TP n°6 : Utilisation des IPC Unix

1 Présentation

Les trois mécanismes ainsi désignés ont un certain nombre de propriétés en commun :

- à chacun des trois mécanismes est affecté une table de taille fixe.
- Dans chacune des tables, toute entrée active est associée à une clé numérique choisie par l'utilisateur, et servant de nom d'identification global.
- Un appel système `...get` permet de créer une nouvelle entrée ou d'accéder à une entrée existante. La fonction renvoie une référence devant être utilisée dans les autres appels système. La clef `IPC_PRIVATE` permet d'obtenir une référence privée vers une nouvelle entrée non utilisée. Des drapeaux permettent de modifier le comportement :
  - ▷ `IPC_CREAT` : permet de créer une nouvelle entrée s'il n'en existe pas déjà une pour cette clef.
  - ▷ `IPC_EXCL` : lorsque ce drapeau est combiné avec le précédent, une erreur est générée si l'entrée existe déjà.
- Un appel système `...ctl` permet d'interroger ou modifier les paramètres d'une entrée, ainsi que de la supprimer.
- Chaque entrée comporte des droits d'accès similaires à ceux des fichiers (n° utilisateur et n° de groupe du propriétaire, droits de lecture écriture).
- Lorsqu'une entrée est supprimée, le système attribue une référence différente pour la prochaine utilisation de cette entrée (par exemple ancienne référence + taille de la table). L'utilisation d'une référence périmée a donc toutes les chances de provoquer une erreur (sauf après un cycle complet !).

Les commandes `ipcs` [1] et `ipcrm` [2] permettent respectivement l'affichage des IPC actifs et la destruction de l'un des IPC (si vous avez les droits nécessaires). Bien entendu, une aide plus complète peut être obtenue sur chaque fonction en utilisant le manuel UNIX.

> **Note :** Si vous avez du mal à trouver une (ou plusieurs) clés d'identification, vous pouvez vous retourner vers la fonction `ftok` [a] qui construit une clé à partir d'un nom de fichier (par exemple votre "home directory").
>
> a. https ://man7.org/linux/man-pages/man3/ftok.3.html

2 Mémoires partagées

2.1 Les principales fonctions

Cette méthode évite de faire des copies de/vers l'espace système pour transférer de l'information. En effet, plusieurs processus voient la même zone mémoire dans leurs espaces virtuels respectifs et cette mémoire leur est accessible comme de la mémoire ordinaire : c'est donc à eux de gérer les problèmes des accès concurrents. Cette méthode est particulièrement recommandée dans la communication de gros volumes de données (images par exemple).

> **Note :** La mémoire reste allouée même lorsque tous les processus qui y accèdent se terminent : le même problème que pour les autres IPC, la mémoire partagée doit donc être libérée explicitement.

Les primitives pour gérer la mémoire partagée sont :

1. https ://man7.org/linux/man-pages/man1/ipcs.1.html
2. https ://man7.org/linux/man-pages/man1/ipcrm.1.html

- `shmget` [3]

  Permet la création d'une zone mémoire partagée, ou l'accès à une zone déjà existante. Les drapeaux permettent de spécifier les modalités d'ouverture, et notamment les droits de lecture et d'écriture.

- `shmat` [4]

  Permet d'attacher la mémoire référencée à l'adresse virtuelle spécifiée en argument. Le résultat est l'adresse à laquelle le système a effectivement réalisé l'attachement (elle peut être différente de l'argument).

- `shmdt` [5]

  Détachement d'une zone partagée de l'espace virtuel du processus.

- `shmctl` [6]

  Permet de consulter ou de modifier les caractéristiques d'un segment mémoire ainsi que de le supprimer. Il est notamment possible de demander son verrouillage en mémoire centrale.

2.2 Un exemple

Ce petit exemple va simplement utiliser (ou créer la première fois) une zone partagée pour l'afficher et la modifier. Ce programme doit être utilisé plusieurs fois pour faire apparaître l'unicité de cette zone mémoire. Bien entendu, les commandes `ipcs` [7] et `ipcrm` [8] sont utilisables.

```c
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define KEY               0x00012347

typedef struct {
    int value;
} COUNTER;

int main(void) {
    /* création ou lien avec la zone partagée */
    int memid = shmget(KEY, sizeof(COUNTER), 0700 | IPC_CREAT);
    assert(memid >= 0);

    /* montage en mémoire */
    COUNTER *c = shmat(memid, NULL, 0);
    printf("compteur␣=␣%d\n", c->value++ );

    return (EXIT_SUCCESS);
}
```

⏩ **Travail à faire :** testez cet exemple et explorez la carte mémoire de ce processus.

3. https ://man7.org/linux/man-pages/man2/shmget.2.html
4. https ://man7.org/linux/man-pages/man2/shmat.2.html
5. https ://man7.org/linux/man-pages/man2/shmdt.2.html
6. https ://man7.org/linux/man-pages/man2/shmctl.2.html
7. https ://man7.org/linux/man-pages/man1/ipcs.1.html
8. https ://man7.org/linux/man-pages/man1/ipcrm.1.html

2.3 Partager un tampon en mémoire

Nous allons utiliser la mémoire partagée IPC pour inplanter un petit exemple de producteur / consommateur. La zone partagée pourrait être du type suivant :

```
struct {
    int the_char;       /* '\0' si vide         */
    int end;            /* signal de fin d'entrée */
} ...
```

Le consommateur peut consommer si, et seulement si `(the_char != 0)`. Le producteur peut produire ssi `(the_char == 0)`. Je vous propose de suivre les étapes ci-dessous :

⏩ Travail à faire :

1. Créez un programme C pour le producteur (en partant de l'exemple précédent).
2. Créez un programme C pour le consommateur (en partant de l'exemple précédent).
3. Prévoyez un fichier `tampon.h` qui contient la définition du tampon.
4. Le producteur lit des caractères au clavier et les place dans le tampon. Le consommateur les récupère et les affiche. Ces deux acteurs appliquent la méthode de l'attente active pour se synchroniser.
5. Faites en sorte que la zone partagée soit créée et initialisée par le producteur. Si le consommateur est plus rapide, il doit attendre que la zone partagée soit disponible.
6. Prenez soin de ralentir le producteur et le consommateur pour tester tous les cas de figure.
7. La fin de la lecture au clavier est indiquée au consommateur par la mise à un du drapeau `end`. Ce dernier a la charge de supprimer la ressource partagée avant de se terminer.
8. Consultez la carte mémoire de ces processus.

3 Envoi et réception de messages

3.1 Les principales fonctions

Elle se fait par l'utilisation de quatre primitives permettant d'opérer sur une des queues de message :

- `msgid = msgget(key,flags)`

  Pour la création ou l'accès à une queue de message de nom `key`. Cette fonction renvoie un entier comme référence, que nous appellerons `msgid`.

- `int msgsnd(msgid,msgbuf,size,flags)`

  Écriture de messages. Elle se fait par copie vers une zone de données du système. Pour envoyer ou recevoir un message, le processus appelant alloue une structure comme celle-ci :

  ```
  struct msgbuf {
     long mtype;    /* type de message ( > 0 ) */
     char mtext[1]; /* contenu du message   */
  };
  ```

  avec une table `mtext` de taille `size`, valeur entière non-négative. Le membre `mtype` doit avoir une valeur strictement positive qui puisse être utilisée par le processus lecteur pour la sélection de messages (voir plus bas).

- `int msgrcv(msgid,msgbuf,maxsize,type,flags)`

  Lecture de messages. Elle se fait par copie depuis une zone de données du système. Si le type est 0, le

premier message de la queue est renvoyé, s'il est positif, le premier message de même type est renvoyé.

- `int msgctl(msgid,command,buffer)`

  Permet de consulter, modifier ou supprimer une queue de messages.

> **Note :** Une file de messages, même privée, doit être détruite explicitement. Elle n'est pas supprimée lors de la mort du processus créateur.

3.2 Un exemple

Commencons pas définir (dans `message.h`) les structures de données communes au serveur et aux clients.

Fichier `message.h`

```
#define REQUEST_KEY       0x00012345
#define LG_MAX            512
#define PID_SERVER        1

typedef struct {
    pid_t pid;
    char mtext[ LG_MAX ];
} DATA;

typedef struct {
    long mtype;
    DATA data;
} MESSAGE;
```

Dans cet exemple, chaque client met son numéro de processus (fonction `getpid`[9]) dans le champ `pid` des messages de requête ce qui permet au serveur de l'identifier. Le serveur copie ce champ dans la réponse, ce qui permet au client de récupérer dans la queue des réponses les seuls messages qui le concernent.

9. https ://man7.org/linux/man-pages/man2/getpid.2.html

Codage du client

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

#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#include "message.h"

int main(void) {
    MESSAGE msg;
    int my_pid = getpid();

    /* se connecter aux IPC de requête et de réponse */
    int requests = msgget(REQUEST_KEY, 0700 | IPC_CREAT);
    assert(requests >= 0);

    /* demander un message à l'utilisateur */
    printf("Enter message : ");
    fgets(msg.data.mtext, LG_MAX, stdin);

    /* envoyer la requête signée par son numéro de processus */
    msg.mtype = PID_SERVER; // pour le serveur
    msg.data.pid = my_pid; // pour donner au serveur mon PID
    int res = msgsnd(requests, & msg, sizeof(DATA), 0);
    assert(res >= 0);

    /* récupérer sa réponse signée par son numéro de processus */
    res = msgrcv(requests, & msg, sizeof(DATA), getpid(), 0);
    assert(res >= 0);

    printf("result : %s", msg.data.mtext);
    return (EXIT_SUCCESS);
}
```

Et voilà le serveur :

Codage du serveur

```c
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <assert.h>

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#include "message.h"

int main(void) {
    MESSAGE msg;

    /* Se connecter aux IPC de requête et de réponse */
    int requests = msgget(REQUEST_KEY, 0700 | IPC_CREAT);
    assert(requests >= 0);

    for(int counter = 0; ; counter++) {
        printf("Waiting a request (%d processed)...\n", counter);
        int res = msgrcv(requests, & msg, sizeof(DATA), PID_SERVER, 0);
        assert(res >= 0);

        /* Traiter la requête (passage en majuscule) */
        for(int i=0; i < strlen(msg.data.mtext); i++) {
            msg.data.mtext[i] = toupper(msg.data.mtext[i]);
        }

        /* Envoyer la réponse (signée pour le client) */
        printf("send '%s' to %d\n", msg.data.mtext, msg.data.pid);
        msg.mtype = msg.data.pid;
        res = msgsnd(requests, & msg, sizeof(DATA), 0);
        assert(res >= 0);
    }

    return (EXIT_SUCCESS);
}
```

Note : les structures partagées ne sont jamais détruites automatiquement. Après avoir testé cet exemple, vous pouvez le vérifier en utilisant la commande `ipcs` [a] qui liste les ressources partagées actives. Vous pouvez également faire le ménage avec `ipcrm` [b].

a. https ://man7.org/linux/man-pages/man1/ipcs.1.html
b. https ://man7.org/linux/man-pages/man1/ipcrm.1.html

Travail à faire : Lancez un serveur et plusieurs clients et vérifiez que chaque client récupère les bonnes données (inspirez-vous du script bash ci-dessous pour lancer plusieurs clients).

```
#/bin/bash
for((c=1; c<100; c++)); do
    ./votre_client <<< "Hello"
done
exit 0
```

⏩ **Travail à faire :** Refaites le même exercice (producteur / consommateur) mais en remplaçant la mémoire partagée par un envoi de message entre le producteur et le consommateur.

4 Montage de fichiers en mémoire

La fonction `mmap` [10] permet de projeter un fichier en mémoire (plus de précisions en cliquant sur le lien).

Un exemple d'utilisation de `mmap`

```c
#include <sys/types.h>
#include <sys/mman.h>
#include <fcntl.h>

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>


void create_file(int size) {
    FILE *f = fopen("data", "w");
    for(int i=1; i<size; i++) {
        fputc('A', f);
    }
    fputc('\n', f);
    fclose(f);
}


int main() {
    int size = 60;

    /* création du fichier pour l'exemple */
    create_file(size);

    /* ouverture en utilisant les fonctions UNIX */
    int file = open("data", O_RDWR, 0700);
    assert(file > 0);

    /* montage en mémoire */
    char *zone = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED, file, 0);
    assert(zone != NULL);

    /* utilisation */
    printf("Affichage␣du␣contenu\n");
    for(int i=0; (i<size); i++) {
        putchar(zone[i]);
    }

    return (EXIT_SUCCESS);
}
```

10. https ://man7.org/linux/man-pages/man2/mmap.2.html

⏩ Travail à faire :

- Exécutez l'exemple et étudiez son fonctionnement.
- Modifiez un des caractères en vérifiant la mise à jour automatique du fichier sur disque.
- Alors que votre processus a déjà projeté le fichier (freinez le avec un `getchar` [a]), modifiez le fichier d'origine (avec un éditeur de texte) et vérifiez que la modification est bien reportée dans la version projetée.
- Consultez la carte mémoire de ce processus.

a. https ://man7.org/linux/man-pages/man3/getchar.3.html