

Sisteme de operare

Laborator 8

POSIX threads, System V IPC

1. Creati un nou subdirector *lab8/* in structura de directoare a laboratorului creat anterior (*SO/laborator*) si subdirectoarele aferente *doc src* si *bin*. Nu uitati sa actualizati variabila de mediu PATH pentru a include directorul *SO/laborator/lab8/bin*.
2. Rescrieti programul C **race.c** din laboratorul 4 in care doua procese, parinte si copil, scriu concurrent propriul string pe ecran, scrierea fiind facuta caracter cu caracter. In loc de procese, folositi doua thread-uri POSIX create in thread-ul *main*. Fiecare thread apeleaza functia *myprint* de 10 ori intr-o bucla pentru a evidenta mai clar race condition-ul. Folositi un *mutex* POSIX pentru a impiedica producerea race condition-ului si a obtine un output neintretesut pe ecran.
3. Scrieti doua programe C **msg-echod.c** si **msg-echo.c** care implementeaza un serviciu de *echo* la fel ca in laboratorul 7, de data aceasta folosind cozi de mesaje System V IPC in loc de FIFOs. Practic, in loc sa foloseasca un fisier *FIFO*, programul **msg-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 pe care le scrie inapoi in coada de mesaje (serviciu de *echo*) folosind *msgsnd*. Programul **msg-echod.c** se lanseaza in background ca serviciu de echo dupa cum urmeaza:

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

Programul **msg-echo.c** foloseste parametrul primit in linie de comanda (*keyfile*) ca argument pt functia *ftok* si astfel poate identifica si obtine cu ajutorul *msgget* un ID pentru coada de mesaje creata de *echod*. Apoi, citeste in bucla caractere de la tastatura folosind functia *fgets* si asambleaza mesaje cu tipul dat de PID-ul sau (obtinut cu *getpid*) pe care le trimit in coada de mesaje cu ajutorul functiei *msgsnd*. Apoi citeste din coada de mesaje cu ajutorul *msgrcv* ecoul caracterelor trimise catre serviciul *echod* pe care, odata primite, le tipareste pe ecran. Programul se termina cand utilizatorul apasa *Ctrl-c*.

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

La final, nu uitati sa faceti curatenie:

```
$ jobs
$ kill %<n>                                # n e numarul de job al msg-echod afisat
                                                # de comanda jobs de mai sus
$ ipcs -q
$ ipcrm -q <id>                            # id e id-ul cozii de mesaje tiparit de comanda
                                                # ipcs de mai sus
```

Pentru a evita ultimele doua comenzi de mai sus, in programul **msg-echod.c** puteti sa prindeti semnalul SIGTERM si sa stergeti coada de mesaje din program atunci cand se executa 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 concurrent (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 comenзii 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 comenзii executate de *exec* si a-l putea trimite inapoi catre client folosind coada de mesaje, puteti folosi un *pipe* redirectat corespunзator. 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 prioritari 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.