Outils IPC

Files de messages

Cet article présente plusieurs programmes permettant de manipuler les files de messages IPC System V. Un premier programme (appelé serveur) crée une file et se met en attente de requêtes de la part d'autres programmes (appelés clients). Dès qu'une requête est reçue, il y répond.

1 Le fichier de déclaration

Ce fichier contient la déclaration des deux structures (avec leur type associé) correspondant aux messages envoyés dans la file. Nous noterons le premier champ de type `long` obligatoire correspondant au type.

```
#define TYPE_REQUETE 1

typedef struct {
  long type;
  int valeur1;
  int valeur2;
} requete_t;

#define TYPE_REPONSE 2
typedef struct {
  long type;
  int resultat;
} reponse_t;
```

2 Le serveur

Dans un premier temps, le serveur crée la file si elle n'existe pas et récupère l'identifiant interne. Dans le cas contraire, le programme s'arrête.

```
int msqid;
if((msqid = msgget((key_t)CLE, S_IRUSR | S_IWUSR | IPC_CREAT | IPC_EXCL)
   ) == -1) {
  if(errno == EEXIST)
    fprintf(stderr, "Erreur : file (cle=%d) existante\n", CLE);
  else
    perror("Erreur lors de la creation de la file ");
  exit(EXIT_FAILURE);
}
```

Le serveur se met alors en attente d'une requête à l'aide de l'appel système `msgrcv`. Il spécifie comme type de message attendu `TYPE_REQUETE`.

```
if(msgrcv(msqid, &requete, sizeof(requete_t) - sizeof(long),
    TYPE_REQUETE, 0) == -1) {
  perror("Erreur lors de la réception d'une requête ");
  exit(EXIT_FAILURE);
}
```

Quand il reçoit une requête, il y répond. Il initialise une structure `reponse_t` qu'il envoie à l'aide de l'appel système `msgsnd`.

```
reponse.type = TYPE_REPONSE;
reponse.resultat = requete.valeur1 + requete.valeur2;

if(msgsnd(msqid, &reponse, sizeof(reponse_t) - sizeof(long), 0) == -1) {
  perror("Erreur lors de l'envoi de la reponse ");
  exit(EXIT_FAILURE);
}
```

3 Le client

Dans un premier temps, le client doit récupérer l'identifiant interne de la file de messages qui a été créée.

```
if((msqid = msgget((key_t)CLE, 0)) == -1) {
  perror("Erreur lors de la récupération de la file ");
  exit(EXIT_FAILURE);
}
```

Il peut alors initialiser une structure `requete_t` qu'il envoie au serveur à l'aide de `msgsnd`.

```
requete.type = TYPE_REQUETE;
requete.valeur1 = 3;
requete.valeur2 = 6;

if(msgsnd(msqid, &requete, sizeof(requete_t) - sizeof(long), 0) == -1) {
  perror("Erreur lors de l'envoi de la requête ");
  exit(EXIT_FAILURE);
}
```

Il se met ensuite en attente d'une réponse.

```
if(msgrcv(msqid, &reponse, sizeof(reponse_t) - sizeof(long),
    TYPE_REPONSE, 0) == -1) {
  perror("Erreur lors de la réception de la réponse ");
  exit(EXIT_FAILURE);
}
```

4 Suppression de la file de messages

Pour supprimer la file de messages à l'aide d'un programme en C, nous devons dans un premier temps récupérer l'identifiant interne de la file de messages à l'aide de `msgget`. Puis nous utilisons l'appel système `msgctl` avec la commande `IPC_RMID` (le troisième paramètre est inutile).

```
if(msgctl(msqid, IPC_RMID, 0) == -1) {
  perror("Erreur lors de la suppression de la file ");
  exit(EXIT_FAILURE);
}
```

5 Compilation et exécution

Le `makefile` fourni permet de compiler les programmes précédents. Dans la section `ARGUMENTS ET COMPILATEUR`, la variable `CLE_MSG` correspond à la clé de la file de messages utilisées dans les différents programmes. Elle est spécifiée par `gcc` grâce à l'option `-D`.

Pour compiler les programmes précédents, saisissez la commande suivante :

```
make
```

Pour tester les programmes, dans un premier terminal, exécutez le serveur :

```
./fileServeur
```

Dans un second terminal (ou dans un nouvel onglet), exécutez le client :

```
./fileClient
```

Pour supprimer la file, vous pouvez soit utiliser le programme `fileSupprime`, soit la règle du `makefile` `cleanIPC`, soit utiliser la commande `ipcrm` (où X est la clé de la file de messages) :

```
./fileSupprime
make cleanIPC
ipcrm -Q X
```