Sisteme de operare
Laborator 4
Controlul proceselor, semnale

1. Creati un nou subdirector *lab4/* in structura de directoare a laboratorului creata anterior (*SO/laborator*) si subdirectoarele aferente *doc src* si *bin*. Nu uitati sa actualizati variabila de mediu PATH pentru a include directorul *SO/laborator/lab4/bin*.

2. Scrieti un program C **my-exec-engine.c** care primeste ca parametru in linie de comanda un program executabil si apeleaza *fork* pentru a crea un nou proces. Procesul copil executa comanda primita ca argument folosind apelul sistem *execvp*. Parintele asteapta terminarea executiei copilului folosind apelul sistem *wait*.

3. Scrieti un program C **system.c** care simuleaza comportamentul programului anterior folosind functia de biblioteca *system*. Cum ati scrie varianta proprie a acestui program folosind *fork*/*exec*?

Indicatie: pentru a folosi functia *system* e nevoie sa concatenati string-urile furnizate ca parametri de apel ai programului (argv[i]). In acest sens, puteti folosi functia de biblioteca *strcat*.

4. Scrieti un program C **usr-signal.c** care prinde semnalele rezervate utilizatorilor SIGUSR1 si SIGUSR2, folosind apelul sistem *signal* pentru a le asocia handler de tratare a celor doua semnale. Apoi, procesul intra intr-o bucla infinita in care apeleaza *pause*.

Handlerul de semnale SIGUSR1 si SIGUSR2 identifica numarul de semnal primit si il afiseaza pe ecran, dupa care re-armeaza handlerul pentru semnalul primit.

Dupa compilare, lansati programul in background si folositi comanda shell *kill* pentru a trimite procesului semnale de tip SIGUSR1 si SIGUSR2, de maniera urmatoare:

```
$ gcc -o usr-signal usr-signal.c
$ usr-signal &
$ jobs
$ kill -USR1 %1
$ kill -USR2 %1
```

Presupunerea de mai sus este ca nu aveti decat un proces in background, identificat de numarul de job 1 (comanda *jobs* de mai sus va va spune in realitate care este numarul de job asociat cu procesul *usr-signal*). In mod normal, avand in vedere ca dupa primirea fiecarui semnal in handler rearmati dispozitia de tratare a semnalului respectiv, ar trebui sa puteti trimite semnalele SIGUSR1 si SIGUSR2 de ori de cat ori vreti fara ca programul sa se termine.

Pentru a termina programul, ii trimiteti semnalul SIGTERM ca mai jos:

```
$ kill -TERM %1
```

sau, mai simplu,

```
$ kill %1
```

Exersati variante ale comenzii *kill* care folosesc PID-ul procesului pe care il puteti afla cu comanda *ps*.

5. Scrieti un program C **signal.c** care foloseste apelul *fork* pentru a crea un nou proces. Procesul parinte simuleaza executia unui task de lunga durata apeland functia de biblioteca *getchar*, si se va bloca in asteptarea unui caracter introdus de utilizator de la tastatura. Procesul copil tipareste pe ecran PID-ul sau si al parintelui sau si termina cu cod de stare 44.

Pentru a putea procesa asincron evenimentul terminarii procesului copil, parintele prinde semnalul SIGCHLD inregistrand un handler de semnal pt acest semnal inainte de a apela *fork*, cu ajutorul apelului sistem *signal*.

Handlerul de semnal SIGCHLD tipareste un mesaj care afiseaza numarul de semnal primit si cheama neblocant (cu flag-ul WNOHANG) apelul sistem *waitpid* pentru a afla PID-ul procesului copil care tocmai s-a terminat si afiseaza acest PID pe ecran impreuna cu starea procesului terminat.

La final, utilizatorul va debloca procesul parinte apasand pe o tasta iar procesul parinte, odata iesit din apelul *getchar*, se termina cu cod de stare 0.

6. Modificati programul **pwnam.c** din primul laborator (cel care foloseste functia *getpwnam* pentru a accesa informatiile asociate unui utilizator in */etc/passwd*) pentru a testa reentranta functiei *getpwnam* atunci cand e apelata dintr-un handler de semnal. Mai exact, programul prinde semnalul SIGALRM (i.e., ii asociaza un handler) si cheama *alarm(1)* pentru a genera semnalul dupa 1 secunda. Apoi, intr-o bucla infinita, foloseste functia *getpwnam* pentru a afla informatii despre utilizatorul cu al carui nume a fost apelat programul. Dintre informatiile obtinute din */etc/passwd*, programul nu tipareste pe ecran decat UID-ul de utilizator.

Handlerul de semnal SIGALRM tipareste pe ecran numarul de semnal cu care a fost apelat, dupa care incearca sa citeasca din */etc/passwd* informatiile despre utilizatorul *nobody* si sa-i tipareasca UID-ul pe ecran. Apoi reseteaza alarma chemand *alarm(1)*.

Ce se intampla cand rulati programul?

7. Folosind apelul sistem *alarm* scrieti propria versiune *mysleep*

unsigned int mysleep(unsigned int seconds);

a functiei *sleep*, care pune procesul apelant sa doarma un numar de secunde primit ca parametru si intoarce numarul de secunde care au ramas de dormit, daca procesul apelant a fost intrerupt de un semnal.

Scrieti un scurt program C, **mysleep.c**, care testeaza functionalitatea *mysleep* in modul urmator: programul prinde semnalul SIGINT (Ctrl-C de la tastatura), apeleaza *mysleep(60)* si tipareste pe ecran numarul de secunde care i-au ramas procesului de dormit. Handlerul de semnal nu face nimic (apeleaza direct return).

La rularea programului, utilizatorul va tasta Ctrl-C dupa cateva secunde pentru a termina programul.

Exista posibile probleme de tip race condition in implementarea pe care ati facut-o?