



HE VINCI

15 juin 2025

BINV2181 : Linux : Programmation distribuée

Professeurs : J. Vander Meulen, A. Legrand, X. Gillard, A. Lonfils

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 pas d'erreurs de compilation. Notez que nous vous demandons de compiler votre code avec les flags utilisés lors du cours :

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

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

1. Communication bidirectionnelle (10/30 points)

Écrivez un programme `messenger` qui ne reçoit aucun argument sur la ligne de commande et qui crée deux processus (père et fils) qui communiquent via deux pipes pour simuler un système de messagerie simple.

Le père lit des messages à l'entrée standard et les envoie à son fils via le premier

pipe. Chaque message est précédé par un préfixe "[PARENT]".

Le processus fils lit les messages envoyés par son père, compte le nombre de mots dans chaque message, puis renvoie cette information au père via le deuxième pipe sous la forme "[CHILD] Message has X words".

Le père affiche le message original qu'il a envoyé, puis affiche la réponse reçue de son fils.

Quand l'utilisateur introduit un ctrl-D à la place d'une ligne à l'entrée standard, le père ferme l'écriture sur le premier pipe. Le fils détecte la fin de fichier, envoie un message final au père indiquant le nombre total de messages traités ("[CHILD] Total messages processed: X"), puis se termine.

Le père affiche ce message final et se termine également.

Exemple d'exécution :

```
Hello world
[PARENT] Hello world
[CHILD] Message has 2 words
How are you today
[PARENT] How are you today
[CHILD] Message has 4 words
^D
[CHILD] Total messages processed: 2
```

Indications utiles :

1. Vous avez les fichiers `messenger.c`, `Makefile` et la librairie `utils_v2`.
2. Les messages lus au clavier ne feront jamais plus de 100 caractères.
3. Un mot est défini comme une séquence de caractères non-espaces séparée par des espaces.
4. Utilisez la fonction `printf` pour les affichages.

Pour cette question, il vous est demandé de :

1. Compléter le programme `messenger.c`
-

2. Compteur partagé avec synchronisation (10/30 points)

Cette question concerne l'implémentation d'un compteur partagé entre plusieurs processus fils utilisant la mémoire partagée et un sémaphore.

Vous devez implémenter un programme `shared_counter` qui permet à plusieurs processus fils d'incrémenter un compteur partagé de manière synchronisée.

Fonctionnement :

1. Le processus père prend deux arguments : le nombre de processus fils (N) et le nombre d'incréments par fils (M).
2. Le père initialise :
 - Un compteur entier en mémoire partagée (initialisé à 0)
 - Un sémaphore pour protéger l'accès au compteur (mutex)
 - Un pipe pour recevoir les notifications de fin
3. Le père crée N processus fils.
4. Chaque processus fils :
 - Effectue M incréments du compteur partagé (chaque incrément doit être protégé par le sémaphore)
 - Affiche son PID et la valeur du compteur après chaque incrément
 - Envoie son PID via le pipe pour signaler qu'il a terminé
5. Le père attend que tous les fils aient terminé, puis affiche la valeur finale du compteur et vérifie qu'elle est correcte (doit être égale à $N \times M$).

Utilisation :

```
./shared_counter 2 3
```

(2 processus fils, 3 incréments chacun)

Exemple d'exécution :

```
$ ./shared_counter 2 3
Process 1234: counter = 1
Process 1235: counter = 2
Process 1234: counter = 3
Process 1234: counter = 4
Process 1235: counter = 5
Process 1235: counter = 6
Process 1234 finished
Process 1235 finished
Final counter value: 6
Expected: 6 - CORRECT
```

Indications utiles :

1. Utilisez `sem_create()`, `sem_down0()`, `sem_up0()` et `sem_delete()` de la librairie `utils_v2`.
2. Le sémaphore doit être initialisé à 1 pour fonctionner comme un mutex.
3. Chaque incrément doit être atomique (protégé par le sémaphore).

Pour cette question, il vous est demandé de :

1. Compléter le programme `shared_counter.c`
2. Compléter le fichier `Makefile`

3. Serveur d'enregistrement de clients (10/30 points)

Écrivez un programme `registration_server` qui implémente un serveur d'enregistrement simple permettant à plusieurs clients de se connecter, d'envoyer leur nom, et de stocker ces informations en mémoire partagée.

Fonctionnement :

1. Le serveur prend un argument : le port d'écoute.
2. Le serveur initialise :
 - Un socket d'écoute sur le port spécifié

- Une zone de mémoire partagée pour stocker les informations des clients
 - Un sémaphore pour protéger l'accès à la mémoire partagée
3. Le serveur accepte les connexions entrantes en boucle et pour chaque client :
 - Crée un processus fils pour gérer le client
 - Le processus fils lit le nom du client (maximum 50 caractères)
 - Ajoute le nom et le socket du client dans la mémoire partagée (protégé par sémaphore)
 - Affiche "Client [nom] enregistré avec socket [fd]"
 - Ferme la connexion et termine le processus fils
 4. Le serveur peut gérer jusqu'à 10 clients simultanément.
 5. Quand le serveur reçoit un signal SIGINT (Ctrl+C) :
 - Il affiche la liste de tous les clients enregistrés
 - Nettoie les ressources (mémoire partagée et sémaphore)
 - Se termine proprement

Utilisation :

```
./registration_server 8080
```

Exemple d'exécution côté serveur :

```
$ ./registration_server 8080
Serveur d'enregistrement démarré sur le port 8080
Client Alice enregistré avec socket 4
Client Bob enregistré avec socket 5
Client Charlie enregistré avec socket 6
^C
=== Liste des clients enregistrés ===
Nombre de clients : 3
Client 1 : Alice (socket 4)
Client 2 : Bob (socket 5)
Client 3 : Charlie (socket 6)
Nettoyage des ressources...
Serveur arrêté
```

Test avec telnet :

Vous pouvez tester votre serveur avec plusieurs terminaux en utilisant telnet :

```
telnet localhost 8080
```

Puis tapez un nom et appuyez sur Entrée.

Test avec le client fourni :

Un programme client `registration_client` est également fourni pour tester votre serveur :

```
./registration_client 127.0.0.1 8080 Alice  
./registration_client 127.0.0.1 8080 Bob  
./registration_client 127.0.0.1 8080 Charlie
```

Indications utiles :

1. Utilisez `ssocket()`, `sbind()`, `slisten()`, `saccept()` de la librairie `utils_v2`.
2. Utilisez `sshmget()`, `sshmat()`, `sshmdt()`, `sshmdlete()` pour la mémoire partagée.
3. Utilisez `sem_create()`, `sem_down0()`, `sem_up0()`, `sem_delete()` pour les sémaphores.
4. Utilisez `fork_and_run1()` pour créer les processus fils.
5. Gérez le signal `SIGINT` avec `ssigaction()`.
6. Les noms des clients ne dépasseront jamais 50 caractères.

Structure suggérée pour la mémoire partagée :

```
#define MAX_CLIENTS 10  
#define MAX_NAME_LEN 50  
  
struct client_info {  
    char name[MAX_NAME_LEN];  
    int sockfd;  
};  
  
struct shared_data {  
    struct client_info clients[MAX_CLIENTS];  
    int nb_clients;  
};
```

Pour cette question, il vous est demandé de :

1. Compléter le programme `registration_server.c`
 2. Compléter le fichier `Makefile`
-

4. Fichiers fournis

Sur EvalMoodle, vous trouverez les fichiers suivants :

1. `messenger.c` (Q1)
 2. `shared_counter.c` (Q2)
 3. `registration_server.c` (Q3)
 4. `registration_client.c` (client de test pour Q3)
 5. `Makefile` (Q1, Q2 & Q3)
 6. `utils_v2.h` (Q1, Q2 & Q3)
 7. `utils_v2.c` (Q1, Q2 & Q3)
 8. Ce fichier d'instructions
-

5. À remettre

Sur EvalMoodle, les fichiers `messenger.c`, `shared_counter.c`, `registration_server.c` et `Makefile` complétés.

Bonne chance !