Communication par file de messages [IPC System V]

Rendu des travaux :
Vous avez ***5 jours*** (date exacte à voir sur le dossier des travaux) après la fin du TP pour envoyer les codes sources (1 seul fichier compressé) en utilisant **Chamilo** (pas de mail aux enseignants !)

EXERCICE 1 :

On vous demande de programmer un serveur de calcul utilisant les files de messages. Le serveur exécute des opérations simples (+, -,...). Il fonctionne de la façon suivante :

- Les clients déposent leurs requêtes (opération, opérandes et pid) dans un message dont le type est 1.
- Le serveur effectue l'opération demandée (+, -, /, *) et écrit le résultat dans un message dont le type est le `pid` du client.

Le client, qui s'était bloqué en attente sur un message dont le type est son propre `pid` peut alors lire le résultat. Pour construire la partie texte des messages envoyés par le client on utilisera une structure de données qui contient :

- le type du message (long).
- Une autre structure contenant le pid du client, l'opération à effectuer et les opérandes.

Le serveur répond en envoyant un message structuré de la même façon. Le résultat de l'opération est enregistré sur le premier opérande.

Proposez une structure de message pour cet exercice (sur « `calcul.h` »), et completez les deux fichiers fournis « `serveur.c` » et « `client.c` » (voir Annexe 1).

EXERCICE 2 :

Un processus client envoie au moyen d'une file de messages à un processus serveur un texte passé en ligne de commande. Le serveur « traduit » ce message (traduction : minuscules -> MAJUSCULES) et renvoie au client le message traduit par *une autre* file de messages. Le client affiche le message traduit et s'arrête. Le serveur s'arrête à la réception d'un message « @ ».

Annexe 1

/ calcul.h */*

```
#ifndef __CALCUL_H
#define __CALCUL_H

#define MSGKEY 50            /* cle d'entree dans la table d'IPC (cf. ipcs) */

struct msg_struct               /* structure d'un message */
{
        long type;
         /* A COMPLETER */
};

#endif
```

/ serveur.c */*

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>

#include "calcul.h"

void raz_msg(int);                      /* routine de traitement des signaux */

int msg_id;                                     /* identificateur de la file de messages */

int main(int argc, char const *argv[])
{
        struct msg_struct msg;
        int i_sig;
        int result;

        /* liberer la zone de messages sur reception
           de n'importe quel signal */
        for (i_sig = 0; i_sig < NSIG; i_sig++)
        {
             /* A COMPLETER */
        }

        msg_id = msgget(/* A COMPLETER */);
        printf("SERVEUR: pret!\n");
```

```
        while(1)
        {
             /* reception */
             msgrcv(/* A COMPLETER */);
             printf("SERVEUR: reception d'une requete de la part de: %d\n",
                        /* A COMPLETER */);

             /* preparation de la reponse */
             /* A COMPLETER */

             /* envoi de la reponse */
             msgsnd(/* A COMPLETER */);
        }
        return 0;

}

void raz_msg(int signal)
{
        printf("Suppression de la file de message!\n");
        msgctl(/* A COMPLETER */);
        exit(0);
}
```

/ client.c */*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "calcul.h"

int main(int argc, char const *argv[])
{
        int msg_id;
        struct msg_struct msg;

        if (argc != 4)
        {
                printf("Usage: %s operande1 {+|-|*|/} operande2\n", argv[0]);
                exit(-1);
        }

        /* il faut eviter la division par zero */
        /* A COMPLETER */

        /* ATTENTION : la file de messages doit avoir ete creee par
                 le serveur.
```

```
                Il faudrait  tester la valeur de retour (msg_id) pour
                verifier que cette creation a bien eu lieu.*/
        msg_id = msgget(/* A COMPLETER */);

    printf("CLIENT %d: preparation du message contenant l'operation suivante: %d %c %d\n",
               getpid(), atoi(argv[1]), argv[2][0], atoi(argv[3]));

        /* On prepare un message de type 1 à envoyer au serveur
            avec les informations necessaires */
        /* A COMPLETER */

        /* envoi du message */
        msgsnd(/* A COMPLETER */);

        /* reception de la reponse */
        msgrcv(/* A COMPLETER */);

        printf("CLIENT: resultat recu depuis le serveur %d : %d\n",
                /* A COMPLETER */);
        return 0;
}
```

Annexe 2

Parmi les différents outils de gestion de communication interprocessus, Unix propose les files de messages. Un message est un couple : <type, message>. System V fournit l'ensemble des primitives suivantes :

- msgget, (création de la file de messages)
- msgctl, (consultation de cette file)
- msgsnd, (déposer un message)
- msgrcv, (extraire un message)

Les structures de données nécessaires à la gestion des messages sont répertoriées dans :

`/usr/include/sys/msg.h`

`/usr/include/sys/ipc.h`

`/usr/include/sys/types.h`

Nous allons décrire les quatre fonctions : `msgget`, `msgctl`, `msgsnd` et `msgrcv`.

➢ msgget

```
int msgget (int msg_clef, int drapeau)
```

`msgget` renvoie un identificateur local (appelé par la suite `msgid`) pour la file de messages identifiée par la clef `msg_clef`. Si le bit `IPC_CREAT` est positionné dans la variable drapeau ou si `msg_clef` = `IPC_PRIVATE`, alors `msgget` crée une zone de messages, c'est à dire un objet de type `msgid_ds` (décrit dans `/usr/include/sys/msg.h`). Sinon, il positionne simplement l'utilisateur sur la file de messages de clef

msg_clef.

➢ msgctl

```
int msgctl (int msgid, int commande, struct msgid_ds *tab)
```

msgctl envoie la commande commande sur la zone msgid. Suivant la commande commande envoyée, cette fonction renvoie des informations dans tab, ou modifie les caractéristiques de la file de messages.

➢ msgsnd

```
int msgsnd (int msgid, struct msgbuf *adresse, int taille, int drapeau)
```

msgsnd envoie un message dans la zone de messages msgid, adresse est un pointeur sur un objet de type msgbuf dont la partie texte seule est de dimension taille, drapeau indique l'action à entreprendre si l'opération demandée n'a pu être réalisée et msgbuf est une structure composée d'un long et d'un texte de taille octets :

type (sur 4 octets)	texte (sur taille octets)

➢ msgrcv

```
int msgrcv (int msgid, struct msgbuf *adresse, int taille, int type, int drapeau)
```

msgrcv récupère un message dont on peut spécifier le type, adresse est un pointeur sur l'objet contenant le type du message dont le texte seul est de dimension taille. Et type indique quel message on veut lire, en particulier :

- (type = 0) => le premier message,
- (type > 0) => le premier message du type désigné par type.
- (type < 0) => le premier message dont le type est inférieur ou égal à valeur absolue de type.

Enfin drapeau indique l'action à entreprendre si l'opération demandée n'a pu être réalisée.