Matière : PRS (Programmation Système)
Année : LA1
Année scolaire : 2019-2020
Enseignant : Isabelle Le Glaz

TP n°5 - Boites aux lettres (Queues)

Ce conquième TP aborde l'utilisation boîtes aux lettres pour échanger des données entre threads et entre processus.

Ce TP fera l'objet d'un compte-rendu contenant :

- Les réponses aux diverses questions présentes dans cet énoncé,
- Les codes sources commentés des programmes réalisés,
- Les jeux d'essais obtenus,
- Une conclusion sur le travail réalisé, les difficultés rencontrées, les connaissances acquises ou restant à acquérir, …

Il sera tenu compte de la qualité de vos pratiques de production de code :

- Lisibilité du code : emploi de commentaires, indentation, constantes symboliques
- Automatisation : Fichiers makefile avec macros …
- Robustesse : Vérification des valeurs de retours des appels systèmes

1. 1 Contexte

Les boîtes à lettres (appelées aussi files de messages ou queues) implémentent le mécanisme d'une queue de messages où plusieurs processus peuvent lire et écrire.

1.1 Création de la file de messages

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

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

L'appel système **msgget()** renvoie l'identifiant de la file de messages associée à la clé ***key***. Une nouvelle file de messages est créée si *key* a la valeur IPC_PRIVATE ou bien si n'est pas IPC_PRIVATE, aucune file de message n'est associée à *key*, et si la valeur IPC_CREAT a été introduite dans *msgflg*.
Si *msgflg* indique à la fois IPC_CREAT et IPC_EXCL et si une file de messages associée à *key* existe déjà, la fonction msgget() échouera et *errno* contiendra la valeur EEXIST. (C'est analogue au comportement de open(2) avec la combinaison O_CREAT | O_EXCL).

Le choix du nom **IPC_PRIVATE** est malheureux, **IPC_NEW** aurait mieux décrit sa fonction.

Lors de la création, les bits de poids faibles de l'argument *msgflg* définissent les permissions d'accès à la file de message. Ces bits de permission ont le même format et la même signification que les permissions indiquées dans l'argument *mode* de l'appel open(2). (Les permissions d'exécution ne sont pas utilisées).
Pendant la création, la structure de données associée *msqid_ds* est initialisée de la manière suivante :

- *msg_perm.cuid* et *msg_perm.uid* sont remplis avec l'UID effectif du processus appelant.
- *msg_perm.cgid* et *msg_perm.gid* sont remplis avec le GID effectif du processus appelant.
- Les 9 bits de poids faibles de *msgflg* sont copiés dans les 9 bits de poids faibles de *msg_perm.mode*.
- *msg_qnum*, *msg_lspid*, *msg_lrpid*, *msg_stime* et *msg_rtime* sont fixés à 0.
- *msg_ctime* est rempli avec l'heure actuelle.
- *msg_qbytes* est rempli avec la limite système MSGMNB.
-

Si la file de message existe déjà, les permissions d'accès sont contrôlées, et une vérification est faite pour voir si la file est prête à être détruite.

1.2 Contrôle de la boîte aux lettres

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

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

msgctl() permet d'effectuer l'opération indiquée par *cmd* sur la file de messages ayant l'identifiant *msqid*.
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 */
};
```

Les valeurs possibles pour *cmd* sont :

IPC_STAT : Copier les informations depuis la structure de données du noyau associée à *msqid* dans la structure *msqid_ds* pointée par *buf*. L'appelant doit avoir des privilèges d'accès en lecture sur la file de messages.

IPC_SET : Écrire la valeur de certains champs de la structure *msqid_ds* pointée par *buf* dans la structure de données du noyau associée à cette file de messages, en mettant à jour le champ **msg_ctime**. Les champs suivants de la structure peuvent être mis à jour : *msg_qbytes*,*msg_perm.uid*, *msg_perm.gid* et (les 9 bits poids faible de) *msg_perm.mode*.
L'UID effectif du processus appelant doit être celui du propriétaire (*msg_perm.uid*) ou celui du créateur (*msg_perm.cuid*) de la file de messages ou l'appelant doit être privilégié. Des privilèges appropriés (sous Linux, la capacité **CAP_IPC_RESOURCE**) sont nécessaires pour augmenter la valeur de *msg_qbytes* au-dessus de la constante système **MSGMNB**.

IPC_RMID : Effacer immédiatement la file de messages, en réveillant tous les processus écrivains et lecteurs en attente. Ils obtiendront un code d'erreur, et *errno* aura la valeur **EIDRM**. Le processus appelant doit avoir les privilèges associés ou bien son UID effectif doit être celui du créateur ou du propriétaire de la file de messages.

IPC_INFO (Spécifique à Linux) : Renvoyer les informations sur les limites de files de messages à l'échelle du système et les paramètres dans une structure pointée par *buf*. Cette structure est de type *msginfo* (aussi, un transtypage est nécessaire), définie dans *<sys/msg.h>* si la macro de test de fonctionnalités **_GNU_SOURCE** est définie :

```
struct msginfo {
```

```
   int msgpool; /* Taille en kibioctets du tampon utilisé pour
                   stocker les données messages ;
                   inutilisé par le noyau */
   int msgmap;  /* Nombre maximum d'entrées dans la table des
                   messages ; inutilisé dans le noyau */
   int msgmax;  /* Nombre maximum d'octets pouvant
                   être écrit dans un message */
   int msgmnb;  /* Nombre maximum d'octets pouvant être écrits
                   dans une file ; utilisé pour initialiser
                   msg_qbytes lors de la création de la file
                   (msgget(2)) */
   int msgmni;  /* Nombre maximum de files de messages */
   int msgssz;  /* Taille du segment de message ;
                   inutilisé par le noyau */
   int msgtql;  /* Nombre maximum de messages dans toutes les files
                   du système ; inutilisé dans le noyau */
   unsigned short int msgseg;
                /* Nombre maximum de segments ;
                   inutilisé par le noyau */
};
```

Les réglages *msgmni*, *msgmax* et *msgmnb* peuvent être modifié via les fichiers */proc* de même nom ; voir **proc**(5) pour plus de détails.
MSG_INFO (Spécifique à Linux) : Renvoie une structure *msginfo* contenant les mêmes informations que pour **IPC_INFO**, excepté que les champs suivants sont renvoyés avec une information relative aux ressources système consommées par les files de messages : le champ *msgpool* renvoie le nombre de files de messages qui existent actuellement sur le système ; le champ *msgmap* renvoie le nombre total de messages dans toutes les files du système ; et le champ *msgtql* renvoie le nombre total d'octets de tous les messages de toutes les files du système.
MSG_STAT (Spécifique à Linux) : Renvoie une structure *msqid_ds* comme pour **IPC_STAT**. Toutefois, l'argument *msqid* n'est pas un identifiant de file, mais plutôt un index dans un tableau interne au noyau qui contient des informations sur toutes les files de messages du système.

1.3 Lecture et écriture dans une boîte aux lettres

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

Envoi de message (send)

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

Réception de message (receive)

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

Les appels système **msgsnd**() et **msgrcv**() servent respectivement à envoyer et recevoir des messages dans une file d'attente de messages. Le processus appelant doit avoir la permission d'écriture sur la file pour envoyer un message et la permission de lecture pour en recevoir.

L'argument *msgp* est un pointeur vers une structure, définie par l'appelant, dont la forme est la suivante:

```
struct msgbuf {
   long  mtype;    /* type de message ( doit être > 0 ) */
   char  mtext[1]; /* contenu du message */
1
```

Le champ *mtext* est un tableau (**ou une autre structure**) de taille *msgsz*, valeur entière non négative. Les message de taille nulle (sans champ *msgsz*) sont autorisés. Le membre **mtype** doit avoir une valeur strictement positive. Cette valeur peut être utilisée par le processus lecteur pour la sélection de messages (voir la description de **msgrcv**() plus loin).

Ce type de messages se prête particulièrement bien à la mise en place d'un protocole avec des messages typés.

msgsnd

Si un même message est destiné à plusieurs boîtes, il faudra répéter autant de fois l'opération qu'il y a de destinataires. Le contrôle du programme passe à l'instruction suivante dès que le message est posé dans la boîte, ce qui signifie quasiment instantanément, à moins que la boîte du destinataire ne soit pleine, auquel cas le processus est bloqué jusqu'à ce que la boîte du destinataire soit suffisamment dégagée pour recevoir le message (... à moins que l'option IPC_NOWAIT ne soit activée.)

Rappel du man de msgsnd:

L'appel système **msgsnd**() insère une copie du message pointé par l'argument *msgp* dans la file dont l'identifiant est indiqué par la valeur de l'argument *msqid*.
S'il y a assez de place dans la file, **msgsnd**() réussit immédiatement. (La capacité de la file est définie par le champ *msg_bytes* de la structure associée à la file de messages. Durant la création de la file, ce champ est initialisé à **MSGMNB** octets, mais cette limite peut être modifiée avec **msgctl**(2).) S'il n'y a pas assez de place, alors le comportement par défaut de **msgsnd**() est de bloquer jusqu'à obtenir suffisamment d'espace. En indiquant **IPC_NOWAIT** dans *msgflg*, le message ne sera pas envoyé et l'appel système échouera en retournant **EAGAIN** dans *errno*.
Un appel à **msgsnd**() bloqué peut également échouer si :

- la file est supprimée, auquel cas l'appel système échouera en retournant **EIDRM** dans *errno* ; ou
- un signal a été intercepté, auquel cas l'appel système échouera en retournant **EINTR** dans *errno* ; voir **signal**(7). (**msgsnd**() n'est jamais relancé automatiquement après interruption par un gestionnaire de signal, quelle que soit la configuration de **SA_RESTART** lors de l'installation du gestionnaire.)

Si l'appel système réussit, la structure de la file de messages sera mise à jour ainsi :

- *msg_lspid* contient le PID du processus appelant.
- *msg_qnum* est incrémenté de 1.
- *msg_stime* est rempli avec l'heure actuelle.

msgrcv

C'est ici qu'intervient le type des messages. En effet, il est possible de ne recevoir que les messages d'un certain type ! tout en gardant bien à l'esprit que le fonctionnement général est de type FIFO. Voici la

signification de ce paramètre :

- Si typeM==0 le premier message de n'importe quel type est extrait
- Si typeM > 0 le premier message de type==typeM est extrait
- Si typeM < 0 le premier message de type < typeM est extrait

tailleMax spécifie la taille maximale du message que l'on peut récupérer à l'adresse adresseReception. Il y a toutefois un détail particulièrement dérangeant : le type du message n'est pas compris dans tailleMax. Donc, supposons que l'emplacement mémoire pointé par adresseReception soit de 256 octets. Lorsque l'on récupère un message, le type est extrait avec les autres informations. Le paramètre tailleMax doit donc être fixé à 256 - sizeof(long) sous peine de surprise désagréable ! Mais, que se passe-t-il si le message est trop long ? Par défaut, **msgrcv** retourne un code d'erreur, à moins que MSG_NOERROR ne soit inclus dans les options, auquel cas, le message est tronqué pour rentrer dans l'espace prévu.

Rappels du man de msgrcv

L'appel système **msgrcv**() supprime un message depuis la file indiquée par *msqid* et le place dans le tampon pointé par *msgp*.
L'argument *msgsz* indique la taille maximale en octets du membre **mtext** de la structure pointée par l'argument *msgp*. Si le contenu du message est plus long que *msgsz* octets, le comportement dépend de la présence ou non de **MSG_NOERROR** dans *msgflg*. Si **MSG_NOERROR**est spécifié , alors le message sera tronqué (et la partie tronquée sera perdue) ; si **MSG_NOERROR** n'est pas spécifié, le message ne sera pas extrait de la file, et l'appel système échouera en renvoyant -1 et en indiquant **E2BIG** dans **errno**
L'argument *msgtyp* indique le type de message désiré :

- Si *msgtyp* vaut 0, le premier message est lu.
- Si *msgtyp* est supérieur à 0, alors le premier message de type *msgtyp* est extrait de la file. Si *msgflg* contient **MSG_EXCEPT** l'inverse est effectué, le premier message de type **différent** de *msgtyp* est extrait de la file.
- Si *msgtyp* est inférieur à 0, le premier message de la file avec un type inférieur ou égal à la valeur absolue de *msgtyp* est extrait.

L'argument *msgflg* est composé d'un **OU** binaire « | » avec les options suivantes :
IPC_NOWAIT : Revient immédiatement si aucun message du type désiré n'est présent. L'appel système échoue et **errno** est renseignée avec **ENOMSG**.
MSG_EXCEPT : Utilisé avec *msgtyp* supérieur à 0 pour lire les messages de type différent de *msgtyp*.
MSG_NOERROR : Tronque silencieusement les messages trop longs

Si aucun message du type requis n'est disponible et si on n'a pas demandé **IPC_NOWAIT** dans *msgflg*, le processus appelant est bloqué jusqu'à l'occurrence d'un des événements suivants.

- Un message du type désiré arrive dans la file.
- La file de messages est supprimée. L'appel système échoue et *errno* contient **EIDRM**.

Le processus appelant reçoit un signal à intercepter. L'appel système échoue et *errno* est renseignée avec **EINTR**. (**msgrcv**() n'est jamais automatiquement relancé après avoir été interrompu par un gestionnaire de signal, quelle que soit la configuration de **SA_RESTART** lors de l'installation du gestionnaire.)
Si l'appel système réussit, la structure décrivant la file de messages est mise à jour comme suit :

- *msg_lrpid* est rempli avec le PID du processus appelant.
- *msg_qnum* est décrémenté de 1
- *msg_rtime* est rempli avec l'heure actuelle.

1.4 Remarques

- Un message ne peut être lu qu'une seule fois, même s'il est tronqué
- Un message ne contient pas d'informations identifiant son émetteur. Si l'émetteur désire être identifié, il devra placer lui même des informations dans le message, par exemple, une adresse de retour (clef ou identificateur de boîtes aux lettres)

2 Questions

1 Ecrire le programme bal_1.c qui crée une boîte aux lettres, et affiche les informations concernant la boite aux lettres ainsi créée.

2 Ecrire le programme bal2_c qui crée une boîte aux lettres, écrit un message dans la boîte aux lettres, et examine les informations concernant la boite aux lettres avant et après l'envoi d'un message.

3 Ecrire les deux programmes suivants :

- Le programme de type serveur bal3_s, qui crée une boîte aux lettres, qui affiche les informations de cette boîte aux lettres après création, puis qui réalise une boucle sans fin de lecture des messages de type 1 où il affiche le message reçu et ses informations (Cf + bas).
- Le programme de type client bal3_c, qui ouvre la boîte aux lettres créée par bal3_s, qui écrit 10 fois des messages saisis par l'utilisateur. Ces messages sont de type 1.

Pour disposer d'informations utiles, vous passerez comme message une structure dans laquelle vous stockerez le message de l'utilisateur, mais aussi le PID du processus client.

4 Faire évoluer les deux programmes ci-dessus en bal4_s et bal4_c.

A présent, bal4_s envoie une réponse au client pour lui indiquer que son message a été bien reçu. Le message qu'il envoie est de type X, où X est le pid du client destinataire du message.

A présent, bal4_c :

- traite les saisies de l'utilisateur dans une boucle.
- se termine quand l'utilisateur a tapé "SALUT". une boucle sans fin de saisie des messages utilisateurs
- traite la réception des messages issus du serveur en lisant dans la boîte aux lettres les messages qui lui sont destinés.

5 4 Faire évoluer les deux programmes ci-dessus en bal5_s et bal5_c.

A présent, bal5_s mémorise les clients avec lesquels il a entamé un dialogue et tient à jour une liste de ces clients. Quand il reçoit un message, il vérifie si ce message est issu d'un client déjà connu ou non dans sa liste; lorsqu'il reçoit le message SALUT, il gère le départ du client.

bal5_c est identique à bal4_c

6 Que faut-il faire pour transformer notre serveur en serveur de type broadcast ?

7 Que faut-il faire pour transformer notre serveur de type broadcast en serveur de type point à point ?