Sisteme de operare
Laborator 9
System V IPC

1. Creati un nou subdirector *lab9/* 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/lab9/bin*.

2. Scrieti un program C **mem-layout.c** care tipareste adresele (aproximative) ale sectiunilor unui program: cod (.text), date neinitializate (.bss), date initializate, stiva, heap (+ program break), memorie partajata. Pentru a realiza acest lucru, tipariti urmatoarele adrese:

- adresa functiei *main*
- adresa *argv*
- adresa unei variabile globale initializate
- adresele de inceput si sfarsit ale unui array global de octeti neinitializat (de dimensiune relativ mare, sa zicem 40000 bytes)
- adresa primei variabile locale declarate in *main* (va fi alocata pe stiva)
- adresa unui vector de 10000 octeti alocat in *main* cu *alloca*
- adresa unui vector de 10000 de octeti alocat dinamic in *main* cu *malloc* + program break-ul (folositi apelul *sbrk*)
- creati o zona de memorie partajata privata (*shmget* cu IPC_PRIVATE) si tipariti valoarea adresei la care se ataseaza in program (obtinuta cu *shmat*).

La final, apelati *shmctl* cu IPC_RMID pentru a sterge zona de memorie partajata din sistem.

Indicatie: adresele de variabile se tiparesc folosind *%p* in formatul *printf*.

3. Rescrieti programul C **race.c** din laboratorul 5 in care doua procese, parinte si copil, scriu concurent propriul string pe ecran, scrierea fiind facuta caracter cu caracter. Pentru a evidentia mai clar race condition-ul, procesele parinte si copil repeta operatia de tiparire pe ecran de un numar de ori, sa zicem de 10 ori, in bucla. Folositi un semafor System V pentru a impiedica producerea race condition-ului si a obtine un output neintretesut pe ecran. La final, apelati *semctl* cu IPC_RMID pentru a sterge semaforul din sistem.

Indicatie: fiind vorba de procese inrudite, puteti crea semaforul cu IPC_PRIVATE. De asemenea, e util sa scrieti functii ajutatoare pentru implementarea operatiilor *down* si *up* cu semnificatiile de la curs.

4. Scrieti un program C **shm.c** care creeaza o zona de memorie partajata intre un proces parinte si un copil. Parintele creeaza zona de memorie partajata (puteti folosi IPC_PRIVATE) si o ataseaza la proces inainte de apelul *fork*. Procesul copil executa un loop cu 10 iteratii in care scrie succesiv, la fiecare iteratie, urmatorul mesaj in zona de memorie partajata:

"this is the child printing " + numarul iteratiei

Pentru a compune asemenea mesaje puteti folosi functia de biblioteca *sprintf*. Mesajele din fiecare iteratie se adauga (*append*) la sfarsitul mesajului scris in iteratia anterioara (cu exceptia primei iteratii).

Procesul parinte executa si el un loop cu 10 iteratii in care, la fiecare iteratie, citeste unul dintre mesajele scrise de procesul copil si le afiseaza pe ecran.

Dezvoltati o solutie folosind semafoare System V care sa permita coordonarea (sincronizarea) accesului celor doua procese la zona de memorie partajata a.i. executia programului sa fie cea anticipata, si anume dupa fiecare mesaj scris de copil in memoria partajata parintele il citeste si il scrie pe ecran. (*Obs*: *ambele procese executa un loop cu 10 iteratii dar nu puteti face nici o presupunere asupra vitezei relative de executie a celor doua procese*).

La final, apelati *shmctl* si *semctl* cu IPC_RMID pentru a sterge memoria partajata si semafoarele din sistem.

5. Folositi semafoare si memorie partajata System V pentru a implementa problema producator-consumator. Cele doua procese trebuie sa fie independente (i.e., nu sunt create cu *fork*). Buffer-ul circular va fi implementat folosind o zona de memorie partajata.