Notes sur la gestion des files de messages

Master 1^{iere} année, Bruno Jacob

1 Introduction

Les files de messages sont une implémentation du concept de boîtes aux lettres (BAL) : un processus peut déposer des lettres (messages) dans une BAL pour qu'ils soient lus par d'autres processus ou il peut extraire des lettres qui lui sont destinés. L'idée est d'accoler un *type* aux structures de messages qui sont déposés dans la boîte aux lettres afin que les processus qui extraient ces messages puissent distinguer ceux qui les intéressent.

Donc les structures de messages utilisables par les BAL sont du type

```
struct exemple_msg
{
  int type ; /* entier > 0 */
 /*
  * corps du message
  */
} ;
```

Les primitives de gestion des files de messages utilisent le type générique suivant

```
struct msgbuf
{
  long        mtype;          /* message type */
  char        mtext[1];       /* message text */
};
```

mais celui ci ne peut pas être utilisé tel quel.

Les files de messages sont un des 3 mécanismes de communication et de synchronisation (IPC) entre processus d'un même système[1]. Tous les objets IPC créés peuvent être manipulés par les commandes shell suivantes :

ipcs permet de consulter tous les objets IPC ; avec l'option **-m** elle affiche toutes les files de messages actives.

[1] Les deux autres étant la mémoire partagée et les sémaphores

ipcrm supprime un objet IPC. Avec l'option **-q** *msgid* on suprime la file de message d'identifiant *msgid* donné dans la deuxième colonne `ID` de la commande `ipcs`.

Une propriété essentielle d'un message est qu'il peut être vu comme un objet indivisible : quand un message est déposé dans une BAL, il est obligatoirement extrait dans sa totalité. Ceci est à opposer aux tubes par exemple, qui transfèrent des flots de caractères sans repères (plusieurs caractères peuvent être écrits en un seul `write` et lus avec une suite de `read`).

Enfin, une file de messages est gérée comme une file FIFO.

2 Création d'une file de messages

Les files de messages, comme tout objet IPC, peuvent être repérés par un *identifiant* géré par le système et/ou par une *clé* gérée par l'utilisateur. Des primitives permettent d'obtenir l'identifiant à partir de la clé.

La primitive `msgget` permet de créer une file de messages ou d'obtenir l'identification d'une file déjà existante.

```
#include <sys/msg.h>
int msgget(key_t cle,          /* Cle de file de messages */
           int   option);      /* Option de creation      */
```

`msgget` renvoie l'identifiant de la file de messages associée à `cle`.
Une nouvelle file de messages est créée si l'une des conditions suivantes est vérifiée :

- `cle = IPC_PRIVATE`
- `cle` n'a pas de file de messages associée et `option&IPC_CREAT` est vrai.

3 Envoi d'un message

```
#include <sys/msg.h>
int msgsnd(int          fileid, /* Identifiant de file      */
           const void *p_mess, /* Pt sur message a envoyer */
           size_t       lg,    /* Longueur du message      */
           int          option);/* Option d'emission       */
```

Cette primitive correspond à l'envoi dans la file identifiée par `fileid` d'un message contenu dans une zone mémoire pointée par `p_mess`. Cette zone mémoire doit contenir :

- le type du message (un entier strictement positif)
- le corps du message

Le paramètre `lg` contient la longueur du **corps** du message (sans la place occupée par le type)

Quand la file de messages est pleine,
- si `option` = `0` alors `msgsnd` est bloquant : le processus est mis en sommeil et il sera réveillé quand l'écriture dans la file sera rendu possible. Si un signal interrompt un appel bloquant et qu'on ne reprend pas les appels système (c'est à dire si `sigaction.sa_flags` ≠ `SA_RESTART`) alors `msgsnd` renvoie `-1` et `errno` = `EINTR`.
- si `option` = `IPC_NOWAIT` alors `msgsnd` est non bloquant : la primitive renvoie `-1` et positionne `errno` à `ENOMSG`.

4 Réception d'un message

```
#include <sys/msg.h>
ssize_t msgrcv(int      fileid , /* Identifiant de la file  */
               void    *p_mess , /* Pt sur zone reception   */
               size_t   lgmax ,  /* Longueur max prevue     */
               long int type ,   /* Type de message attendu */
               int      option);/* Option de reception     */
```

La primitive `msgrcv` extrait de la file identifiée par `fileid` un message de type `type`

`lgmax` donne la taille maximale du corps du message que l'on peut extraire.
- Si le message que l'on veut extraire est trop long alors
  - si `option` contient `MSG_NOERROR` alors le message est extrait dans sa totalité mais le message est tronqué à `lgmax` dans la zone de réception
  - sinon `msgrcv` échoue et `errno` = `E2BIG`.
- sinon tout est OK, le message est stocké dans la zone mémoire pointée par `p_mess`.

Quand la file ne contient pas de message du type souhaité
- si `option` = `0` alors `msgrcv` est bloquant : le processus est mis en sommeil jusqu'à ce qu'un message du bon type arrive. Si l'appel bloquant est interrompu alors `msgrcv` renvoie `-1` et `errno` = `EINTR`.
- si `option` = `IPC_NOWAIT` alors `msgrcv` est non bloquant : la primitive renvoie `-1` et positionne `errno` à `ENOMSG`.

Si le message est extrait de la BAL, alors `msgrcv` renvoie la longueur du corps du message stocké (donc si le message est tronqué, on ne sait pas combien d'octets ont été perdus).

Le paramètre `type` indique quel message on souhaite extraire :

1. `type = 0` : le premier message est extrait
2. `type > 0` : le premier message de type `type` est extrait
3. `type < 0` : le premier message de type $\leqslant$ `|type|` est extrait

dans les cas 1 et 3, le champ `mtype` de `msgbuf` contient au retour le type du message extrait.

5 Contrôle d'une file

```
#include <sys/msg.h>
int msgctl(int              fileid,  /* Identifiant de la file    */
           int              cmd,     /* Operation a effectuer     */
           struct msqid_ds *p_buf);  /* Arguments de l'operation */
```

Cette primitive permet de réaliser sur la file d'identifiant `fileid` l'opération `cmd` avec les paramètres `p_buf`.
Les opération possibles sur les files de messages sont

- `IPC_RMID` : suppression de la file `fileid` (`p_buf = NULL`)
- `IPC_STAT` : affecte dans la zone pointée par `p_buf` une structure contenant les informations de l'entrée de `fileid` dans la table des files de messages. Une entrée dans la table des files de messages a la strucure suivante :

```
struct msqid_ds {
    struct ipc_perm msg_perm;      /* structure des droits d'acces */
    struct msg      *msg_first;    /* ptr sur le 1er message */
    struct msg      *msg_last;     /* ptr sur le dernier message */
    msglen_t        msg_cbytes;    /* nb total d'octets dans la file */
    msgqnum_t       msg_qnum;      /* nb de messages dans la file */
    msglen_t        msg_qbytes;    /* nb max d'octets */
    pid_t           msg_lspid;     /* pid du dernier processus emetteur */
    pid_t           msg_lrpid;     /* pid du dernier processus recepteur */
    time_t          msg_stime;     /* date du dernier envoi */
    time_t          msg_rtime;     /* date de la derniere reception */
    time_t          msg_ctime;     /* date dernier changement par msgctl */
};
```

- `IPC_SET` : modification de l'entrée de la file `fileid` dans la table des files de messages avec la structure pointée par `p_buf`. Les seuls champs modifiables par un utilisateur standard sont `msg_perm.uid`, `msg_perm.gid` et `msg_perm.mode`.

6 Exemple

C'est un exemple de client/serveur sur une seule machine.

Structures des messages échangés (messages.h)

```
#ifndef _MESSAGES_H_
#define _MESSAGES_H_

#include <unistd.h>

#define SERVEUR_ENCODAGE 1
#define SERVEUR_DECODAGE 2

typedef struct req_corps
{
     pid_t pid_client  ;
     char msg[256] ;
} req_corps_t ;

typedef struct requete
{
     int type ;
     req_corps_t corps ;
} requete_t ;

typedef struct reponse
{
     int type ;
     char msg[256] ;
} reponse_t ;

#endif
```

Types des messages échangés (serveurs.h)

```
#ifndef _SERVEURS_H_
#define _SERVEURS_H_

#define CLE_INTERNE 10

#endif
```

Serveur d'encodage

Le serveur encode des chaines de caractères envoyées par des clients. Il reçoit des messages du type `SERVEUR_ENCODAGE` et envoie des messages dont le type a pour valeur le pid du client qui lui a fait la requête.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <string.h>
#include <signal.h>
#include <messages.h>
#include <serveurs.h>
```

```c
/* variable globales */
int file_id ;

/* Handler pour Arret du serveur */
void hand_arret(int sig)
{
     msgctl( file_id , IPC_RMID , 0 );
     printf("Terminaison du serveur\n");
     exit(0);
}

/* On suppose que c'est le serveur d'encodage qui doit creer la BAL */
int
main( int nb_arg , char * tab_arg[] )
{
     int file_cle ;
     int i , p ;
     requete_t requete ;
     reponse_t reponse ;
     /*-----*/

     if( nb_arg != 2 )
     {
          fprintf( stderr, "usage : %s <cle de file>\n" , tab_arg[0] );
          exit(-1);
     }

     if( sscanf( tab_arg[1] , "%d" , &file_cle) != 1 )
     {
          fprintf( stderr, "%s : erreur , mauvaise cle de file (%s)\n" ,
                   tab_arg[0]  , tab_arg[1] );
          exit(-2);
     }

     signal( SIGINT , hand_arret );
     signal( SIGQUIT , hand_arret ) ;

     /* Creation de la file de messages */
     if( (file_id = msgget( file_cle , IPC_CREAT | IPC_EXCL | 0666 )) == -1 )
     {
          perror( "pb creation file");
          exit(-3);
     }

     while(1)
     {
          /* Attente requete d'un client */
          if( msgrcv( file_id ,
                      &requete, sizeof(req_corps_t),
                      SERVEUR_ENCODAGE,
                      0 ) ==-1 )
          {
               perror("Pb msgrcv");
               exit(-4);
          }

          printf( "%s : message recu = %s\n", tab_arg[0] , requete.corps.msg) ;
          /* Encodage du message de la requete */
          p = (requete.corps.pid_client * CLE_INTERNE) % 26 ;
          for( i=0 ; i< strlen(requete.corps.msg) ; i++ )
               requete.corps.msg[i] = (requete.corps.msg[i] + p )  ;
```

```
            /* Envoi du message crypte au client */
            reponse.type = requete.corps.pid_client ;
            strcpy( reponse.msg , requete.corps.msg );

            if( (msgsnd( file_id ,
                        &reponse, strlen(reponse.msg)+1,
                        0 )) == -1 )
            {
                 perror("Pb msgsnd");
                 exit(-5);
            }
            printf( "%s : message envoye = %s\n", tab_arg[0] , requete.corps.msg) ;

      }
}
```

Serveur de décodage

Le serveur décode des chaines de caractères envoyées par des clients. Il reçoit des messages du type `SERVEUR_DECODAGE` et envoie des messages dont le type a pour valeur le pid du client qui lui a fait la requête.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <string.h>
#include <signal.h>
#include <messages.h>
#include <serveurs.h>

/* variable globales */
int file_id ;

/* Handler pour Arret du serveur */
void hand_arret(int sig)
{
     msgctl( file_id , IPC_RMID , 0 );
     printf("Terminaison du serveur\n");
     exit(0);
}

int
main( int nb_arg , char * tab_arg[] )
{
     int file_cle ;
     int i , p ;
     requete_t requete ;
     reponse_t reponse ;
     /*-----*/

     if( nb_arg != 2 )
     {
          fprintf( stderr, "usage : %s <cle de file>\n" , tab_arg[0] );
          exit(-1);
     }

     if( sscanf( tab_arg[1] , "%d" , &file_cle) != 1 )
     {
          fprintf( stderr, "%s : erreur , mauvaise cle de file (%s)\n" ,
```

```
                tab_arg[0]   , tab_arg[1] );
        exit(-2);
    }

    signal( SIGINT , hand_arret );
    signal( SIGQUIT , hand_arret ) ;

    /* Creation de la file de messages */
    if( (file_id = msgget( file_cle , 0 )) == -1 )
    {
        perror( "pb recherhe file");
        exit(-3);
    }

    while(1)
    {
        /* Attente requete d'un client */
        if( msgrcv( file_id ,
                    &requete, sizeof(req_corps_t),
                    SERVEUR_DECODAGE,
                    0 ) ==-1 )
        {
            perror("Pb msgrcv");
            exit(-4);
        }
        printf( "%s : message recu = %s\n", tab_arg[0] , requete.corps.msg) ;

        /* Decodage du message de la requete */
        p = (requete.corps.pid_client * CLE_INTERNE) % 26 ;
        for( i=0 ; i< strlen(requete.corps.msg) ; i++ )
            requete.corps.msg[i] = (requete.corps.msg[i] - p )  ;

        /* Envoi du message crypte au client */
        reponse.type = requete.corps.pid_client ;
        strcpy( reponse.msg , requete.corps.msg );

        if( (msgsnd( file_id ,
                     &reponse, strlen(reponse.msg)+1,
                     0 )) == -1 )
        {
            perror("Pb msgsnd");
            exit(-5);
        }
        printf( "%s : message envoye = %s\n", tab_arg[0] , requete.corps.msg) ;
    }
}
```

Client

Le client envoie un message au serveur avec le type `SERVEUR` et attend un message ayant un type égal à son pid.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <string.h>
#include <messages.h>

int
```

```
main( int nb_arg , char * tab_arg[] )
{
     int file_cle ;
     requete_t requete ;
     reponse_t reponse ;
     int file_id ;
     /*-----*/

     if( nb_arg != 2 )
     {
          fprintf( stderr, "usage : %s <cle de file>\n" , tab_arg[0] );
          exit(-1);
     }

     if( sscanf( tab_arg[1] , "%d" , &file_cle) != 1 )
     {
          fprintf( stderr, "%s : erreur , mauvaise cle de file (%s)\n" ,
                   tab_arg[0]  , tab_arg[1] );
          exit(-2);
     }

     /* Recherche de la file de messages */
     if( (file_id = msgget( file_cle , 0 )) == -1 )
     {
          perror( "pb recherche file");
          exit(-3);
     }

     /* Saisie d'un message */
     requete.type = SERVEUR_ENCODAGE ;
     requete.corps.pid_client = getpid();
     printf( "Saisissez le message a encoder\n");
     scanf( "%s" , requete.corps.msg );

     /* Envoi du message en clair au serveur */
     if( (msgsnd( file_id ,
                  &requete, sizeof(req_corps_t),
                  0 )) == -1 )
     {
          perror("Pb msgsnd");
          exit(-4);
     }

     /* Attente reponse du serveur */
     if( msgrcv( file_id ,
                 &reponse, sizeof(reponse.msg),
                 requete.corps.pid_client ,
                 0 ) ==-1 )
     {
          perror("Pb msgrcv");
          exit(-5);
     }

     printf( "Message encode = %s\n" , reponse.msg );


     /* Envoi du message encode au serveur decodage */
     requete.type = SERVEUR_DECODAGE ;
     requete.corps.pid_client = getpid();
     strcpy( requete.corps.msg , reponse.msg );

     if( (msgsnd( file_id ,
```

```
                &requete, sizeof(req_corps_t),
                0 )) == -1 )
    {
        perror("Pb msgsnd");
        exit(-4);
    }

    /* Attente reponse du serveur */
    if( msgrcv( file_id,
                &reponse, sizeof(reponse.msg),
                requete.corps.pid_client,
                0 ) ==-1 )
    {
        perror("Pb msgrcv");
        exit(-5);
    }
    printf( "Message decode = %s\n" , reponse.msg );

    exit(0);
}
```