Programmation Système et Réseau

Communication Inter-Processus (IPC)
Les Files de messages

Équipe pédagogique

CY Tech

Bibliographie - Sitographie

- Slides de Juan Ángel Lorenzo del Castillo, Seytkamal Medetov et Son Vu
- Linux : Programmation système et réseau de Joëlle Delacroix Dunod
- Système d'exploitation de Andrew Tanenbaum, Pearson Education

Programmation Système et Réseau

Communication Inter-Processus (IPC)
Les Files de messages

Équipe pédagogique

CY Tech

Bibliographie - Sitographie

- Slides de Juan Ángel Lorenzo del Castillo, Seytkamal Medetov et Son Vu
- Linux : Programmation système et réseau de Joëlle Delacroix Dunod
- Système d'exploitation de Andrew Tanenbaum, Pearson Education
- Slides de A. Silberschatz, P. B. Galvin et G. Gagne
- Slides de A. Frank - P. Weisberg (Introduction to Operating System)
- Slides H. Bourzoufi , Arnaud Lewandowski, François Bourdon, Joelle Delacroix, Mirian Halfeld-Ferrari, A. B. Dragut
- `https://www.lri.fr/~anab/teaching/DevLog/cours4-threads.pdf`
- `http://cours.polymtl.ca/inf2610/documentation/notes/chap4.pdf`

Bibliographie - Sitographie

- Les files de messages IPC : http://supertos.free.fr/supertos.php\?page=37
- La fonction ftok() : http://manpagesfr.free.fr/man/man3/ftok.3.html et http://supertos.free.fr/supertos.php\?page=36
- Manuel de msgctl() : http://manpagesfr.free.fr/man/man2/msgctl.2.html#lbAB
- Communication Interprocessus : https://mtodorovic.developpez.com/linux/programmation-avancee/\?page=page_5
- Inter Processus Communications (I.P.C.) : http://www-igm.univ-mlv.fr/~dr/NCS/node197.html

Rappels

Par communication inter-processus on entend :

- Existence de plusieurs processus sur la même machine travaillant “simultanément”.

Rappels

Par communication inter-processus on entend :

- Existence de plusieurs processus sur la même machine travaillant “simultanément”.
- Échange d’information entre ces processus.

Rappels

Par communication inter-processus on entend :

- Existence de plusieurs processus sur la même machine travaillant “simultanément”.
- Échange d’information entre ces processus.
- Informe les processus de l’occurrence d’événements asynchrones et permet de faire exécuter à un processus une action relative à ces événements.

Les IPC (Inter Processus Communication)

- Fournit des facilités de communication entre processus sur une même machine

Les IPC (Inter Processus Communication)

- Fournit des facilités de communication entre processus sur une même machine
- Externes au SGF

Les IPC (Inter Processus Communication)

- Fournit des facilités de communication entre processus sur une même machine
- Externes au SGF
- 3 fonctions : Files de messages, segments de mémoire partagée, sémaphores

Les IPC (Inter Processus Communication)

- Fournit des facilités de communication entre processus sur une même machine
- Externes au SGF
- 3 fonctions : Files de messages, segments de mémoire partagée, sémaphores
- Identifiés et manipulés à travers une clé numérique

Les IPC (Inter Processus Communication)

- Fournit des facilités de communication entre processus sur une même machine
- Externes au SGF
- 3 fonctions : Files de messages, segments de mémoire partagée, sémaphores
- Identifiés et manipulés à travers une clé numérique
- Interface #include <sys/ipc.h> #include <sys/shm.h> #include <sys/types.h>

Les IPC (Inter Processus Communication)

- Fournit des facilités de communication entre processus sur une même machine
- Externes au SGF
- 3 fonctions : Files de messages, segments de mémoire partagée, sémaphores
- Identifiés et manipulés à travers une clé numérique
- Interface #include <sys/ipc.h> #include <sys/shm.h> #include <sys/types.h>
- Deux commandes : ipcs (voir un IPC) et ipcrm (supprimer un IPC)

Les IPC (Inter Processus Communication)

- type.h définit des types par rapport à la machine (POSIX peut ne pas l'exiger suivant les implémentations de Linux)

Les IPC (Inter Processus Communication)

- type.h définit des types par rapport à la machine (POSIX peut ne pas l'exiger suivant les implémentations de Linux)
- ipc.h :

```
type desf long mtyp_t; /* ipc type message */
struct ipc_perm {
uid_t uid;
/* identification du proprietaire */
gid_t gid;
/* identification du groupe */
uid_t cuid;
/* identification du createur */
gid_t uguid;
/* identification du groupe a la creation */
mode_t mode; /* mode d'acces */
}
```

Les IPC (Inter Processus Communication)

- type.h définit des types par rapport à la machine (POSIX peut ne pas l'exiger suivant les implémentations de Linux)
- ipc.h :

```
type desf long mtyp_t; /* ipc type message */
struct ipc_perm {
uid_t uid;
/* identification du proprietaire */
gid_t gid;
/* identification du groupe */
uid_t cuid;
/* identification du createur */
gid_t uguid;
/* identification du groupe a la creation */
mode_t mode; /* mode d'acces */
}
```

- mode : droits classiques d'écriture ou lecture (exemples 0400 lecture par le propriétaire, 0020 écriture par le groupe ...)

Les IPC (Inter Processus Communication)

- un identifiant unique de type key_t

Les IPC (Inter Processus Communication)

- un identifiant unique de type key_t
  - clé unique qui permet d'accéder à une ressource depuis plusieurs processus

Les IPC (Inter Processus Communication)

- un identifiant unique de type key_t
  - clé unique qui permet d'accéder à une ressource depuis plusieurs processus
  - valeur numérique existante de 2 manières :

Les IPC (Inter Processus Communication)

- un identifiant unique de type key_t
  - clé unique qui permet d'accéder à une ressource depuis plusieurs processus
  - valeur numérique existante de 2 manières :
    - ★ figée dans le code source

Les IPC (Inter Processus Communication)

- un identifiant unique de type key_t
  - clé unique qui permet d'accéder à une ressource depuis plusieurs processus
  - valeur numérique existante de 2 manières :
    - ★ figée dans le code source
    - ★ calculée par le système à partir d'un référence commune : nom de fichier de type entier

```
key_t ftok (const char *ref, int
    numero);
```

Les IPC (Inter Processus Communication)

- un identifiant unique de type key_t
  - clé unique qui permet d'accéder à une ressource depuis plusieurs processus
  - valeur numérique existante de 2 manières :
    - ★ figée dans le code source
    - ★ calculée par le système à partir d'un référence commune : nom de fichier de type entier

```
key_t ftok (const char *ref, int
    numero);
```

  - une commande ipcs permet de visualiser toutes les IPCs d'un système

Files de messages

Les files de messages ou MQ

Files de messages

- Contrairement aux tubes (pipes), une file de messages (ou queues de messages) contient des données structurées composées

Files de messages

- Contrairement aux tubes (pipes), une file de messages (ou queues de messages) contient des données structurées composées
  - d'un type de message

Files de messages

- Contrairement aux tubes (pipes), une file de messages (ou queues de messages) contient des données structurées composées
  - d'un type de message
  - d'un message de longueur variable

Files de messages

- Contrairement aux tubes (pipes), une file de messages (ou queues de messages) contient des données structurées composées
  - d'un type de message
  - d'un message de longueur variable
- A la différence des tubes, il sera possible d'attendre uniquement des messages d'un type donné : s'il n'y a que des messages d'un autre type, le processus sera bloqué (attente passive)

Files de messages

- Contrairement aux tubes (pipes), une file de messages (ou queues de messages) contient des données structurées composées
  - d'un type de message
  - d'un message de longueur variable
- A la différence des tubes, il sera possible d'attendre uniquement des messages d'un type donné : s'il n'y a que des messages d'un autre type, le processus sera bloqué (attente passive)
- Comme les tubes, les files de message sont gérées selon un algorithme FIFO : on lit toujours le message du type cherché qui a séjourné le plus longtemps dans la file

Files de messages

- Contrairement aux tubes (pipes), une file de messages (ou queues de messages) contient des données structurées composées
  - d'un type de message
  - d'un message de longueur variable
- A la différence des tubes, il sera possible d'attendre uniquement des messages d'un type donné : s'il n'y a que des messages d'un autre type, le processus sera bloqué (attente passive)
- Comme les tubes, les files de message sont gérées selon un algorithme FIFO : on lit toujours le message du type cherché qui a séjourné le plus longtemps dans la file
- L'attente est par défaut bloquante, mais, comme pour les tubes, et si cela est nécessaire, le processus peut ne pas attendre en positionnant le drapeau IPC_NOWAIT. Il recevra un code d'erreur lui indiquant s'il a ou non obtenu un message.

Files de messages

- Contrairement aux tubes (pipes), une file de messages (ou queues de messages) contient des données structurées composées
  - d'un type de message
  - d'un message de longueur variable
- A la différence des tubes, il sera possible d'attendre uniquement des messages d'un type donné : s'il n'y a que des messages d'un autre type, le processus sera bloqué (attente passive)
- Comme les tubes, les files de message sont gérées selon un algorithme FIFO : on lit toujours le message du type cherché qui a séjourné le plus longtemps dans la file
- L'attente est par défaut bloquante, mais, comme pour les tubes, et si cela est nécessaire, le processus peut ne pas attendre en positionnant le drapeau IPC_NOWAIT. Il recevra un code d'erreur lui indiquant s'il a ou non obtenu un message.
- Comme les tubes, les files de messages ont des droits d'accès de type Unix (rwx, rwx, rwx)

Files de messages

- Ce mécanisme permet l'échange d'information entre processus en réalisant une implémentation du concept de boite aux lettres.

Files de messages

- Ce mécanisme permet l'échange d'information entre processus en réalisant une implémentation du concept de boite aux lettres.
- C'est un mode de communication structuré, utilisant des paquets identifiables et indivisibles, à la différence des tubes qui utilisent un flot continu de données.

Files de messages

- Ce mécanisme permet l'échange d'information entre processus en réalisant une implémentation du concept de boite aux lettres.
- C'est un mode de communication structuré, utilisant des paquets identifiables et indivisibles, à la différence des tubes qui utilisent un flot continu de données.
- Une demande de lecture de message extraira exactement le produit d'une opération d'écriture.

Files de messages

- Ce mécanisme permet l'échange d'information entre processus en réalisant une implémentation du concept de boite aux lettres.
- C'est un mode de communication structuré, utilisant des paquets identifiables et indivisibles, à la différence des tubes qui utilisent un flot continu de données.
- Une demande de lecture de message extraira exactement le produit d'une opération d'écriture.
- L'implémentation des passages de messages intègre un mécanisme permettant le multiplexage :

Files de messages

- Ce mécanisme permet l'échange d'information entre processus en réalisant une implémentation du concept de boite aux lettres.
- C'est un mode de communication structuré, utilisant des paquets identifiables et indivisibles, à la différence des tubes qui utilisent un flot continu de données.
- Une demande de lecture de message extraira exactement le produit d'une opération d'écriture.
- L'implémentation des passages de messages intègre un mécanisme permettant le multiplexage :
  - chaque message transmis possède un type ;
  - ainsi un processus peut extraire un message d'une file en utilisant ce type comme critère de sélection.

Files de messages

- Un identificateur MQ : entier fourni par le système à la création

Files de messages

- Un identificateur MQ : entier fourni par le système à la création
  - msqid (similaire à un numéro de descripteur)

Files de messages

- Un identificateur MQ : entier fourni par le système à la création
  - msqid (similaire à un numéro de descripteur)
- Un ensemble de messages typés

Files de messages

- Un identificateur MQ : entier fourni par le système à la création
  - msqid (similaire à un numéro de descripteur)
- Un ensemble de messages typés
  - déposés par un ou plusieurs processus

Files de messages

- Un identificateur MQ : entier fourni par le système à la création
  - msqid (similaire à un numéro de descripteur)
- Un ensemble de messages typés
  - déposés par un ou plusieurs processus
  - relus par un ou plusieurs processus

Files de messages

- Un identificateur MQ : entier fourni par le système à la création
  - msqid (similaire à un numéro de descripteur)
- Un ensemble de messages typés
  - déposés par un ou plusieurs processus
  - relus par un ou plusieurs processus
- Aucune filiation exigée

Files de messages

- Un identificateur MQ : entier fourni par le système à la création
  - msqid (similaire à un numéro de descripteur)
- Un ensemble de messages typés
  - déposés par un ou plusieurs processus
  - relus par un ou plusieurs processus
- Aucune filiation exigée
- Lecture destructrice

Files de messages

- Un identificateur MQ : entier fourni par le système à la création
  - msqid (similaire à un numéro de descripteur)
- Un ensemble de messages typés
  - déposés par un ou plusieurs processus
  - relus par un ou plusieurs processus
- Aucune filiation exigée
- Lecture destructrice
- Structure associée : fichier msg.h

Files de messages

- Les messages étant typés, chaque processus peut choisir les messages qu'il veut lire (extraire de la file)

Files de messages

- Les messages étant typés, chaque processus peut choisir les messages qu'il veut lire (extraire de la file)
- Le fichier header <sys/msg.h> contient (entre autres) une structure générique msgbuf pour les messages, définie comme suit :

```
struct msgbuf {
long mtype;     /* type of received/sent message
    :\__syscall_slong_t mtype; */
char mtext[1]; /* text of the message */
}
```

Files de messages

- Cette structure constitue un modèle : une application utilisera probablement une structure pour les messages différente, mais elle devra contenir :

Files de messages

- Cette structure constitue un modèle : une application utilisera probablement une structure pour les messages différente, mais elle devra contenir :
  - un première champ de type long qui constitue le type de message (entier strictement positif), il fournit aux processus utilisant la file de messages un critère de sélection des messages (multiplexage).

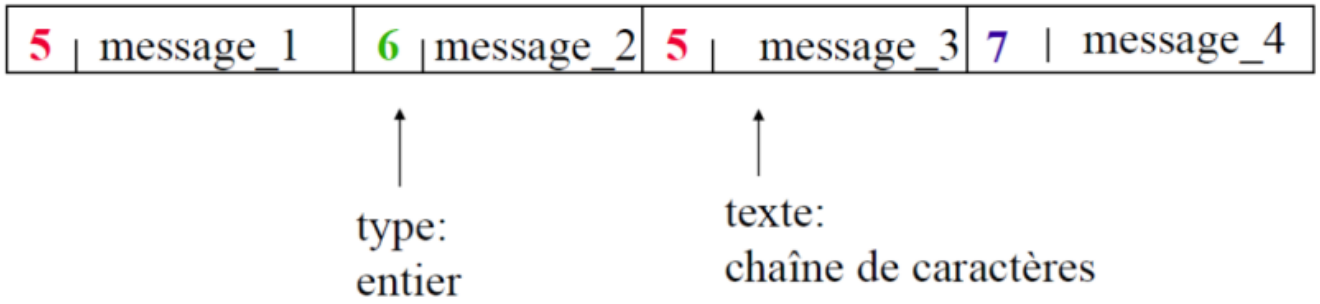
Files de messages

- Cette structure constitue un modèle : une application utilisera probablement une structure pour les messages différente, mais elle devra contenir :
  - un première champ de type long qui constitue le type de message (entier strictement positif), il fournit aux processus utilisant la file de messages un critère de sélection des messages (multiplexage).
  - un second champ constitué d'une suite d'octets consécutifs en mémoire. Il peut correspondre à des objets de type quelconque : on peut utiliser par exemple un tableau ou une structure.

Files de messages

- Cette structure constitue un modèle : une application utilisera probablement une structure pour les messages différente, mais elle devra contenir :
  - un première champ de type long qui constitue le type de message (entier strictement positif), il fournit aux processus utilisant la file de messages un critère de sélection des messages (multiplexage).
  - un second champ constitué d'une suite d'octets consécutifs en mémoire. Il peut correspondre à des objets de type quelconque : on peut utiliser par exemple un tableau ou une structure.
    - ★ il n'est par contre pas possible de réaliser des indirections par le biais de pointeurs !

Files de messages



Files de messages

Utilisation

- On obtient une file de messages en utilisant la fonction msgget :

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

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

- Cette fonction permet la création d'une file de messages et renvoie l'identifiant de cette file (rôle analogique à la fonction open pour les fichiers) ou -1 en cas d'erreur.

Utilisation

- Ses arguments sont les suivants :
  - key (de type key_t), il s'agit d'une clé IPC obtenue par exemple par la fonction ftok. On peut aussi utiliser la clé symbolique IPC_PRIVATE, auquel cas une nouvelle file est crée (pourvu que IPC_CREAT soit présent dans le second argument)

Files de messages

Utilisation

- Ses arguments sont les suivants :
  - key (de type key_t), il s'agit d'une clé IPC obtenue par exemple par la fonction ftok. On peut aussi utiliser la clé symbolique IPC_PRIVATE, auquel cas une nouvelle file est crée (pourvu que IPC_CREAT soit présent dans le second argument)
  - msgflg (de type int), il doit contenir les droits d'accès en cas de création de la file de messages (usage traditionnel d'un nombre octal de type 0664 par exemple). Il existe aussi des valeurs symboliques comme IPC_CREAT ou IPC_EXCL qu'il faut composer éventuellement avec les droits à l'aide d'un OU bits à bits.
  - man msgget (`http://manpagesfr.free.fr/man/man2/msgget.2.html`)

Files de messages

Contrôle d'une file de message

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

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

- La fonction msgctl permet d'effectuer l'opération indiquée par cmd sur une file de message indiquée par le première argument ayant l'identifiant msqid.

Files de messages

Contrôle d'une file de message

- La fonction renvoi 0 s'il réussi, ou -1 s'il échoue auquel cas errno contient le code d'erreur.
  - msqid (de type int), identificateur de la fdm visée
  - cmd (de type int), type d'opération à effectuer sur la fdm :
    - IPC_RMID : détruire la fdm
    - IPC_STAT : copier les information depuis la structure représentant la fdm dans la structure pointée par buf
    - IPC_SET : écrire la valeurs de certains champs de la structure msqid_ds pointée par buf dans la structure représentant la fdm
    - IPC_INFO, MSG_INFO, MSG_STAT, ...
  - buf, pour récupérer la table associée à la fdm (la page suivante)
  - man msgctl (`http://manpagesfr.free.fr/man/man2/msgctl.2.html`)

Files de messages

Contrôle d'une file de message

La structure de données msqid_ds est définie dans <sys/msg.h> de la manière suivante :

```
struct msqid_ds {
 struct ipc_perm msg_perm;     //Appartenance,
     permissions
 time_t          msg_stime;    // Heure dernier msgsnd
 time_t          msg_rtime;    // Heure dernier msgrcv
 time_t          msg_ctime;    // Heure derniere modif
 unsigned long   __msg_cbytes; // Nb actuel octets
     file
 msgqnum_t       msg_qnum;     // Nb actuel messages
     file
 msglen_t        msg_qbytes;   // Nb max d'octets file
 pid_t           msg_lspid;    // PID dernier msgsnd
 pid_t           msg_lrpid;    // PID dernier msgrcv
};
```

Files de messages

Contrôle d'une file de message La structure ipc_perm est définie dans <sys/ipc.h> de la façon suivante (les champs mis en évidence sont configurables en utilisant IPC_SET) :

```
struct ipc_perm {
    key_t key;              /* Cle fournie a
        msgget(2) */
    uid_t uid;              /* UID effectif
        proprietaire */
    gid_t gid;              /* GID effectif
        proprietaire */
    uid_t cuid;             /* UID effectif createur
        */
    gid_t cgid;             /* GID effectif createur
        */
    unsigned short mode;    /* Permissions */
    unsigned short seq;     /* Numero de sequence */
};
```

Files de messages

Émission/envoi de messages

- Un message peut être déposé dans une fdm à l'aide de la fonction msgsnd. Cette fonction renvoie 0 en cas de succès ou -1 en cas d'erreur :

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

int msgsnd( int msqid, const void *msgp, size_t
    msgsz, int msgflg);
```

Files de messages

Émission/envoi de messages

- Ses arguments sont les suivants :
  - msqid (de type int), identificateur de la fdm (valeur renvoyée par la fonction msgget).
  - msgp (de type const void*), il s'agit d'un pointeur sur une structure analogue à msgbuf comme vue précédemment
  - msgsz (de type int), cet argument indique la taille du message en octets diminuée de la taille du message. Il s'agit donc de ma taille en octets du corps du message.
  - msgflg (de type int), s'il n'est pas nul, il peut prendre la valeur symbolique IPC_NOWAIT : dans ce cas, un appel à msgsnd lorsque la fdm est pleine n'est pas bloquant
- man msgsnd (`http://manpagesfr.free.fr/man/man2/msgop.2.html`)

Files de messages

Réception de message

- L'extraction d'un message d'une file peut être obtenu par la fonction msgrcv. L'appel système msgrcv supprime un message depuis la file indiquée par msqid et le place dans le tampon pointé par msgp. La valeur retournée est la taille en octets du message extrait en cas de succès ou -1 en cas d'échec :

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

ssize_t msgrcv(int msqid, void *msgp, size_t
    msgsz, long msgtyp, int msgflg);
```

Files de messages

Réception de message

- Ses arguments sont les suivants :
  - msqid, identificateur de la fdm (valeur renvoyée par la fonction msgget)
  - msgp, il s'agit d'un pointeur sur une structure analogue à msgbuf comme vue précédemment
  - msgsz, cet argument indique la taille en octets du corps du message.
  - msgtyp, il s'agit du type de message à extraire. Si msgtyp > 0, le message extrait sera le plus ancien de ce type dans la file. Si msgtyp $= 0$, le plus ancien message de la file (quel que soit son type) est extrait. Si msgtyp < 0, le message extrait sera plus ancien de la file dont le type est inférieur ou égal à la valeur absolue de cet argument.
  - msgflg (de type int), s'il n'est pas nul, il peut prendre la valeur symbolique IPC_NOWAIT : dans ce cas, un appel à msgrcv lorsque la fdm ne contient pas de message de type attendu n'est pas bloquant
- man msgrcv (`http://manpagesfr.free.fr/man/man2/msgop.2.html`)

Commandes liées aux IPC système V

- Il existe essentiellement deux commandes concernant les IPC système V : ipcs et ipcrm

Files de messages

Commandes liées aux IPC système V

- Il existe essentiellement deux commandes concernant les IPC système V : ipcs et ipcrm
- La première permet la consultation des tables IPC du système. Une exécution dans un terminal fournira une sortie analogue à ce qui suit :

```
smv@smv:~/EISTI/Enseignement/Progr_Syst_et_Reseau/4-File_de_messages/exemple$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x30051efa 1          smv        640        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32771      smv        600        53248      2          dest
0x00000000 32772      smv        600        53248      2          dest
0x00000000 59         smv        600        524288     2          dest
0x00000000 60         smv        600        524288     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

Files de messages

Commandes liées aux IPC système V

```
smv@smv:~/EISTI/Enseignement/Progr_Syst_et_Reseau/4-File_de_messages/exemple$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x30051efa 1          smv        640        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32771      smv        600        53248      2          dest
0x00000000 32772      smv        600        53248      2          dest
0x00000000 59         smv        600        524288     2          dest
0x00000000 60         smv        600        524288     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

- On voit l'ensemble des files de messages, puis l'ensemble des segments partagés en cours d'utilisation, avec un certain nombre de caractéristiques (clé IPC en héxa, identifiant en décimal, le propriétaire, les permissions en octal, la taille en octets, le nombre d'attachements et l'état), et les sémaphores.

Files de messages

Commandes liées aux IPC système V

```
smv@smv:~/EISTI/Enseignement/Progr_Syst_et_Reseau/4-File_de_messages/exemple$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x30051efa 1          smv        640        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32771      smv        600        53248      2          dest
0x00000000 32772      smv        600        53248      2          dest
0x00000000 59         smv        600        524288     2          dest
0x00000000 60         smv        600        524288     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

- On voit l'ensemble des files de messages, puis l'ensemble des segments partagés en cours d'utilisation, avec un certain nombre de caractéristiques (clé IPC en héxa, identifiant en décimal, le propriétaire, les permissions en octal, la taille en octets, le nombre d'attachements et l'état), les sémaphores.
  - La commande ipcs possède bien évidemment des options, consulter le manuel.
- La commande ipcrm permet la suppression d'une entrée dans l'une des tables de segments, sémaphores et files.

Files de messages

Les files de messages Posix

- Dans l'API standard Unix, il existe deux implémentations pour les files de message : les message queue Système V et les message queues Posix.
- Les POSIX MQ sont apparues pour standardiser les précédentes MQ de AT&T System V (cf. msgget, msgctl, msgsnd, msgrcv)
- Contrairement aux tubes ou aux sockets, elles utilisent des modules « temps réel » du noyau : en pratique dans l'industrie, elles sont utilisées pour acheminer des informations importantes entre processus, cela explique l'importance de la réactivité malgré le fonctionnement global asynchrone.

Files de messages

Les files de messages Posix

- Les MQ fonctionnent, comme leur nom l'indique, comme un service de gestion de files. Les files contiennent des messages avec des priorités variables, et les messages les plus prioritaires et les plus anciens sont lus en premiers. Les files sont approvisionnées par un ou des processus, et sont lues par un ou des processus.
- Les MQ contiennent des propriétés qui peuvent être locales à la file, ou globales à leur fonctionnement sur le système. Par exemple, on peut fixer localement le nombre maximum de messages que la file pourra contenir, mais c'est le système qui fixe la priorité maximale qu'un message peut avoir.
- Les files contiennent des messages dont la longueur maximale est fixe. Cette dernière information nous permettra une implémentation beaucoup plus facile avec un buffer de taille fixe, ou des chaînes de taille fixe.

Files de messages

Les files de messages Posix

- POSIX impose quelques règles lors de la création des MQ quant à leur nom. Celles-ci doivent « au moins » commencer par un slash (« / »), et être suivies de caractères alphanumériques.
- Les POSIX MQ servent essentiellement à transférer des messages courts entre plusieurs processus.
- Il est tout de même possible de lire ses propres messages.
- La lecture d'une file est « destructrice » : une fois un message lu, il est retiré de la file.
- La file extraira le message le plus prioritaire stocké en premier automatiquement. C'est à l'écriture que l'on indiquera la priorité du message.
- POSIX impose au minimum 32 niveaux de priorités, et suggère que tous les messages « normaux » passent avec la priorité 0, et seuls les messages réellement « urgents » passent avec des priorités supérieures.

Files de messages

Les files de messages Posix

- Les fonctions pour accéder aux POSIX MQ sont très similaires aux fonctions des autres IPC : ouverture/fermeture, lecture/écriture. La principale différence réside dans le fait que les MQ travaillent avec des messages de longueurs fixes, ainsi qu'avec un numéro de priorité.
- Les prototypes des 5 principales fonctions :

```
mqd_t mq_open(const char *, int, ...);
ssize_t mq_receive(mqd_t, char *, size_t,
    unsigned *);
int mq_send(mqd_t, const char *, size_t,
    unsigned);
int mq_close(mqd_t);
int mq_unlink(const char *);
```

Files de messages

Les files de messages Posix

- Pour plus d'informations vous pouvez consulter :
  - Introduction aux POSIX MQ : `https://fabrice-boissier.developpez.com/articles/introduction-posix-mq/`
  - Efficacité des IPC : les files de messages Posix : `https://www.blaess.fr/christophe/2011/09/17/efficacite-des-ipc-les-files-de-mess`
  - Page manuel : `https://man.cx/mq_open(3)/fr`

Files de messages

Exemple

```
#include <stdio.h>  #include <stdlib.h>
#include <string.h> #include <sys/types.h>
#include <sys/ipc.h> #include <sys/msg.h>
#define CLE 17 // cle de la MSQ
#define TAILLEMSG 80 // taille maximum du message
struct msgbuf{
long le_type; // type du message
char mtext[TAILLEMSG]; // texte du message };
int main() {
struct msgbuf msg; int msqid;
msqid = msgget((key_t)CLE, IPC_CREAT|0750); //
    creation MSQ
msg.le_type = 12; // type du message
char unmsg[TAILLEMSG];
strcpy(unmsg, "Message"); //message longueur 30
// on peut faire aussi char *unmsg = "Message";
strcpy(msg.mtext,unmsg); msgsnd(msqid, &msg,
    strlen(unmsg), IPC_NOWAIT);
exit(0); }
```

Files de messages

Exemple

```
#include <stdio.h> #include <stdlib.h>
#include <string.h> #include <sys/types.h>
#include <sys/ipc.h> #include <sys/msg.h>
#define CLE 17 // cle de la MSQ
#define TAILLEMSG 80 // taille maximum du message
struct msgbuf { long le_type; // type du message
char mtext[TAILLEMSG]; // texte du message };
int main() {
struct msgbuf msg; int msqid; int result;
while(1) {msqid = msgget((key_t)CLE, 0); // recup
    msqid
result = msgrcv(msqid,&msg,30,(long)12,IPC_NOWAIT);
// message de taille 30, de type 12
if (result > -1) { msg.mtext[result]='\0'; // fin
    de chaine de caractere
printf("Message lu %d : %s \n",result,msg.mtext);
msgctl(msqid,IPC_RMID,NULL); // destruction MSQ
exit(0); }
}
exit(0); }
```

Files de messages

Exemple

Pour exécuter :
Ouvrir un terminal et lancer :
./ecrivainMSQ
ipcs

Ouvrir un autre terminal et lancer :
./lecteurMSQ
ipcs