Nom 1 : DEBEIR Luca **Nom 2 :** SERAFIN Solène

TP Synchronisation des processus, tubes et signaux

Module F.A.R. IG3 _ Feuille à rendre en fin de TP

0 - Echauffement

Chargez les fichiers du TP depuis Moodle sur votre machine dans un dossier spécifique que vous rangerez au bon endroit, puis ouvrez un terminal positionné dans le dossier.

I - C'est du lourd

Ouvrez `cpt-lourd.c` dans un éditeur de code (sublime text par exemple, s'il n'est pas sur votre machine, téléchargez la version 2 -- gratuite pour votre S.E.).

Répondez aux questions suivantes SANS COMPILER ET EXECUTER le programme (ne pas tricher, merci ;)

1) combien de processus sont créés par ce programme

500

2) Quel est le but de ce programme ?

Le but de ce programme est de créer des processus fils et d'afficher leur ordre.
Il termine également les processus.

3) Que va-t-il se passer lors de l'exécution, c'est-à-dire que va produire l'exécution de ce programme à l'écran ?

Normalement la somme doit s'incrémenter à chaque création d'un processus fils

Ca y est, vous avez enfin le droit de compiler le programme et de l'exécuter.

4) Etes-vous surpris du résultat de l'exécution du programme ? pourquoi ? expliquez ce qui ne marche pas bien :

Oui car la somme vaut toujours 1
Le problème vient du fait que la somme est une variable qui n'est pas partagée.
C'est donc une variable locale.

5) Pourquoi c'est toujours la ligne du processus père qui s'affiche en dernier ?

La ligne du processus père s'affiche toujours en dernier à cause de la fonction wait(0) qui attend que tous les processus fils meurent

II - Se sentir plus léger

Ouvrez maintenant dans l'éditeur de code `cpt-leger.c` et répondez aux questions suivantes SANS LE COMPILER ET L'EXECUTER (hop hop hop, on ne triche pas non plus) :

1) Quelle(s) différence(s) avec le programme précédent ?

La fonction void
La somme est une variable glocale
Il utilise des threads à la place de fork() pour créer des processus fils

2) Prévoyez-vous que son exécution va donner ou pas un résultat différent du programme précédent ? pourquoi ?

La somme devrait s'incrémenter correctement car il y a un partage des données dorénavant

Allez-y, vous avez maintenant le droit de compiler et exécuter ce programme.

3) Que constatez-vous ? expliquez (sauf si vous aviez deviné juste ci-dessus).

Cette fois ci la variable est partagée la somme est donc bien incrémentée.

4) Rappelez quel est l'avantage des threads sur les processus classiques (créés par un appel système `fork()`) :

L'avantage des threads et le partage de données

5) Mais au fait vous aurait-on menti : vérifions si les processus créés par `fork()` ont bien un numéro de processus différent les uns des autres. Comment afficher le numéro de processus (pid) de chaque processus (et du père aussi) ?

On affiche le numéro de processus de chaque processus avec getpid()
On affiche le numéro du processus père avec getppid()

6) Modifiez `cpt-lourd.c` pour que chaque processus (père et fils) affichent leur pid puis indiquez ce que vous constatez :

Chaque fils ont un pid différent avec le même ppid du père

7) Faites la même modification pour `cpt-leger` et indiquez ce que vous constatez :

Tous les processus ont tous le même pid et ppid

8) Dans les deux cas, si vous avez la chance que la commande `pstree` soit installée, lancez-la depuis un autre terminal pendant que chacun des programmes ci-dessus s'exécute, et vérifiez les filiations éventuelles (sinon avec la commande `ps -l`). Ce que vous observez est comme attendu ou pas (pourquoi) ?

On rajoute sleep(10) juste au début du if et on rentre « ps -edf » pour les processus lourds et « sc -T -C cpt-leger » pour les processus légers (juste apèrs long i dans le main()

9) Vérifions maintenant la charge pour le système d'un programme comparé à l'autre. Après avoir consulté `man time`, expliquez à quoi sert cette commande en deux lignes (max) :

Elle permet d'afficher le temps que met 1 programme à s'executer, ainsi que le temps consommer par le système et le temps mis pour exécuter la sortie d'erreur standard

10) Lancez maintenant les deux programmes précédés de la commande time. Quels temps obtenus pour `cpt-leger` en moyenne détaillez user et système :

time ./cpt1 —>real : 0m4.024s user 0m0.000s sys 0m0.024s

11) Idem pour `cpt-lourd` :

time ./cpt2 —> real 0m1.010s user 0m0.000s sys 0m0.032s

12) Conclusion : quel programme est le plus coûteux pour la machine à faire tourner ? Comment expliquer ça ?

Il y a création de 500 processus contre 1

III Les bons comptes font les bons amis

Dans `cpt-leger` passez le nombre de processus créés de 500 à 10 000. Lancez plusieurs exécutions du programme, que constatez-vous ?

Le programme ne fonctionne pas car la somme n'est pas correctement calculée.

1) Complétez les phrases suivantes avec les mots proposés en dessous :

Le problème constaté vient du fait que le programme contient une section critique et que l'accès à celle-ci n'est pas protégé. Pour que le programme fonctionne il faut garantir une exclusion mutuelle. Ceci peut être réalisé grâce au mécanisme des sémaphores qui permet de plus d'éviter attente active contrairement à d'autres solutions logicielles.

Mots à utiliser (vous pouvez les barrer au fur et à mesure pour un total retour en enfance) : exclusion mutuelle - tube - inclusion indépendante - ~~attente active~~ - ~~pâte à crêpe~~ - sémaphore - ~~c'est moi l' plus fort~~ - processus - signaux - attention - section christique - section critique - attente passive - ~~attente lascive~~.

2) Pour remédier au problème ci-dessus, on va maintenant regarder le programme `mutex-thread.c` . Ouvrez ce programme et remarquez comment en faisant attendre les processus fils un temps non prévisible à un endroit critique, on peut utiliser bien moins de threads pour provoquer le soucis repéré précédemment : compilez et exécutez le programme pour vérifier.

Devinette : que fait l'instruction pthread_mutex_init (&mutex, NULL); située dans le main ? (aide : consultez la diapo 73 du cours 1 et ses copines autour)

L'instruction pthread_mutex_init (&mutex, NULL) permet d'initialiser le mutex.Plus précisément des objets de type pthread_mutex_t

3) A quelle valeur le sémaphore est initialisé après cette instruction ? Est-il possible d'utiliser la même instruction différemment pour donner une valeur plus importante au sémaphore ? si oui comment ?

Il est initialisé à 1 car un mutex est toujours un sémaphore de valeur 1
On ne peut pas donner une valeur plus importante au mutex, il faudrait utiliser un autre sémaphore.

4) Dans quel cas voudrait-on donner une valeur plus importante à un sémaphore ?

On voudrait donner une valeur plus important à un sémaphore dans le cas où l'on souhaite autoriser plus d'un thread à entrer dans la section critique, sachant que la valeur du sémaphore est le nombre maximum autorisé.

5) Placez les instructions suivantes au bon endroit dans le code (à nouveau voir diapos du cours, si nécessaire) :

```
pthread_mutex_lock (&mutex);

pthread_mutex_unlock (&mutex);
```

Puis compiler et exécuter le code pour voir si le soucis de compteur est réglé ou pas (vérifiez sur plusieurs exécutions). Vérifiez déjà sur plusieurs exécutions du programme que le soucis est réglé, puis pour vérifier d'une autre façon, commentez la partie qui fait dormir les processus et passez à 10 000 processus. Est-ce ok au bout de 10 essais à 10 000 processus ?

En rajoutant les instructions ci dessus tout fonctionne.

IV - Signaux et tube

Revenez sur le cours numéro 2 et écrivez les programmes nécessaires pour les exercices suivant :

- exercices p13

- exercices p20

V - Le tout à la fois

Ecrivez un programme qui prend deux arguments en entrée : un nom de dossier de départ, et un nom de dossier d'arrivée. Le programme doit copier tous les fichiers (non récursivement) du dossier de départ dans le dossier d'arrivée (qu'on suppose créé à la main dans un premier temps), en faisant appel à un thread différent pour chaque fichier à copier. Si le fichier ne peut pas être copié, alors le thread enverra un signal indiquant une situation à problème en écrivant son numéro et le fichier à problème dans un fichier partagé du nom de `errors.log` . Bien sûr l'accès à ce fichier ne doit être réalisé que par un seul thread à la fois (vous utiliserez un (des ?) sémaphore(s) pour cela).

Pour tester le programme, vous choisirez un dossier de départ où vous aurez enlevé les droits d'accès à certains fichiers (`chmod`).