Laboratoire

I.P.C.

jcj mba mwa nvs

septembre 2021

Table des matières

Chapitre 1

mémoire partagée & permanence

1.1 IPC - LaboIPC 01-01 - permanence et suppression

Titre :	IPC - LaboIPC 01-01 - permanence et suppression
Support :	MDV2007 Installation Classique
Date :	07/2011

1.1.1 Énoncé

Écrire un programme FreeShm.c qui efface toute les mémoires partagées qui existent sur le système après avoir affiché les informations concernant ces mémoires. Afin de tester ce programme, écrire un programme AllouеShm.c qui réserve et initialise de la mémoire partagée. Ajouter le programme AttacheShm.c qui consulte les zones de mémoire allouées.

1.1.2 Une solution

```
/*
NOM      : FreeShm.c
CLASSE   : IPC - LaboIPC 01-01
#OBJET  : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
 */
#include <stdlib.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <time.h>
int main(int argc, char * argv[])
{  int shmid; struct shmid_ds infos;
   int i;
   // test de tous les identifiants possibles
   //for (i=0;i<2147483647;i++)
   for (i=1;i<65536 ;i++) // 2147483647 est un peu long ...
   {  if ((shmid = shmget(i,1,0777)) > 0 &&
           shmctl(shmid,IPC_STAT,&infos)==0)
      {  printf("\n\nClé identifiante : %5d\n",i);
         printf("Identifiant noyau : %5d\n",shmid);
         printf("Taille : %10d bytes\n",(int)infos.shm_segsz);
         printf("Process créateur : %d\n",infos.shm_cpid);
         printf("Process dernier accès : %d\n",infos.shm_lpid);
         printf("Nombre d'attachements : %d\n",(int)infos.shm_nattch);
         printf("Propriétaire de ce process : %d\n",infos.shm_perm.uid);
         printf("Date de dernier attachement : %s",ctime(&infos.shm_atime));
         printf("Du dernier détachement : %s",ctime(&infos.shm_dtime));
         printf("De la dernière modif droits : %s",ctime(&infos.shm_dtime));
         // marquage pour suppression
         if (shmctl(shmid,IPC_RMID,0) <0)
```

```
            perror ("semctl");
         else
            printf("\nidentifiant %d marqué pour libération\n",shmid);
      }
   }
   exit(0);
}
```

```
/*
NOM      : AllOueShm.c
CLASSE   : IPC - LaboIPC 01-01
#OBJET  : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
   : Ajout du while pause N. Vansteenkiste 2021
*/
#include <stdlib.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>  // pour pause
int main(int argc, char * argv[])
{  int shmid1,shmid2;
   char * c;
   int * i;
   shmid1 = shmget(256,1,0777|IPC_CREAT);
   shmid2 = shmget(512,4,0777|IPC_CREAT);
        printf ("shmid1=%d, shmid2=%d\n", shmid1,shmid2);
   c=shmat(shmid1,0,0777);
   i=shmat(shmid2,0,0777);
   *c='a';
   *i=1234;
   // shmdt(c);
   shmdt(i);

   while (1) pause();
   // boucle pour ne pas terminer Alloue
   // comme on ne s'est pas détaché de c, lorsque
   // FreeShm est exécuté, seule la variable partagée i
   // est détruite, pas celle associée à c
   // utiliser
   // $ ipcs -m
   // pour voir les mémoires partagées actives
   // quand on tue Alloue, il y a un détachement automatique
   // de c => à ce moment la zone mémoire liée à c est
   // détruite

   exit(0);
}
```

```
/*
NOM      : AttachShm.c
CLASSE   : IPC - LaboIPC 01-01
#OBJET  : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
   : MBA 11/2017
   : NVS 2021
*/
#include <stdlib.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
int main(int argc, char * argv[])
{  int shm1,shm2;
   char * c;
   int * i;
   shm1 = shmget(256,1,0777/*|IPC_CREAT*/);
   shm2 = shmget(512,4,0777/*|IPC_CREAT*/);
   if (shm1 >= 0 && shm2 >= 0)
   {
      c=shmat(shm1,0,0777);
      i=shmat(shm2,0,0777);
      printf ("shm1=%d, c=%c, shm2=%d, i=%d\n", shm1, *c, shm2, *i);
      shmdt(c);
```

```
        shmdt(i);
    }
    else
    {
        printf("%d %d\n", shm1, shm2);
    }
    exit(0);
}
```

1.1.3 Commentaires

— Les zones de mémoire allouées sont permanentes. Un même identifiant indique qu'il s'agit de la même zone.
— On ne peut libérer que les mémoires partagées dont on est propriétaire, sauf le root qui a tous les droits.

1.1.4 En roue libre

— Décommentez le deuxième appel à shmdt dans le code et vérifiez la différence de comportement. (make ; ./Demo).
— Montrez que l'adresse d'attachement de la mémoire partagée à l'espace d'adressage du programme est un multiple de la taille d'une page (2^{12}) en l'affichant avec
— Un processus peut-il continuer à modifier la zone de mémoire attachée après son "marquage par suppression" par le programme FreeShm ? Quel comportement vérifie-t-on lors de nouveaux appels à shmget et shmat pour la même clé après marquage pour suppression ?

Chapitre 2

mémoire partagée & sémaphores

2.1 IPC - LaboIPC 02-01 - section critique

Titre :	IPC - LaboIPC 02-01 - section critique
Support :	MDV2007 Installation Classique
Date :	07/2011

2.1.1 Énoncé

Deux process, père et fils, affichent à l'écran ce qui est saisi au clavier. Il faut synchroniser les process de telle façon que l'utilisateur sait à quel process il s'adresse et que l'affichage précise quel process affiche. Ce programme utilise un argument qui vaut soit s (les process sont synchronisés), soit ns (les process ne sont pas synchronisés). Prendre s par défaut.

2.1.2 Une solution

```
/*
NOM      : Critique.c
CLASSE   : IPC - LaboIPC 02-01
#OBJET  : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
*/
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/wait.h>
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include <unistd.h>

int opsem(int sem, int i)
{  int n; struct sembuf op[1];
   op[0].sem_num = 0; // premier et unique sémaphore
   op[0].sem_op = i;
   op[0].sem_flg = SEM_UNDO; //
   if ((n=semop(sem,op,1))==-1)
   {  perror ("semop");
      exit(1);
   }
   return(n);
}
int main (int argc, char * argv[])
{  int n,sem,s;
   char buff[100];
   if ((argc==2) && (strcmp(argv[1],"ns")==0)) s=0; else s=1;
   if (s==1) {
           if (( sem=semget(IPC_PRIVATE,1,0666|IPC_CREAT))==-1)
```

```
            {   perror ("semget");
             exit(-1);
            }

            if (semctl(sem,0,SETVAL,1)==-1) // une ressource pour une SC
            {       perror ("semctl");
               exit(1);
            }
    }
    if (fork()==0)
    {  do
       {  if (s==1) opsem (sem, -1); // down
          printf("[fils] : Je veux lire\n");
          n=read(0,buff,100);
          printf("Le fils a lu : [%s]\n",buff);
          if (s==1) opsem (sem, +1); // up
          buff[n-1]=0;
       } while (strcmp(buff,"quit")!=0);
       printf("Le fils a terminé \n");
       exit(0);
    }
    do
    {  if (s==1) opsem (sem, -1); // down
       printf("[père] : Je veux lire\n");
       n=read(0,buff,100);
       printf("Le père a lu : [%s]\n",buff);
       if (s==1) opsem (sem, +1); // up
       buff[n-1]=0;
    } while (strcmp(buff,"quit")!=0);
    printf("Le père a terminé \n");
    wait(0);
    if (s==1)
            if (semctl(sem,0,IPC_RMID)!=0)
            {   perror ("semctl");
               exit(1);
            }
    exit(0);
}
```

2.1.3 Commentaires

- Dans le cas d'un père et une fils l'identifiant est transmis au moment du fork. Nous choisissons d'utiliser IPC_PRIVATE.
- La ressource est mise à 1 au début pour écrire la section critique (lecture + affichage) de chaque processus
- La section critique doit être la plus petite possible sinon ce n'est pas la peine d'avoir la multiprogrammation.
- Le père utilise l'appel système wait() avant la suppression du sémaphore par IPC_RMID(). Est-ce indispensable ?
- Les variables n et buff sont des instances différentes chez le père et chez le fils.

2.1.4 En roue libre

Vérifiez si un sémaphore est supprimé après la mort des process qui l'utilisent.
En vous inspirant de l'exemple donné, créez un module source c (une librairie) avec les outils suivants :

- int creeSem () ; // sans paramètres, crée un nouveau sémaphore unique et en retourne l'id
- void initsem (int sem, int val) ; // initialise le compteur de ressources à val
- void supsem (int sem) ; // supprime le sémaphore
- void down (int sem) ; // obtient une ressource
- void up (int sem) ; // restitue une ressource
- void zero (int sem) ; // attend que le compteur vaille 0

testez vos fonctions dans l'exercice précédent.

2.2 IPC - LaboIPC 02-02 - synchronisation pour Affichage continu

Titre :	IPC - LaboIPC 02-02 - synchronisation pour Affichage continu
Support :	MDV2007 Installation Classique
Date :	07/2011

2.2.1 Énoncé

Écrire un programme AffContinu.c qui affiche de façon permanente un caractère saisi au clavier. L'affichage ne peut commencer qu'après avoir saisi un premier caractère. Exemple, si on saisi 'a' suivi de enter, des 'a' s'affichent de façon continue à l'écran...etc. Le programme s'arrête si on frappe 'q' suivi de enter. L'ordre de lancement est quelconque.

2.2.2 Une solution

```
/*
NOM      : AffContinu.c
CLASSE   : IPC - LaboIPC 02-02
#OBJET  : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/shm.h>
#include<sys/wait.h>
#include <stdlib.h>
#include <unistd.h>

int opsem(int sem, int i)
{  int n; struct sembuf op[1];
   op[0].sem_num = 0; // premier et unique sémaphore
   op[0].sem_op = i;
   op[0].sem_flg = SEM_UNDO;
   if ((n=semop(sem,op,1))==-1)
   {  perror ("semop");
      exit(1);
   }
   return(n);
}

int main ( int argc, char * argv[])
{  int n,sem,r;
   int shm;
   char t[2];
   char *c;
   if (( sem=semget(123,1,0666|IPC_CREAT))==-1)
   {  perror ("semget");
            exit(-1);
   }
   if (semctl(sem,0,SETVAL,0)==-1) // 0 ressources (feu rouge au début)
   {       perror ("semctl");
           exit(1);
   }
   if ((shm = shmget (IPC_PRIVATE, 1, 0666|IPC_CREAT)) == -1)
   {  perror ("shmget");
            exit(-1);
   }
   c = shmat (shm, NULL, 0666);
   if (c == NULL){
                perror ("shmat"); exit (3);
        }
   if ((r=fork())==0) //fils
   {  opsem (sem, -1); // down - feu rouge ?
      while (*c != 'q') {write(1,c,1);usleep(10000);}
      shmdt (c);
      exit(0);
```

```
    }
    n=0;
    do
    {  read(0,t,2);
       *c=t[0];
       if (n==0) opsem (sem, +1); // up
       n=1;
    } while (t[0] != 'q');
    wait(0);
    shmdt (c);
    if (semctl(sem,0,IPC_RMID)!=0)
    {  perror ("semctl");
            exit(1);
    }
    exit(0);
}
```

2.2.3 Commentaires

— La lecture est bloquante, l'affichage est continu. Il faut donc deux process.
— Il faut communiquer le caractère lu à l'autre process. Nous utilisons une mémoire partagée à ce fin
— Le sémaphore est utilisé pour empêcher le process qui affiche de commencer avant celui qui lit.
— Au clavier, il faut lire au minimum 2 caractères (celui souhaité + enter). Seul le premier est affiché.
— Il est utile de vérifier que tous ces process sont terminés à l'aide de ps.

2.2.4 En roue libre

— Modifiez la logique de synchronisation de départ pour que l'on puisse utiliser un processus afficheur indépendant (compilé séparément).
— Comment l'adapter si le nombre d'afficheurs n'est pas connu ?

2.3 IPC - LaboIPC 02-03 - synchronisation accès aux douches

Titre :	IPC - LaboIPC 02-03 - synchronisation accès aux douches
Support :	MDV2007 Installation Classique
Date :	07/2011

2.3.1 En roue libre

— Gérez une douche où ne peuvent se trouver en même temps filles (processus F) et garçons (processus G).
— Un garçon ne pourra entrer en présence d'au moins une fille et vice-versa.
— Le nombre de filles et garçons n'est pas connu et est quelconque.

2.4 IPC - LaboIPC 02-04 - producteur-consommateur

Titre :	IPC - LaboIPC 02-04 - producteur-consommateur
Support :	MDV2007 Installation Classique
Date :	11/2017

2.4.1 Énoncé

Écrire deux programmes prod.c et cons.c. Ils simulent le problème du producteur-consommateur. Le producteur lit stdin. Le consommateur affiche sur stdout. La mémoire partagée sera un tableau de 5 caractères.

2.4.2 Une solution

```c
/*
NOM      : cons.c
CLASSE   : IPC - LaboIPC 02-04
#OBJET  : réservé au makefile
AUTEUR   : MBA 11/2017
 */
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/types.h>
#include "prodcons.h"
int main(void)
{  int tete, vides,pleines,shm;
   unsigned char * t; // tableau partagé

   tete = 0;
   if ((shm = shmget(11,TAILLE,0777|IPC_CREAT)) < 0){perror ("shmget"); exit (201); }
   t=shmat(shm,0,0777);
   if ((pleines=semget(21,1,0666|IPC_CREAT))< 0) {perror ("semget pleines"); exit(202); }
   if ((vides=semget(31,1,0666|IPC_CREAT)) < 0) {perror ("semget pleines"); exit(203); }
   if (semctl(pleines,0,SETVAL,0)<0) {perror("semctl pleines"); exit(204); }
   if (semctl(vides,0,SETVAL,5)<0) {perror("semctl vides"); exit(205); }

   int status = 0;
   while (status != -1) // suppression du sémaphore ?
   {
      if ((status = opsem(pleines,-1)) != -1) { // down pleines

         write (1, t+tete, 1); // Afficher le caractère en tête
         tete = (tete + 1) % TAILLE; // chaise musicale
         status = opsem(vides,+1); // up vides, une nouvelle case vide
      }
   }
   shmdt(t);
   exit(0);
}
```

```c
/*
NOM      : prod.c
CLASSE   : IPC - LaboIPC 02-04
#OBJET  : réservé au makefile
AUTEUR   : MBA 11/2017
*/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/types.h>
#include "prodcons.h"
int main(void)
{  int queue, vide,plein,shm;
```

```
    unsigned char * t;
    unsigned char b;
    queue = 0;
    if ((shm = shmget(11,TAILLE,0777|IPC_CREAT)) < 0){perror ("shmget"); exit (201); }
    t=shmat(shm,0,0777);
    if ((plein=semget(21,1,0666|IPC_CREAT))< 0) {perror ("semget plein"); exit(202); }
    if ((vide=semget(31,1,0666|IPC_CREAT)) < 0) {perror ("semget plein"); exit(203); }
         if (semctl(plein,0,SETVAL,0)<0) {perror("semctl plein"); exit(204); }
         if (semctl(vide,0,SETVAL,5)<0) {perror("semctl vide"); exit(205); }
    // produire ce qu'on lit sur stdin
    while (read (0, &b,1) > 0)
    {
       opsem(vide,-1); // down vide
       t[queue]=b;
       queue = (queue + 1) % TAILLE;
       opsem(plein,+1);// up  plein
    }
         semctl(plein,0,IPC_RMID,0);
         semctl(vide,0,IPC_RMID,0);
    shmdt(t); // explicite
         shmctl(shm,IPC_RMID,0);
    exit(0);
}
```

2.4.3 Commentaires

- Au cas où ce programme ne donne pas les résultats espérés, commencer par vérifier que les sémaphores sont bien libres.
- Le sémaphore PLEIN signifie nombre de cases pleines, 0 au début
- Le sémaphore VIDE signifie nombre de cases libres, 5 au début
- La fonction opsem est celle définie au laboratoire 0202.

2.4.4 En roue Libre

- Généralisez ce programme au cas où il y aurait plusieurs consommateurs qui tournent en même temps. Il faut partager et protéger l'accès à la variable tete. Vous aurez besoin d'une section critique.
- La suppression des sémaphores est immédiate, elle provoque une erreur dans le consommateur.

Chapitre 3

Exercices

+ Ipc014 : Définissez une section critique qu'un seul process peut accéder à la fois. Écrire un process qui utilise cette section critique. Immédiatement après y être entré, et juste avant d'en sortir, il affiche '1'. Écrire un deuxième process qui utilise la même section critique. De la même façon, il affiche '2'. Quelle sera la séquence logique des caractères 1 et 2 qui vont s'afficher ?
+ Ipc029 : Écrivez 3 process. Synchronisez ces process de telle façon que le process 3 ne peut s'exécuter que quand les process 1 et 2 sont terminés. L'ordre de lancement des process est quelconque. Prouvez que votre synchronisation est correcte. Vous devez utiliser les sémaphores SystemV pour résoudre cet exercice.
+ Ipc031 : Soient deux processus ; le premier affichant des caractères A à raison de 1 par seconde, le deuxième affichant des caractères B à raison de 1 par seconde. Réalisez exactement l'alternance AA B AA B AA B AA B AA B AA B ... par synchronisation des deux processus.
+ Ipc032 : Un process crée une réserve de N cacahouètes. N est un nombre compris entre 1000 et 2000. 3 processus fils sont des mangeurs de cacahouètes (M à la fois). M est un nombre aléatoire compris entre 100 et 200. Les processus fils affichent sur 3 lignes "je suis le processus <pid> , je vole M cacahuètes, je pars". Les fils meurent quand il ne reste plus suffisament de cacahouètes dans la réserve. Le parent meurt quand les 3 fils sont morts. Les messages ne peuvent pas être mélangés et le nombre de cachuètes disponibles respecté. Vous ne pouvez pas utiliser de mémoire partagée dans cet exercice.
+ Ipc046 : Soient deux processus qui affichent respectivement les caractères A et B à répétition. Arrangez-vous pour obtenir un affichage alterné des deux lettres.
+ Ipc059 : Soient deux processus qui affichent respectivement `bonjour` et `le monde` à la réception du signal SIGUSR1. Arrangez-vous pour que la phrase `bonjour le monde` soit toujours écrite dans le bon ordre.