Université de Sciences et de la Technologie Houari Boumediene

Rapport 2

Outils de Communication Inter-Processus

Sémaphores + Mémoire partagée

Mahmoud Bacha rabah sami

M1 RSD - GROUP : 02

La bibliothèque personnel semaphore.h :

Dans cette partie on va créer une nouvelle bibliothèque de sémaphore contenant un ensemble de fonctions pour gérer les sémaphores :

- **P , V , Z**
- **Semcreate :** pour la création d'une sémaphore
- **Seminit** et **Seminitall** : pour l'initialisation
- **Semdestroy** : pour détruire le groupe de sémaphore
- **Barriere** : création d'une sémaphore de blocage a N processus donné comme argument

En se basant sur les primitives suivantes de la bibliothèque **sys/sem.h** et **sys/ipc.h**:

- **key_t key = ftok (char* pathname, char project)**

  Pour la création du chemin de la clé

- **int semid=semget(key, n, IPC_CREAT| IPC_EXCL| 0666)**

  pour la creation d'un groupe de sémaphore de taille n

- **int semop(int semid, struct sembuf *ops, unsigned nbops)**

  pour manipuler le semaphore et utiliser les opérations P V Z

- **int semctl (semid, numsum, flag , Valeur)**

  Pour l'initialisation du sémaphore , **SETVAL** ou **SETALL**
  pour la récupération de la valeur du sémaphore **GETVAL**
  pour la suppression du groupe de sémaphore , **IPC_RMID**
  pour récupère des statistiques sur le groupe de sémaphores , **IPC_STAT**

Implementation de la bibliothèque :

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/wait.h>
#include<sys/types.h>


int p (int semid , int semnum){
    struct sembuf op ;
    op.sem_num=semnum;
    op.sem_op=-1;
    op.sem_flg = SEM_UNDO;
    int r=semop(semid, &op, 1) ;
    if (r== -1) {
                perror("semop");
                return 0;
    }
    return r;
}

int v(int semid  , int semnum){
    struct sembuf op ;
    op.sem_num=semnum;
    op.sem_op=+1;
    op.sem_flg = SEM_UNDO;

    if (semop(semid, &op, 1) == -1) {
                perror("semop");
                exit(0);
    }

}

int z(int semid  , int semnum){
    struct sembuf op ;
    op.sem_num=semnum;
    op.sem_op=0;
    op.sem_flg = 0;

    if (semop(semid, &op, 1) == -1) {
                perror("semop");
                exit(0);
    }

}


int seminit(int idsem, int numsem , int initval ){

    if(semctl(idsem , numsem , SETVAL,initval)==-1)perror("semctl initialisation
error");
}
```

```c
int seminitall(int idsem, unsigned short  initval[]){

        if(semctl(idsem , 0 , SETALL,initval)==-1)perror("semctl initialisation
error");

}

int semcreate(key_t key,int n){
  int semid = semget(key,n, IPC_CREAT | IPC_EXCL | 0666);
  if(semid == -1){
         perror("semget semcreate");
         semid = semget(key,n,0);
  }
  return semid;
}

int semdestroy (int semid){
    if(semid==-1){
        printf("semaphore doesnt exist ");
    }
    else{
        semctl(semid , 0 , IPC_RMID);
    }
}


void  barriere (int semid , unsigned short Nombre_final ){



    p( semid , 2);
    p(semid , 1);
    v(semid , 2);

    if(semctl(semid , 1 , GETVAL)==0){
        printf("je suis le dernier %d  , je les libere \n" , getpid());
        for(int i =0 ; i <= Nombre_final ; i++)v(semid , 0);
        exit(0);
    }else{
        printf("je suis %d et je suis bloqué \n", getpid());
        p(semid , 0);
        printf("je suis libere %d \n",getpid());

        exit(0);

    }


}
```

La sémaphore barrière :

Explication :

Un processus père créer N processus fils , qui tous exécute la sémaphore bloquante initialise a 0 appelé barrière
Quand le N ieme processus arrive , il les libères tous par une boucle de V.

Pour ne pas créer une variable partagé qui compte le nombre de processus arrivé , on utilise un sémaphore de comptage initialiser a N et un mutex pour protéger ce dernier sémaphore

Implémentation :

```
#include "my_semaphore.c"

void main( ){

    key_t key = ftok("~/SE_RSD/semfolder","br");
    int Nombre_final;
    printf("Donner le nombre total de processus fils : ");scanf("%d" ,
&Nombre_final);

    int semid = semcreate(key , 3);
    unsigned short tab[3]={0,Nombre_final,1 };
    seminitall(semid , tab);

    int father_pid = getpid();
    printf("Je suis le pere %d , mes fils vont faire le processus barriere
\n",getpid());
    int pid;

    for (int i = 0; i < Nombre_final ; i++)
    {
        if(getpid()==father_pid) pid = fork();
    }
    if (pid==0)
    {
        barriere(semid ,Nombre_final);
    }else{
        while(wait(NULL)!=-1);}

    printf("Je suis le pere %d , mes fils sont terminer , i die \n",getpid());
    semdestroy(semid);

return 0;
```

Résultat d'exécution :

```
sami@laptop:~/SE_RSD/tp2$ gcc barrier.c -o barrier  -w
sami@laptop:~/SE_RSD/tp2$ ./barrier
Donner le nombre total de processus fils : 6
Je suis le pere 8792 , mes fils vont faire le processus barriere
je suis 8793 et je suis bloqué
je suis 8794 et je suis bloqué
je suis 8795 et je suis bloqué
je suis 8798 et je suis bloqué
je suis 8796 et je suis bloqué
je suis le dernier 8797  , je les libere
je suis libere 8793
je suis libere 8794
je suis libere 8798
je suis libere 8795
je suis libere 8796
Je suis le pere 8792 , mes fils sont terminer , i die
sami@laptop:~/SE_RSD/tp2$ S
```

Exercice 3 (l'imprimante) :

Expliquation :

N processus fils s'exécute en exclusion mutuelle d'où l'utilisation d'un sémaphore binaire .

Le problème de processus mort :
Si un processus est accidentellement terminer , il **bloquera** tous les autres processus qui attendent l'imprimante car il n'a pas exécuter le **V**

Solution :

il faut rajouter le flag , **SEM_UNDO** dans la fonction V et P de la bibliothèque pour faire l'opération inverse et et libérer le sémaphore

Implémentation :

```c
#include "my_semaphore.c"
#include<signal.h>

int main(void){

    printf("*****informations general sur le processu :***** \n\n");
    key_t key = ftok("~/SE_RSD/semfolder",'kkk');
    printf("         key := %0x\n",key);


    int semid= semcreate(key , 1);
    seminit(semid , 0 , 1);  // sempahore binaire
    printf("         smid = %d \n", semid);


    int father_pid=getpid();
    printf("    Je suis le pere %d mes fils vont utilser l'imprimante
\n\n\n",father_pid);


    int N ;
    printf("Donner le nombre de processus N : "); scanf("%d" , &N);

    int pid ;
    for (int i = 0; i < N; i++){
        if(getpid()==father_pid){
            pid = fork();
        }
    }


    if(pid==0){
        printf(" i am  %d  , j'attends l'imprimante \n", getpid());

        p(semid,0);

        printf(" i am %d , j'utilise l'imprimante \n", getpid());

        sleep(10);

        printf(" i am %d , terminer l'utilisation \n", getpid());


        v(semid , 0);


        exit(0);
    }else{
        while(wait(NULL)!=-1);
    }

    printf("Je suis le pere %d mes fils sont terminer , i die now \n",getpid());

    semdestroy(semid);
return 0;
```

Résultat d'exécution :

- Avant rajout de SEM_UNDO :
  On lance le processus père , puis on exécute un **KILL -9 child_pid** , pour forcer la terminaison d'un fils a partir d'un autre invité de commande

```
sami@laptop:~/SE_RSD/tp2$ kill -9 9097
sami@laptop:~/SE_RSD/tp2$
```

```
sami@laptop:~/SE_RSD/tp2$ gcc exo3_spool.c -o spool -w
sami@laptop:~/SE_RSD/tp2$ ./spool
*****informations general sur le processu :*****

        key := ffffffff
        smid = 12
    Je suis le pere 9095 mes fils vont utilser l'imprimante


Donner le nombre de processus N : 10
 i am  9096  , j'attends l'imprimante
 i am 9096 , j'utilise l'imprimante
 i am  9097  , j'attends l'imprimante
 i am  9098  , j'attends l'imprimante
 i am  9100  , j'attends l'imprimante
 i am  9101  , j'attends l'imprimante
 i am  9102  , j'attends l'imprimante
 i am  9099  , j'attends l'imprimante
 i am  9105  , j'attends l'imprimante
 i am  9104  , j'attends l'imprimante
 i am  9103  , j'attends l'imprimante
 i am 9096 , terminer l'utilisation
 i am 9097 , j'utilise l'imprimante
```

On voit bien qu'il a bloqué les autres processus de s'exécuter .

- Apres rajout du SEM_UNDO :

```
sami@laptop:~/SE_RSD/tp2$ kill -9 9683
sami@laptop:~/SE_RSD/tp2$ kill -9 9685
sami@laptop:~/SE_RSD/tp2$
```

```
sami@laptop:~/SE_RSD/tp2$ gcc exo3_spool.c -o spool -w
sami@laptop:~/SE_RSD/tp2$ ./spool
*****informations general sur le processu :*****

        key := ffffffff
semget semcreate: File exists
        smid = 13
    Je suis le pere 9681 mes fils vont utilser l'imprimante


Donner le nombre de processus N : 6
 i am  9684  , j'attends l'imprimante
 i am 9684 , j'utilise l'imprimante
 i am  9683  , j'attends l'imprimante
 i am  9685  , j'attends l'imprimante
 i am  9687  , j'attends l'imprimante
 i am  9686  , j'attends l'imprimante
 i am  9682  , j'attends l'imprimante
 i am 9684 , terminer l'utilisation
 i am 9683 , j'utilise l'imprimante
 i am 9685 , j'utilise l'imprimante
 i am 9687 , j'utilise l'imprimante
 i am 9687 , terminer l'utilisation
 i am 9686 , j'utilise l'imprimante
 i am 9686 , terminer l'utilisation
 i am 9682 , j'utilise l'imprimante
 i am 9682 , terminer l'utilisation
Je suis le pere 9681 mes fils sont terminer , i die now
sami@laptop:~/SE_RSD/tp2$
```

La terminaison des fils **9683** et **9685** n'a pas empêcher les autres de s'exécuter

Exercice 4 (Producteur / Consommateur) :

Explication :

L'utilisation 4 sémaphores :

- 2 mutex **mutexC** et **mutexP** pour l'accès aux indices de chaque classe
- 1 sémaphore compteur **NVIDE** initialiser a N (nombre de tampons disponible)
- 1 sémaphore bloquante **NPLEIN** pour bloquer les consommateur

L'utilisation d'une mémoire partagé :

- Les deux indices de lecture et d'écriture
- Le tableau du tampon

Un producteur ne peut produire que si Nombre de tampons vide est > 0 , il exécute P(NVIDE) , insère dans le tampon puis V(NPLEIN) pour donner autorisation aux consommateurs de lire une valeur
Un producteur ne peut consommer que si Nombre de tampons plein > 0 , il exécute P(NPLEIN) , insère dans le tampon puis V(NVIDE) pour donner autorisation aux producteurs d'écrire une valeur

Implémentation :

Create.c
Contient l'initialisation des sémaphores et de la mémoire partagé

```
#include <sys/shm.h>
#include "my_semaphore.c"
typedef struct data{
    int idxr;
    int idxl;
    int tab[10];
}sdata;

int main(){

    key_t key = ftok("~/SE_RSD/semfolder","dd");
    int shmid=shmget(key,sizeof(sdata),IPC_CREAT|IPC_EXCL|0666);
    if(shmid==-1){ //la zone existe deja !

        shmid=shmget(key,sizeof(sdata),0); //recuperer son id
        printf("Segment existe deja id: %d \n",shmid);

    }else printf("Segment mémoire d'id:%d\n",shmid);
    sdata *sd=NULL;
    sd=shmat(shmid,sd,0);

    // on teste si la fonction shmat a bien marché
```

```
    if (sd == NULL)
    {
        printf(" shmat didnt work ");
    }else{

        sd->idxl=0;
        sd->idxr=0;
        printf("val écrites :%d, %d\n",sd->idxl,sd->idxr);
    }

    int semid = semcreate(key ,4 );
    unsigned short tabinit[4]={10,0,1,1};

    seminitall(semid,tabinit );

    return 0;
}
```

Producteur.c

Créer N processus fils producteur , N est donné par argv[1]

```
#include <sys/shm.h>
#include "my_semaphore.c"

#define nvide 0
#define nplein 1
#define mutexC 2
#define mutexP 3

typedef struct data{
    int idxr;
    int idxl;
    int tab[10];
}sdata;

int main(int argc, char const *argv[])
{

    if(argc !=2 ){
        printf("Nombre d'argument incorrect ");
        return 1;
    }
    int nombre_producteur = atoi(argv[1]);


    int valeur_ecrite = 0;

    key_t key = ftok("~/SE_RSD/semfolder","dd");
```

```
    int semid = semget(key, 4 , 0);
    printf(" i am producer id semaphore recuperer %d \n",semid);

    int shemid = shmget(key , sizeof(sdata) , 0);
    printf(" i am producer id shared memory recuperer %d \n", shemid);
    sdata *sd = shmat(shemid , sd , 0);

    int father_pid = getpid();
    for (int i = 0; i < nombre_producteur-1; i++)
    {
        if (getpid()==father_pid) fork();

    }
    while(1){
    p(semid , nvide);
    p(semid , mutexP);
    sd->tab[sd->idxr]=valeur_ecrite++;
    printf(" je suis producteur %d , j'ai disposer a l'indice %d , la valeur %d \n" ,
getpid(),sd->idxr , sd->tab[sd->idxr]);
    sd->idxr = (sd->idxr +1) %10;
    v(semid ,mutexP);
    v(semid , nplein);

    }
    return 0;
}
```

Consomateur.c

```
#include <sys/shm.h>
#include "my_semaphore.c"

#define nvide 0
#define nplein 1
#define mutexC 2
#define mutexP 3

typedef struct data{
    int idxr;
    int idxl;
    int tab[10];
}sdata;

int main(int argc, char const *argv[])
{

    if(argc !=2 ){
        printf("Nombre d'argument incorrect ");
        return 1;
    }
    int nombre_consomateur = atoi(argv[1]);

    key_t key = ftok("~/SE_RSD/semfolder","dd");
```

```
    int semid = semget(key, 4 , 0);
    printf(" i am consom id semaphore recuperer %d \n",semid);

    int shemid = shmget(key , sizeof(sdata) , 0);
    printf(" i am consom id shared memory recuperer %d \n", shemid);

    sdata *sd = shmat(shemid , sd , 0);

    int father_pid = getpid();
    for (int i = 0; i < nombre_consomateur-1; i++)
    {
        if (getpid()==father_pid) fork();

    }

    while(1){
        p(semid , nplein);
        p(semid , mutexC);
        printf(" je suis consomatteur %d j'ai lu a l'indice %d , la valeur %d \n"
,getpid(), sd->idxl , sd->tab[sd->idxl]);
        sd->idxl = (sd->idxl+1)%10;
        v(semid , mutexC);
        v(semid , nvide);

    }
return 0;
}
```

Résultat :

Au début , le tableau est vide , les consommateurs ne peuvent consommer

```
sami@laptop:~/SE_RSD/tp2$ gcc exo4_consm.c -o consomateur -w
sami@laptop:~/SE_RSD/tp2$ ./consomateur 3
 i am consom id semaphore recuperer 14
 i am consom id shared memory recuperer 32823
```

Si on lance le producteur les processus s'exécute en pseudo parallèle :

```
je suis producteur 10280 , j'ai disposer a l'indice 1 , la valeur 1256
je suis producteur 10281 , j'ai disposer a l'indice 2 , la valeur 1350
je suis producteur 10282 , j'ai disposer a l'indice 3 , la valeur 1095
je suis producteur 10280 , j'ai disposer a l'indice 4 , la valeur 1257
je suis producteur 10281 , j'ai disposer a l'indice 5 , la valeur 1351
je suis producteur 10282 , j'ai disposer a l'indice 6 , la valeur 1096
je suis producteur 10280 , j'ai disposer a l'indice 7 , la valeur 1258
je suis producteur 10281 , j'ai disposer a l'indice 8 , la valeur 1352
je suis producteur 10282 , j'ai disposer a l'indice 9 , la valeur 1097
```

```
je suis consomatteur 10266 j'ai lu a l'indice 3 , la valeur 1208
je suis consomatteur 10264 j'ai lu a l'indice 4 , la valeur 1360
je suis consomatteur 10265 j'ai lu a l'indice 5 , la valeur 1475
je suis consomatteur 10266 j'ai lu a l'indice 6 , la valeur 1209
je suis consomatteur 10264 j'ai lu a l'indice 7 , la valeur 1361
je suis consomatteur 10265 j'ai lu a l'indice 8 , la valeur 1476
je suis consomatteur 10266 j'ai lu a l'indice 9 , la valeur 1210
je suis consomatteur 10265 j'ai lu a l'indice 0 , la valeur 1362
je suis consomatteur 10265 j'ai lu a l'indice 1 , la valeur 1477
```

On a créer 3 **producteur 10280** , **10281** et **10282** et aussi 3 **consommateur 10266 , 10264** et **10265** on remarque bien qu'à chaque écriture ou lecture l'index est **incrementer** donc **l'exclusion mutuelle** est bien appliquée

Si On arrête le processus producteur , les consommateurs vont consommer tous les 10 cases possibles et ensuite se bloque eux aussi
Dans l'autre sense , si on arrête les consommateurs , les producteurs vont écrire dans les 10 cases et puis ils se bloquent.

L'état du sémaphore dans les deux cas :

```
sami@laptop:~/SE_RSD/tp2$ ipcs -s -i 15

Semaphore Array semid=15
uid=1000        gid=1000        cuid=1000       cgid=1000
mode=0666, access_perms=0666
nsems = 4
otime = Sun Dec 10 01:56:09 2023
ctime = Sun Dec 10 01:55:59 2023
semnum     value      ncount     zcount     pid
0          0          3          0          10443
1          10         0          0          10443
2          1          0          0          10436
3          1          0          0          10443

sami@laptop:~/SE_RSD/tp2$
```

```
Semaphore Array semid=15
uid=1000        gid=1000        cuid=1000       cgid=1000
mode=0666, access_perms=0666
nsems = 4
otime = Sun Dec 10 01:57:15 2023
ctime = Sun Dec 10 01:55:59 2023
semnum     value      ncount     zcount     pid
0          10         0          0          10449
1          0          3          0          10449
2          1          0          0          10449
3          1          0          0          10443

sami@laptop:~/SE_RSD/tp2$
```

Exercice 6 (ipcs clone) :

Explication :

On utilise la fonction semctl avec flag = IPC_STAT , afin de récupérer les infos qui concerne le processus , on utilise une variable partagé **kill_flag** pour savoir si un changement est fait

Implémentation :

```
#include"my_semaphore.c"
#include<time.h>
#include <sys/shm.h>

typedef struct data{    int kill_flag;
}sdata;

void showinfos(int semid ){

    struct semid_ds buffer;



    if(semctl(semid , 0,IPC_STAT , &buffer)==-1){
        perror("error semtcl");
        exit(EXIT_FAILURE);
    }


    printf("****************Informations sur les semaphroes*********************
\n\n");
    printf("Semaphore  ID: %d\n", semid);
    printf("Owner UID: %d\n", buffer.sem_perm.uid);
    printf("Owner GID: %d\n", buffer.sem_perm.gid);
    printf("Mode: %#o\n", buffer.sem_perm.mode);
    printf("Number of semaphores in set: %ld\n", (long)buffer.sem_nsems);
    printf("Otime = %s",ctime(&buffer.sem_otime));
    printf("Ctime = %s",ctime(&buffer.sem_ctime));
    printf("\n numseum     value\n");
    printf("    %d             %d\n" , 0 , semctl(semid , 0, GETVAL));




}

int main(int argc, char const *argv[])
{
```

```
    if (argc <2){
        printf("Nombre d'arguments insuffisant\n");
        exit(EXIT_FAILURE);
    }

    int semid = atoi(argv[1]);

    struct semid_ds buffer ;

    int numsum = buffer.sem_nsems;



    key_t key = ftok("~/SE_RSD/semfolder","bb");

    int shmid=shmget(key,sizeof(sdata),IPC_CREAT|IPC_EXCL|0666);

    if(shmid==-1){ //la zone existe deja !

        shmid=shmget(key,sizeof(sdata),0); //recuperer son id
        printf("Segment existe deja id: %d \n",shmid);

    }else printf("Segment mémoire d'id:%d\n",shmid);

    sdata *sd=shmat(shmid,sd,0);
    sd->kill_flag=0;

    while (1)
    {
        while (sd->kill_flag==1)
        {
            sleep(1);
        }
        sleep(5);
        showinfos(semid);
        sd->kill_flag =0;


    }


    return 0;
}
```

Résultat :

```
**********Informations sur les semaphroes**********

Semaphore  ID: 16
Owner UID: 1000
Owner GID: 1000
Mode: 0666
Number of semaphores in set: 1
Otime = Sun Dec 10 02:09:19 2023
Ctime = Sun Dec 10 02:08:56 2023

 numseum     value
   0           0
```

```
**********Informations sur les semaphroes**********

Semaphore  ID: 16
Owner UID: 1000
Owner GID: 1000
Mode: 0666
Number of semaphores in set: 1
Otime = Sun Dec 10 02:09:31 2023
Ctime = Sun Dec 10 02:08:56 2023

 numseum     value
   0           1
```

.

END 🙂