

Programmation Système et Réseau

Communication Inter-Processus (IPC) 4 – Files de messages

Mohamed Maachaoui
Juan Ángel Lorenzo del Castillo
Seytkamal Medetov
Son Vu

CY Tech
ING2 - GSI
2020 – 2021

Content

- Introduction
- Messages : msgbuf
- Files de messages : utilisation
- Contrôle d'une file de message
- Émission/envoi de messages
- Réception de message
- Commande liées aux IPC système V
 - Exemple
- Les files de messages Posix
 - Exemple

Communication (rappel)

Par communication inter-processus on entend :

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

Files de messages : Introduction

- 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 pipes, 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 pipes, les files de messages ont des droits d'accès de type Unix (`rw`, `rx`, `wx`)

Files de messages : Introduction

- 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.

Messages : msgbuf

- 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 */  
}
```

- Cette structure constitue un modèle : une application de votre cru utilisera probablement une structure pour les messages différente, mais elle devra contenir :
 - ♦ un premier 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  TECH pointeurs !

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 analogue à la fonction `open` pour les fichiers) ou `-1` en cas d'erreur. 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 créée (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>)

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**. La fonction renvoi 0 s'il réussit, 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>)

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 et permissions */
    time_t      msg_stime;      /* Heure du dernier msgsnd(2) */
    time_t      msg_rtime;      /* Heure du dernier msgrcv(2) */
    time_t      msg_ctime;      /* Heure de la dernière modif */
    unsigned long __msg_cbytes;  /* Nombre actuel d'octets dans la file (non standard) */
    msgqnum_t   msg_qnum;       /* Nombre actuel de messages dans la file */
    msglen_t     msg_qbytes;     /* Nombre maximum d'octets autorisés dans la file */
    pid_t       msg_lspid;       /* PID du dernier msgsnd(2) */
    pid_t       msg_lrpid;       /* PID du dernier msgrcv(2) */
};
```

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;                  /* Clé fournie à msgget(2) */
    uid_t uid;                  /* UID effectif du propriétaire */
    gid_t gid;                  /* GID effectif du propriétaire */
    uid_t cuid;                 /* UID effectif du créateur */
    gid_t cgid;                 /* GID effectif du créateur */
    unsigned short mode;        /* Permissions */
    unsigned short seq;         /* Numéro de séquence */
};
```

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);
```

- 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 la 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>)

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);
```

- Ses arguments sont les suivants :
 - `msqid` (de type `int`), identificateur de la fdm (valeur renvoyée par la fonction `msgget`).
 - `msgp` (de type `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 la taille en octets du corps du message.
 - `msgtyp` (de type `long`), il s'agit du type de message à extraire. Si c'est une valeur positive, le message extrait sera le plus ancien de ce type dans la file. Si cette valeur est nulle, le plus ancien message de la file (quel que soit son type) est extrait. Si c'est une valeur négative, 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>)

Commande 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
```

- 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 hexa, identifiant en décimal, le propriétaire, les permissions en octal, la taille en octets, le nombre d'attachements et l'état*), et enfin l'ensemble des 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.

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.
- 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.

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.

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 *);`
- Pour plus d'information 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-messages-posix/>
 - Page manuel : [https://man.cx/mq_open\(3\)/fr](https://man.cx/mq_open(3)/fr)

Références

- 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>
<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>