

Sisteme de operare

Laborator 8

POSIX threads, System V IPC

1. Creați un nou subdirector *lab8/* în structura de directoare a laboratorului creată anterior (*SO/laborator*) și subdirectoarele aferente *doc src* și *bin*. Nu uitați să actualizați variabila de mediu *PATH* pentru a include directorul *SO/laborator/lab8/bin*.

2. Rescrieți programul C **race.c** din laboratorul 4 în care două procese, părinte și copil, scriu concurent propriul string pe ecran, scrierea fiind făcută caracter cu caracter. În loc de procese, folosiți două thread-uri POSIX create în thread-ul *main*. Fiecare thread apelează funcția *myprint* de 10 ori într-o buclă pentru a evidenția mai clar race condition-ul. Folosiți un *mutex* POSIX pentru a împiedica producerea race condition-ului și a obține un output neîntreșut pe ecran.

3. Scrieți două programe C **msg-echod.c** și **msg-echo.c** care implementează un serviciu de *echo* la fel ca în laboratorul 7, de data aceasta folosind cozi de mesaje System V IPC în loc de FIFOs. Practic, în loc să folosească un fișier *FIFO*, programul **msg-echod.c** creează o coadă de mesaje cu ajutorul *msgget* (parametrul *key* îl generați cu funcția *ftok*). Apoi apelează în buclă funcția *msgrcv* pentru a primi mesaje de la clienți pe care le scrie înapoi în coadă de mesaje (serviciu de *echo*) folosind *msgsnd*. Programul **msg-echod.c** se lansează în background ca serviciu de echo după cum urmează:

```
$ gcc -o msg-echod msg-echod.c
$ msg-echod keyfile &          # keyfile e un fișier pe care îl creați voi
$ ipcs -q                      # vizualizați coada creată
```

Programul **msg-echo.c** folosește parametrul primit în linie de comandă (*keyfile*) ca argument pt funcția *ftok* și astfel poate identifica și obține cu ajutorul *msgget* un ID pentru coadă de mesaje creată de *echod*. Apoi, citește în buclă caractere de la tastatură folosind funcția *fgets* și assemblează mesaje cu tipul dat de PID-ul său (obținut cu *getpid*) pe care le trimite în coadă de mesaje cu ajutorul funcției *msgsnd*. Apoi citește din coadă de mesaje cu ajutorul *msgrcv* ecoul caracterelor trimise către serviciul *echod* pe care, odată primite, le tipărește pe ecran. Programul se termină când utilizatorul apasă *Ctrl-c*.

```
$ gcc -o msg-echo msg-echo.c
$ msg-echo keyfile
```

La final, nu uitați să faceți curățenie:

```
$ jobs
$ kill %<n>                    # n e numărul de job al msg-echod afișat
                                # de comandă jobs de mai sus

$ ipcs -q
$ ipcrm -q <id>               # id e id-ul cozii de mesaje tipărit de comandă
                                # ipcs de mai sus
```

Pentru a evita ultimele două comenzi de mai sus, în programul **msg-echod.c** puteți să prindeți semnalul *SIGTERM* și să stergeți coada de mesaje din program atunci când se execută comanda *kill* de mai sus.

4. Modificati programul C **msg-echod.c** de mai sus pentru a oferi un serviciu **multi-threaded** de *echo* folosind cozi de mesaje System V IPC. Noul program, **msg-mt-echod.c**, creeaza o coada de mesaje cu ajutorul *msgget* (parametrul *key* il generati cu functia *ftok*). Apoi apeleaza in bucla functia *msgrcv* pentru a primi mesaje de la clienti, Spre deosebire de **msg-echod.c**, care este *single-threaded*, **msg-mt-echod.c** creeaza un thread POSIX separat pentru fiecare nou mesaj primit. Serverul copiaza mesajul primit cu *msgrcv* si il paseaza noului thread creat care il scrie inapoi in coada de mesaje (serviciu de *echo*) folosind *msgsnd*. Programul **msg-mt-echod.c** se lanseaza in background ca serviciu de *echo* dupa cum urmeaza:

```
$ gcc -o msg-mt-echod msg-mt-echod.c -lpthread
$ msg-mt-echod keyfile &          # keyfile e un fisier pe care il creati voi
$ ipcs -q                          # vizualizati coada creata
```

Pentru a testa functionalitatea serverului folositi programul client **msg-echo.c** dezvoltat in exercitiul anterior. Ca sa testati functionalitatea serverului multithreaded lansati mai multe programe client in mod concurent (de pilda folosind un shell script simplu care lanseaza in background un numar de clienti).

Cand sunt mai multi clienti activi simultan, cum izolati traficul unui client cu serverul de traficul altor clienti? Mai exact, cum faceti ca raspunsul serverului sa ajunga exact la clientul caruia trebuie sa-i raspunda, si nu altui client pe care-l deserveste de asemenea?

La final, ca la exercitiul anterior, nu uitati sa faceti curatenie in sistem stergand coada de mesaje fie prin program, fie din shell.

5. Modificati programul C **msg-mt-echod.c** de mai sus a.i. sa emuleze comportamentul unui server de remote shell, **rshd.c**. Cand serverul **rshd** primeste un string de la un client **msg-echo** il interpreteaza ca pe o comanda shell, o executa si intoarce rezultatul executiei catre client. Mai jos aveti un exemplu de executie al serverului **rshd**:

```
$ gcc -o rshd rshd.c -lpthread
$ rshd keyfile &          # keyfile e un fisier pe care il creati voi
$ ipcs -q                  # vizualizati coada creata
```

in vreme ce clientul trebuie apelat cu mesaje care au sens, i.e. string-uri care reprezinta de fapt comenzi:

```
$ msg-echo keyfile
ls -l
...
ps
...
Ctrl-c
$
```

unde punctele de suspensie de mai sus inlocuiesc rezultatul executiei comenzii trimis inapoi de catre **rshd**.

Indicatie: in thread-ul de serviciu, *rshd* apeleaza secventa *vfork/exec/wait* pentru a executa comanda primita de la client. Pentru a recupera output-ul comenzii executate de *exec* si a-l putea trimite inapoi catre client folosind coada de mesaje, puteti folosi un *pipe* redirectat corespunzator. Pentru a

nu mai parsa string-ul care reprezinta comanda, folositi in *exec* comanda “*sh -c*” care executa o comanda care urmeaza dupa flagul *-c*.

6. Modificati programul C **rshd.c** a.i. sa emuleze comportamentul OOB (Out-Of-Band data) prin care serverele de retea sunt anuntate prioritar de sosirea unui caracter (sau secventa de caractere) de control, cum ar fi de pilda *Ctrl-c* intr-o sesiune de *ssh* pentru a termina o comanda aflata in executie la distanta. Noua versiune a programului, **rshd-oob.c**, primeste caracterul de control, *Ctrl-c* in exemplu nostru, il interpreteaza (*Ctrl-c* inseamna terminarea comenzii) si executa dispozitia asociata secventei de control (termina comanda).

N.B. Cand programul client, **msg-echo**, primeste *Ctrl-c* de la tastatura se va termina abrupt. Nu aceasta este functionalitatea pe care trebuie sa o implementati, deci trebuie sa modificati si codul lui **msg-echo.c** pentru a evita terminarea programului client si a fi capabil sa execute o secventa de felul urmator:

```
$ msg-echo keyfile
sleep 300
Ctrl-c
ps
...
ls -l
...
$
```

In exemplul de mai sus, comanda *sleep 300* este folosita pentru a emula executia unei comenzi care dureaza mult.