Rapport Exo 4 et 5

Belmouloud Mustapha Abdellah

212131092524

Groupe 1

Exo 4:

Creation des semaphores: 3 semaphore pour 1 consomateur et 1 producteur.

- mutex pour la section critique
- full pour les consomateurs
- empty pour les producteur

full et empty vont etre utilisé comme des compteur

pour le cas de n consomateurs et n producteur on vat ajouter a semaphore

pour chaque un, donc 5 semaphores en total

la memoire partagé vat etre créé comme l'exercise le dicte

creat.c

```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>


typedef struct{
int tab[10];
int indxl;
int indxr;
} madata;

int main(){

key_t key=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'b');
int shmid=shmget(key,sizeof(madata),IPC_CREAT|IPC_EXCL|0666);
if (shmid == -1) shmid = shmget(key, sizeof(madata), 0666);

madata *shmval = shmat(shmid, NULL, 0);
for (int i = 0; i < 9; i++){
shmval->tab[i]=0;
}
shmval->indxl = 0;
shmval->indxr = 0;
```

```
shmdt(shmval);

key_t mutexkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'm');

key_t fullkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'f');
key_t emptykey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'e');
key_t prokey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'p');
key_t conkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'c');

int mutex = semget(mutexkey, 1, IPC_CREAT | IPC_EXCL | 0666);
int full = semget(fullkey, 1, IPC_CREAT | IPC_EXCL | 0666);
int empty = semget(emptykey, 1, IPC_CREAT | IPC_EXCL | 0666);
int pro = semget(prokey, 1, IPC_CREAT | IPC_EXCL | 0666);
int con = semget(conkey, 1, IPC_CREAT | IPC_EXCL | 0666);

if (full == -1) mutex = semget(mutexkey, 1, 0666);
if (full == -1) full = semget(fullkey, 1, 0666);
if (empty == -1) empty = semget(emptykey, 1, 0666);
if (pro == -1) pro = semget(prokey, 1, 0666);
if (con == -1) con = semget(conkey, 1, 0666);

semctl(mutex, 0, SETVAL, 1);
semctl(full, 0, SETVAL, 0);
semctl(empty, 0, SETVAL, 10);
semctl(pro, 0, SETVAL, 1);
semctl(con, 0, SETVAL, 1);
return 0;
}
```

consomateur.c :

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

typedef struct{
int tab[10];
int indxl;
int indxr;
} madata;

struct sembuf p = {0, -1, 0};
struct sembuf v = {0, +1, 0};
// semop(semid, &p, 1);
```

```
void consume(madata *shm){
int in = shm->indxl;
shm->tab[in] = 0;
if (in == 9) shm->indxl = 0;
else shm->indxl = in + 1;
sleep(1);
printf("consumer: %d \n", in);
}

int main(){

key_t key=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'b');
int shmid = shmget(key, sizeof(madata), 0666);

madata *shmval = shmat(shmid, NULL, 0);

key_t mutexkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'm');
key_t fullkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'f');
key_t emptykey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'e');
key_t prokey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'p');
key_t conkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'c');

int mutex = semget(mutexkey, 1, 0666);
int full = semget(fullkey, 1, 0666);
int empty = semget(emptykey, 1, 0666);
int pro = semget(prokey, 1, 0666);
int con = semget(conkey, 1, 0666);

while (1)
{
semop(con, &p, 1);
semop(full, &p, 1);
semop(mutex, &p, 1);

consume(shmval);

semop(mutex, &v, 1);
semop(empty, &v, 1);
semop(con, &v, 1);
}

return 0;

}
```

producteur.c

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


typedef struct{
int tab[10];
int indxl;
int indxr;
} madata;

struct sembuf p = {0, -1, 0};
struct sembuf v = {0, +1, 0};
// semop(semid, &p, 1);

void produce(madata *shm){
int in = shm->indxr;
shm->tab[in] = 1;
if (in==9) shm->indxr = 0;
else shm->indxr = in + 1;
sleep(1);
printf("produced: %d \n", in);
}

int main(){

key_t key=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'b');
int shmid = shmget(key, sizeof(madata), 0666);

madata *shmval = shmat(shmid, NULL, 0);

key_t mutexkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'm');
key_t fullkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'f');
key_t emptykey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'e');
key_t prokey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'p');
key_t conkey=ftok("/home/crow/uni/rsd-s1/sys/tp1/exo4", 'c');

int mutex = semget(mutexkey, 1, 0666);
int full = semget(fullkey, 1, 0666);
int empty = semget(emptykey, 1, 0666);
int pro = semget(prokey, 1, 0666);
int con = semget(conkey, 1, 0666);

while (1)
```

```
{
semop(pro, &p, 1);
semop(empty, &p, 1);
semop(mutex, &p, 1);

produce(shmval);

semop(mutex, &v, 1);
semop(full, &v, 1);
semop(pro, &v, 1);
}

return 0;

}
```

Exo 5:

Explication de la solution:

- il faut avoir un fichier pour creer et initialiser les memoires partagé et les semaphores necessaires
- il faut creer une memoire partagé pour guarder compte de combien de processus sont arrivés a la barriere
- il faut creer un semaphore pour l'accée a cette memoire partager
- il faut un semaphore pour bloquer les 2 premier processus, on peut pas blocker dans le semaphore précedent donc on vat avoir une variable temp = variable attaché a la memoir partagé pour evaluer si il faut bloquer ou liberer
- un semaphore pour la section critique

base.c

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



int main(){

/*create shared memory and semaphores*/
/*shared memory to check if all three arrived + semaphore for it*/
/*semaphore to block , not inside the shm semaphore so gotta make a temp variable*/
/*sem for the actual critical section*/
key_t shmkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'M');
int shmid = shmget(shmkey, sizeof(int), IPC_CREAT | IPC_EXCL | 0666);
if(shmid == -1) shmid = shmget(shmkey, sizeof(int), 0666);
int *mem = shmat(shmid, NULL, 0);
```

```
*mem = 0;

key_t checkkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'C');
int checkid = semget(checkkey, 1, IPC_CREAT | IPC_EXCL | 0666);
if(checkid == -1) checkid = semget(checkkey, 1, 0666);
semctl(checkid, 0, SETVAL, 1);

key_t blockkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'B');
int blockid = semget(blockkey, 1, IPC_CREAT | IPC_EXCL | 0666);
if(blockid == -1) blockid = semget(blockkey, 1, 0666);
semctl(blockid, 0, SETVAL, 0);

key_t sectionkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'S');
int sectionid = semget(sectionkey, 1, IPC_CREAT | IPC_EXCL | 0666);
if(sectionid == -1) sectionid = semget(sectionkey, 1, 0666);
semctl(sectionid, 0, SETVAL, 1);

shmdt(mem);

return 0;
}
```

oxygen.c

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

struct sembuf p = {0,-1,0};
struct sembuf v = {0,+1,0};

void bondOxygen(){
printf("oxygen bonding, molécule H2O formée H2O %d \n", getpid());
}

void arrived(){

int temp;

key_t shmkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'M');
int shmid = shmget(shmkey, sizeof(int), 0666);

key_t checkkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'C');
int checkid = semget(checkkey, 1, 0666);
```

```
key_t blockkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'B');
int blockid = semget(blockkey, 1, 0666);

semop(checkid, &p, 1);
int *mem = shmat(shmid, NULL, 0);
*mem = *mem + 1;
temp = *mem;
semop(checkid, &v, 1);

if(temp==3){
semop(blockid, &v, 1);
semop(blockid, &v, 1);
}else{
semop(blockid, &p, 1);
}
}

int main(){

key_t sectionkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'S');
int sectionid = semget(sectionkey, 1, 0666);

arrived();
semop(sectionid, &p, 1);
bondOxygen();
semop(sectionid, &v, 1);

return 0;
}
```

hydrogen.c

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

struct sembuf p = {0,-1,0};
struct sembuf v = {0,+1,0};

void bondOxygen(){
printf("oxygen bonding, molécule H2O formée H2O %d \n", getpid());
}
```

```
void arrived(){

int temp;

key_t shmkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'M');
int shmid = shmget(shmkey, sizeof(int), 0666);

key_t checkkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'C');
int checkid = semget(checkkey, 1, 0666);

key_t blockkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'B');
int blockid = semget(blockkey, 1, 0666);

semop(checkid, &p, 1);
int *mem = shmat(shmid, NULL, 0);
*mem = *mem + 1;
temp = *mem;
semop(checkid, &v, 1);

if(temp==3){
semop(blockid, &v, 1);
semop(blockid, &v, 1);
}else{
semop(blockid, &p, 1);
}
}

int main(){

key_t sectionkey = ftok("/home/crow/uni/rsd-s1/sys/tp1", 'S');
int sectionid = semget(sectionkey, 1, 0666);

arrived();
semop(sectionid, &p, 1);
bondOxygen();
semop(sectionid, &v, 1);

return 0;
}
```