Les IPC SYSTEM V

I. Introduction

I. 1. Généralités

- Première version du SYSTEM V
- Mécanismes de communication entre processus locaux
- Principales caractéristiques :
  - Extérieurs au SGF
  - C'est le système qui gère ces objets ➔ une table des sémaphores, une table pour les segments de mémoire partagée et une table pour les files de messages
  - Chaque objet a un identifiant interne (nombre >= 0) qui permet son utilisation par un processus. Cet identifiant s'obtient par héritage ou demande explicite au système.
  - D'un point de vue externe, ces objets sont identifiés par un mécanisme de clé (~ à la référence d'un fichier). Le type de clé est `key_t` et est prédéfini dans le fichier `<sys/types.h>`

I. 2. Fichier standard `<sys/ipc.h>`

Définition de structures et macro-définition de constantes utilisées pour la manipulation des objets des trois types.

Constante	Rôle	Primitive
`IPC_PRIVATE`	Clé privée : création d'un objet dont l'identité ne peut pas être demandée ultérieurement au système	`*get`
`IPC_CREAT`	Créer l'objet s'il n'existe pas	`*get`
`IPC_EXCL`	Utilisée avec `IPC_CREAT`. Message d'erreur si l'objet existe déjà	`*get`
`IPC_NOWAIT`	Opération non bloquante	`semop, msgrcv, msgsnd`
`IPC_RMID`	Suppression d'identification	`*ctl`
`IPC_STAT`	Extraction des caractéristiques	`*ctl`
`IPC_SET`	Modification de caractéristiques	`*ctl`

Définition de la structure `ipc_perm` :

```
struct ipc_perm
  {
   uid_t         uid   ;     /*identification du propriétaire*/
   gid_t         gid ;       /*identification du groupe propriétaire*/
   uid_t         cuid ;      /*identification du créateur*/
   gid_t         cgid ;      /*identification du groupe créateur*/
   unsigned short  mode ;    /*droits d'accès*/
   unsigned short  _seq ;    /*nombre d'utilisations de l'entrée*/
   key_t         key ;       /* clé */
  };
```

Remarques :
- l'identification interne d'un objet est calculée par le système à partir de l'indice dans la table de l'entrée associée à l'objet et du nombre de fois que cette entrée a été utilisée (champ `seq` de la structure `ipc_perm`) → `seq*constante+indice`
- Les droits d'accès sont construits selon le même schéma que pour les fichiers avec deux types d'accès : consultation/lecture et modification/écriture.

I. 3. Gestion des clés

Utilisation de la fonction `ftok` pour la construction des clés des objets manipulés. Cette fonction relie l'espace de nommage des objets IPC au SGF : une clé unique est associée à une référence de fichiers.

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

Cet appel génère une clé unique à partir de `ref` et de `num`. Cette clé pourra être utilisée avec une file de messages, un segment de mémoire partagée ou un ensemble de sémaphores. La fonction utilise en fait l'identité interne du fichier, c'est-à-dire le numéro de disque logique et d'index.

Conséquences :
- la référence doit correspondre à un fichier existant
- la fonction retourne un résultat différent après le déplacement du fichier

Exemple :
```
#include<sys/types.h>
#include<sys/stat.h>
#include<sys/ipc.h>

int main (int argc, char *argv[])
{
 printf("Première clé : %x\n", (int)ftok(argv[1], 0)) ;
 printf("Deuxième clé : %x\n", (int)ftok(argv[1], 1)) ;
}
```

On doit passer en premier paramètre à la fonction `main` un nom de fichier.

I. 4. Les commandes shell `ipcs` et `ipcrm`

Commande `ipcs` : permet la consultation des tables d'IPC du système. Principales informations fournies :
- **T** : le type de l'objet (**q** pour les files de messages, **m** pour les segments de mémoire partagée et **s** pour les ensembles de sémaphores)
- **ID** : identification interne de l'objet
- **KEY** : clé de l'objet sous forme hexadécimale
- **MODE** : les droits d'accès à l'objet
- **OWNER** : identité du propriétaire de l'objet
- **GROUP** : identité du groupe propriétaire de l'objet

Commande `ipcrm` : demande la suppression d'une entrée dans l'une des tables. L'entrée est désignée soit par la clé correspondante soit par son identification interne.

II. Les files de messages

- listes chaînées gérées par le noyau, ensemble de messages
- les informations échangées sont des paquets identifiables → une opération de lecture extrait exactement le produit d'une opération d'écriture

II. 1. Le fichier <sys/msg.h>

Constantes symboliques :

Constante	Interprétation
MSG_NOERROR	Un message trop long est tronqué et son extraction n'entraîne pas d'erreur
MSG_R	Autorisation de lire dans la file
MSG_W	Autorisation d'écrire dans la file
MSG_RWAIT	Indication qu'un processus est bloqué en lecture
MSG_WWAIT	Indication qu'un processus est bloqué en écriture

Structure `msqid_ds`
Correspond à la structure d'une entrée dans la table des files de messages. L'accès à une telle structure est réalisé par la primitive `msgctl`.

```
struct msqid_ds
  {
     struct ipc_perm    msg_perm;   /*droits d'accès à l'objet*/
     struct _ _msg      *msg_first; /*pointeur sur le premier message*/
     struct _ _msg      *msg_last;  /*pointeur sur le dernier message*/
     unsigned short int msg_qnum;   /*nombre de messages dans la file*/
     unsigned short int msg_bytes;  /*nombre maximum d'octets*/
     pid_t              msg_lspid;  /*pid du dernier processus émetteur*/
     pid_t              msg_lrpid;  /*pid du dernier processus récepteur*/
     time_t             msg_stime;  /*date de dernière émission (msgsnd)*/
     time_t             msg_rtime;  /*date de dernière réception (msgrcv)*/
     time_t             msg_ctime;  /*date de dernier changement (msgctl)*/
     unsigned short int msg_cbytes; /*nombre total actuel d'octets*/
  } ;
```

Toute file de messages est définie par une structure de type `msqid_ds` qui est allouée et initialisée lors de sa création.

En général, un message est composé d'un entier long qui représente son type et d'un tableau de caractères qui représente le corps du message. Le type d'un message permet de déterminer la nature de l'information constituant le corps du message. Il permet également le multiplexage entre processus.

Structure générique d'un message

```
struct msgbuf
  {
     long mtype ;      /*type du message*/
     char mtext[1] ;   /*texte du message*/
  } ;
```

Les utilisateurs définissent leur propre structure de message sur le modèle de `msgbuf`. Toute structure de message doit obligatoirement contenir un champ de type long qui joue le rôle de type du message (permet le multiplexage). Ce champ est strictement positif.
La définition de la structure d'un message ne peut pas contenir des indirections (doit être contiguë en mémoire).

Exemples :

1. `struct msgbuf1 {long mtype ; float n1, int tab[4] ;} ;` est valide.
2. `struct msgbuf2 {long mtype; char *p;} ;` n'est pas valide.

Structure `_ _ msg`
Correspond à un message.

```
struct _ _ msg
  {
     struct _ _ msg     *msg_next ; /*pointeur sur le message suivant*/
     long               msg_type ;  /*type du message*/
     unsigned short int msg_ts ;    /*taille du texte du message*/
     long               msg_spot ;  /*adresse du texte du message*/
  } ;
```

La structure `msginfo` : contient différents champs qui correspondent à des valeurs limites imposées par le système.

II. 2. Obtention de l'identification d'une file : la fonction `msgget`

- permet la création d'une nouvelle file ou la recherche de l'identification d'une file existante à partir de sa clé.

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

Retour de l'appel :

- identifiant d'une file de messages, créée à partir du paramètre `cle`. Cet identifiant permet à plusieurs processus d'utiliser la même file.
- -1 en cas d'erreur

Paramètre `cle` : clé caractérisant la file désirée (résultat de `ftok`) ou IPC_PRIVATE
Paramètre `option` : combinaison (ou bit à bit |) des constantes IPC_CREAT, IPC_EXCL et des droits d'accès (voir man de `msgget` pour les détails).

Constante	Rôle
IPC_PRIVATE	Clé privée : création d'un objet dont l'identité ne peut pas être demandée ultérieurement au système
IPC_CREAT	Créer l'objet s'il n'existe pas
IPC_EXCL	Utilisée avec IPC_CREAT. Message d'erreur si l'objet existe déjà

Déroulement de l'appel :

si `cle = IPC_PRIVATE`, une nouvelle file est créée ;

sinon, **si** `cle` ne correspond pas à une file existante alors

si l'indicateur `IPC_CREAT` est positionné dans le paramètre `option`, une nouvelle file est créée et est associée à cette clé. Les caractéristiques de cette file sont : droits définis dans le paramètre `option`, propriétaire et créateur correspondent au propriétaire effectif du processus, groupes propriétaire et créateur correspondent au groupe propriétaire effectif du processus

sinon, une erreur est détectée

si `cle ≠ IPC_PRIVATE` et correspond à une file existante, alors

si les indicateurs `IPC_CREAT` et `IPC_EXCL` ne sont pas tous les deux positionnés dans le paramètre `option`, la fonction renvoie l'identification de la file,

si ces deux indicateurs sont tous deux positionnés dans le paramètre `option`, une erreur est envoyée.

Exemples :

```
file = msgget(IPC_PRIVATE, 0x600) ;
```
Obtention d'une file réservée au processus appelant et à ses descendants.

```
ma_cle = ftok(argv[0], 0) ;
file = msgget (ma_cle, IPC_CREAT | 0x660) ;
```
Accès à une file permettant de dialoguer avec d'autres processus d'un même ensemble d'applications.

```
ma_cle = ftok(argv[0], 0) ;
file = msgget(ma_cle, IPC_CREAT | IPC_EXCL | 0x622) ;
```
Création d'une nouvelle file. On autorise tous les utilisateurs du système à envoyer un message sur la file, mais seul le créateur de la file peut les lire.

```
ma_cle = ftok(fichier_de_creation, 0) ;
file = msgget(ma_cle, 0) ;
```
Le processus doit uniquement utiliser une file créée par un autre.

III. 3. L'envoi de message : la fonction `msgsnd`

```
#include <sys/msg.h>
int msgsnd(int msgid, const void *p_msg, int lg, int option) ;
```

Envoie dans la file d'identification `msgid` le message pointé par `p_msg`. La longueur du texte du message est indiquée par le paramètre `lg` (les octets occupés par le type ne sont pas comptabilisés).

Remarques *:*

- La structure du paramètre `p_msg` est calquée sur celle de `msgbuf`.
- Le paramètre `option` est égal à `0` en général, ou `IPC_NOWAIT` pour un appel non bloquant dans le cas où la file est pleine.

Appel bloquant par défaut. Il est alors interruptible : si un signal capté arrive au processus ou si la file de messages est supprimée alors qu'un processus est bloqué sur émission d'un message, le processus est réveillé et la primitive renvoie –1.

Retour : 0 en cas de succès, -1 en cas d'échec.

II. 4. Réception d'un message : la fonction `msgrcv`

```
#include <sys/msg.h>
int msgrcv(int msgid, void *p_msg, int taille, long type,
int option) ;
```

Retour :

- longueur du texte du message écrit à l'adresse `p_msg` en cas de succès. Cette longueur doit être ≤ paramètre `taille` et ne comprend pas le nombre d'octets occupés par le type
- -1 en cas d'échec

- demande de lecture dans la file `msgid`
- `p_msg` pointe sur une zone mémoire susceptible de recevoir un message de longueur ≤ `taille` (longueur du texte, sans le type du message).
- Paramètre `option` : combinaison des constantes `IPC_NOWAIT` (appel non bloquant dans le cas où la file ne contient pas de message du type spécifié), `MSG_NOERROR` (si la longueur du message est > taille, le message est extrait et tronqué, par défaut on a un message d'erreur) et `MSG_EXCEPT` (réclame un message de n'importe quel type différent de celui indiqué en paramètre de la fonction, strictement positif dans ce cas).

Paramètre `type` (type du message à extraire) :

- si `type > 0` : le message le plus ancien de type égal à `type` est extrait de la file,
- Si `type = 0`, le message le plus ancien est extrait (quel que soit son type),
- Si `type < 0`, le message le plus ancien dont le type est le plus petit entier ≤ à `|type|` est extrait (définition de priorités entre les messages).

II. 5. Lecture/modification des caractéristiques d'une file : la fonction `msgctl`

```
#include <sys/msg.h>
int msgctl(int msgid, int op, …/* strucut msqid_ds *p_msqid */) ;
```

Lecture et/ou modification des caractéristiques de la file d'identification `msgid` contenues dans l'entrée de la table des files du système associée à cette file.

Valeurs possibles du paramètre `op` :

- `IPC_STAT` : l'entrée de la table associée à la file d'identification `msgid` est écrite à l'adresse `p_msqid`,
- `IPC_SET` : l'objet pointé par `p_msqid` est écrit dans l'entrée de la table des files de message correspondant à `msgid`. Les seuls champs que l'on peut modifier de cette manière sont `msg_perm.uid`, `msg_perm.gid` et `msg_perm.mode`.
- `IPC_RMID` : la file de messages est supprimée.

Retour :

- 0 en cas de succès
- -1 en cas d'échec

III. Les ensembles de sémaphores

- mécanisme de synchronisation de processus
- permettent d'obtenir l'accès en exclusion mutuelle à une ressource (empêcher plusieurs processus d'accéder en même temps à une ressource)
- sémaphore `S` = variable entière à valeurs non négatives utilisée à travers deux opérations :
  - `P(S) : si S = 0 alors mettre le processus en attente sinon S ← S – 1`
  - `V(S) : s ← S + 1 ; réveiller un ou plusieurs processus en attente`

III. 1. Le fichier `<sys/sem.h>`

La structure `__sem`
Correspond à la structure dans le noyau d'un sémaphore individuel.

```
struct __sem
  {
     unsigned short int semval ;    /*valeur du sémaphore*/
     unsigned short int sempid ;    /*pid du dernier processus utilisateur*/
     unsigned short int semncnt ;   /*nombre de processus attendant
                                    l'augmentation du sémaphore */
     unsigned short int semzcnt ;   /*nombre de processus attendant la
                                    nullité du sémaphore */

  } ;
```

La structure `semid_ds`
Elle correspond à la structure d'une entrée dans la table des sémaphores accessible par la primitive `semctl`.

```
struct semid_ds
  {
     struct ipc_perm   sem_perm ; /*structure décrivant les droits*/
     struct _ _ sem    *sem_base ; /*pointeur sur premier sem de
                                    l'ensemble*/
     time_t            sem_otime ; /*date de dernière opération*/
     time_t            sem_ctime ; /*date de dernier changement par
                                    semctl*/
     unsigned short int sem_nsems ; /*nombre de sémaphores de l'ensemble*/
  } ;
```

La structure `sembuf`
Correspond à une opération sur un sémaphore.

```
struct sembuf
  {
     unsigned short int     sem_num ;  /*numéro de sémaphore*/
     short                  sem_op ;   /*opération sur le sémaphore*/
     short                  sem_flg ;  /*option*/
  } ;
```

Remarques :
- les numéros de sémaphores dans l'ensemble commencent à 0 ;
- c'est le signe du champ `sem_op` d'un objet qui détermine l'opération (négatif → opération `Pn`, positif → opération `Vn` et nul → opération `Z`)

Autres structures
Correspondent à des objets manipulés par le système :

- `sem_undo` pour annulation automatique d'opérations antérieures en cas de terminaison
- `seminfo` contient les valeurs limites imposées par le système

III. 2. Obtention de l'identité d'un ensemble de sémaphores : la primitive `semget`

```
#include <sys/sem.h>
int semget (key_t cle, int nb_sem, int option) ;
```

Fonctionnement : idem que `msgget`
Rôle de `option` : idem que `msgget`
Le paramètre `nb_sem` correspond au nombre de sémaphores de l'ensemble

III. 3. Opérations sur les ensembles de sémaphores : la primitive `semop`

```
#include <sys/sem.h>
int semop (int semid, struct sembuf *tab_op, int nb_op) ;
```

- Les `nb_op` opérations placées à l'adresse `tab_op` sont réalisées atomiquement (de manière séquentielle en mode noyau)
- Chaque opération du tableau `tab_op` peut être rendue individuellement non bloquante en positionnant l'indicateur `IPC_NOWAIT` dans le champ `sem_flg` de l'élément correspondant.

- L'aspect bloquant ou non bloquant d'un appel à la primitive `semop` dépend de celui de la première opération de l'ensemble qui n'est pas réalisable.

Interprétation des opérations : valeur `n` du champ `sem_op` de l'élément
- si `n > 0`, opération `Vn`. La valeur du sémaphore est augmentée de `n`. Tous les processus en attente de l'augmentation de la valeur de ce sémaphore sont réveillés.
- Si `n = 0`, opération `Z`. Le processus est bloqué tant que le sémaphore n'est pas nul.
- Si `n > 0`, opération $P_{|n|}$. Si l'opération n'est pas réalisable, le processus est par défaut bloqué et le nombre de processus en attente sur ce sémaphore est incrémenté. Si l'exécution de l'opération annule le sémaphore, tous les processus en attente de la nullité du sémaphore sont réveillés.

Les options : applicables aux opérations élémentaires sur un sémaphore.
- `IPC_NOWAIT` : rend une opération non bloquante
- `SEM_UNDO` : annulation automatique d'opérations antérieures en cas de terminaison

La primitive `semctl`

Permet de réaliser des consultations et/ou modifications de l'entrée de la table des sémaphores associée à un sémaphore particulier. Permet également de réaliser des initialisations des sémaphores.

```
#include <sys/sem.h>
int semctl (int semid, int semnum, int op, … /* arg */) ;
```

Valeur de `op`	Interprétation de `semnum`	Interprétation de `arg`	Valeur de retour et effet
GETNCNT	Numéro d'un sémaphore	ignoré	Nombre de processus en attente d'augmentation du sémaphore
GETZCNT	Numéro d'un sémaphore	ignoré	Nombre de processus en attente de la nullité du sémaphore
GETVAL	Numéro d'un sémaphore	ignoré	Valeur du sémaphore
GETALL	Nombre de sémaphores	Tableau d'entiers courts non signés	0 ou –1 Le tableau `arg` contient les valeurs des `semnum` premiers sémaphores
GETPID	Numéro d'un sémaphore	ignoré	Identification du dernier processus ayant réalisé une opération sur le sémaphore
SETVAL	Numéro d'un sémaphore	Entier	0 ou –1 Initialisation du sémaphore à `arg`

SETALL	Nombre de sémaphores	Tableau d'entiers courts non signés	0 ou –1 Initialisation des `semnum` premiers sémaphores au contenu de `arg`
IPC_RMID	ignoré	ignoré	Suppression de l'entrée dans la table des sémaphores
IPC_STAT	ignoré	Pointeur sur `struct semid_ds`	Extraction de l'entrée dans la table des sémaphores
IPC_SET	ignoré	Pointeur sur `struct semid_ds`	Modification de l'entrée dans la table des sémaphores

IV. Les segments de mémoire partagée

- Les processus partagent des pages physiques par l'intermédiaire de leur espace d'adressage
- Ces pages sont des ressources critiques dont les accès doivent être synchronisés
- Un segment de mémoire partagée a une existence indépendante des processus
- Un processus peut demander le rattachement d'un segment de mémoire partagée (smp)

IV. 1. Le fichier <sys/shm.h>

La structure `shmid_ds`
Correspond à une entrée dans la table des smp :

```
struct shmid_ds
  {
     struct ipc_perm    shm_perm ;  /*droits d'accès*/
     int                shm_segsz ; /*taille du segment en octets*/
     pid_t              shm_lpid ;  /*pid ayant fait la dernière opération*/
     pid_t              shm_cpid ;  /*pid créateur*/
     unsigned short int shm_nattch ;/*nombre d'attachements*/
     time_t       shm_atime ;       /*date du dernier attachement*/
     time_t       shm_dtime ;       /*date du dernier détachement*/
     time_t       shc_ctime ;       /*date   du   dernier   changement   par
                                    shmctl*/
  } ;
```

La structure `shminfo` contient les valeurs limites imposées par le système.

IV. 2. Obtention de l'identification d'un smp : la primitive shmget

```
#include <sys/shm.h>
int shmget(key_t cle, int taille, int option) ;
```

Fonctionnement identique à celui de `msgget` et `semget`.
Paramètre `taille` : taille du segment

IV. 3. La primitive d'attachement

```
#include <sys/shm.h>
void *shmat(int shmid, const void *adr, int option) ;
```

Demande d'attachement du segment identifié par `shmid` à l'adresse `adr` de l'espace d'adressage du processus.

Retour :
- adresse où l'attachement a été effectivement réalisé en cas de succès
- -1 en cas d'échec

Règles pour le choix de l'adresse `adr` :
- ne pas entrer en conflit avec des adresses déjà utilisées ou susceptibles de le devenir. Solution générale : les adresses associées à des segments de mémoire partagée appartiennent à une tranche particulière d'adresses
- ne pas violer la forme générale des adresses imposées par le système

Cas où `adr = NULL`
L'adresse est choisie par le système. Permet la portabilité de l'application et la garantie d'une adresse correcte.

Cas où `adr ≠ NULL`
L'adresse est choisie par l'utilisateur. Si l'indicateur `SHM_RND` est positionné dans le paramètre `option`, le système réalisera l'attachement à l'adresse la plus proche dont la valeur est un multiple de la constante système `SHMLBA`.

Option `SHM_RDONLY`
Si elle est utilisée dans le paramètre `option`, permet l'attachement en lecture seule d'un smp.

Attachements multiples par un même processus
Attachement d'un même segment à différentes adresses et éventuellement avec des modes différents par un processus. Possible dans certains systèmes.

IV. 4. Primitive de détachement

```
#include <sys/shm.h>
int shmdt (const void *adr) ;
```

- Demande de détachement d'un smp dont la demande d'attachement a retourné l'adresse `adr`.
- Si le nombre d'attachements d'un segment devient nul, le segment est effectivement libéré
- La terminaison d'un processus entraîne le détachement de tous les segments qu'il a attachés

IV. 5. La primitive `shmctl`

```
#include <sys/shm.h>
int shmctl (int shmid, int op, struct shmid_ds *p_shmid) ;
```

Permet de réaliser différentes opérations de contrôle sur les smp.

Valeurs possibles du paramètre `op` :

- `IPC_RMID` : demande de suppression du segment identifié par `shmid`
- `IPC_STAT` : demande de chargement à l'adresse `p_shmid` des informations contenues dans la table des segments correspondant à l'identification `shmid`
- `IPC_SET` : demande de modification de l'entrée dans la table des smp associée à l'identification `shmid` avec les valeurs définies dans l'objet d'adresse `p_shmid`. Les seuls champs modifiables sont `shm_perm.uid`, `shm_perm.gid` et `shm_perm.mode`.

Référence

J.-M. Rifflet, « La programmation sous Unix », 3ème édition, McGraw-Hill