Posix Message Queue

Abdelkader Gouaich

IUT de Montpellier

2014-2015

Objectifs du cours :

- introduction aux files de messages POSIX
- File de messages et notification des processus

- Nous avons présenté les files de message dans le cours précédent
- Il s'agissait des "message queue" system V
- Nous allons présenter dans ce chapitre les message queue POSIX

- Sur le principe les deux systèmes sont similaires :
- Ils proposent une communication inter-processus en utilisant les messages

Il existe cependant des différences notables entre les deux systèmes :

- Les files POSIX utilisent un compteur de références. Quand une file est marquée pour l'effacement, elle est effectivement effacée quand tous les processus qui l'avaient ouverte terminent.
- Chaque message dans les files System V possède un entier qui représente sont type et plusieurs mécanismes sont disponibles pour sélectionner les messages avec msgrcv(). Les messages POSIX possède une priorité et la sélection des messages se fait toujours en fonction de cette priorité.
- POSIX offre la possibilité aux processus d'être notifiés de façon asynchrone dès l'arrivée d'un message dans une file vide

L'API des message queue POSIX repose sur les fonctions suivantes :

- `mq_open()` pour la création d'une nouvelle ou l'ouverture d'une file existante
- `mq_send()` pour envoyer un message dans une file
- `mq_receive()` pour recevoir un message dans une file
- `mq_close()` pour refermer une file ouverte
- `mq_unlink()` marquer la file pour l'effacement

En plus de ces fonctions élémentaires, deux autres fonctionnalités sont importantes à connaitre :

- Gestion des attributs de la file de message : chaque file de message possède un ensemble d'attributs qui peuvent être modifiés :
  - soit à l'ouverture avec la fonction mq_open()
  - en utilisant les fonctions mq_getattr() et mq_setattr()
- Gestion des notifications : la fonction mq_notify() permet de s'enregistrer pour être notifié à la réception d'un message dans une file vide. Deux méthodes pour gérer l'arrivée d'un message : soit par un signal ; soit par l'exécution d'une fonction dans un thread

Ouverture/Création d'une file de message

#include <mqueue.h>
mqd_t mq_open(**const char** *name, **int** oflag, ...
/ *mode_t mode, struct mq_attr *attr */*);

- name : le nom donné à la file de message. Ce nom doit être conforme aux noms des objets IPC POSIX et commencer par un slash et inférieur à 255.

Ouverture/Création d'une file de message

#include <mqueue.h>
mqd_t mq_open(**const char** *name, **int** oflag, ...
/ *mode_t mode, struct mq_attr *attr */*);

- oflag : les flags d'ouverture de la file de message :
  - O_CREAT : création de la file si elle n'existe pas déjà
  - O_EXCL : en combinaison avec O_CREAT, création de la file exclusivement
  - O_RDONLY : ouverture en mode lecture
  - O_WRONLY : ouverture en mode écriture
  - O_RDWR : ouverture en mode lecture/écriture
  - O_NONBLOCK : ouverture en mode non bloquant. Si les appels de mq_receive() ou mq_send() ne peuvent aboutir alors ces fonctions retournent immédiatement et EAGAIN est mise comme erreur dans errno.

Si mq_open est utilisé pour ouvrir une file existante alors deux arguments sont utilisés. Cependant, s'il s'agit d'une création d'une file alors deux autres arguments supplémentaires sont nécessaires :

- mode : masque des permissions (comme pour le fichiers)
- attr : est une structure mq_attr qui donne les attributs de la file de message

La fonction mq_open retourne un descripteur de la file de message.

Exemple

```
int main(int argc, char *argv[]){
    int flags, opt;
    mode_t perms;
    mqd_t mqd;
    struct mq_attr attr, *attrp;
    attrp = NULL;
    attr.mq_maxmsg = 50;
    attr.mq_msgsize = 2048;
    flags = O_RDWR;
    perms = S_IRUSR | S_IWUSR ;
mqd = mq_open(argv[optind], flags, perms, attrp);
if (mqd == (mqd_t) -1)
            errExit("mq_open");
exit(EXIT_SUCCESS);
```

```
#include <mqueue.h>
int mq_getattr(mqd_t mqdes, struct mq_attr *attr);
```

La structure `struct mq_attr` contient les champs suivants :

- mq_maxmsg : nombre max de messages dans la file
- mq_msgsize : taille max d'un message
- mq_flags : flag associé à la file. Seul le flag O_NONBLOCK. est utilisé ou 0. Cet attribut peut être modifié par la fonction mq_setattr().
- mq_curmsgs : nombre de messages actuellement dans la file.

Modifier les attributs d'une file de message :

```
#include <mqueue.h>

int mq_setattr(mqd_t modes,
const struct mq_attr *newattr, struct mq_attr *oldattr);
```

Les attributs de la file de message sont changés avec les valeurs trouvées dans newattr
Si oldattr est un poitneur non nulle alors les anciennes valeurs sont copiées dans cette structure
La seule valeur possible de modifier avec setattr concerne O_NONBLOCK.

Exemple

```
if (mq_getattr(mqd, &attr) == -1) errExit("mq_getattr");
attr.mq_flags |= O_NONBLOCK;
if (mq_setattr(mqd, &attr, NULL) == -1)
        errExit("mq_getattr");
```

```
#include <mqueue.h>
int mq_send(mqd_t mqdes, const char *msg_ptr, size_t
    msg_len, unsigned int msg_prio);
```

- msg_len : longueur du message pointé par msg_ptr
- msg_prio : priorité du message (0 est la priorité la plus faible)

Si la file est pleine (nombre max de message est atteint) alors mq_send bloque si O_NONBLOCK n'est pas spécifié
Si O_NONBLOCK est spécifié alors mq_send echoue immédiatement avec un code erreur EAGAI

Réception d'un message

ssize_t mq_receive(mqd_t mqdes, **char** *msg_ptr, size_t msg_len, **unsigned int** *msg_prio);

msg_len : taille du buffer pour recevoir le message
si msg_prio n'est pas NULL, alors la priorité du message reçu est mise à cette adresse.
Si le flag O_NONBLOCK n'est pas activé alors mq_receive bloque jusqu'au retrait d'un message
Autrement, elle retourne immédiatement avec un échec et un code d'erreur EAGAIN.

C'est la grande nouveauté par rapport aux files de messages system V.
Au lieu d'avoir un blocage au niveau de la réception, on s'enregistre pour être notifié dès qu'un message arrive dans la file.
Attention nous allons être notifié seulement si la file est vide et reçoit un nouveau message.
Autrement c'est au processus de traiter tous les messages jusqu'à vider la file
La notification se fera soit par signal ou l'exécution d'une fonction dans un thread

Remarques sur le mécanisme de notification

Un seul processus peut s'enregistrer pour la notification. Si un processus est déjà enregistré alors un code erreur EBUSY est mis dans errno.
Le processus sera notifié seulement si un message arrive dans une file vide.
Quand la notification est envoyée au processus, celui-ci est retiré de la liste d'écoute. Cela veut dire qu'il devra se ré enregistrer s'il veut recevoir d'autres notifications.
Le processus enregistré pour la notification sera notifié seulement si aucun autre processus n'est bloqué sur l'écoute par mq_receive(). Dans ce cas, c'est le processus qui est bloqué qui reçoit le message.
Le processus enregistré reste enregistré sur la file.
Un processus peut se retirer de l'écoute d'une file en utilisant mq_notify() avec le paramètre notification égale à NULL

Enregistrement pour être notifié

```
#include <mqueue.h>
    int
    mq_notify(mqd_t
    mqdes,
    const struct sigevent *notification);
```

Le paramètre notification va spécifier la méthode de notification.

La structure sigevent

```
union sigval
{ int sival_int; /* valeur entier value for accompanying data */
  void *sival_ptr; /* pointeur sur un entier */
};
struct sigevent {
    int    sigev_notify; /* Notification method */
    int    sigev_signo; /* Notification signal for SIGEV_SIGNAL
     */
    union sigval sigev_value; /* Value passed to signal handler or
           thread function */
void (*sigev_notify_function) (union sigval);/* Thread
     notification function */
void *sigev_notify_attributes; /* Really 'pthread_attr_t' */ };
```

Le champs sigev_notify peut avoir les valeurs suivantes :

- SIGEV_NONE : enregistre le processus pour la notification, mais quand le message arrive sur une file vide aucune action n'est entreprise.
- SIGEV_SIGNAL : Recevoir la notification comme un signal
- SIGEV_THREAD : recevoir la notification par l'exécution d'une fonction dans un thread

Un exemple complet sera traité en TP.