

---

# **Compte-rendu TP6 OS302**

Dougy Hugo & Creton Pierre

## Introduction

Durant ce TP, nous allons nous intéresser à la synchronisation par sémaphores. Pour cela, nous allons utiliser les fonctions vues en cours pour gérer les sémaphores. Nous allons également utiliser les segments de mémoire partagée, la gestion de processus et les signaux.

## Exercice 1

### Implémentation

Pour implémenter cet exercice, nous avons tout d'abord dû initialiser et créer les structures demandées (sémaphores, segment de mémoire partagée, signaux). Nous avons ensuite défini le cas où N était pair ou impair.

### Code

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <signal.h>
5  #include <sys/ipc.h>
6  #include <sys/sem.h>
7  #include <sys/shm.h>
8  #include <sys/wait.h>
9  #include <sys/msg.h>
10
11 //nombre d'ouvriers
12 #define N 7
13
14 //on définit sem_id en variable globale car on devra l'utiliser dans le
    handle
15 int sem_id;
16 //on crée trois structure sembuf pour les opérations P,V et le cas nul
17 struct sembuf sem_p, sem_v, sem_n;
18
19 //fonction s'exécutant à la réception du signal SUGUSR1 pour les fils
20 void handle(int signal){
21     semop(sem_id, &sem_v, 1);
22     printf("Un ouvrier sort de l'ascenseur\n");
23     exit(0);
24 }
25
26 int main(int argc, char const *argv[]){
```

```
27
28 //creation sempahore
29 key_t key1 = ftok("fich1",0);
30
31 if ((sem_id = semget (key1, 1, IPC_CREAT | 0666))== -1){
32     printf("Erreur création semaphore\n");
33     return EXIT_FAILURE;
34 }
35 //on initialise notre semaphore à 2
36 if(semctl(sem_id, 0, SETVAL, 2) == -1){
37     printf("Erreur initiamisation semaphore\n");
38     return EXIT_FAILURE;
39 }
40
41 //on initialise les valeurs pour les opérations P et V et le cas nul
42 sem_v.sem_num = 0;
43 sem_v.sem_op = 1;
44 sem_v.sem_flg = 0;
45
46 sem_p.sem_num = 0;
47 sem_p.sem_op = -1;
48 sem_p.sem_flg = 0;
49
50 sem_n.sem_num = 0;
51 sem_n.sem_op = 0;
52 sem_n.sem_flg = 0;
53
54
55 pid_t fils[N];
56
57 //creation segment de mémoire partagée
58 key_t key2 = ftok("fich2",2);
59 int mem_id;
60 if ((mem_id = shmget(key2, 10*sizeof(pid_t), IPC_CREAT|0666)) ==
61     -1) {
62     printf("Erreur création segment\n");
63     return EXIT_FAILURE;
64 }
65 /*initialisation à 0 pour que les fils sachent dans quel "case" du
66    segment de mémoire partagée écrire*/
67 pid_t *ini = malloc(2*sizeof(pid_t));
68 ini = (pid_t*)shmat(mem_id, NULL, 0);
69 if(ini == (pid_t*)-1){
70     printf("Erreur d'attachement avec le pere\n");
71     return EXIT_FAILURE;
72 }
73 ini[0] = 0;
74 ini[1] = 0;
75 if(shmdt(ini)==-1){
76     printf("Erreur de détachement initialisation\n");
```

```
76     return EXIT_FAILURE;
77 }
78
79 //création du segment de mémoire pour stocker les pid
80 pid_t *attache = malloc(2*sizeof(pid_t));
81
82 //on crée les N fils
83 for(int i = 0; i<N;i++){
84     fils[i] = fork();
85     if (fils[i] == -1){
86         printf("Erreur création fils\n");
87         return EXIT_FAILURE;
88     }
89     else if (fils[i] == 0){
90         //on est dans le ième fils
91
92         //on execute l'opération p (-1)
93         semop(sem_id, &sem_p, 1);
94
95         //on affecte la fonction handle à la réception du signal
96         signal(SIGUSR1,handle);
97
98         //on alloue un espace mémoire aléatoire (argument NULL)
99         //pour le segment de mémoire partagée
100        attache = (pid_t*)shmat(mem_id, NULL, 0);
101        if(attache == (pid_t*)-1){
102            printf("Erreur d'attachement avec le fils\n");
103            return EXIT_FAILURE;
104        }
105        //on stocke les pid dans fils dans le segment de mémoire
106        //partagée, dans attache[0] si on est le premier à rentrer
107        //dans l'ascenseur ou si on est seul, et dans attache[1]
108        //si on est le deuxième a entrer dans l'ascenseur
109        if (attache[0] == 0){
110            attache[0]=getpid();
111        }
112        else{
113            attache[1]=getpid();
114        }
115        if(shmdt(attache)==-1){
116            printf("Erreur de détachement avec un des fils\n");
117            return EXIT_FAILURE;
118        }
119        //printf("%d attend \n",getpid());
120
121        //en attente d'un signal pour stoper les fils
122        pause();
123    }
124 }
```

```
122 //on est dans le père
123 int i = N;
124
125 // on associe également dans le père le segment de mémoire partagée
126 attache = (pid_t*)shmat(mem_id, NULL, 0);
127 if(attache == (pid_t*)-1){
128     printf("Erreur d'attachement avec le pere\n");
129     return EXIT_FAILURE;
130 }
131
132 while (i !=0){
133     if (i != 1){ //cas pair
134         semop(sem_id, &sem_n, 1); //processus bloqué tant que le
135             semaphore n'est pas nul
136         sleep(1); //temps que les ouvriers montent dans l'
137             ascenseur
138         //on récupère les pids des ouvriers (fils) monté dans l'
139             ascenseur
140         pid_t fils1 = attache[0];
141         pid_t fils2 = attache[1];
142         printf("%d et %d sont montés dans l'ascenseur\n",fils1,
143             fils2);
144         attache[0] = 0;
145         attache[1] = 0;
146         sleep(1); //temps que l'ascenseur monte
147         printf("%d et %d sont arrivés en haut\n",fils1,fils2);
148         //on envoie un signal aux fils pour qu'ils sortent de l'
149             ascenseur
150         kill(fils1,SIGUSR1);
151         kill(fils2,SIGUSR1);
152         sleep(1); //temps que les ouvriers sortent
153         i-=2;
154         sleep(1); //le temps que l'ascenseur redescende
155     }
156     else{ //cas impair, si N est impair, à la fin il reste un
157         ouvrier
158         if (semctl(sem_id, 0, GETVAL, 0) == 1){ //si on a bien un
159             ouvrier dans l'ascenseur, la valeur du semaphore est 1
160         sleep(1); //temps que l'ascenseur monte
161         pid_t fils = attache[0];
162         printf("%d est monté dans l'ascenseur\n",fils);
163         attache[0] = 0;
164         sleep(1); //temps que l'ascenseur monte
165         printf("%d est arrivé en haut\n",fils);
166         kill(fils,SIGUSR1);
167         sleep(1); //temps que l'ouvrier sorte
168         i--;
169         sleep(1); //le temps que l'ascenseur redescende
170     }
171 }
172 }
```

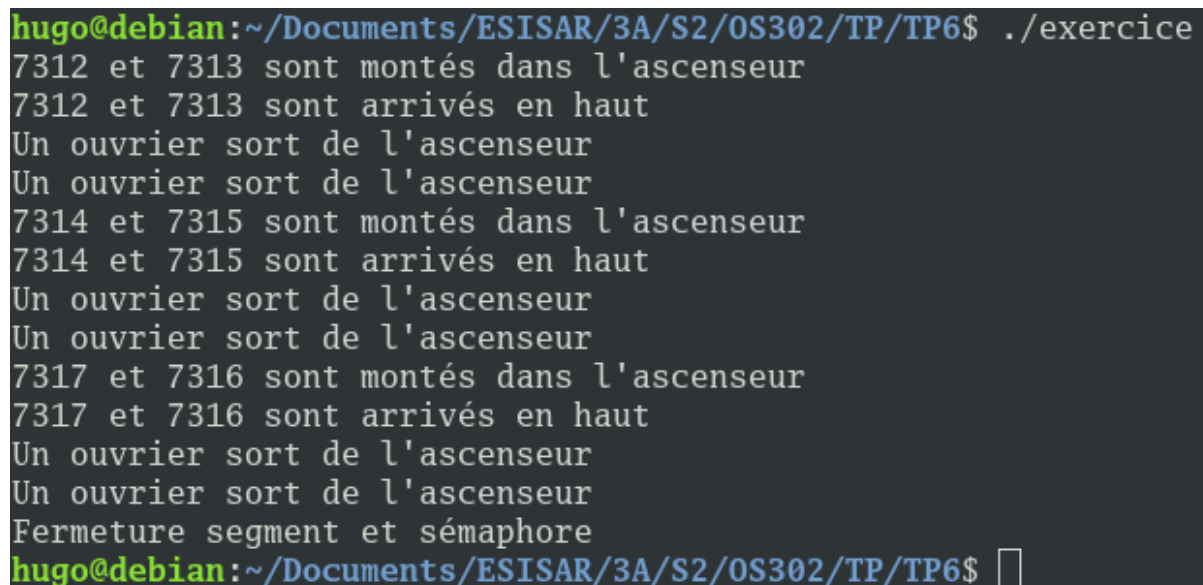
```
166     printf("Fermeture segment et sémaphore\n");
167     if(shmdt(attache)==-1){
168         printf("Erreur de détachement avec le pere\n");
169         return EXIT_FAILURE;
170     }
171     if(semctl(sem_id, 0, IPC_RMID, NULL)==-1){
172         printf("Erreur de suppression du semaphore\n");
173         return EXIT_FAILURE;
174     }
175     if(shmctl(mem_id, 0, IPC_RMID)==-1){
176         printf("Erreur de suppression du segment\n");
177         return EXIT_FAILURE;
178     }
179
180     return 0;
181 }
```

## Résultat

Nous avons testé notre programme pour 2 cas.

### 1: si N est pair

Nous obtenons alors le résultat suivant:



```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP6$ ./exercice
7312 et 7313 sont montés dans l'ascenseur
7312 et 7313 sont arrivés en haut
Un ouvrier sort de l'ascenseur
Un ouvrier sort de l'ascenseur
7314 et 7315 sont montés dans l'ascenseur
7314 et 7315 sont arrivés en haut
Un ouvrier sort de l'ascenseur
Un ouvrier sort de l'ascenseur
7317 et 7316 sont montés dans l'ascenseur
7317 et 7316 sont arrivés en haut
Un ouvrier sort de l'ascenseur
Un ouvrier sort de l'ascenseur
Fermeture segment et sémaphore
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP6$
```

**Figure 1:** résultat du programme exercice avec N = 6 ouvriers

On voit bien que les ouvriers entrent deux par deux dans l'ascenseur et qu'ils ressortent deux par deux.

À la fin, on a bien la suppression des sémaphores et du segment de mémoire partagée.

Pour le constater, on peut mettre en pause le programme pendant son exécution et regarder avec la commande `ipcs` si il existe bien un ensemble de sémaphore et un segment de mémoire partagée.

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP6$ ./exercice
8053 et 8054 sont montés dans l'ascenseur
8053 et 8054 sont arrivés en haut
Un ouvrier sort de l'ascenseur
Un ouvrier sort de l'ascenseur
^Z
[1]+  Stoppé                  ./exercice
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP6$ ipcs

----- Files de messages -----
clef          msqid          propriétaire perms          octets utilisés messages

----- Segment de mémoire partagée -----
clef          shmid          propriétaire perms          octets          nattch          états
0x00000000    2129920        hugo           600           524288          2              dest
0x00000000    393217         hugo           600           524288          2              dest
0x00000000    425986         hugo           600           524288          2              dest
0x00000000    2031619        hugo           600          16777216         2              dest
0x00000000    524292         hugo           600           524288          2              dest
0x00000000    2162693        hugo           600          4194304          2              dest
0x00000000    851974         hugo           600           524288          2              dest
0x00000000    2195463        hugo           600          67108864         2              dest
0x00000000    3407880        hugo           600          16777216         2              dest
0x020671f0    3440649        hugo           666            40              1
0x00000000    2424842        hugo           600           524288          2              dest

----- Tableaux de sémaphores -----
clef          semid          propriétaire perms          nsems
0x00067366    98304          hugo           666            1

hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP6$
```

**Figure 2:** Résultat de la commande `ipcs` pendant l'exécution du programme `exercice`

On voit bien qu'il y a un ensemble de sémaphore créé ainsi qu'un segment de mémoire partagée de 40 octets ( $10 * \text{sizeof}(\text{pid}_t)$ ).

Après l'exécution normal du programme, la commande `ipcs` nous retourne bien aucun sémaphore et aucun segment de mémoire partagée.

## 2: si N est impair

Dans ce cas, nous obtenons le résultat suivant:

```
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP6$ ./exercice
7210 et 7211 sont montés dans l'ascenseur
7210 et 7211 sont arrivés en haut
Un ouvrier sort de l'ascenseur
Un ouvrier sort de l'ascenseur
7212 et 7213 sont montés dans l'ascenseur
7212 et 7213 sont arrivés en haut
Un ouvrier sort de l'ascenseur
Un ouvrier sort de l'ascenseur
7214 et 7215 sont montés dans l'ascenseur
7214 et 7215 sont arrivés en haut
Un ouvrier sort de l'ascenseur
Un ouvrier sort de l'ascenseur
7216 est monté dans l'ascenseur
7216 est arrivé en haut
Un ouvrier sort de l'ascenseur
Fermeture segment et sémaphore
hugo@debian:~/Documents/ESISAR/3A/S2/OS302/TP/TP6$ █
```

**Figure 3:** résultat du programme exercice avec  $N = 7$  ouvriers

On constate qu'à la fin il reste un ouvrier dans l'ascenseur et le programme gère correctement sa sortie de l'ascenseur.

## Travail collaboratif

Pour ce TP, nous nous sommes réparti le travail de la manière suivante:

Exercice 1:

- *Hugo & Pierre*: réflexion sur les différentes structures du code
- *Hugo & Pierre*: implémentation & tests
- *Hugo & Pierre*: commentaires

## Conclusion

Pour conclure sur ce TP, nous avons pu apprendre à manipuler les ensembles de sémaphores pour synchroniser des processus. De plus le programme réalisé nous a permis d'utiliser une grande partie



des notions vues en cours pour réaliser un problème plus complexe.

Pour utiliser les sémaphores et les segments de mémoire partagée, nous nous sommes référés aux pages man des commandes, ainsi qu'au cours.