Communication Interprocessus

Plan

1. Introduction
2. Mécanisme Général de la communication
3. Communication par boîte aux lettres
4. Tubes
5. Files de message
6. La mémoire partagée
7. Les IPC

utbm
université de technologie Belfort-Montbéliard

Introduction

- Communication interprocessus:
  - Ensemble des moyens permettant les échanges d'information entre les processus
    - Problèmes d'accès concurrents → section critique
    - Mécanisme d'exclusion mutuelle
      - Attente active
      - Verrous
      - Sémaphore
    - Modes de communication :
      - Signaux
      - Mémoire commune
      - Variables communes
      - Envoi de message
      - Protocole de communication (Producteur / Consommateur)
      - Boîte aux lettres..

Mécanisme Général de communication

- Met en œuvre un **protocole de communication** entre deux processus utilisant des ressources duales.
  - Emetteur de l'information (producteur),
  - Récepteur de l'information (consommateur).
    - Il n'est souvent pas nécessaire de faire attendre le producteur jusqu'à ce que le consommateur ait reçu l'information → tampon (message) qui mémorise les informations produites non consommées.
    - Le tampon peut recevoir un nombre limité de ces messages.
    - Les deux processus doivent se synchroniser entre eux de façon à respecter certaines contraintes de bon fonctionnement.
    - Le producteur ne peut déposer un message dans le tampon s'il n'y a plus de place libre.
    - Le consommateur ne peut retirer un message depuis le tampon s'il n'y en a pas.
    - Le consommateur ne doit pas retirer un message que le producteur est en train de déposer.

Mécanisme Général de communication

- Exemple 1: Producteur – Consommateur de bonbons
  - Soient deux processus :
    - Un producteur produit des bonbons
    - Un consommateur qui consomme des bonbons
    - Contrainte : le nombre de papiers est limité (N)
    - Deux ressources (les bonbons et les papiers) et deux sémaphores pour les protéger :
      - Sémaphore papier initialisé à N
      - Sémaphore bonbon initialisé à 0

Algorithme

Producteur	*Consommateur*
P(Papier)	*P(Bonbon)*
fabrique bonbon	*mange un bonbon*
V(bonbon)	*V(papier)*

Communication par boite aux lettres

- Objet système qui implante **le schéma producteur consommateur** en conservant le découpage en messages tels que les a déposés le producteur.
- Objectif : Permettre à des processus dont les espaces mémoire sont disjoints de pouvoir désigner un même espace de stockage.
- Principe : la boite aux lettres, ressources privée, est une variable commune à tous les processus qui communiquent :
  - Seul le propriétaire a le droit de lire,
  - Tous les autres processus peuvent écrire,
  - Filtrage des messages reçus → entête du message,
  - Aucune garantie de consommation du message,
  - Capacité limité de messages → nulle dans le cas d'un récepteur synchrone.

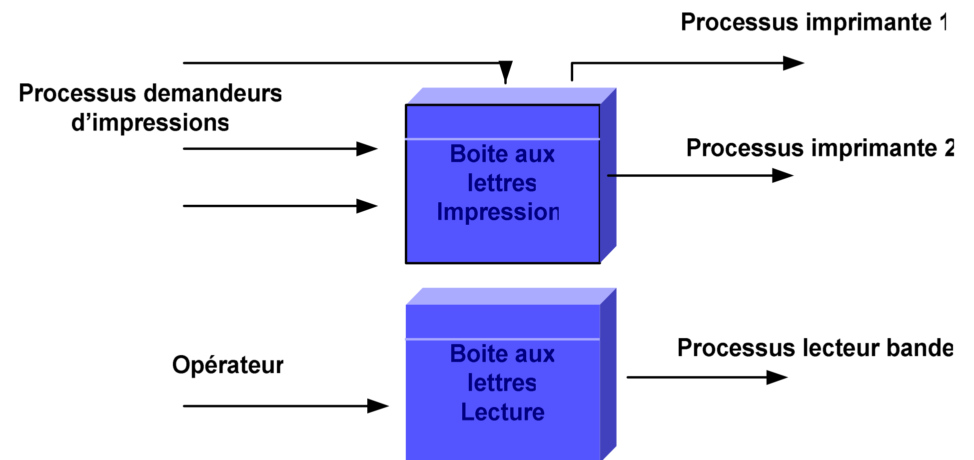
- Les files de messages (Unix) réalisent des boites aux lettres partagées, mettant en œuvre des droits d'accès pour distinguer les différents processus destinataires.

Communication par boite aux lettres

- _Modalité de son implantation
  - Donner un nom symbolique à la boite aux lettres
  - Le système maintient une table des noms symboliques qui ont déjà été créées et non encore détruites.
  - Lorsqu'un processus demande l'accès à une boîte, le système consulte cette table, et retourne au demandeur le descripteur de la boîte aux lettres (par exemple, son indice dans le table).
  - Dans certains cas, cette demande d'accès entraîne sa création implicite si elle n'existe pas, alors que d'autres systèmes séparent l'opération de création explicite de la demande d'accès.

Communication par boite aux lettres

- Exemple : Soit un système d'E/S gérant un lecteur de disque et deux imprimantes, ces trois périphériques fonctionnent en parallèle. Les opérations possibles sont :
  - Imprimer le contenu d'un fichier (demandé par un user)
  - Lire le fichier sur un disque (demandée par un opérateur)
  - Les demandes d'E/S utilisent 2 boites aux lettres selon le schéma suivant :



Principe des tubes

- Objets externe au SGF
- Files d'octets
- Les tubes Unix sont une implantation de la communication suivant le schéma producteur consommateur, avec un tampon de taille fixe, interne au système.
  - Pour un processus, un tube se présente comme deux flots, l'un ouvert en écriture (pour le producteur), l'autre ouvert en lecture (pour le consommateur).
    - Lors d'une écriture de n octets par le producteur, celui-ci sera mis en attente s'il n'y a pas assez de place dans le tube pour y mettre les n octets. Il sera réactivé lorsque les n octets auront pu être tous écrits dans le tube.
    - Le système interrompra l'exécution normale du processus s'il n'y a plus de consommateur pour ce tube.
    - Lors d'une lecture de p octets par le consommateur, celui-ci sera mis en attente s'il n'y a pas assez d'octets dans le tube pour satisfaire la demande.

Principe des tubes

- Fichiers spéciaux gérés selon une méthodologie FIFO
- Utiliser de façon unidirectionnel
- Taille limitée, généralement 10 blocs
- Deux types de tubes :
  - Tubes ordinaires, non visibles dans l'arborescence, entre processus d'une même filiation
  - Tubes nommés, visibles dans l'arborescence, entre processus non forcément liés par filiation

- Création ou ouverture de tube retourne des descripteurs de fichiers ouverts gérés comme les autres fichiers

utbm
université de technologie Belfort-Montbéliard

Tubes ordinaires

- Accès par création ou héritage
- Les descripteurs créés sont ajoutés à la table des descripteurs de fichiers du processus appelant.
- Perte par un processus = perte d'accès définitive par ce processus
- Position courante entièrement déterminée par les lectures/écritures
  - Primitive lseek interdite
- Taille maximum déterminée par PIPE_BUF octets
- Opération de lecture par défaut bloquante
  - Bloquant jusqu'à lecture de la taille demandée ou disparition de tous les écrivains
  - Pour non bloquante : utiliser fcntl avec le drapeau 0_NONBLOCK
- Opération d'écriture atomique (tout est écrit) si opération bloquante
  - Signal SIGPIPE si aucun lecteur.

Création et utilisation d'un Tube

- La création d'un tube correspond à celle de deux descripteurs de fichiers, l'un permettant d'écrire dans le tube et l'autre d'y lire par les opérations classiques read et write de lecture/écriture dans un fichier. Pour cela, on utilise la fonction prototypée par :

```
int pipe (int *p_desc)
```

p_desc[0] désigne le n° du descripteur par lequel on lit dans le tube
p_desc [1] désigne le n° du descripteur par lequel on écrit dans le tube.

- La fonction retourne 0 si la création s'est bien passée et - 1 sinon.

Tube ordinaire

- Le processus appelant crée un tube accessible par lui-même et par tous les processus de sa descendance qui seront créés ultérieurement (fils, petitsfils…).
- Le vecteur tube est un vecteur de descripteurs de fichiers où :
  - tube[0] est ouvert en lecture,
  - tube[1] ouvert en écriture ;
- les données écrites dans tube[1] (voir write) sont lues dans tube[0] (voir read) dans l'ordre First-In, First-Out (FIFO),

- Exemple :
  - write(tube[1], "bonjour", 8) :
    - envoi de caractères dans le tube, on indique le nombre de caractères à déposer (ici 8 pour écrire aussi le '\0' de fin de chaîne).
  - read(tube[0], tabcar, i) :
    - lecture de i caractères dans le tube, les caractères sont copiés dans le tableau tabcar (qui doit pouvoir contenir au moins i caractères).

utbm

Manipulation Tube ordinaire: Lecture

- int **read** (int fildes, char buf[], unsigned int)
  - read tente de lire n octets dans le fichier associé au descripteur fildes (les octets lus sont copiés dans buf).
  - renvoie le nombre d'octets effectivement lus

  - Algorithme de lecture :

    - si le tube n'est pas vide :
      - extrait les n octets du tube ;
    - si le tube est vide :
      - s'il y a des descripteurs ouverts en écriture :attend une écriture dans le tube (la lecture est bloquante par défaut);
      - s'il n'y a plus de descripteurs ouverts en écriture : retourne 0.

  - Conseil : toujours fermer les descripteurs dont on a pas besoin

Manipulation Tube ordinaire :Ecriture

- int **write** (int fildes, char buf[], unsigned int)
  - write tente d'écrire n octets dans le fichier associé au descripteur fildes (les octets sont lus dans buf).
  - renvoie le nombre d'octets effectivement écrits (<=nbyte). Par défaut, un write sur un tube réussit toujours mais peut éventuellement être bloquant si le buffer alloué au tube est plein.

  - Algorithme d'écriture :

    - s'il n'y a pas de descripteur en lecture d'ouvert :
      - » *le signal SIGPIPE (broken pipe) est envoyé au processus écrivain (qui se termine par défaut) ;*
    - s'il y a des descripteurs en lecture d'ouverts :
      - » l'écriture est bloquante par défaut :attend une lecture sur le pipe ;

Manipulation Tube ordinaire

- Fermeture

  int **close** (int fildes)

  - Ferme le fichier associé au descripteur fildes (on ferme un tube en lecture avec close(tube[0])).

Sécurité sous Unix

- Un processus peut fermer un tube, mais alors il ne pourra plus le rouvrir. Lorsque tous les descripteurs d'écriture sont fermés, une fin de fichier est perçue sur le descripteur de lecture : read retourne 0.
- Un processus qui tente d'écrire dans un tube plein est suspendu jusqu'à ce que de la place se libère. On peut éviter ce blocage en positionnant le drapeau O_NDELAY (mais l'écriture peut n'avoir pas lieu). Lorsque read lit dans un tube insuffisamment rempli, il retourne le nombre d'octets lus.
- Attention : les primitives d'accès direct sont interdites; on ne peut pas relire les informations d'un tube car **la lecture dans un tube est destructive**.
- Dans le cas où tous les descripteurs associés aux processus susceptibles de lire dans un tube sont fermés, un processus qui tente de lire reçoit le signal 13 SIGPIPE et se trouve interrompu s'il ne traite pas ce signal.

Tubes nommés

- Tube qui possède un nom dans le SGF.
- Tous processus est autorisé à l'ouvrir à condition de connaître son nom et de posséder les droits d'accès.
- Un processus ayant ouvert un tube en écriture est suspendu tant qu'il n'y a pas de processus lecteur.
- Le tube persiste même après la terminaison du processus qui l'a créé
- Les données du tube ne sont pas gardés d'une session à l'autre.
- Un tube nommé conserve toutes les propriétés d'un tube : taille limitée, lecture destructive, lecture/écriture réalisées en mode FIFO.
- Un tube nommé permet à des processus sans lien de parenté dans le système de communiquer.

Tubes nommés

- Création par la primitive mknod qui permet de créer les fichiers spéciaux ou par mkfifo
  - mknod était préalablement réservée au super-utilisateur
- Ouverture possible (Open) par les processus connaissant le nom du tube
  - Par défaut, ouverture bloquante : lecteur et écrivain s'attendent → synchronisation
- Suppression du tube lorsque explicitement demandé et pas d'ouverture en cours
  - Si suppression alors qu'il existe des lecteurs et écrivain, fonctionnement comme un fichier ordinaire
- Primitives correspondantes :
  - int mknod(const char *nom_fich, mode_t mode, dev_t n_periph)

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

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

- Le paramètre ref définit le chemin d'accès au tube et le paramètre mode les droits d'accès des différents utilisateurs à cet objet,

utbm
université de technologie
Belfort-Montbéliard

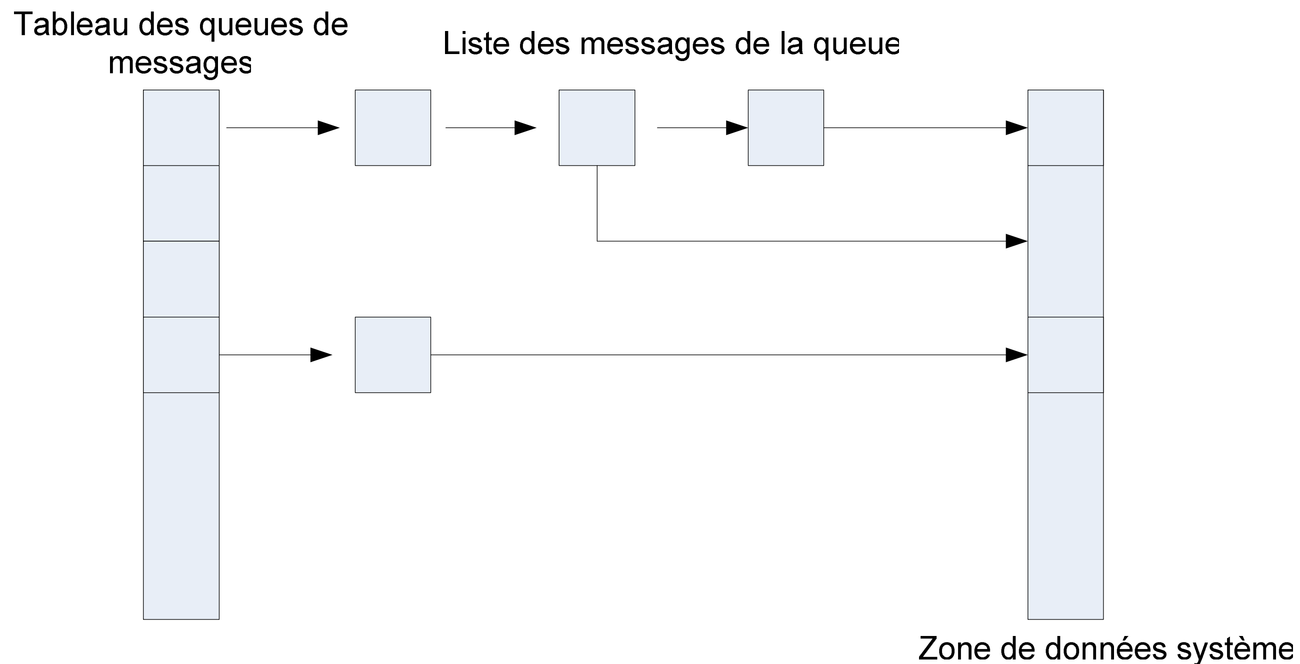
Remarque

- Les instructions read() et write() sont bloquantes:
  - tentative de lecture dans un FIFO vide: le processus attend un remplissage suffisant du tube.
  - tentative d'ecriture dans un FIFO plein: le processus attend que le FIFO soit suffisamment vide.

Les Files de messages

- Implémentation du concept de boite à lettres.
- Le but est d'échanger ,de façon asynchrone, des données stockées dans le noyau du système, sous forme de paquets d'octets identifiables.
- Mode de communication opposé à celui des tubes parce qu'une opération de lecture ne peut extraire que le produit d'une opération d'écriture.
- Ensemble de messages accessibles par plusieurs processus
  - Identifié par un numéro (Unix) externe
  - Chaque message contient un type et un corps
- Les communications se font à travers deux opérations fondamentales :
  - send (destinataire,message)
  - receive (source, message)
- Message de taille fixes ou variables
- Opérations d'envoi et de réception peuvent être :
  - directes entre les processus
  - indirectes par l'intermédiaire d'une boite aux lettres
  - Synchrone ou asynchrone

Les Files de messages

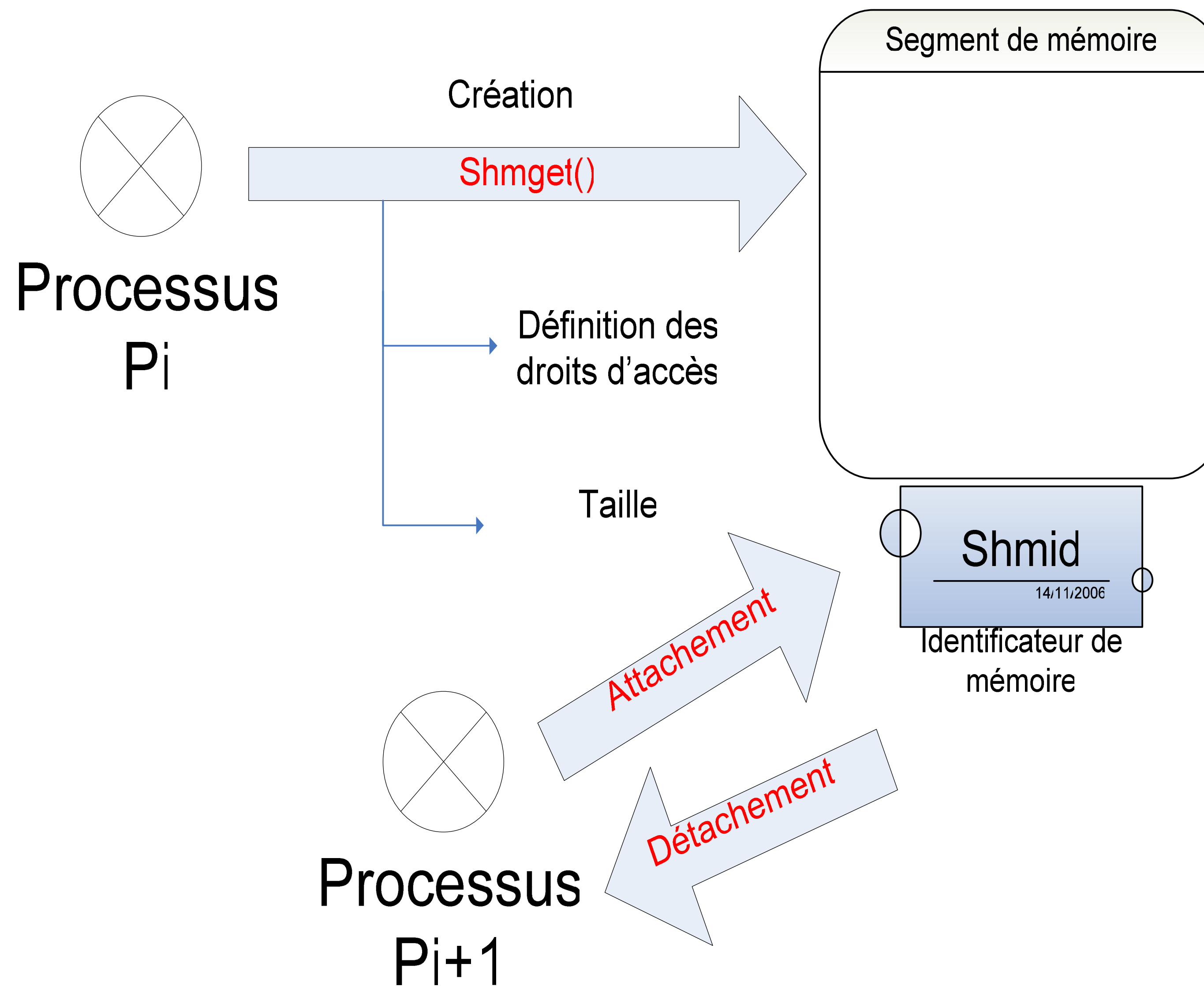


Une file de messages, même privée, doit être détruite explicitement : elle n'est pas supprimée par un appel de exit() par le processus créateur

Les segments de mémoire partagée

- Principe :
  - Un segment de mémoire partagée est une zone d'octets désignée par un pointeur (adresse de base).
  - L'expéditeur et le destinataire se partagent une zone de mémoire et dès que les données sont placées par l'expéditeur, elles sont instantanément disponibles pour le destinataire.
  - Un sémaphore ou un message sert à empêcher le destinataire de lire trop tôt les données, et sert à empêcher l'expéditeur d'écrire de nouvelles données avant que le lecteur n'ait terminé la lecture.
  - Les processus partagent des pages physiques par l'intermédiaire de **leur espace d'adressage**. Il n'y a donc plus de recopie des informations. Les pages partagées par les processus sont des ressources critiques dont les accès doivent être synchronisés (au moyen de sémaphore par exemple).
  - L'adresse de base de la mémoire partagée, vue de deux processus, n'a aucune obligation d'être identique : les processus ne sont pas obligés d'utiliser la zone d'octets à partir de son "début".

Mécanisme de création segment



Les segments de mémoire partagée

– Dispositions d'Unix :

1. Il peut y avoir plusieurs segments partagés, chacun étant partagé par un ensemble de processus actifs.
2. Un processus peut accéder à plusieurs segments de mémoire partagée.
3. Les segments de mémoire partagée ont une existence indépendante des processus.
4. Un segment, une fois créé, pourra continuer d'exister même si aucun processus ne l'utilise.
5. Un processus aura la possibilité de demander le rattachement d'un segment à son espace d'adressage, après quoi il pourra accéder aux données qu'il contient comme il le fait pour les autres données, c'est-à-dire via leur adresse (et ceci sans passer par l'intermédiaire d'un appel système).

- **Procédure :**
  - Un segment est d'abord créé à l'extérieur de l'espace d'adressage de tout processus par un premier processus
    - le processus décrit les droits d'opérations globaux pour le segment de mémoire partagée, décrit sa taille en octets, et à la possibilité de spécifier que l'attachement d'un processus a ce segment de mémoire se fera en lecture seule.
  - Chaque processus voulant accéder au segment va exécuter une primitive pour le translater dans son propre espace d'adressage
    - Lorsqu'un segment de mémoire partagée est crée, un processus peut réaliser deux opérations:
      - attachement de mémoire partagée
      - détachement de mémoire partagée.

IPC System V

- Trois types d'objets système :
  - Files de messages : échange de flots de données formatées
  - Mémoire partagée : mise à disposition d'un espace commun
  - Sémaphores : outils de synchronisation

- Utilisation d'une clé pour identifier les objets système :
  Primitive key_t ftok(char *nom_fichier, char nom_projet)

- Gestion des ressources :
  - Obtention d'information sur les objets existants : ipcs
  - Suppression d'un objet : ipcrm {sem|shm|msg} {id}

utbm
université de technologie Belfort-Montbéliard

IPC System V

- Mécanisme en dehors du SGF
- il existe une table système par type d'objet.
  - On ne désigne donc pas ces objets par des descripteurs.
  - On ne peut pas rediriger les E/S standards d'un processus sur un objet de ce type.
- Apport de fonctionnalités nouvelles et augmentation des performances en matière de partage d'objets.
- Pour partager un objet IPC, les processus partagent la clé externe qui lui est associée et utilisent les méthodes propres à chaque type d'objet IPC (notion de "classe d'objets").

utbm
université de technologie Belfort-Montbéliard

Consultation des tables systèmes

- La commande ipcs permet de consulter les tables systèmes, alors que la commande ipcrm supprime une entrée de la table
- chaque objet IPC possède :
  - Un identificateur interne (ID) équivalent pour les fichiers aux descripteurs dans la table des fichiers ouverts de chaque processus.
  - Une clé externe équivalente aux références (noms symboliques) pour les fichiers.
- Pour qu'un processus utilise un objet IPC, il doit connaître son ID. Pour cela il utilise la clé externe associée à cet objet.

Consultation des tables systèmes

- La fonction ftok permet à l'utilisateur de créer ses propres clés, en reliant l'espace de nommage des objets IPC à celui du système de gestion des fichiers.
- Une clé unique va être créée à partir d'une référence de fichier et d'une constante.

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

key_t  ftok  (char pathname, char proj) ;
                                /* ou "int proj"  */
```

ce paramètre permet pour un même nom de fichier du SGF d'UNIX de générer plusieurs clés différentes.

Consultation des tables systèmes

- La clé est calculée par la fonction suivante (opérations bit à bit) :
- Une clé unique va être créée à partir d'une référence de fichier et d'une constante.

```
Cle = (st_ino & 0xFFFF) | ((st_dev & 0xFF) << 16) | (proj << 24)
```

- Ce calcul est réalisé par une combinaison entre le numéro de l'i-noeud du fichier, le numéro du périphérique sur lequel se trouve le fichier et le dernier paramètre (proj), de telle manière qu'un nombre unique soit généré.

utbm
université de technologie Belfort-Montbéliard

IPC System V : constantes

<sys/ipc.h>

- Drapeaux :
  - IPC_CREAT : créer une entrée si elle n'existe pas
  - IPC_EXCL : échouer si elle existe
  - IPC_NOWAIT : ne pas attendre
  - IPC_PRIVATE : clé privée, pas de lien avec le système de fichiers
- Opérations :
  - IPC_RMID : suppression
  - IPC_SET : mise à jour des attributs
  - IPC_STAT : lecture des attributs
- Permissions :
  - Analogues à celles du système de fichiers UNIX

IPC System V : Files de messages

– Modèle de structure générique d'un message

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

Table des files de messages

```
struct msqid_ds {
        struct ipc_perm     msg_perm;    /*   droits d'acces
        struct __msg        *msg_first;  /*   pointeur sur le 1er message
        struct __msg        *msg_last;   /*   pointeur sur le dernier message
        int                 msg_qnum;    /*   nombre de message dans la file
        ...
        time-t              shm-stime;   /*   date du derniere émission (msgsnd)
        time-t              shm-rtime;   /*   date du dernier réception (msgrcv)
        time-t              shm-ctime;   /*   date dernier changement par msgctl
        ...

}
```

IPC System V : Files de messages

- Création ou trouver une file existante à partir de la clé : retourne un n° de file

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

- Un appel à cette fonction renvoie l'identification d'une file de messages ou –1 en cas d'erreur.

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

int idMsg;

idMsg = msgget (ftok(« /opt/app/msg/essai», 1), IPC_CREAT | IPC_EXCL | 0666);
```

IPC System V : Files de messages

- Émission (bloquante si file pleine et d ne contient pas IPC_NOWAIT) :

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

- Envoi dans la file d'identification msgid le message pointé par p_msg.
- Lg : longueur du message
- La primitive renvoie la valeur 0 en cas de succès et –1 sinon.
- Le paramètre optionnel peut avoir la valeur IPC_NOWAIT. Dans ce cas si la file est pleine, l'appel à la primitive msgsnd n'est pas bloquant alors qu'il l'est par défaut.

IPC System V : Files de messages

- Réception (bloquante si file vide et d ne contient pas IPC_NOWAIT)

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

- Un appel cette primitive est une demande de lecture dans la file d'identification **msgid**. Le pointeur **p_msg** pointe sur une zone mémoire susceptible de recevoir un message de longueur inférieure ou égale à **lg** (les octets occupés par le type ne sont pas comptabilisés).
- Le paramètre option est une combinaison des constantes :
  - **IPC_NOWAIT** : si la file ne contient pas de message du type attendu, l'appel est non bloquant.
  - **MSG_NOERROR** : si le texte du message à extraire est de longueur supérieure à lg, le message extrait est tronqué (alors que par défaut une erreur est détectée).
- Le paramètre type permet de spécifier le type du message à extraire :
  - **> 0** le message le plus vieux de type égal à type est extrait de la file
  - **0** le message le plus vieux quel que soit son type est extrait de la file
  - **< 0** le message le plus vieux du type le plus petit inférieur ou égal à type est extrait. Ce mécanisme permet de définir des priorités entre les messages.
- La valeur renvoyée en cas de réussite est la longueur du texte du message qui a été écrit à l'adresse **p_msg**.

utbm
université de technologie Belfort-Montbéliard

IPC System V : Files de messages

- Elle permet d'accéder aux informations contenues dans l'entrée de la table des files du système associée à une file particulière et d'en modifier certains attributs.

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

- Les valeurs possibles du paramètre ***op*** sont :
  - **IPC_RMID** demande la suppression de la file de message identifié par ***msgid***;
  - **IPC_STAT** demande de chargement à l'adresse ***p_msqid*** des informations contenues dans l'entrée de la table des files de message correspondant à l'identification ***msgid***.

    IPC_SET : demande de modification de l'entrée dans la table des files de messages associée à l'identification ***msgid*** avec les valeurs définies dans l'objet d'adresse ***p_ msqid***.
- Les seuls champs modifiables sont msg_perm.uid, msg _perm.gid et msg _perm.mode.
- La primitive renvoie la valeur 0 en cas de succès et –1 sinon.

IPC System V : Mémoire partagée

- Entrée dans la table des segments de mémoire

```
struct shmid_ds {
            struct ipc-perm         shm-perm;     /*      droits d'acces
            int                     shm-segsz;    /*      taille du segment en octets
            pid-t                   shm-lpid;     /*      pid ayant fait la derniere opération*/
            pid-t                   shm-cpid;     /*      pid createur */
            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                  shm-ctime;    /*      date dernier changement par shmctl
      }
```

IPC System V : Mémoire partagée

- **Création :** Tout nouveau segment de mémoire créé a comme propriétaire et créateur le propriétaire effectif du processus et comme groupe propriétaire et créateur le groupe propriétaire effectif du processus.

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

- Le paramètre ***taille*** spécifie la taille du segment : dans le cas où la clé donnée en paramètre est associée à un segment existant, la taille donnée en paramètre doit être inférieure on égale à celle du segment.
- Le paramètre **option** est construit comme une combinaison (opérateur ou bit à bit) des constantes : IPC_CREAT, IPC_EXCL et des droits d'accès.
- Les neufs bits de droite du paramètre **option** sont les droits d'accès ; l'autorisation en lecture permet seulement d'extraire les données du segment, alors que l'autorisation en écriture permet à la fois le stockage et l'extraction.

IPC System V : Mémoire partagée

Exemple de création

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int idShm;
idShm = semget(ftok(‘’/opt/app/shm1’’, 1), *sizeof(int), IPC_CREAT | IPC_EXCL | 0666);
```

Valeur de option	Explication
IPC_PRIVATE | droits	Un nouveau segment de mémoire privé est systématiquement créé.
IPC_CREAT | IPC_EXCL | droits	Création d’un nouveau segment de mémoire associé à la **clé** passée en paramètre. Si le segment existe déjà, une erreur est détectée.
IPC_CREAT | droits	Récupération de l’identification d’un segment de mémoire existant dont la **clé** est passée en paramètre. Si le segment n’existe pas, il est créé.

IPC System V : Mémoire partagée

- Attachement : C'est la primitive qui permet à un processus connaissant l'identification d'un segment de mémoire de lui associer une adresse dans son espace d'adressage au moyen de laquelle il pourra y accéder directement.

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

- Un appel à la primitive **shmatt** correspond à une demande d'attachement d'un segment identifié par **shmid** à l'adresse **adr** de l'espace d'adressage du processus.
- La valeur de retour est l'adresse où l'attachement a été effectivement réalisé, c'est-à-dire au premier octet du segment et –1 en cas d'échec.
- **Adresse choisie par le système**
  - Si l'adresse passée en argument est égale à NULL (**adr** = NULL) c'est le système d'exploitation qui choisira l'adresse.
  - Cette solution garantie d'une part que l'adresse sera correctement construite et d'autre part permettra la portabilité de l'application.

IPC System V : mémoire partagée

- La primitive **shmdt** correspond à une demande de détachement du segment de mémoire dont la demande d'attachement par la primitive **shmat** a été réalisée à l'adresse ***adr***.

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

- Si une demande de, suppression de l'entrée dans la table des segments a été formulée et qu'après ce détachement le nombre d'attachements du segment devient nul, le segment est effectivement libéré.
- La terminaison d'un processus entraîne le détachement de tous les segments qu'il a préalablement attachés.
- Libération de l'espace mémoire uniquement lors du dernier détachement
- Positionnement d'un drapeau interdisant des attachements
- Retour : 0 ou -1

IPC System V : mémoire partagée

- Cette primitive permet de réaliser les différentes opérations de contrôle sur les segments de mémoire partagée.

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

- Les valeurs possibles du paramètre ***op*** sont :
- **IPC_RMID** demande de suppression du segment identifié par ***shmid*** ; la suppression sera différée si le segment est encore attaché dans un processus.
- **IPC_STAT** demande de chargement à l'adresse ***p_shmid*** des informations contenues dans l'entrée de la table des segments correspondant à l'identification ***shmid***.
- **IPC_SET** : demande de modification de l'entrée dans la table des segments 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.