Communications Inter Processus

Methode basique pour faire communiquer 2 processus :

Utilisation de fichiers

- Erciture par un processus
- Lecture par l'autre processus

Problèmes

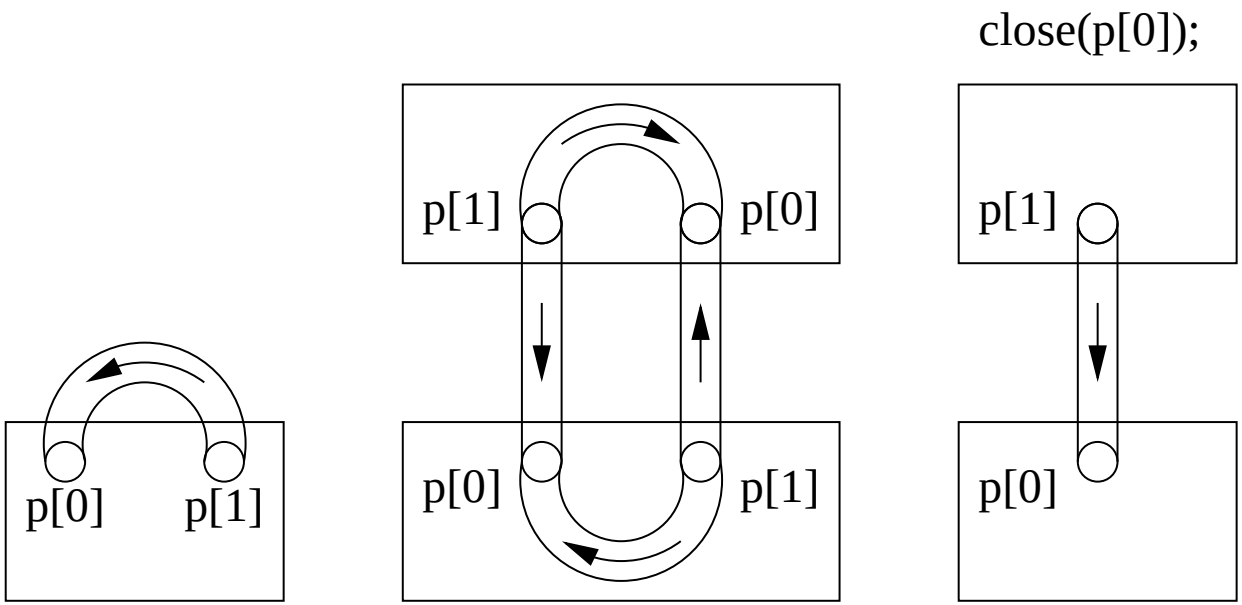
Synchronisation sur les acces au fichier

Les différents mécanismes de communication

- Signaux
- Tubes
- FIFOs
- IPC System V
  - Sémaphores
  - Files de messages
  - Mémoire partagée
- Sockets

Les tubes : rappel

`int pipe(int filedes[2])` : crée un tube; retourne -1 si la fonction échoue.



Les tubes : rappel

```
int main(){
   int n, fd[2];
   char line[15];
   pipe(fd);
   if (fork()>0){ /*processus pere*/
      close(fd[0]);
      write(fd[1],"hello son",9);
      wait();
   }
   else{ /*processus fils*/
      close(fd[1]);
      n = read(fd[0], line, 15);
   }
   exit(0);
}
```

Les tubes

Tubes avec un programme externe

- `FILE * popen(const char * cmdstring, const char * type);`
- `int pclose(FILE *fp);`

Fonctionnement

- *type* = "*r*" la sortie standard du programme appelé est redirigée sur le descripteur retourné.
- *type* = "*w*" L'entrée standard du programme appelé est le descripteur retourné.

Les tubes

Exercice

Compter le nombre de caractères que retourne un "ls -l".

FIFOs

Définition

Similaires aux tubes, les FIFOs permettent de faire communiquer deux processus qui n'ont pas un lien de parenté connu.

Création

```
int mkfifo(const char * pathname, mode_t mode);
```

retourne -1 sur une erreur.
Créer une entrée dans le système de fichier.

Utilisation

avec les primitives `open`,`close`,`read`, `write` et `unlink`.

Généralités

Les 3 types de communication définis par System V ont beaucoup de similarités.

- Chacun de ces mécanismes est référencé dans le noyau par un identifiant.
- Un certain nombre de structures sont communes entre les 3 types de mecanismes
- Le fonctionnement reste similaire

Fonctionnement

Les différents processus peuvent acceder au mécanisme IPC via son identifiant.

- Un processus créé le mécanisme IPC en specifiant une clé et récupère l'identifiant généré. Celui ci peut alors etre stocké dans un fichier et partagé pour les autres processus.
- En utilisant la clé.
- On peut utiliser une clé privée (*IPC_PRIVATE*). Dans le cas d'une relation de parenté entre les processus, ceux-ci partagent naturellement l'identifiant du mécanisme.
- Via la fonction *ftok*()

Permissions

```
struct ipc_perm{
   uid_t uid;
   gid_t gid;
   uid_t cuid; /*creator id*/
   gid_t cgid;
   mode_t mode; /*acces mode*/
   ulong seq;
   key_t key;
}
```

Inconvénients des mécanismes IPC

- Mécanismes Globaux sur le systeme
- Lourdeur
- La terminaison de processus utilisant le mécanisme n'entraine pas la supperession de celui-ci.

Rappels

Un sémaphore S (le plus simple) est un drapeau qui est :

- soit levé
- soit baissé

Avant d'entrer en S.C, un processus attend que le drapeau soit levé, puis le baisse : opération $P()$ A la sortie de la S.C., le processus lève le drapeau opération $V()$ et le S.E réveille un processus en attente

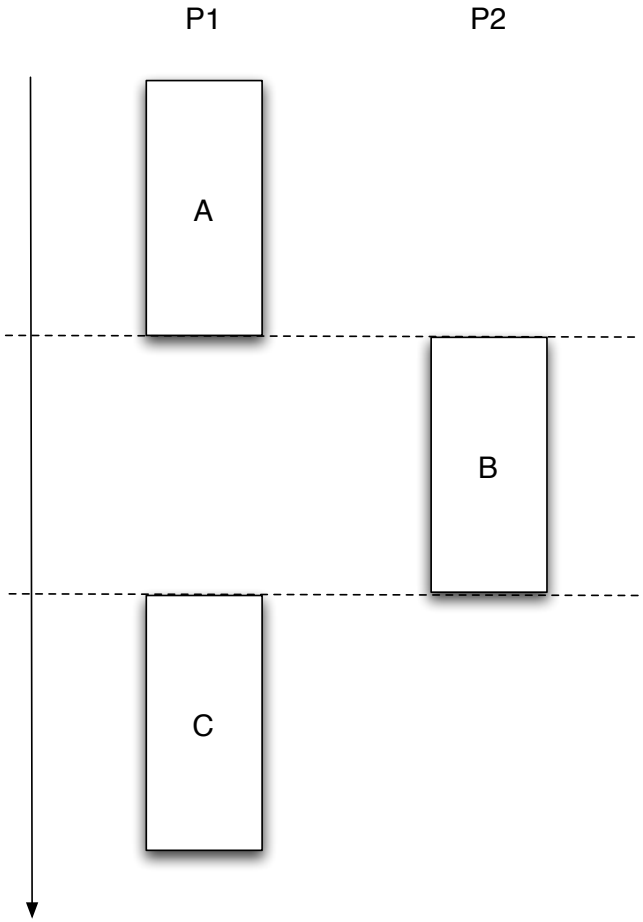
Rappels

Les opérations P() et V() sont atomiques

- $P()$: le test et la décrémentation sont **atomiques**.
- $V()$: l'incrémentation est atomique

2 opérations atomiques ne peuvent être exécutées simultanément En aucun cas, 2 processus ne verront le drapeau levé et l'abaisseront

Rappels



Fonctionnement

Primitives associées

- `int semget(key_t key, int nsems, int flag);` crée ou référence à un jeu de sémaphores. `key` est la clé IPC, `nsems` est le nombre de sémaphores et `flags` contient les permissions et `IPC_CREAT` pour créer un nouveau jeu.Retourne l'ID IPC.
- `int semctl(int semid, int semnum, int cmd, union semun arg);` manipule un jeu de sémaphores. `semid` est l'ID IPC du jeu de sémaphores, `semnum` est le sémaphore concerné et `cmd` doit contenir une macro :
  - `SETVAL` : modifie la valeur du sémaphore `semnum`.
  - `GETVAL` : retourne la valeur d'un des sémaphores du jeu.
  - `IPC_RMID` : permet de supprimer un jeu de sémaphores.

Fonctionnement

Primitives associées

- `int semop(int semid, struct sembuf semoparray[], size_t nops);` réalise des opérations sur les sémaphores. `semid` est l'identifiant IPC. Chaque opération est de type `sembuf` et est placée dans le tableau `semoparray`. `nops` est le nombre d'opérations.

Fonctionnement

Structures de contrôle

```
union semun {
    int val; // Valeur
    struct semid_ds *buf;
    u_short *array;
};
```

Dans le cas de `SETVAL`, nous n'utiliserons que le champ `val`. D'autres opréations de contrôle sont possibles : `IPC_STAT`, `IPC_SET`, `GETPID`, `GETALL`, `SETALL`, ... La page de manuel vous en dira plus...

Fonctionnement

Structures de contrôle

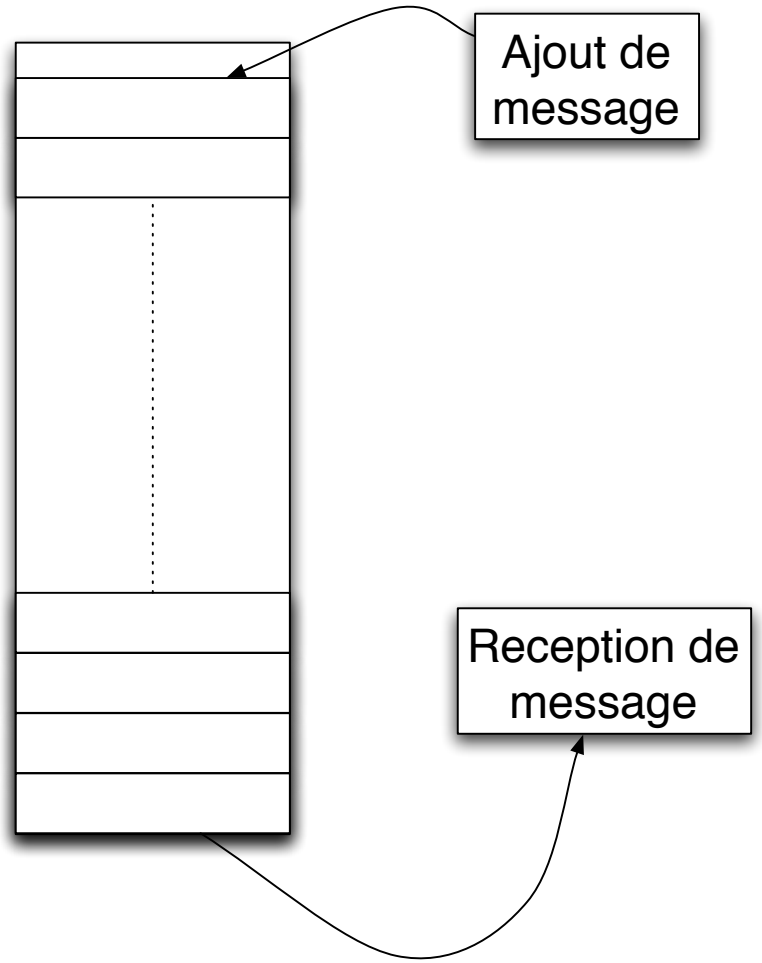
```
struct sembuf {
    u_short sem_num; Nuero du semaphore
    short sem_op; opération
    short sem_flag;
};
```

`sem_flag` peut prendre la valeur `IPC_NOWAIT` pour fair un appel non bloquant.

Exemple

```
 union semun s;
struct sembuf t;
idIpc=semget(3700, 3, IPC_CREAT|0666);
s.val = 1;
semctl(idIpc, 0, SETVAL, s);
v = semctl(idIpc, 0, GETVAL);
printf("Valeur de la sémaphore :  %d", v);
t.sem_num = 0;
t.sem_op = -1;
t.sem_flg = 0;
semop(idIpc, &t, 1);
semctl(idIpc, 1, IPC_RMID);
```

Principes



Principes

- Plusieurs processus peuvent ajouter des messages dans la meme file de processus.
- D'autres processus (ou les memes) peuvent recevoir les différents messages.

Structure

```
 struct msqid_ds{
    struct ipc_perm msg_perm; /*permissions*/
    struct msg *msg_first; /*premier message*/
    struct msg *msg_last;/*dernier message*/
    ulong msg_cbytes;/*Nb octets dans la file*/
    ulong msg_qnum;/*Nb de messages*/
    ulong msg_qbytes;/*Max du Nb d'octets dans la file*/
    pid_t msg_lspid;/*dernier processus ayant proceder à un
envoi*/
    pid_t msg_lrpid;/*dernier processus ayant proceder à une
reception*/
    time_t msg_stime;/*date dernier envoi*/
    time_t msg_rtime;/*date derniere reception*/
    time_t msg_ctime;/*date derniere operation*/
};
```

Fonctions

`int msgget(key_t key, int flags);`

Retourne un identifiant de file de message. Le paramètre `key` est la clé ipc. Le paramètre `flags` peut prendre la valeur `IPC_CREAT` pour créer une nouvelle file de message.

`int msgctl(int msqid, int cmd, struct msqid_ds *buf);`

Effectue une operation de controle sur la file de message associée à `msqid`. Le paramètre `cmd` peut prendre les valeurs suivantes :

- `IPC_STAT` récupère la structure `msqid_ds` dans le paramètre `buf`.
- `IPC_SET` positionne les paramètres contenus dans `buf`.
- `IPC_RMID` supprime la ressource IPC

Messages

```
struct mymsg{
    long mtypes;
    char mtext[512];
};
```

Envoi d'un message

```
int msgsnd(int msqid, const void * ptr, syze_t nbytes,
int flag);
```

Ecrit un message `ptr` de longueur `nbytes`($<$512) à la fin de la file référencée par `msqid`. L'écriture peut être bloquante si la file est pleine. On utilise dans ce cas le flag `IPC_NOWAIT`.

Reception d'un message

```
int msgrsv(int msqid, void * ptr, syze_t nbytes, long type, int flag);
```

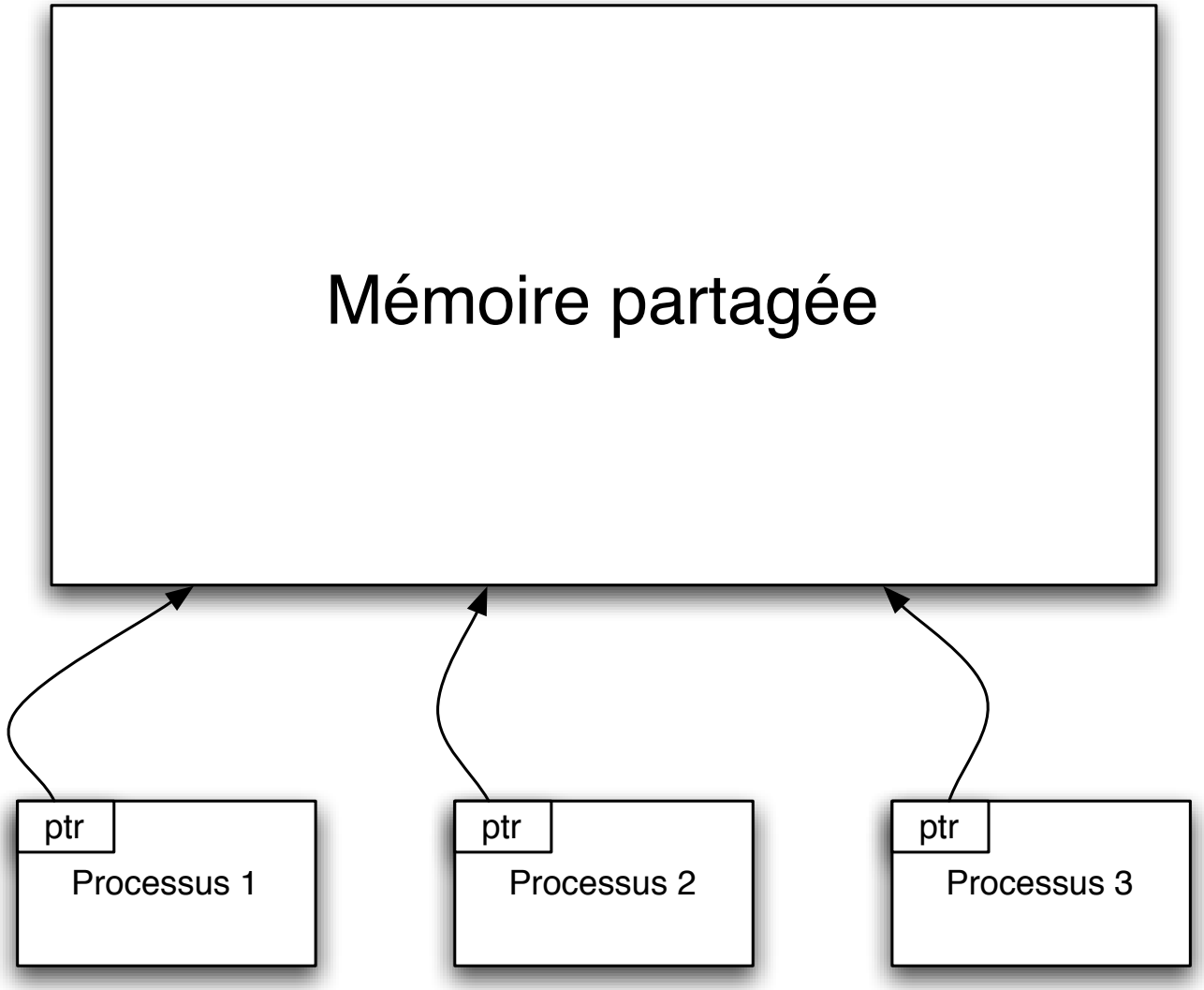
Lit un message `ptr` de longueur `nbytes`($<$512) au debut de la file référencée par `msqid`. La lecture peut être bloquante si la file est vide. On utilise dans ce cas le flag `IPC_NOWAIT`. La valeur de `type` permet de spécifier le message que l'on souhaite recevoir.

- `type == 0` premier message de la fle
- `type` $>$ 0 premier message dont le champ `mtypes` est égal à `type`
- `type` $<$ 0 premier message dont le champ `mtypes` est inferieur à la valeur de `- type`

Exercice

Soit une file de message contenant des messages de trois types : urgent , usuel, touriste. La priorité des messages est une valeur qui s'echellonne de 1 à 10 (1-4 urgent, 5-7, usuel, 8-10 touriste). Ecrire la portion de code qui permet de ranger chacun des messages soit dans la file urgent, soit dans la file, usuel, soit dans la file touriste.

Principes


Mémoire partagée
ptr
Processus 1
ptr
Processus 2
ptr
Processus 3

Structure

```
 struct shmid_ds{
    struct ipc_perm shm_perm; /*permissions*/
    struct anon_map *shm_map; /*adresse dans le noyau*/
    int shm_segsz;/*taille*/
    ushort shm_lkcnt;/*Nb de verrouillage*/
    pid_t shm_lspid;/*dernier processus ayant opérer*/
    pid_t shm_cpid;/*créateur*/
    ulong shm_nattch;/*Nb de processus attachés à la
mémoire*/
    time_t shm_atime;/*date dernier attachement*/
    time_t shm_dtime;/*date derniere détachement*/
    time_t shm_ctime;/*date dernier changement*/
};
```

Fonctions

```
int shmget(key_t key, int size, int flags);
```

Retourne un identifiant de mémoire partagée. Le paramètre `key` est la clé ipc. Le paramètre `flags` peut prendre la valeur `IPC_CREAT` pour créer une nouvelle zone de mémoire partagée de taille `size`.

Fonctions

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Effectue une operation de controle sur la mémoire partagée associée à `msqid`. Le paramètre `cmd` peut prendre les valeurs suivantes :

- `IPC_STAT` récupère la structure `shmid_ds` dans le paramètre `buf`.
- `IPC_SET` positionne les paramètres contenus dans `buf`.
- `IPC_RMID` supprime la ressource IPC
- `SHM_LOCK` Positionne un verrou sur la memoire partagée
- `SHM_UNLOCK` Retire le verrou sur la memoire partagée

Attachement / Détachement

```
void * shmat(int shmid, void *addr, int flag);
```

Attache la mémoire partagée à l'espace mémoire du processus pour pouvoir être utilisée. Le paramètre `addr` permet de spécifier l'adresse mémoire. S'il vaut `NULL`, c'est le noyau UNIX qui doit choisir l'adresse. La valeur de retour est une adresse `(void*)` représentant l'adresse de départ du segment partagé. En cas d'échec, la valeur `(void*)(-1)` est retournée.

```
int shmdt(void *addr)
```

détache la mémoire partagée.

Remarques

- Nécessité de cartographier l'espace memoire utilisé.
- Lecture / Ecriture : Necessité de gèrer la concurence, utilisation de sémaphore.