Système d'exploitation (SE)
3. Communications Inter-processus

Jilles Dibangoye

Dépt. Télécommunications, Usages et Services

Outline

1 Introduction

2 Communications par tubes
- Tubes anonymes
- Tubes nommés

3 Communications par ipc System V
- Files de messages
- Mémoires partagées
- Semaphores

Communications inter-processus

Motivation

Idéal des mécanismes de communication inter-processus

1. **coopération** entre plusieurs processus différents
2. **synchronisation & sécurisation** des accès aux ressources partagées

Communications inter-processus classiques

1. Communication via un fichier :
   - `write()` et `read()` : coordination par accès concurrents à un **fichier**.
2. Communication entre processus père et fils :
   - `exit()` et `wait()` : un processus **père** reçoit la valeur de retour du **fils**.
   - `kill()` et `signal()` : synchronisation par **signalisation** d'un évènement.

Outline

1 Introduction

2 Communications par tubes
- Tubes anonymes
- Tubes nommés

3 Communications par ipc System V
- Files de messages
- Mémoires partagées
- Semaphores

Définition d'un tube

Définition

→ Un tube est un mécanisme de communication **unidirectionnel et de capacité finie** fondé sur un type spécifique de fichiers (`S_IFIFO`).

→ On distingue les tubes **anonymes** (*pipe*) des tubes **nommés** (*fifo*), mais tous deux ont des caractéristiques communes.

Caractéristiques communes

1. **Objets du système de fichiers**
   - accessibles via un descripteur de fichier,
   - manipulables via les primitives *read(...)* et *write(...)*,
   - redirection des entrées/sorties standard depuis/vers un tube
2. **Communication unidirectionnelle**
   - un descripteur pour écrire (option *O_WRONLY*)
   - un descripteur pour lire (option *O_RDONLY*)
3. **Communication en flot continu de caractères**
   - opérations de lecture et d'écriture sont indépendantes
     (ex. : écrire 5 caractères, en lire 2, en écrire 3, en lire 6, etc.)
   - l'opération de lecture dans un tube est destructrice
     (une information ne peut-être lue qu'une seule fois)

Caractéristiques communes (*cont'd*)

1. **Communication en mode *fist-in, first-out***
   - premier caractère écrit est le premier lu
2. **Nombre de lecteurs**
   - nombre de descripteurs associés à la lecture
   - si nul, alors le tube est par défaut bloquant en écriture
3. **Nombre d'écrivains**
   - nombre de descripteurs associés à la écriture
   - si nul, alors le tube est par défaut bloquant en lecture
4. **Capacité finie**
   - un tube peut-être plein et une écriture être bloquante
   - PIPE_BUF est la valeur max prédéfinie dans `<limits.h>`

Concepts et constantes communs

Quelques concepts utiles

→ Les **modes d'accès** définissent les droits en lecture ou écriture du processus appellant.

→ Les **bits d'états** définissent le comportement lors des opérations de lecture ou d'écriture (e.g., bloquantes ou non).

→ Les **options** correspondent à une disjonction des bits dédiés aux *modes d'accès* et les *bits d'états* entre autres (e.g., O_WRONLY | O_NONBLOCK).

Constantes	**Type**	**Role**
O_WRONLY	mode d'accès	lecture seulement
O_RDONLY	mode d'accès	écriture seulement
O_NONBLOCK	bits d'états	lecture/écriture non bloquante
F_GETFL	Commande	récupérer les **options**
F_SETFL	Commande	modifier les **options**

Structure commune *stat*

```
#include <sys/types.h>
#include <sys/stat.h>

struct stat{
 dev_t     st_dev  ;     /*Device                          */
 ino_t     st_ino  ;     /*File serial number              */
 mode_t    st_mode ;     /*File mode                       */
 nlink_t   t_nlink ;     /*Link count                      */
 uid_t     st_uid  ;     /*User ID of the file's owner     */
 gid_t     st_gid  ;     /*Group ID of the file's owner    */
 off_t     st_size ;     /*File size in bytes              */
 time_t    st_atime;     /*Time of last access             */
 time_t    st_mtime;     /*Time of last data modification*/
 time_t    st_ctime;     /*Time of last change             */
};
```

Primitive de récupération des statistiques *fstat()*

Primitive d'extraction des caractéristiques d'un tube (*fstat*)

```
1  #include <sys/stat.h>
2  int fstat (int p, struct stat *bf);
```

`int p`	descripteur du tube
`struct stat *bf`	pointeur sur la structure *stat* à remplir

Interprétation

Retourne 0 si succès et -1 sinon.

Accès aux caractéristiques *(cont'd)*

Primitive d'extraction et de mofication *fcntl()*

Primitive *fcntl(...)*

```
1  #include <sys/fcntl.h>
2  int fcntl (int p, int cmd);
3  int fcntl (int p, int cmd, int options);
```

`int p`	descripteur du tube
`int cmd`	constante associée à une opération
`int options`	une des disjonctions des bits d'états (e.g., O_NONBLOCK) et les modes d'accès (e.g., O_WRONLY et O_RDONLY)

Interpretation

cmd	options	**retourne**
F_GETFL	—	retourne les bits d'états et les modes d'accès
F_SETFL	int	modifie les modes d'accès (e.g., O_WRONLY et O_RDONLY) et les bits d'états (e.g., O_NONBLOCK)

Récupérer un descripteur d'un fichier

Attention non applicable aux fichiers anonymes

Primitive de récupération d'un descripteur (*open(...)*)

```
1  #include <sys/types.h>
2  #include <sys/stat.h>
3  int open(const char *ref, int options);
```

*const char * ref*	chemin d'accès au fichier
int options	disjonction bit-à-bit des bits d'états (e.g., O_NONBLOCK) et des modes d'accès (e.g., O_WRONLY ou O_RDONLY)

Interprétation de la primitive (*open(...)*)

retour (*int*)	Mode d'accès	Bits d'états	#RD	#WR	Interprétations
descripteur	O_RDONLY	—	—	—	lecture (bloquante)
descripteur	O_WRONLY	—	—	—	écriture (bloquante)
descripteur	O_RDONLY	O_NONBLOCK	—	non nul	lecture (¬bloquante)
descripteur	O_RDONLY	O_NONBLOCK	—	0	lecture (bloquante)
descripteur	O_WRONLY	O_NONBLOCK	non nul	—	écriture (¬bloquante)
−1	O_WRONLY	O_NONBLOCK	0	—	échec

Écriture dans un fichier (y compris un tube)

Primitive de lecture d'un tube anonyme (*write(...)*)

```
1  #include <unistd.h>
2  ssize_t write(int p, const void* bf, size_t lg);
```

`int p`	le descripteur en écriture du tube
`const void* bf`	le pointeur vers la mémoire contenant les octets
`size_t lg`	le nombre d'octets (size_t = entier non signé)

Interpretation de la primitive *write(...)*

#**lecteurs**	**écriture**	lg	**retour** [1]	**processus**
0	—	—	—	Fin (SIGPIPE)
non nul	—	$lg + lg* \leq$ PIPE_BUF	lg	—
non nul	bloquante	$lg + lg* >$ PIPE_BUF	—	**bloqué**
non nul	¬bloquante	$lg + lg* >$ PIPE_BUF	$< lg$	—

1. supposons que le tube contienne lg* octets.

Lecture d'un fichier (y compris un tube)

Primitive d'écriture d'un tube anonyme (*read(...)*)

```
1  #include <unistd.h>
2  ssize_t read(int p, const void* bf, size_t lg);
```

`int p`	le descripteur en lecture du fichier
`const void* bf`	le pointeur vers la mémoire destination des octets
`size_t lg`	le nombre d'octets (size_t = entier non signé)

Interpretation de la primitive *read(...)*

tube	#écrivaints	lecture	processus	retour
$\neg vide$	—	—	—	$\min(lg*, lg)$
vide	0	—	—	0 (*EOF*)
vide	non nul	*bloquante*	**bloqué**	—
vide	non nul	$\neg$*bloquante*	—	-1 (échec)

Fermeture d'un fichier (y compris un tube)

Primitive *close*(...)

```
1  #include <unistd.h>
2  int close (int p);
```

`int p` descripteur (en lecture ou écriture) du fichier

Interpretation

Retourne 0 en cas de succès et -1 sinon.

Attention

Ne conserver que les descripteurs utiles. Fermer systématiquement les autres.

Caractéristiques particulières d'un tube anonyme

→ Seuls les processus dans la **descendance** du créateur d'un tube anonyme dont les ancêtres ont eux-mêmes eu connaissance du tube peuvent communiquer au travers de ce tube.

→ Il n'est pas possible d'obtenir les descripteurs d'un tube anonyme via la primitive *open(...)* car il ne possède pas de référence dans le système de fichiers.

→ Un tube anonyme est **détruit automatiquement** au terme de son utilisation (si le nombre de lecteurs et d'écrivains est nul).

Création d'un tube anonyme

Primitive de création d'un tube anonyme

```
1  #include <unistd.h>          → librairie requise
2  int pipe(int desc[2]);       → prototype de pipe(...)
```

`int desc[2]` les deux descripteurs (en lecture et en écriture) du tube

Interprétations de la primitive *pipe(...)*

retour (*int*)	**descripteur** (*int desc[2]*)
0 (création réussie)	*desc[0]* ← descripteur en lecture *desc[1]* ← descripteur en écriture
−1 (échec de création)	—

Exemple d'utilisation des tubes anonymes

Communication entre processus père et fils

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(){
  int pid, note, descr[2];                    // déclaration des descripteurs
  if(pipe(descr) == 0){                       // creation d'un tube anonyme
    if((pid = fork()) == 0){
      close(descr[1]);                        // fils : fermeture du descripteur en écriture
      do{
        sleep(1);
        read(descr[0], &note, sizeof(int));   // fils : lecture (bloquante) d'une note
        printf("reçu %d.\n", note);
      }while(note >= 0);
      close(descr[0]);                        // fils : fermeture du descripteur en lecture
    }else if(pid > 0) {
      close(descr[0]);                        // père : fermeture du descripteur en lecture
      prinft("Entrez une suite de notes.\n");
      do{
        scanf("%d", &note);
        write(descr[1], &note, sizeof(int));  // père : écriture de la note dans le tube
      }while(note >= 0);
      close(descr[1]);                        // père : fermeture du descripteur en écriture
    }
  }
}
```

Caractéristiques particulières d'un tube nommé

→ Contrairement aux tubes anonymes, les tubes nommés permettent à des processus **sans lien** de communiquer.

→ Tout processus connaissant la référence du tube nommé peut obtenir au travers de la primitive *open(...)* un descripteur en lecture et/ou en écriture.

→ Contrairement aux tubes anonymes, la fermeture n'est pas automatique.

Création d'un tube nommé

Primitive de création d'un tube nommé (*mkfifo(...)*)

```
1  #include <sys/types.h>
2  #include <sys/stat.h>
3  int mkfifo(const char* ref, mode_t mode);
```

*const char * ref*	référence du chemin d'accès au tube
mode_t mode	mode d'accès (e.g., O_RDONLY ou O_WRONLY)

Interprétation de la primitive (*mkfifo(...)*)

Retourne 0 en cas de succès et -1 sinon.

Exemple d'utilisation des tubes nommé

Communication entre processus père et fils

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
int main(){
  int pid, note, descr[2];                    // déclaration des descripteurs
  if(mkfifo("fifo", 0666) == 0){              // création d'un tube nommé
    if((pid = fork()) == 0){
      descr[0]=open("fifo", O_RDONLY);        // fils : récupère un descripteur en lecture
      do{
        sleep(1);
        read(descr[0], &note, sizeof(int));   // fils : lecture (bloquante) d'une note
        printf("reçu %d.\n", note);
      }while(note >= 0);
      close(descr[0]);                        // fils : fermeture du descripteur en lecture
    }else if(pid > 0) {
      descr[1]=open("fifo", O_WRONLY);        // père : récupère un descripteur en écriture
      printf("Entrez une suite de notes.\n");
      do{
        scanf("%d", &note);
        write(descr[1], &note, sizeof(int));  // père : écriture de la note dans le tube
      }while(note >= 0);
      close(descr[1]);                        // père : fermeture du descripteur en écriture
    }
  }
}
```

Outline

1 Introduction

2 Communications par tubes
- Tubes anonymes
- Tubes nommés

3 Communications par ipc System V
- Files de messages
- Mémoires partagées
- Semaphores

Définition des ipc

Définition

→ Les ipc (System V) sont trois mécanismes de communication **bidirectionnels** entre processus locaux.

→ Les trois type d'ipc sont : *les files de messages*, *les sémaphores*, et *les segments de mémoire partagée*

Caractéristiques communes

- **Objects n'appartenant pas au système de fichiers**
  - Inaccessible via un descripteur de même nature que les fichiers.
- **Maintient d'une table pour chaque type d'ipc**
  - le système gère une table pour les files de messages, idem pour les semaphores et les segments de mémoire partagée ;
- **Chaque objet dispose d'un identifiant interne et externe**
  - identifiant interne : nombre entier positif ou nul
  - identifiant externe : mécanisme de **clé**

Caractéristiques communes (*cont'd*)

Constantes du fichier standard `<sys/ipc.h>`

Constante	**Type**	**Rôle**	**Primitives**
IPC_PRIVATE	key_t	clé privé : un nouvel objet sera crée, dont l'identité ne pourra être réclamée ultérieurement	`*get(...)`
IPC_CREAT	bits d'états	créer l'objet s'il n'existe pas	`*get(...)`
IPC_EXCL	bits d'états	utiliser avec IPC_CREAT, si l'objet existe déjà, une erreur est signalée	`*get(...)`
IPC_NOWAIT	bits d'états	opération non bloquante	`semop`, `msgrcv`, `msgsnd(...)`
IPC_RMID	commande	suppression d'identifiant	`*ctl(...)`
IPC_STAT	commande	extraction des caractéristiques	`*ctl(...)`
IPC_SET	commande	modification des caractéristiques	`*ctl(...)`

Caractéristiques communes (*cont'd*)

Constantes du fichier standard <sys/ipc.h>

Le système maintient un nombre d'informations relatives à chaque objet (ipc) figure dans la structure suivante :

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

Caractéristiques communes (*cont'd*)

Génération de clés uniques

Primitive de génération de clés (*ftok(...)*)

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

*const char * ref*	chemin d'accès d'un fichier existant
int num	nombre quelconque

Interpretation

- Retourne la clé construite à partir d'une référence à un fichier et un numéro.
- Attention, c'est la référence au fichier qui sert à la construction de la clé. Changement de l'emplacement de votre fichier → clé différente.

Caractéristiques communes (*cont'd*)

Commandes shell

Commande `ipcs`

Liste les objets présents dans les tables d'objets ipc du système.

Commande `ipcrm`

→ Supprime une entrée dans les tables d'objets ipc du système.

→ L'entrée à supprimer peut-être désignée par son identifiant interne ou externe.

→ Options minuscules correspondent aux identifiants internes, et les options majuscules aux clés : (-q -Q → files de messages) (-s -S → sémaphores) (-m -M → segments de mémoire partagée).

Définition

Définition

→ C'est un mécanisme de communication inter-processus inspiré du concept de **boîte à lettre (de capacité limitée)**.

→ Contrairement aux tubes, les files de messages communiquent par paquets identifiables (ou **mode datagramme**).

→ Il est possible d'extraire (de cet ipc) des messages possédant une caractéristique particulière en **mode FIFO**.

Structure de donnée relatif aux files de messages

Structure msqid_ds

Définition

→ Structure de données stockant l'ensemble des informations relatives à une entrée dans la table des files de messages.

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

Structure générique des messages

Structure msgbuf

Définition

→ Structure de données stockant l'ensemble des informations relatives à un message.

```
struct msgbuf{
  long   mtype;  /* type de message                                    */
  ...    ...   ;  /* texte du message (de type quelconque sauf pointeur) */
}
```

Exemple de structures msgbuf

(autorisé)

```
struct msgbuf{
  long    mtype ;
  float   n1    ;
  int     tab[4];
}
```

(non autorisé)

```
struct msgbuf{
  long    mtype;
  float   n1   ;
  char*   p    ;
}
```

Création d'une file de messages

Création d'une file de message ou recherche de l'identifiant d'une file déjà existante

Primitive de création d'un tube nommé (*msgget(...)*)

```
1  #include <sys/ipc.h>
2  #include <sys/msg.h>
3  int msgget(key_t cle, int options);
```

`key_t cle`	clé d'identification externe de l'ipc
`int options`	disjonction bit-à-bit des bits d'états et des modes d'accès

Interprétation de la primitive *msgget(...)*

- renvoie l'identifiant interne de la file en cas de succès
- renvoie -1 sinon

Propriétés de la création

1. Si cle = IPC_PRIVATE, alors une nouvelle file est créée
2. Si cle $\neq$ IPC_PRIVATE et ne correspond pas à une file existante
   - Si IPC_CREAT $\subset$ options, une nouvelle file est créée et l'appel retourne l'identifiant interne
   - Sinon, l'appel retourne -1
3. Si cle $\neq$ IPC_PRIVATE et correspond à une file existante
   - Si IPC_CREAT | IPC_EXCL $\subset$ options, une erreur est détectée et la valeur de retour est -1.
   - Sinon, l'identifiant interne de la file est renvoyé en retour.

Emission d'un messages

Primitive d'émission d'un message

```
1   #include <sys/msg.h>
2   int msgsnd(int id, struct msgbuf *msg, int lg, int options);
```

int id — identifiant interne (provenant de `msgget()`)
*struct msgbuf * msg* — message à emettre
int lg — taille du message
int options — paramètre optionnel, si sa valeur est IPC_NOWAIT alors l'appel à `msgsnd(...)` n'est plus bloquant

Interprétations de la primitive *msgsnd(...)*

- renvoie 0 en cas de succès
- renvoie -1 sinon

Propriétés d'une émission

1. Emission bloquante (par défaut)
   - Si la file est pleine, le processus est suspendu jusqu'à :
     - extraction de message de la file
     - suppression du système de la file (renvoie -1)
     - réception d'un signal
   - Sinon,
     - insertion du message dans la file
     - incrémentation du nombre de messages de la file
     - mise à jour de l'identificateur du dernier écrivain
     - mise à jour de la date de dernière écriture
2. Emission non bloquante
   - Si la file est pleine et IPC_NOWAIT $\subset$ options,
     - le message n'est pas envoyé et
     - le processus reprend immédiatement la main.

Extraction d'un messages

Primitive d'extraction d'un message *msgrcv(...)*

```
1   #include <sys/msg.h>
2   int msgrcv(int id, struct msgbuf *msg, int lg, long type, int options);
```

`int id`	identifiant interne de la file (provenant de *msgget(...)*)
`struct msgbuf *msg`	adresse de stockage du message à recevoir
`long type`	type de message à recevoir
`int options`	disjonction bit-à-bit des droits d'accès et bits d'états

Interprétation de la primitive (*msgrcv(...)*)

- Extraction quelconque ou sélective,
  - $type = 0 \rightarrow$ le 1er msg de la file, quel que soit son type,
  - $type > 0 \rightarrow$ le 1er msg du type désigné,
  - $type < 0 \rightarrow$ le 1er msg dont le type est supérieure à $|type|$,
- Opération bloquante par défaut,
- Renvoie la taille du message extrait en cas de succès
- Renvoie -1 en cas d'échec

Propriétés d'une extraction

1. Si aucun message ne répond aux conditions demandées :
   - IPC_NOWAIT$\not\subset$options, alors le processus est suspendu jusqu'à :
     - arrivée d'un message satisfaisant les conditions demandées,
     - destruction de la file (renvoie -1)
     - réception d'un signal
   - IPC_NOWAIT$\subset$ options, alors
     - le processus reprend immédiatement la main,
     - la primitive renvoie -1
2. Sinon
   - extraction effective du message de la file,
   - décrémentation du nombre de messages de la file
   - mise à jour de l'identifiant du dernier lecteur
   - mise à jour de la date de dernière lecture.

Contrôle de l'état d'une file

Consultation, modification des caractéristiques, et suppression d'une file

Primitive d'extraction d'un message (*msgctl*)

```
1  #include <sys/msg.h>
2  int msgctl(int id, int cmd, msqid_id *buf);
```

`int id`	identifiant interne de la file (provenant de *msgget(...)*)
`int cmd`	file (e.g., IPC_RMID, IPC_STAT ou IPC_SET)
`msqid_id *buf`	adresse de stockage des données de la file

Interpretation

Renvoie 0 en cas de succès et -1 sinon.

Exemple d'utilisation des files de messages (serveur.c)

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>

int main(void){
   key_t cle;                                                    // déclaration d'une clé
   int msg_id, note;
   struct msgbuf{
     long mtype;
     int mnote;
   } msg_buf;                                                    // structure de stockage d'une note

   if( (cle = ftok("serveur.c", 0)) != -1){
     if((msg_id = msgget(cle, 0666 | IPC_CREAT)) != -1){         // création d'une file de messages
       printf("Entrer une suite de notes (>= 0): \n");
       do{
         scanf("%d", &note);
         msg_buf.mtype = 1;                                      // définition du type
         msg_buf.mnote = note;                                   // définition de la note
         msgsnd(msg_id, &msg_buf, sizeof(msg_buf.mnote), 0);     // envoie d'un message
       }while(note >= -1);
         msgctl(msg_id, IPC_RMID, NULL);                         // suppression de la file de messages
     }
   }
}
```

Exemple d'utilisation des files de messages (client.c)

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void){
   key_t cle;
   int msg_id, note;
   struct msgbuf{
     long mtype;
     int mnote;
   } msg_buf;

   if( (cle = ftok("serveur.c", 0)) != -1){              // récupération de la clé
     if((msg_id = msgget(cle, 0666)) != -1){             // récupération d'un identifiant interne
       do{
         msgrcv(msg_id, &msg_buf, sizeof(int), 1, 0);    // envoie d'un message
         printf("reçu %d.\n", msg_buf.mnote);
       }while(msg_buf.mnote >= -1);
     }
   }
}
```

Définition

Definition

→ Un segment de mémoire partagée est un mécanisme de communication où les processus **partagent des pages physiques**.

→ Contrairement à d'autres (e.g., tubes ou files de messages), il y'a pas de recopie des données.

→ Attention, le segment de mémoire partagée devient une **ressource critique** (risque d'inter-blocage).

Caractéristiques spécifiques

Structure de données stockant l'ensemble des caractéristiques (shmid_ds)

Définition

→ Structure de données stockant l'ensemble des informations relatives à une entrée dans la table des segments de mémoire partagée.

```
struct shmid_ds{
  struct ipc_perm      shm_perm  ;  /* droits d'accès à l'objet                */
  int                  shm_segsz ;  /* taille du segment en octets             */
  pid_t                shm_lpid  ;  /* pid du dernière processus opérant       */
  pid_t                shm_cpid  ;  /* pid du processus créateur               */
  unsigned short int   shm_nattch;  /* nombre d'attachements                   */
  time_t               shm_atime ;  /* date de dernière attachement (shmatt)   */
  time_t               shm_dtime ;  /* date de dernière detachement (shmdt)    */
  time_t               shm_ctime ;  /* date de dernière changement (shmctl)    */
}
```

Création d'un segment de mémoire partagée

Création d'un segment de mémoire partagée ou recherche de l'identifiant

Primitive de création d'un segment de mémoire partagée

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

`key_t cle`	identifiant interne du segment de mémoire partagée
`int taille`	taille du segment de mémoire partagée
`int options`	disjonction bit-à-bit des bits d'états et modes d'accès

Interpretation

Renvoie l'identifiant interne en cas de succès et -1 sinon.

$\rightarrow$ les propriétés de la création d'un segment de mémoire partagée sont identiques à celles énoncées au slide 33.

Attachement d'un segment de mémoire partagée

Primitive d'attachement d'un segment de mémoire partagée

```
1  #include <sys/shm.h>
2  void* shmat(int id, const void* adr, int options);
```

`int id`	identifiant interne du segment de mémoire partagée
`int options`	disjonction bit-à-bit des bits d'états et modes d'accès
`const void* adr`	adresse où stocker le segment de mémoire partagée

Interpretation

- renvoie l'adresse à laquelle le segment a été attaché
- si 1er attachement, alors allocation effective de l'espace mémoire correspondant
- si adr $= 0$, alors le système choisit l'adresse d'attachement

Détachement d'un segment de mémoire partagée

Primitive détachement d'un segment de mémoire partagée

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

`const void* adr` adresse où stocker le segment de mémoire partagée

Interpretation

Renvoie 0 en cas de succès et -1 sinon.

Contrôle sur les segments de mémoire partagée

Primitive de contrôle d'un segment de mémoire partagée

```
1  #include <sys/shm.h>
2  void* shmctl(int id, int cmd, shmid_ds *buf);
```

`int id`	identifiant interne du segment de mémoire partagée
`int cmd`	commande
`shmid_ds *buf`	adresse où stocker les données du segment de mémoire partagée

Interpretation

- cf. tableau des constantes relatives aux ipc (voir slide 25),
- renvoie 0 en cas de succès et -1 sinon.

Définition

Définition

→ Mécanisme de communication permettant en particulier de synchroniser les processus demandant l'accès à une même ressource.

→ Offre une solution aux problèmes d'accès concurrents à une ressource partagée et d'exclusion mutuelle.

→ Un sémaphore est donné par :

- un compteur : nombre d'accès disponibles avant blocage.
- une file d'attente : processus suspendus en attente d'un accès.

Fonctionnement d'un sémaphore S

1. Demande d'accès ($P(S)$ ou *puis-je?*)
   - Si $S = 0$ alors le processus est suspendu
   - Sinon, $S \leftarrow S - 1$
2. Fin d'accès ($V(S)$ ou *vas-y*)
   - $S \leftarrow S + 1$
   - Réveiller un (ou plusieurs) processus suspendu

Dysfonctionnements

Interblocage

On parle d'interblocage si deux processus p_1 et p_2 sont tels que :

1. p_1 attend la fin d'accès de p_2
2. p_2 attend la fin d'accès de p_1

Famine

On parle de famine si un processus est en attente d'une fin d'accès qui n'arrivera jamais.

Structure d'un sémaphore individuel

Définition

→ Structure de données 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 suspendus attendant      */
                                    l'augmentation du sémaphore
  unsigned short int   semzcnt;  /* nombre de processus suspendus attendant      */
                                    la nullité du sémaphore
}
```

Structure d'une entrée dans la table des sémaphores

Définition

→ Structure de données d'une entrée dans la table des sémaphores correspondant à un **lot de sémaphores individuels**.

```
struct semid_ds{
  struct ipc_perm      sem_perm ;  /* droits d'accès à l'objet          */
  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        */
                                   (semctl)
  unsigned short int   sem_nsems;  /* nombre de sémaphores dans         */
                                   l'ensemble
}
```

Numérotation des sémaphores

Les sémaphores sont numérotés de 0 à `sem_nsems` − 1.

Structure d'une entrée dans la table des sémaphores

Définition

→ Structure de données permettant d'effectuer des opérations sur une sémaphore ($P(S)$ et $V(S)$).

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

Création d'un lot de sémaphores

Création d'un lot de sémaphores ou recherche de l'identifiant d'un lot existant

Primitive de création d'un lot de sémaphores *semget(. . .)*

```
1  #include <sys/ipc.h>
2  #include <sys/sem.h>
3  int semget(key_t clé, int nsems, int options);
```

`key_t clé`	clé identifiant externe du lot de sémaphores
`int nsems`	nombre de sémaphores dans le lot de sémaphores
`int options`	une des disjonctions possibles des bits IPC_CREAT, IPC_EXCL, etc.

Interpretations

→ Renvoie l'identifiant interne en cas de succès et -1 sinon.

→ Propriétés de la création d'un lot de sémaphores sont identiques à celles énoncées au slide 33.

Opérations sur le lot de sémaphores

Primitive de modification ($P(S)$ ou $V(S)$) sur un lot de sémaphores *semop(...)*

```
1  #include<sys/ipc.h>
2  #include<sys/sem.h>
3  int semop(int id, struct sembuf *op, int n_op);
```

`int id`	identifiant interne du lot de sémaphores donné par *semget(...)*
`struct sembuf *op`	adresse de stockage de l'ensemble des opérations à effectuer
`int n_op`	nombre des opérations placées à l'adresse op.

Interpretations

- retour 0 en cas de succès et -1 sinon
- chaque sem_op est exécutée sur le sémaphore correspondant à id et sem_num
- opérations traitées de façon atomique (**toutes** à la fois ou aucune)
- si un processus s'endort, la valeur initiale des sémaphores (avant l'appel) est restaurée

Opérations de contrôle sur les sémaphores

Primitive de contrôle d'un lot de sémaphores *semctl(...)*

```
1  #include<sys/ipc.h>
2  #include<sys/sem.h>
3  int semctl(int id, int semnum, int cmd, union semun arg);
```

`int id`	identifiant interne du lot de sémaphores donné par *semget(...)*
`int semnum`	selon *cmd* soit un numéro de sémaphore, soit les *semnum* premiers sémaphores
`union semun arg`	argument facultatif.

Interpretations

Retourne 0 en cas de succès et -1 sinon.

Opérations de contrôle sur les sémaphores (*cont'd*)

```
union semun arg{
  int val;                    /* Valeur pour SETVAL */
  struct semid_ds *buf;       /* Tampon pour IPC_STAT, IPC_SET */
  unsigned short *array;      /* Tableau pour GETALL, SETALL */
}
```

Propriétés

cmd	semnum	arg	retourne
GETVAL, GETPID, GETNCNT, GETZCNT	numéro	—	entier
SETVAL	numéro	*val*	0 si succès, sinon −1 et initialise le sémaphore à *arg*
GETALL, SETALL	lot	*array*	0 si succès, sinon −1 et *arg* prend ou actualise les valeurs du lot désigné par *semnum*
IPC_STAT, IPC_SET, IPC_RMID	lot	*buf*	extraction, modification ou suppression dans la table des sémaphores

Synchronisation via des sémaphores (*serveur.c*)

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>                                        // (+) stdio.h, stdlib.h, unistd.h
int main(void){
   key_t cle;
   int shm_id, sem_id, *note;
   struct sembuf sem_v = {0, 1, 0}, sem_p = {0, -1, 0};
   if( (cle = ftok("serveur.c", 0)) != -1){                 // Récupérer une clé
        if((shm_id = shmget(cle, 1024, 0666|IPC_CREAT)) != -1){   // Créer un segment de mémoire
        note = (int*)shmat(shm_id, (char*) NULL, 0);        // S'attacher au segment de mémoire
         if((sem_id = semget(cle, 1, 0666|IPC_CREAT)) != -1){     // Créer un lot de sémaphores
           if(semctl(sem_id, 0, SETVAL, 1) != -1){          // Initialiser le lot de sémaphores
              printf("Entrer une suite de notes (>= 0): \n");
              do{
                 semop(sem_id, &sem_p, 1);                  // Puis-je?
                 scanf("%d", note);
                 semop(sem_id, &sem_v, 1);                  // Vas-y!
                  sleep(1);                                 // Ordonnancement des processus
              }while(*note >= -1);
           }
        }
     }
   }
   shmdt(note);                                             // Se détacher du segment de mémoire
   shmctl(shm_id, IPC_RMID, 0);                             // Supprimer le segment de mémoire
   semctl(sem_id, IPC_RMID, 0);                             // Supprimer le lot de sémaphore
}
```

Synchronisation via des sémaphores (*client.c*)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(void){
   key_t cle;
   int shm_id, sem_id, *note;
   struct sembuf sem_v = {0, 1, 0}, sem_p = {0, -1, 0};
   if( (cle = ftok("serveur.c", 0)) != -1){                  // Récupérer une clé
     if((shm_id = shmget(cle, 1024, 0666)) != -1){           // Créer un segment de mémoire
       if((sem_id = semget(cle, 1, 0666)) != -1){            // Créer un lot de sémaphores
         note = (int*)shmat(shm_id, (char*)NULL, 0);         // S'attacher au segment de mémoire
         if(semctl(sem_id, 0, SETVAL, 1) != -1){             // Initialiser le lot de sémaphores
           do{
             semop(sem_id, &sem_p, 1);                       // Puis-je?
             printf("reçu %d.\n", *note);
             semop(sem_id, &sem_v, 1);                       // Vas-y!
             sleep(1);                                       // Ordonnancement des processus
           }while(*note >= -1);
         }
       }
     }
   }
   shmdt(note);
}
```