Support de cours

le cnam

EXERCICES File de Message

Joelle Delacroix

V1

19/08/2024

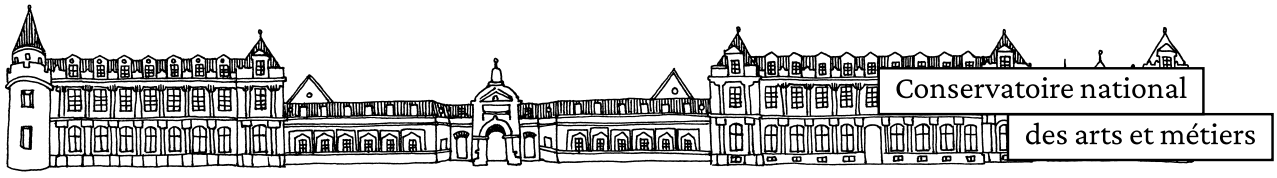

Table des matières

I Préambule

1. Rappels

Généralités sur les IPC

Rappel

Les IPC (*Inter Process Communication*) forment un groupe de trois outils de communication indépendants des tubes anonymes ou nommés, dans le sens où ils n'appartiennent pas au système de gestion de fichiers. Ces trois outils sont :

- les files de messages ou MSQ (*Messages Queues*) ;
- les régions de mémoire partagée ;
- les sémaphores

Ces trois outils sont gérés dans des tables du système, une par outils.

Un outil IPC est identifié de manière unique par un identifiant externe appelé la clé (qui a le même rôle que le chemin d'accès d'un fichier) et par un identifiant interne (qui joue le rôle de descripteur). Un outil IPC est accessible à tout processus connaissant l'identifiant interne de cet outil. La connaissance de cet identifiant s'obtient par héritage ou par une demande explicite au système au cours de laquelle le processus fournit l'identifiant externe de l'outil IPC.

la CLE

Rappel

La clé est une valeur numérique de type key_t. Les processus désirant utiliser un même outil IPC pour communiquer doivent se mettre d'accord sur la valeur de la clé référençant l'outil. Ceci peut être fait de deux manières :

- la valeur de la clé est figée dans le code de chacun des processus ;
- la valeur de la clé est calculée par le système à partir d'une référence commune à tous les processus. Cette référence est composée de deux parties, un nom de fichier et un entier. Le calcul de la valeur de la clé à partir cette référence est effectuée par la fonction ftok(), dont le prototype est :

```
1 #include <sys/ipc.h>
2 key_t ftok (const char *ref, int numero);
```

Commande ipcs

Complément

La commande ipcs permet de lister l'ensemble des outils IPC existants à un moment donné dans le système. Les informations fournies par cette commande sont notamment pour chaque outil IPC :

- le type (q pour messages queues, s pour sémaphores, m pour les régions de mémoire partagée) ;
- l'identification interne ;
- la valeur de la clé ;
- les droits d'accès définis ;

- le propriétaire ;
- le groupe propriétaire.

```
delacroix@vesuve:~> ipcs
------ Segments de mémoire partagée --------
touche      shmid  propriétaire perms     octets   nattch   statut
0x00000000 1015808    root      644        118784     3      dest
0x00000000 1114113    root      644        118784     2      dest
0x00000000 1146882    root      644        110592     2      dest
0x00000000 1179651    root      644        110592     2      dest
0x00000000 1212420    root      644        151552     4      dest
0x00000000 1245189    root      644        151552     4      dest

------ Tables de sémaphores --------
touche    semid   propriétaire perms      nsems
0x0000000c 0       delacroix     600        1

------ Files d'attente de messages --------
touche     msqid    propriétaire perms    octets utilisés messages
0x0000013a  0        delacroix    750      0            0

```

Commande ipcrm Complément

La suppression d'une file de messages peut également être réalisée depuis le prompt du shell par la commande :

```
ipcrm - q identifiant ou ipcrm -Q cle.
```

2. Les fonctions utiles pour les exercices

Accès à une file de message

L'accès à une file de message s'effectue par l'intermédiaire de la primitive msgget(). Cette primitive permet :

- la création d'une nouvelle file de messages ;
- l'accès à une file de messages déjà existante.

Le prototype de la fonction est :

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t cle, int option);
```

Le paramètre cle correspond à l'identification externe de la file de messages. Le paramètres options est une combinaison des constantes IPC_CREAT, IPC_EXCL et de droits d'accès.

La création d'une file de messages est demandée en positionnant les constantes IPC_CREAT et IPC_EXCL. Une nouvelle file est alors créée avec les droits d'accès définis dans le paramètre option. Le processus propriétaire de la file est le processus créateur tandis que le groupe propriétaire de la file est le groupe du processus créateur. Si ces deux constantes sont positionnées et qu'une file d'identifiant externe cle existe déjà, alors une erreur est générée. Si seule la constante IPC_CREAT est positionnée et qu'une file d'identifiant externe égal à cle existe déjà, alors l'accès à cette file est retourné au processus.

Ainsi l'exécution de msgget (cle, IPC_CREAT | IPC_EXCL | 0660) crée une nouvelle file avec des droits en lecture et écriture pour le processus propriétaire de la file et pour les processus du groupe.

Un processus désirant accéder à une file déjà existante effectue un appel à la primitive msgget() en positionnant à 0 le paramètre option.

Droits d'accès

Rappel

Les droits d'accès à un fichier définissent pour un ensemble de trois entités, les actions pouvant être réalisées par les membres de ces entités. Ces entités sont :

- le propriétaire du fichier ;
- le groupe auquel le propriétaire appartient ;
- les autres qui englobent tous les utilisateurs n'appartenant pas au groupe.

Trois types de permissions sont définis sur un fichier. Elles peuvent être notées de deux façons, soit par une lettre, soit par un chiffre en octal :

- la permission en lecture (r ou valeur 4) ;
- la permission en écriture (w ou valeur 2) ;
- la permission en exécution (x ou valeur 1).

Ainsi des droits 666 correspondent à des droits de lecture et écriture pour tout le monde (4 + 2 = 6). Des droits 755 correspondent à des droits de lecture,écriture et exécution pour le propriétaire et lecture avec exécution pour le groupe et les autres.

Envoi et réception de messages

La communication au travers d'une file de messages peut être bidirectionnelle, c'est-à-dire qu'un processus consommateur de messages dans la file peut devenir producteur de messages pour cette même file. La communication mise en œuvre est une communication de type boite aux lettres, préservant les structures des messages.

Chaque message comporte les données en elles-mêmes ainsi qu'un type qui permet de faire du multiplexage dans la file de messages et de désigner le destinataire d'un message.

Un message est toujours composé de deux parties :

- la première partie constitue le type du message. C'est un entier long positif ;
- la seconde partie est composée des données proprement dites.

Toutes les données composant le message doivent être contiguës en mémoire centrale. De ce fait, le type pointeur est interdit.

Un exemple de structure de messages est par exemple :

```
1 struct message {
2               long mtype;
3               int n1;
4               float fl1; };
```

L'envoi d'un message dans une file de messages s'effectue par le biais de la primitive msgsnd(). Le paramètre idint correspond à l'identifiant interne de la file. Le paramètre *msg est l'adresse du message en mémoire centrale tandis que le paramètre longueur correspond à la taille des données seules dans le message msg envoyé. La primitive renvoie 0 en cas de succès, – 1 sinon.

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd (int idint, const void *msg, int longueur, int option);
```

Le réception d'un message depuis une file de messages utilise la primitive msgrcv().

```c
include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv (int idint, const void *msg, int longueur, long letype, int option);
```

Le paramètre idint correspond à l'identifiant interne de la file. Le paramètre *msg est l'adresse d'une zone en mémoire centrale pour recevoir le message tandis que le paramètre longueur correspond à la taille des données seules dans le message msg envoyé.

Le paramètre letype permet de désigner un message à extraire, en fonction du champ type de celui-ci. Plus précisément :

- si letype est strictement positif, alors le message le plus ancien dont le type est égal à letype est extrait de la file ;
- si letype est nul, alors le message le plus ancien est extrait de la file. La file est alors gérée en FIFO ;
- si letype est négatif, alors le message le plus ancien dont le type est le plus petit inférieur ou égal à |letype| est extrait de la file. Ce mécanisme instaure des priorités entre les messages.

Par défaut, la primitive msgrcv() est bloquante c'est-à-dire qu'un processus est suspendu lors d'un retrait de messages si la file ne contient pas de messages correspondant au type attendu. La primitive renvoie la longueur du message prélevé en cas de succès, – 1 sinon.

Destruction d'une file de message

La destruction d'une file de messages s'effectue en utilisant la primitive msgctl() dont le paramètre operation est positionné à la valeur IPC_RMID. La valeur renvoyée est 0 en cas de succès et – 1 sinon.

```c
msgctl (int idint, IPC_RMID, NULL);
```

Obtenir la taille d'une chaine de caractères en C

Complément

La primitive **strlen()** permet d'obtenir la taille d'une chaine de caractères.

```c
#include <string.h>
size_t strlen( const char * theString );
```

Obtenir la taille d'une structure en C

Complément

L'opérateur **sizeof()** fournit la taille (en octets) du type ou de la variable qui suit.

Exemple

```c
#include <stdio.h>
#include <string.h>

struct message {
                long mtype;
                int n1;
```

```
                char st[20];
                float fl1; };

int main() {

struct message m1;
int taille;
char chaine1[20];


    printf ("Donnez une chaine :");
    scanf ("%s", m1.st);
    taille = strlen(m1.st);
    printf("longueur de la chaine  %d\n",taille);

    printf ("La taille de la structure est %d\n", sizeof(struct message));
    printf ("La taille de mtype est %d\n", sizeof(long));
    printf ("La taille de la structure en données est %d\n", sizeof(struct message)- sizeof(long));
 }

```

```
root@ubuntu:/media/sf_source# ./a.out

Donnez une chaine :bonjour

longueur de la chaine  7

La taille de la structure est 40

La taille de mtype est 8

La taille de la structure en données est 32

root@ubuntu:/media/sf_source#
```

Connaitre la taille en données de la structure requête ou réponse pour les primitives msgrcv et msgsnd

Méthode

Dans les primitives msgsnd et msgrcv évoquées ci dessus, le paramètre longueur correspond à la taille des données seules se trouvant dans la structutre envoyée ou reçue.

on calcule longueur comme suit par exemple :

```
int longueur;
longueur = sizeof(struct requete)-sizeof(la_requete.letype)
```

II Exercices corrigés

1. ENONCE EXERCICE 1

Ecrivez le programme permettant de réaliser la communication client-serveur suivante :

- Un processus client demande deux entiers à l'utilisateur et il les envoie à un processus serveur
- Le processus serveur calcule la somme des deux entiers et envoie le résultat au processus client.
- Le processus client affiche le résultat.

Ces processus communiquent via une MSQ. Plusieurs clients s'exécutent en même temps.

2. CORRIGE EXERCICE 1

Contexte

Dans cette application, le serveur doit uniquement consommer les requêtes depuis la file de messages. Les clients de leur côté doivent uniquement lire la réponse qui les concerne.

Pour parvenir à ce schéma, on utilise le champ type dans la structure des messages véhiculés par la file de messages. Ainsi, le serveur désigne le client auquel s'adresse la réponse en remplissant le champ type de la réponse avec la valeur du pid du client. Ce pid est indiqué par le client au niveau de sa requête. Le client quant à lui désigne le serveur comme destinataire de la requête en initialisant le champ type de sa requête avec une valeur positive qui ne peut pas correspondre à un pid de processus utilisateur, par exemple la valeur 1 (cette valeur 1 correspond obligatoirement au processus init).

Partie serveur

```
/*****************************************************************/
/*                 Processus serveur: crée la MSQ                */
/*    et additionne les deux nombres reçus dans chaque message    */
/*****************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>

#define CLE 314

struct requete {
                long letype;
                int nb1;
                int nb2;
                pid_t mon_pid;
               };
struct reponse {
                long letype;
                int res;
                };

main()
{
int msqid,l;
```

```
27 struct requete la_requete;
28 struct reponse la_reponse;
29 /*   allocation  MSQ */
30
31 if((msqid=msgget((key_t)CLE,0750|IPC_CREAT|IPC_EXCL))==-1)
32   {
33    perror("msgget");
34    exit(1);
35   }
36
37 while (1)
38 {
39 /* lecture d'une requête ; le serveur prélève les messages de type 1 */
40 if((l=msgrcv(msqid,&la_requete,sizeof(struct requete)-
   sizeof(la_requete.letype),1,0))==-1)
41   {
42    perror("msgrcv");
43    exit(2);
44   }
45
46 printf("les nombres reçus du client %d sont %d  %d\n", la_requete.mon_pid,
   la_requete.nb1,la_requete.nb2);
47
48 la_reponse.res = la_requete.nb1 + la_requete.nb2;
49 la_reponse.letype=la_requete.mon_pid;
50 /* type associé au message; le pid du client  */
51 if(msgsnd(msqid,&la_reponse,sizeof(struct reponse) -
   sizeof(la_reponse.letype),0)==-1)
52   {
53    perror("msgsnd");
54    exit(2);
55   }
56 }
57 exit(0);
58 }
59
```

(cf. serveurMSQexo1.c)

Partie client

```
1 /*****************************************************************/
2 /*      Processus client: envoi de deux nombres à additionner      */
3 /*****************************************************************/
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <sys/types.h>
7 #include <sys/ipc.h>
8 #include <sys/msg.h>
9
10 #define CLE 314
11
12 struct requete {
13                  long letype;
14                  int nb1;
15                  int nb2;
16                  pid_t mon_pid;
17                 };
18 struct reponse {
```

```
                long letype;
                int res;
                };
main()
{
int msqid, l, nb1, nb2;
struct requete la_requete;
struct reponse la_reponse;

/* récupération du msqid */
if((msqid=msgget((key_t)CLE,0))<0)
  {
   perror("msgget");
   exit(1);
  }

/* préparation de la requête et envoi */

printf ("Je suis le client %d Donnez moi deux nombres à additionner: ", getpid());
scanf ("%d %d", &nb1, &nb2);

la_requete.letype = 1;
la_requete.nb1 = nb1;
la_requete.nb2 = nb2;
la_requete.mon_pid = getpid();

if(msgsnd(msqid,&la_requete,sizeof(struct requete)-
sizeof(la_requete.letype),0)==-1)
  {
   perror("msgsnd");
   exit(2);
  }

/* réception de la réponse */
if((l=msgrcv(msqid,&la_reponse,sizeof(struct reponse)-
sizeof(la_reponse.letype),getpid(),0)==-1))
  {
   perror("msgrcv");
   exit(2);
  }

printf ("le resultat reçu est: %d\n", la_reponse.res);
exit(0);
}

```

(cf. clientMSQexo1.c)

Remarques sur l'exécution et traces Remarque

(cf. exo1 traces correction)

3. ENONCE EXERCICE 2

Ecrivez le programme permettant de réaliser la communication client-serveur suivante :

Deux serveurs Serveur_1 et Serveur_2 dialoguent avec les clients par l'intermédiaire d'une file de messages.

- Le serveur Serveur_1 effectue l'addition de deux nombres entiers envoyés par le client et renvoie le résultat de l'opération
- Le serveur Serveur_2 effectue la multiplication des trois nombres flottants envoyés par le client et renvoie le résultat de l'opération au client.

Chacun des serveurs ne doit prélever dans la file de messages que les requêtes qui le concerne. Un client doit évidemment recevoir le résultat correspondant à sa requête.

4. CORRIGE EXERCICE 2

Partie serveur addition

Il crée la file des messages et reçoit des messages de type 1

```
/*****************************************************************/
/*                  Processus serveur: crée la MSQ                */
/*     et additionne les deux nombres reçus dans chaque message    */
/*****************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>
#define CLE 314

struct requete {
                long letype;
                int nb1;
                int nb2;
                pid_t mon_pid;
               };
struct reponse {
                long letype;
                int res;
                };

main()
{
int msqid,l;
key_t cle;
struct requete la_requete;
struct reponse la_reponse;
/*   allocation  MSQ */

if((msqid=msgget((key_t)CLE,0750|IPC_CREAT|IPC_EXCL))==-1)
  {
   perror("msgget");
   exit(1);
  }

while (1)
```

```
{
/* lecture d'une requête */
if((l=msgrcv(msqid,&la_requete,sizeof(struct requete)-
sizeof(la_requete.letype),1,0))==-1)
  {
   perror("msgrcv");
   exit(2);
  }

printf(" SERVEUR ADD : les nombres reçus du client %d sont %d  %d\n",
la_requete.mon_pid, la_requete.nb1,la_requete.nb2);

la_reponse.res = la_requete.nb1 + la_requete.nb2;
la_reponse.letype=la_requete.mon_pid;
/* type associé au message; le pid du client  */
if(msgsnd(msqid,&la_reponse,sizeof(struct reponse) -
sizeof(la_reponse.letype),0)==-1)
  {
   perror("msgsnd");
   exit(2);
  }
}
exit(0);
}

```

(cf. serveurMSQadditionexo2.c)

Partie serveur multiplication

Il récupère la file des messages et reçoit des messages de type 2

```
/*****************************************************************/
/*          Processus serveur: multiplie les trois nombres          */
/*                      reçus dans chaque message                    */
/*****************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdlib.h>

#define CLE 314
struct requetem {
                 long letype;
                 float nb1;
                 float nb2;
                 float nb3;
                 pid_t mon_pid;
                };
struct reponsem {
                 long letype;
                 float res;
                 };

main()
{
int msqid,l;
key_t cle;
```

```
28 struct requetem la_requete;
29 struct reponsem la_reponse;
30
31 /*  récupération  MSQ */
32
33 if((msqid=msgget((key_t)CLE,0))==-1)
34   {
35    perror("msgget");
36    exit(1);
37   }
38 while (1)
39 {
40 /* lecture d'une requête */
41 if((l=msgrcv(msqid,&la_requete,sizeof(struct requetem)-
   sizeof(la_requete.letype),2,0))==-1)
42   {
43    perror("msgrcv");
44    exit(2);
45   }
46
47 printf(" SERVEUR MUL : les nombres reçus du client %d sont %f %f  %f\n",
   la_requete.mon_pid, la_requete.nb1,la_requete.nb2, la_requete.nb3);
48
49 la_reponse.res = la_requete.nb1 * la_requete.nb2 * la_requete.nb3;
50
51
52 la_reponse.letype=la_requete.mon_pid;
53 /* type associé au message; le pid du client  */
54 if(msgsnd(msqid,&la_reponse,sizeof(struct reponsem) -
   sizeof(la_reponse.letype),0)==-1)
55   {
56    perror("msgsnd");
57    exit(2);
58   }
59 }
60 exit(0);
61 }
62
```

(cf. serveurMSQmulexo2.c)

Partie client

Le client récupère la file de message. Il envoie des messages de type 1 au processus serveur addition et des messages de type 2 au processeur serveur multiplication. Chaque client indique son pid dans la requête.

```
1 /*****************************************************************/
2 /*      Processus client: envoi de deux entiers à additionner      */
3 /*                 ou de trois flottants à multiplier                */
4 /*****************************************************************/
5 #include <stdio.h>
6 #include <sys/types.h>
7 #include <sys/ipc.h>
8 #include <sys/msg.h>
9 #include <stdlib.h>
10
11 #define CLE 314
12
13 struct requete {
```

```
                long letype;
                int nb1;
                int nb2;
                pid_t mon_pid;
               };
struct reponse {
                long letype;
                int res;
                };
struct requetem {
                long letype;
                float nb1;
                float nb2;
                float nb3;
                pid_t mon_pid;
               };
struct reponsem {
                long letype;
                float res;
                };


main()
{
int msqid, l, nb1, nb2, choix;
key_t cle;
float fb1, fb2, fb3;
struct requete la_requete_add;
struct reponse la_reponse_add;
struct requetem la_requete_mul;
struct reponsem la_reponse_mul;

/* récupération du msqid */

if((msqid=msgget((key_t)CLE,0)) < 0)
  {
   perror("msgget");
   exit(1);
  }

printf ("Quel service desirez-vous? Addition (1), Multiplication (2)?\n");
scanf ("%d",&choix);
if (choix == 1) {
  /* préparation de la requête et envoi */
printf ("Je suis le client %d Donnez moi deux nombres à additionner: ", getpid());
  scanf ("%d %d", &nb1, &nb2);

  la_requete_add.letype = 1;
  la_requete_add.nb1 = nb1;
  la_requete_add.nb2 = nb2;
  la_requete_add.mon_pid = getpid();

if(msgsnd(msqid,&la_requete_add,sizeof(struct requete)-
sizeof(la_requete_add.letype),0)==-1)
  {
   perror("msgsnd");
   exit(2);
```

```
70    }
71
72  /* réception de la réponse */
73  if((l=msgrcv(msqid,&la_reponse_add,sizeof(struct reponse)-
    sizeof(la_reponse_add.letype),getpid(),0)==-1))
74    {
75     perror("msgrcv");
76     exit(3);
77    }
78  printf ("le resultat reçu est: %d\n", la_reponse_add.res);
79   }
80   else
81   {
82    /* préparation de la requête et envoi */
83    printf ("Je suis le client %d Donnez moi trois nombres à multiplier:", getpid());
84    scanf ("%f %f %f", &fb1, &fb2, &fb3);
85    la_requete_mul.letype = 2;
86    la_requete_mul.nb1 = fb1;
87    la_requete_mul.nb2 = fb2;
88    la_requete_mul.nb3 = fb3;
89    la_requete_mul.mon_pid = getpid();
90
91  if(msgsnd(msqid,&la_requete_mul,sizeof(struct requetem)-
    sizeof(la_requete_mul.letype),0)==-1)
92    {
93     perror("msgsnd");
94     exit(2);
95    }
96  /* réception de la réponse */
97  if((l=msgrcv(msqid,&la_reponse_mul,sizeof(struct reponsem)-
    sizeof(la_reponse_mul.letype),getpid(),0)==-1))
98    {
99     perror("msgrcv");
100    exit(3);
101   }
102 printf ("le resultat reçu est: %f\n", la_reponse_mul.res);
103 }
104 }
105
```

(cf. clientMSQexo2.c)

Remarques sur l'exécution et traces

Remarque

(cf. exo2 traces correction)