HE VINCI Août 2022

BINV2181 : Linux : Programmation distribuée

J. Vander Meulen, A Legrand, O Choquet

Durée de l'examen : 120 minutes (pas de sortie durant les 30 premières minutes)

Consignes importantes :

- Nous vous demandons de faire particulièrement attention à ce que votre code ne produise par d'erreurs de compilation. Notez que nous vous demandons de compilez votre code avec les flags utilisés lors du cours :

```
gcc -std=c11 -pedantic -Wall -Wvla -Werror
    -Wno-unused-variable -D_DEFAULT_SOURCE
```

- A la fin de l'examen, soumettez vos fichiers sur EvalMoodle. Voir la page 5 de cet énoncé pour plus d'informations.

1. Sockets - IPC(10/20 points)

Pour qu'un serveur soit le plus souvent disponible et à l'écoute de ses clients, plusieurs techniques existent. Il existe notamment la technique "fork" qui consiste à créer un fils à chaque connexion d'un nouveau client. Cette technique présente l'avantage de rendre plus disponible le serveur central et d'isoler les requêtes des clients. Le serveur central et ses fils utilisent une mémoire partagée. Cette mémoire partagée contient les socket id et le nom de tous les clients.

Une (grosse) partie du travail d'implémentation de cette technique a été effectuée. Le but est donc ici de rentrer le code d'une autre personne et de le compléter. Le programme client est complet. Le programme server est à compléter.

Voici ce que font/doivent faire ces 2 programmes :

1. Un programme "server" qui initialise la mémoire partagée et le socket d'écoute. La structure de la mémoire partagée est définie dans « server.h ». Il crée ensuite un fils à chaque connexion d'un client. Le(s) fils accède(nt) à la mémoire partagée pour mémoriser le nom du client et son socket id. Le(s) fils se termine dès qu'il a effectué cette opération. Lors de l'arrêt du serveur (CTRL-C), celui affiche les informations présentes dans la mémoire partagée. Tous les IPC doivent être détruits lors de l'arrêt du serveur.
2. Un programme "client" qui se connecte au serveur en envoyant son nom ("test" dans notre cas pour tous les clients).

Voici un exemple d'exécution :

Le serveur a reçu ici 2 clients.

```
olivier@bilout:~/Téléchargements/solution$ ./server
Le serveur tourne sur le port 9090
Nom client reçu : test
Nom client reçu : test
^CFin serveur - Affichage des clients
Nb clients : 2
Client name :  test
Client sockfd :  4
Client name :  test
Client sockfd :  5
olivier@bilout:~/Téléchargements/solution$
```

```
olivier@bilout:~/Téléchargements/solution$ ./client
olivier@bilout:~/Téléchargements/solution$ ./client
olivier@bilout:~/Téléchargements/solution$
```

Le modules *utils* vous est fourni ainsi que les fichiers server.h, server.c, client.c

Pour cette question, il vous est demandé de :

1. compléter le programme server.c

2. rédiger un fichier *makefile*

2. Signaux –pipe – fork - alarm (10/20 points)

Ecrivez un programme « testmult » qui permet à un père de tester chez son fils la connaissance des tables de multiplication.

Le programme reçoit comme argument sur la ligne de commande le temps que le fils a pour répondre à l'ensemble des questions. Le nombre de questions est défini dans « testmult.h ».

Le programme génère des couples de nombres aléatoires compris entre 1 et 10 (A et B), il envoie à son fils le calcul à faire c'est-à-dire « A * B = », attend la réponse avant d'envoyer la question suivante.

Le père et le fils communiquent via deux pipes. Le père écrit sur le premier pipe des enregistrements qui ont la structure suivante :

```
struct mult {

int a;

int b;

}mult;
```

Le fils écrit un nombre entier sur le deuxième pipe pour envoyer à son père le résultat que l'utilisateur a introduit au clavier.

Pour chaque calcul que le processus père envoie à son fils, il attend sa réponse, la compare avec la bonne réponse qu'il a calculée et comptabilise les bonnes et les mauvaises réponses.

Le processus fils lit les questions proposées par son père, demande une réponse à l'utilisateur et la communique à son père.

Il est aussi demandé de vérifier que le fils ne prend pas plus de temps pour répondre que la limite imposée. Cette limite est une limite globale pour le questionnaire. Pour ce faire, le père lancera une alarme dès le début du questionnaire.

Le programme se termine quand le fils a introduit un ctrl-D à la place d'une réponse ou que le nombre de questions est atteint. Avant de rendre la main au shell, le père

affiche le nombre de bonnes, de mauvaises réponses et également si le fils a dépassé le temps imparti.

Voici un exemple d'exécution :

Le questionnaire contient 3 questions et a une durée maximum de 10 secondes. Le fils répond correctement à la première question, incorrectement à la deuxième et ne répond pas à la troisième.

```
olivier@bilout:~/Téléchargements/solution$ ./testmult 10
5 * 6 = 30

réponse recue fils : 30
1 * 6 = 7

réponse recue fils : 7
9 * 2 =
réponse recue fils : -1

Fils a obtenu 1 bonne(s) reponse(s), 1 mauvaise(s)
Temps imparti écoulé
olivier@bilout:~/Téléchargements/solution$
```

Le modules *utils* vous est fourni.

Pour cette question, il vous est demandé de :

1. compléter le programme testmult.*c*
2. compléter le fichier *makefile* de la question 1

3. Fichiers fournis

Sur EvalMooldle, vous trouverez les fichiers suivants :

1. server.h (Q1)
2. server.c (Q1)
3. client.c (Q1)
4. Makefile (Q1 & Q2)
5. testmult.c (Q2)
6. utils_v1.h (Q1 & Q2)
7. utils_v1.c (Q1 & Q2)
8. HEVINCI-2021-2022-BINV2180-EXAMEN-SEPT.pdf (ce fichier)

4. À remettre

Sur EvalMoodle, tous les fichiers de la section 3 « Fichiers fournis ». Attention les fichiers server.c, Makefile et testmult.c doivent être complétés.