# Laboratoires 16

# *La communication interprocessus : les sémaphores (suite)*

**6. Primitive *semop()***

*#include <sysitypes .h>*

*#include <syslipc.h>*

*#include <sys/sem.h>*

*int semop(semid, sops, nsops) ;*

*struct sembuf sops[];*

*int semid, nsops;*

Returned value: the semval value of the last semaphore used, or -1 in the event of an error.

The semop () system call allows you to perform operations on semaphores. It takes three arguments: a semaphore set identifier (semid), a pointer to an array of structures of type struct sembuf (sops), and an nsops integer. Each of the nsops elements in the array pointed to by sops indicates an operation to be performed on a semaphore using a struct sembuf structure:

*struct sembuf {*

*ushort sem\_num; // numéro du sémaphore*

*short sem\_op; // opération à réaliser*

*short sem\_flg // indicateur d’opération*

*};*

The possible options for sem\_flg are IPC\_NOWAIT and SEM\_UNDO. If an operation specifies the SEM\_UNDO option, it will be canceled when the process ends.

The sembuf structure specifies the number of the semaphore that will be processed, the operation that will be performed on that semaphore, and the operation control flags.

The type of operation depends on the value of sem\_op:

* Si *sem\_op < 0* (demande de ressource) :
  + si *semval* >= |*sem\_op*| alors *semval* = *semval* - |*sem\_op*| : décrémentation du sémaphore,
  + si *semval* < |*sem\_op*| alors le processus se bloque jusqu’à ce que *semval* >= |*sem\_op*|,
    - Si *sem\_op* = 0 :
      * si *semval* = 0, alors l’appel retourne,
      * si *semval* !=0, alors le processus se bloque jusqu’à ce que *semval* = 0,
        + Si *sem\_op* > 0 (restitution de ressource): alors *semval = semval* + *sem\_op.*

The positioning of certain flags modifies the result of semop () type operations:

¬ IPC\_NOWAIT: avoids blocking the process and returns an error code.

¬ SEM\_UNDO: Resource requests and returns are automatically balanced at the end of the process. All modifications made to semaphores by a process are undone on its death. During the life of a process, the operations performed with the SEM\_UNDO flag on all the semaphores of all the identifiers are cumulated, and when this process dies, the system redoes these operations in reverse. This makes it possible not to block processes on semaphores indefinitely, after the accidental death of a process. Note that this process is expensive, both in terms of CPU time and memory space reservation.

**Programme 1**:

A first process (program1) creates a set of semaphores, sets the value of one of the semaphores to 1, then requests a resource. It goes on hold for 10 seconds. A second process (program2) gets the semid identifier from the set of semaphores, then also requests a resource. It remains blocked until the first process has finished its wait and then releases the resource

*// Programme 1.cpp*

*#include <errno.h>*

*#include <stdio.h>*

*#include <sys/types.h>*

*#include <sys/ipc.h>*

*#include <sys/sem.h>*

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <unistd.h>*

*#define SEM\_W 200 // permission de modification*

*#define SEM\_R 400 // permission en lecture*

*#define CLE 123*

*int semid ;*

*struct sembuf operation[1] ;*

*union {*

*int val ;*

*struct semid\_ds \*buf ;*

*ushort array[4] ; } arg ;*

*int main()*

*{*

*// Création d'un ensemble de 4 sémaphores*

*if (( semid = semget(CLE,4,IPC\_CREAT|SEM\_W|SEM\_R))==-1){*

*perror("impossible de créer l'ensemble de sémaphores") ;*

*exit(1) ;*

*}*

*printf("processus1: je viens de créer un ensemble de sémaphore :*

*%d\n",semid) ;*

*// mise à 1 du troisième sémaphore*

*arg.val=1 ;*

*if ((semctl(semid,2,SETVAL,arg))==-1){*

*perror("semctl") ;*

*exit(1);*

*}*

*// Demande de ressource au troisième sémaphore*

*printf("process1: je vais demander une ressource\n") ;*

*operation[0].sem\_num = 2; //opération sur le troisième sémaphore*

*operation[0].sem\_op = -1 ; //opération de décrémentation/*

*operation[0].sem\_flg = SEM\_UNDO; // pour défaire les opérations*

*if ( semop(semid,operation,1) == -1){*

*perror("semop:opération de décrémentation non effectuée") ;*

*exit(1) ;*

*}*

*// Attente pour bloquer le second processus*

*printf("processus1 : j'attends 10 sec\n") ;*

*sleep(10) ; // attente ...*

*printf("processus1 : j'ai fini d'attendre: je libère la ressource\n") ;*

*// libération de ressource*

*operation[0].sem\_op = 1 ; // incrémentation*

*if ( semop(semid,operation,1) == -1){*

*perror("semop:operation d'incrémentation non effectuée") ;*

*exit(1) ;*

*}*

*printf("mort de processus1\n") ;*

*return 0;*

*}*

*// Programme2.cpp*

*#include <errno.h>*

*#include <stdio.h>*

*#include <sys/types.h>*

*#include <sys/ipc.h>*

*#include <sys/sem.h>*

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <unistd.h>*

*#define CLE 123*

*int semid ;*

*struct sembuf operation[1] ;*

*int main()*

*{*

*// récupération du semid*

*if (( semid = semget(CLE,0,0))==-1){*

*perror("impossible de retrouver l'ensemble de sémaphores") ;*

*exit(1) ;*

*}*

*printf("processus2 : traite les sem : semid %d\n",semid) ;*

*// Boucle d'attente de la disponibilité du sémaphore.*

*// On demande de ne pas rester bloquer en attente*

*// en positionnement le drapeau IPC\_NOWAIT*

*operation[0].sem\_num = 2;*

*operation[0].sem\_op = -1 ;*

*operation[0].sem\_flg = IPC\_NOWAIT + SEM\_UNDO ;*

*for (;;){*

*if ( semop(semid,operation,1) != -1) break ;*

*printf(" demande du processus2 : sémaphore non disponible\n") ;*

*sleep(1) ;*

*}*

*printf(" sémaphore alloue au processus2\n") ;*

*// Libération du segment de sémaphore*

*// libération de ressource*

*operation[0].sem\_op = 1 ; // incrémentation*

*if ( semop(semid,operation,1) == -1){*

*perror("semop:operation d'incrémentation non effectuée") ;*

*exit(1) ;*

*}*

*return 0;*

*}*

**Programme 2 :**

Program a schema between a parent process and its child to access a common variable. The parent process will modify the variable adding 1 each time while the child will only read the value of the modified variable.

*#include<stdio.h>*

*#include<stdlib.h>*

*#include<sys/ipc.h>*

*#include<sys/shm.h>*

*#include<sys/sem.h>*

*#include<errno.h>*

*#include <unistd.h>*

*#include <iostream>*

*using namespace std;*

*/\*On définit une valeur de clé\*/*

*#define CLE 340*

*typedef union*

*{*

*int valeur;*

*struct semid\_ds \* buffer;*

*unsigned short int \* table;*

*} semun\_t;*

*int main ()*

*{*

*int shmid;*

*int semid;*

*int \*Zone;*

*semum\_t semum;*

*struct sembuf SemOp;*

*// Première étape : On crée une zone de mémoire partagée*

*shmid=shmget(CLE,1024,IPC\_CREAT|IPC\_EXCL|SHM\_R|SHM\_W)*

*/\*On vérifie que l'allocation s'est bien déroulée\*/*

*if(shmid==-1)*

*{perror("Pb d'allocation zone memoire partagee :");*

*exit(1);*

*}*

*cout << "Zone de memoire allouee "<<shmid<<endl;*

*// On rattache le processus à cette zone de mémoire*

*Zone=(int \*)shmat(shmid,NULL,0);*

*if(Zone==NULL)*

*{perror("Pb d'attachement de la zone au processus :");*

*exit(1);*

*}*

*cout <<"Rattachement effectue !"<<endl;*

*// On initialise la valeur du nombre pointé en zone mémoire*

*Zone[0]=112;*

*cout << "Valeur mise "<< Zone[0]<<endl;*

*sleep(2);*

*// On crée un sémaphore, et on l’initialise à 1*

*semid=semget(CLE,1,IPC\_CREAT|IPC\_EXCL|SHM\_R|SHM\_W);*

*semun.valeur=1;*

*semctl(semid,0,SETVAL,semun);*

*// On crée ensuite un processus fils*

*n = fork();*

/ We now want to distinguish the father from the son, the father and the son cannot access

// at the same time to the memory area, you must use a semaphore to block

// access. One of the two processes (father or son) will access the memory area and not

// both at the same time)

*if (n !=0) // Si c’est le père*

*{ // Je suis le père, je souhaite écrire dans la zone de mémoire partagée, uniquement*

*// lorsque je peux le faire*

*while(1)*

*{*

*// Je décrémente d'un la valeur de la sémaphore*

*SemOp.sem\_num=0;*

*SemOp.sem\_op=-1;*

*SemOp.sem\_flg=0;*

*semop(semid,&SemOp,1);*

*// Ensuite, soit je suis bloqué, car le fils a accès, il est en train de lire, dans ce*

*// cas, j'attends, soit j'ai accès et dans ce cas, je peux modifier la valeur*

*Zone[0]=Zone[0]+1;*

*cout << "Le pere modifie la valeur -> "<<Zone[0]<<endl;*

*// Une fois la modification effectuée, je peux remettre la valeur du sémaphore à 1*

*SemOp.sem\_num=0;*

*SemOp.sem\_op=+1;*

*SemOp.sem\_flg=0;*

*semop(semid,&SemOp,1);*

*}*

*}*

*else // Je suis le fils*

*{*

// We do the same as for the father, I get the semaphore id

*semid=semget(CLE,0,0);*

// I want to read in the shared memory area, only when I can

// make

*while(1)*

*{*

*// Je décrémente d'un la valeur du sémaphore*

*SemOp.sem\_num=0;*

*SemOp.sem\_op=-1;*

*SemOp.sem\_flg=0;*

*semop(semid,&SemOp,1);*

*// Ensuite, soit je suis bloqué, car le père a accès, il est en train d'écrire, dans ce*

*// cas, j'attends, soit j'ai accès et dans ce cas, je peux lire la valeur*

*cout << "Le fils lit la valeur-> "<<Zone[0]<<endl;*

*// Une fois la lecture effectuée, je peux remettre la valeur du sémaphore à 1*

*SemOp.sem\_num=0;*

*SemOp.sem\_op=+1;*

*SemOp.sem\_flg=0;*

*semop(semid,&SemOp,1);*

*}*

*}*

*return 0;*

*}*

**Programme 3 :**

We want to simulate the behavior of several cars at a road junction. A car should only enter the junction if no other car is already there. Crossing the crossroads is therefore done in mutual exclusion. The movement of each car is done according to the following diagram:

Car process:

- arrival at the crossroads,

- crossing the crossroads in mutual exclusion,

- departure from the crossroads

The function void car (int num) defines the process simulating the movement of a car. num is the car number.

Complete the program below and test by checking that it is not possible to have 2 cars at the same time in the intersection.

*#include<stdio.h>*

*#include<stdlib.h>*

*#include<sys/ipc.h>*

*#include<sys/shm.h>*

*#include<sys/sem.h>*

*#include<errno.h>*

*#include <unistd.h>*

*#include <iostream>*

*#include <sys/wait.h>*

*using namespace std;*

*#define CLE 222*

*typedef union*

*{*

*int valeur;*

*struct semid\_ds \* buffer;*

*unsigned short int \* table;*

*} semun\_t;*

*int semid;*

*semun\_t semun;*

*struct sembuf SemOp;*

*void voiture (int num)*

*{*

*int r;*

*if (fork()==0)*

*{*

*r=rand()%5;*

*sleep(r);*

*cout << "voiture # "<<num << " arrivee "<<endl;*

*//Je décrémente d'un la valeur de la sémaphore*

*SemOp.sem\_num=0;*

*SemOp.sem\_op=-1;*

*SemOp.sem\_flg=0;*

*semop(semid,&SemOp,1);*

*cout << "voiture # "<<num << " traverse "<<endl;*

*r=rand()%5;*

*sleep (r);*

*cout << "voiture # "<<num << " depart "<<endl;*

*// Une fois la traversé effectuée, on remet la valeur du sémaphore à 1*

*SemOp.sem\_num=0;*

*SemOp.sem\_op=+1;*

*SemOp.sem\_flg=0;*

*semop(semid,&SemOp,1);*

*exit(0);*

*}*

*}*

*int main ()*

*{*

*int r,status;*

*/\*On cree une sémaphore\*/*

*semid=semget(CLE,1,IPC\_CREAT| IPC\_EXCL|SHM\_R|SHM\_W);*

*/\*On l'initialise à 1\*/*

*semun.valeur=1;*

*semctl(semid,0,SETVAL,semun);*

*srand(time(0));*

*cout << "Arrivées aleatoire des voitures 1, 2, 3 et 4 " <<endl;*

*voiture(1);*

*voiture(2);*

*voiture(3);*

*voiture(4);*

*for(int i=1; i<=4;i++) wait(&status);*

*return 0;*

*}*

**Programme 4:** Homework

We want to simulate the operation of a swimming pool. Bathers share two resources:

- the baskets, necessary to enter the enclosure of the swimming pool;

- cabins, necessary for changing (dressing or undressing).

Each bather is represented by a process whose behavior is as follows:

The bather process:

1. takes a basket;

2. enter a cabin;

3. change;

4. exit the cabin;

5. bathe;

6. enter a cabin;

7. gets changed;

8. leaves the cabin;

9.free the basket

10. quit

The following function represents the behavior of a swimmer:

*void baigneur (int i)*

*{*

*if (fotk() == 0)*

*{*

*.*

*.*

*.*

*}*

*exit(0);*

*}*

Take :

- the number of bathers = 10

- the number of baskets = 3

- the number of cabins = 2

1. Write the program and test, checking that it is not possible to exceed the planned capacities (in terms of baskets and cabins).

2. Modify the schedule so that a bather uses two baskets (one for clothes and one for shoes) instead of one.