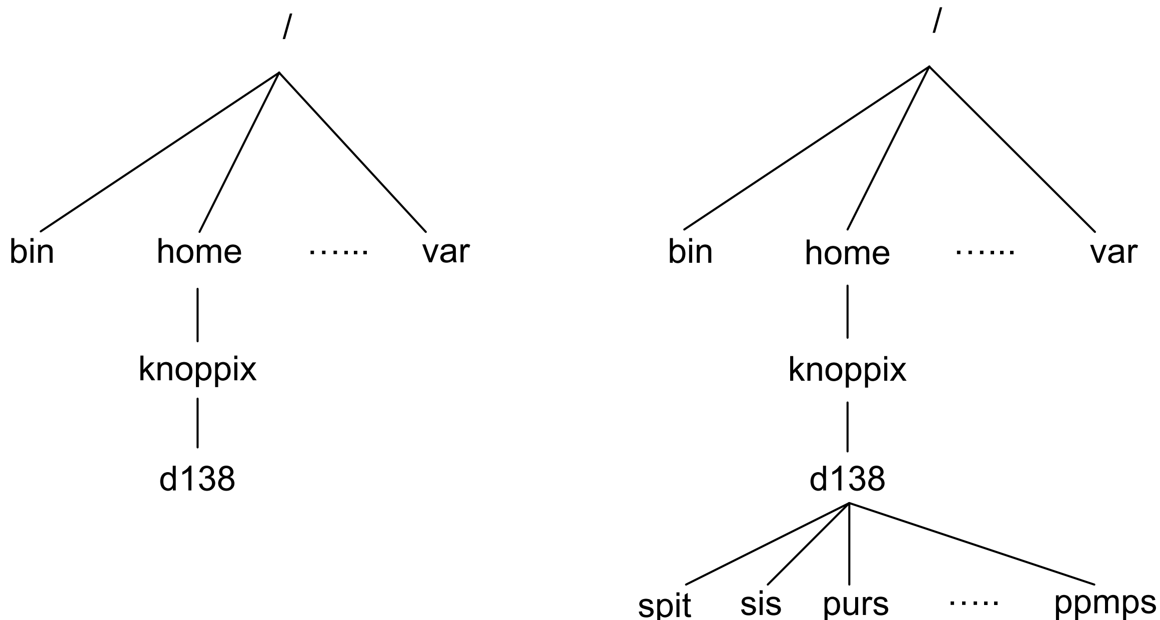
Potrebno ju je izvršiti kao *root* (password je prazan). Opcija `-t smbfs` određuje da se radi o samba file systemu (zapravo protokol za mrežni pristup podatcima). Prilikom *mountanja* ne smijete biti u `d138` direktoriju jer ćete inače dobiti poruku o grešci (*device is busy*). Ako se želi maknuti taj dijeljeni direktorij potrebno je:

```
# umount d138
```

Unutar `d138/ppmps` kreirajte direktorij sa svojim imenom gdje ćete onda spremati svoje datoteke koje neće nestati nakon gašenja računala.

Proces proširenja stabla datoteka nakon mount-anja, prikazan je slikom.

Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.



2. PREGLED PROCESA

Proces je program koji je pokrenut i nezavisno postoji na dotičnom računalu. Svaki proces ima određeni set pripadajućih resursa (dio procesorskog vremena, radna memorija...). Popis procesa moguće je dobiti naredbom:

```
$ ps
```

Rezultat se odnosi na procese koji su nastali iz trenutnog terminala, je nešto tipa:

```
  PID TTY          TIME CMD
21808 pts/1    00:00:00 bash
21809 pts/1    00:00:00 ps
```

Svaki proces ima svoj process id (PID). CMD je naziv programa koji je stvorio proces. U našem slučaju `bash` je ljuska koja je aktivna unutar terminala a `ps` program kojim smo dobili te podatke.

Popis svih procesa moguće je dobiti naredbom:

```
$ ps aux
```

Naredba `ps` daje izvještaj o procesima koji postoje na računalu. Ona daje trenutni snimak stanja procesa u trenutku zadavanja naredbe. U kasnijem dijelu vježbe bit će više riječi o programu pomoću kojeg se mogu pratiti promjene stanja procesa u vremenu. Isprobajte i kombinacije slijedećih naredbi koje omogućavaju da bolje promotrimo izlaz naredbe `aux`:

`$ ps aux > procesi` izlaz preusmjeravamo u datoteku procesi
`$ ps aux | head –n 5` ponekad je dovoljno prikazati samo prvih pet redaka...
`$ ps aux | tail` ...ili zadnjih nekoliko redaka

Stupac STAT govori u kojem stanju (running R, ready S (sleep), blocked D) je proces.

Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

U `man` stranicama možete vidjeti da opcije `a`, `u` i `x` znače slijedeće:

```
a    show processes of other users too
u    user format: gives user name and start time
x    show processes without controlling terminal
```

Znak x stoji za procese koji nisu vezani za nijedan pravi ili pseudoterminal. Većinom su to razni daemoni. Daemoni su procesi koji su cijelo vrijeme aktivni i o njima će biti više riječi u nekoj od slijedećih vježbi.
Slijedeću naredbu:

```
$ ps aux | grep knoppix
```

Koristim kad želimo saznati kojim je sve procesima vlasnik korisnik pod imenom knoppix. Slijedeća naredba daje sve retke u kojima se nalazi slovo R, dakle i one koji su u stanju izvođenja (running)

```
$ ps aux | grep R
```

Istu smo stvar mogli doznati i jednostavnije, izdavanjem naredbe `ps r`.

Svaki proces ima svog jednog *roditelja* (*Parent Process ID*) i može imati više *djece* (procesi koje je sam stvorio). Dakle procesi se mogu posložiti u stablo. Prvi proces koji nastane, koji je na vrhu stabla i kojem su svi ostali procesi potomci jest `init` proces. Njegov je PID (*Process ID*) 1. Dalje se PID-ovi sekvencijalno dodjeljuju.

Da bi vidjeli i *ppid* polje svakog procesa možemo se poslužiti sljedećom `ps` naredbom gdje ćemo sami kontrolirati što će se ispisivati. (`-e` znači: svi procesi)

```
$ ps -e –o user,pid,ppid,stat,command
```

Bitno je primijetiti da je upravo pozvani 'ps' proces dijete od procesa ljuske ('bash'). Dakle kada se od ljuske želi da pokrene program `ps` onda ona kreira proces dijete koje izvrši taj program. Čije je dijete proces ljuske?

3. PRIORITETI

Relativni priotiteti procesa kreću se u rasponu od –20 do 20. –20 predstavlja najviši prioritet izvođenja procesa, a 20 najniži. Korisnik može procesima kojih je on vlasnik prilikom njihovog pokretanja smanjiti prioritet naredbom `nice`. Sintaksa naredbe `nice` je slijedeća:

```
nice -broj naredba
```

gdje je `broj` relativni prioritet koji se želi postići, dok `naredba` predstavlja proces koji ćemo pokrenuti. Tako npr. možemo u pozadini (operator `&`) pokrenuti naredbu `sleep` koja će završiti nakon 50 sekundi.

```
$ nice -15 sleep 50 &
```

Gornjom smo naredbom procesu `sleep` smanjili relativni prioritet na 15. Naredbom:

```
$  ps –o pid,pri,ni,stat,command
```


Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

moguće je vidjeti njezin prioritet (stupac PRI) i njezin relativni prioritet, tzv. *nice number* (stupac NI). Stupac PRI predstavlja stvarni trenutni prioritet procesa. Stupac NI (*nice number*) predstavlja njegov relativni prioritet, broj prema kojem će mu se u vremenu pridjeljivati stvarni prioritet. Stvarni će prioritet biti manji (proces nevažniji) što je nice broj veći i obrnuto.
Uobičajeno je da proces prilikom pokretanja dobiva relativni prioritet 0. Samo superkorisnik (root) može procesu povećati prioritet. Isprobati naredbu (biti logiran kao root):

```
# nice --10 sleep 50 &
```

Vidimo da je ovu naredbu moguće izvršiti jedino ako smo logirani kao root. Uočite da je ovdje riječ o negativnom *nice* broju (-10), dakle povećavamo relativni prioritet procesa. Ukoliko želimo već aktivnom procesu promijeniti prioritet, koristimo narebu `renice`:

```
$ renice 7 432
```

gdje je 432 primjer PID (*process ID*), a 7 konačni prioritet dotičnog procesa. Vi zamijenite 432 sa PID vašeg *sleep* procesa. Njegov PID je broj koji je ispisan na ekranu nakon pokretanja procesa. Naravno, može ga se doznati i korištenjem naredbe `ps`.
Uočite (korištenjem prije opisane `ps` naredbe) da je stvarni prioritet (PRI) pao na nižu vrijednost nakon što je NI promijenjen na višu vrijednost.

4. C PREVODILAC

U ovom će se dijelu vježbe koristiti C prevodilac `gcc` (*GNU C compiler*). Prevođenje (kompajliranje) je postupak stvaranja izvršne datoteke iz C programa. Ono ima nekoliko osnovnih faza.
U prvoj fazi prevodilac stvara tzv. *object file*, datoteku sa ekstenzijom `.o`. *Object file* je mašinski kod koji je nastao iz originalnog C koda.
U slijedećoj fazi vrši se povezivanje (linkanje) *object file*-a da bi se došlo do izvršne verzije programa. Posao linkera je da uzme `.o` datoteke, poveže ih sa kodom iz biblioteka i generira izvršnu verziju programa. Naime, kôd svih funkcija koje smo koristili u našim programima (npr. *printf()*) mora biti "linkan" u izvršnu verziju programa. Tek je tada izvršna verzija programa potpuna.
Kompajliranje se vrši naredbom:

```
$ gcc program.c –o program
```

gdje je `program.c` naziv C programa, a `–o` prekidač koji kaže da će se izvršna datoteka zvati `program`. Da smo samo napisali `gcc program.c`, izvršna datoteka bi se zvala po defaultu `a.out`.

Tipično se biblioteke nalaze u više direktorija od kojih je najvažniji `/usr/lib`. Postoje dvije vrste biblioteka: statičke (*static*) i dijeljene (shared). Statičke imaju ekstenziju `.a` (archive), dok dijeljene imaju ekstenziju `.so`. Razlika je u tome što se statičke direktno uključuju u izvršni kod prilikom linkanja dok se dijeljene uključuju po pokretanju programa. Naime *linker* postavi samo pozive dijeljenih biblioteka u izvršni kôd a kad se on krene izvršavati tada *loader* poveže te pozive sa pravim adresama tih biblioteka. Znači da pri pokretanju programa kojem su potrebne dijeljene biblioteke na računalu one moraju biti instalirane. Dijeljene su dobile naziv po tome što isti bibliotečni kôd mogu dijeliti nekoliko

Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

programa koji ih koriste dok bi inače svaki program u sebi sadržavao svoju kopiju rutina koje mu trebaju. Koje biblioteke su potrebne pojedinom programu moguće je dobiti naredbom *ldd ime_prog.* Ako uspoređujemo sa Windows OS-om tada se tamo statičke biblioteke označavaju sa ekstenzijom `.lib` a shared sa ekstenzijom `.dll` (dynamically linked library).
Proučite sadržaj direktorija `/usr/lib`. Uočite da sve *library* datoteke imaju prefiks `lib`. Tako se npr. može naći da se matematičke rutine (definirane u `math.h` - `/usr/include` direktorij) nalaze u biblioteci `libm`.
Koliko se `libm` datoteka nalazi u `/usr/lib` direktoriju? Možete li naći *static* i *shared* verziju? Kad postoje obje verzije po default-u se linka sa *shared* verzijom.

Da bismo prilikom linkanja uključili i `libm` biblioteku (npr. ako u C programu koristimo matematičke funkcije) potrebno je izdati slijedeću naredbu:

```
$ gcc program.c –o program -lm
```

gdje je slovo `m` poslije prekidača `-l` došlo od `(lib)m`.

Zadatak:

- Proučiti i iskompajlirati primjere C programa (PRILOG A) u home direktoriju. Ići redom: `moj_prvi.c`, `sinus.c`, `tockice.c`.

U programu `sinus.c` koristi se `sin()` funkcija. Zbog toga je, kao je već prije rečeno potrebno uključiti opciju `–lm`.

```
$ gcc sinus.c –o sinus -lm
```

Pokrenuti programe `moj_prvi`, `sinus` i `tockice`.
Ukoliko je potrebno zaustaviti izvođenje programa, koristiti ^C (Ctrl+C).

- Korištenjem editora (npr. `joe`) izmijeniti program `sinus.c` tako da u beskonačnoj petlji izračunava sinuse kuteva od 0 do 100 radijana.

5. KREIRANJE PROCESA

U programima `fork1.c`, `fork2.c` i `fork3.c` korištena je sistemska funkcija **`fork()`**. *Fork* je osnovni poziv sustava kojim se stvaraju novi procesi. Funkcija `fork()` stvara kopiju procesa iz kojega je pokrenut. U trenutku poziva nastaju dakle dva procesa koji su kopija jedan drugoga, imaju isti programski kôd i odvojene podatke. Jedan od njih se naziva *roditelj*, a drugi *dijete*. Ta se dva procesa mogu razlikovati po tome što će im vratiti poziv funkcije *fork*. U procesu dijete poziv fork vraća vrijednost 0. U procesu roditelj poziv fork vraća pozitivnu vrijednost veću od 0 koja predstavlja PID procesa dijete. Oba nastavljaju sa izvođenjem svoje kopije programa na poziciji iza fork poziva.

- Proučiti, iskompajlirati i pokrenuti primjere C programa (PRILOG A) u home direktoriju. Ići redom: `fork1.c`, `fork2.c` i `fork3.c`. Znati objasniti rezultate izvođenja.

Jednom kada je proces dijete završio on se nezavisno schedule-ira i može se izvršiti i nakon roditelja.

Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

Normalno proces završi na jedan od dva načina. Bilo sistemskim pozivom **exit()** ili tako da main funkcija završi sa `return`. Parametar tih funkcija je *exit-kod* koji se proslijedi roditelju. Ispravan završetak se dogovorno označava sa exit kodom 0. Ostali brojevi su razni error kodovi.

Ponekad je poželjno da roditelj pričeka da dijete završi. Recimo zanima ga sa kakvim exit kodom je proces dijete završio. Za to služi sistemski poziv **wait()**. On blokira proces koji ga je pozvao sve dok jedno od njegovih djeteta ne završi. Kada proces dijete završi roditelj preuzme *exit kod* te obriše proces dijete. Primjer toga je `fork4.c`, pokrenite ga.

Postavlja se pitanje što ako proces dijete završi sa radom a roditelj ga ne čeka sa wait(). Proces dijete neće nestati jer bi se onda izgubili podaci o tome sa kakvim exit kodom je završio. Tada proces dijete postaje *zombie* proces i `ps` će za njega pod STAT ispisati Z. Roditeljeva je briga da počisti zombie djecu za sobom. Ukoliko proces roditelj završi bez wait poziva njegovu djecu naslijedi `init` proces (PID=1) koji onda počisti svu zombie djecu koji dobije. Primjer programa koji dovede do zombie procesa je `zombie.c`. U tom primjeru roditelj 60 sec. ne radi ništa a u međuvremenu je dijete završilo i postaje zombie. Provjeriti preko drugog termianala sa:

```
$ ps -eo pid,ppid,stat,cmd
```

Kad roditelj završi sa izvođenjem on nije pozvao wait pa dijete zombie naslijedi `init` proces koji ga počisti.

Osim fork i wait još jedna bitna sistemska funkcija je **exec()**. Pomoću nje proces koji ju pozove prestaje sa izvođenjem trenutnog programa i počinje sa drugim (koji je naveden kao argument).

Ideja kako da jedan program pokrene drugi program je da prvo forkom stvori proces dijete te da taj novostvoreni proces pomoću exec naredbe počne izvršavati taj drugi program.

- Vaš zadatak je da tim principom napišete program, koji kada se pozove, kreira program koji će izvršiti '`ls -l /`' naredbu, dakle ispis root direktorija. Također na kraju ispisa direktorija mora se ispisati poruka "`Zavrsio sa ispisom`".

Kao pomoć, exec, točnije njegova verzija `execvp()`, se treba koristiti na sljedeći način:

```
/* Treba formirati listu argumenata koji se dostavljaju "ls" naredbi. */
char* arg_list[] = {
            "ls",  // argv[0], tu ide ime programa
            "-l",
            "/",
            NULL  // Lista mora zavrsiti sa NULL
};

execvp("ls",arg_list); // poziv programa ls
```


Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

6. PRAĆENJE STANJA PROCESA

Za interaktivije praćenje stanja procesa pogodan je program `top`. U jednoj od konzola pokrenuti program `top`. Potrebno je biti logiran kao root da bi se po volji moglo mijenjati prioritete procesima. U drugim konzolama pokrenuti program `tockice` i prepravljeni program `sinus`.
Mogućnosti programa `top` su slijedeće:

```
space   Update display
^L      Redraw the screen
fF      add and remove fields
oO      Change order of displayed fields
h or ?  Print this list
S       Toggle cumulative mode
i       Toggle display of idle proceses
c       Toggle display of command name/line
l       Toggle display of load average
m       Toggle display of memory information
t       Toggle display of summary information
k       Kill a task (with any signal)
r       Renice a task
P       Sort by CPU usage
M       Sort by resident memory usage
T       Sort by time / cumulative time
u       Show only a specific user
n or #  Set the number of process to show
s       Set the delay in seconds between updates
W       Write configuration file ~/.toprc
q       Quit
```

Navedene naredbe izvršavaju se pritiskom na odgovarajuće slovo. Podesiti da se prikaz osvježava svake sekunde. Pratiti kako se mijenja opterećenje procesora i korištena memorija te prioriteti procesa. Uključiti opcije `c`, `i`, `P`. Naredbom `f` dodati stupce PPID (*Parent PID*) i UID (*User ID*). Naredbom `r` procesu tockice maksimalno smanjiti prioritet (19), a procesu sinus maksimalno povećati prioritet (-19). Pratiti što se događa na virtualnim terminalima gdje se 'vrte' ta dva procesa.

Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

PRILOG A: C PROGRAMI

moj_prvi.c

```
#include <stdio.h>

int main()
{

        printf("Ovo je moj prvi program\n");

        return 0;
}
```

sinus.c

```
#include <stdio.h>
#include <math.h>

int main()
{
      int i;

      for (i=1;i<1000 ;i++)
      {
             printf("sin(%d)=%g\n",i,sin(i));
      }
      return 0:
}
```

tockice.c

```
#include <stdio.h>

int main()
{
        while(1)
        {
                int i;
                printf(".");
                fflush(stdout);     // ovime se forsira ispis na ekran
                for(i=0;i<100000;i++);
        }
        return 0;
}
```

fork1.c

```
#include <stdio.h>
#include <unistd.h>

int main()
{
        fork();

        printf("Dobro dosli u...\n");
        printf("Linux laboratorij\n");

        return 0;
}
```


Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

fork2.c

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

int main()
{
      int a;
      pid_t my_pid;

      a = fork();

      my_pid = (int) getpid();

      if(a == 0)
      {
            printf("Ja sam DIJETE %d i fork je vratio: %d\n", my_pid, a);
      }
      else
      {
            printf("Ja sam RODITELJ %d a PID mog djeteta je: %d\n", my_pid, a);
      }

      printf("Kraj!!! \n");

      return 0;
}
```

fork3.c

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

int main()
{
      int a,i;
      pid_t my_pid;

      for (i=0; i<4; i++)
      {
            a = fork();

            my_pid = (int) getpid();

            if(a == 0)
            {
                  printf("Ja sam DIJETE i PID je: %d\n", my_pid);
            }
            else
            {
                  printf("Ja sam RODITELJ %d a PID djeteta %d\n", my_pid, a);
            }
            printf("Kraj!!! \n");
      }
      return 0;
}
```


Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.

fork4.c

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

int main()
{
	int a,dijete_status;
	pid_t my_pid;

	a = fork();
	my_pid = (int) getpid();

	if(a == 0)
	{
		printf("Ja sam DIJETE i funkcija fork je vratila: %d\n", my_pid);
		sleep(60);
	}
	else
	{
		printf("Ja sam RODITELJ %d a PID mog djeteta je: %d\n", my_pid, a);
		wait(&dijete_status);
		printf("Dijete je zavrsilo sa statusom %d",dijete_status);
	}
	return 0;
}
```

zombie.c

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

int main ()
{
	int a:
	pid_t child_pid;


	a = fork ();

	if (a > 0) {
		/* Roditelj - spava 60 sec. */
		sleep (60);
	}
	else {
		/* Dijete odmah zavrsi sa radom. Error code 1*/
		exit (1);
	}

	return 0;
}
```


Ovaj dokument predstavlja intelektualno vlasništvo FER, ZESOI, LSS.