TP1

Q1)

Le processus fork permet de créer un nouveau processus à partir d’un processus d’origine. Le nouveau processus créé est une copie du processus d’origine. On appelle cette copie le processus fils et l’origine est le père.

pid\_t pidFils = fork(); permet d’appeler le processus fork

Le processus père et fils se lancent en parallèle

je suis le pere, mon numero est 35954, celui de mon fils est 35955

1sec

je suis le fils, mon numero est 35955, celui de mon pere est 35954

15 sec

Q2)

Pour afficher le numéro du père d’un processus : getppid() dans l'exécution du processus père

je suis le pere, mon numero est 106659, celui de mon fils est 106660, celui de mon pere est 5114, pidFils vaut 106660

je suis le fils, mon numero est 106660, celui de mon pere est 106659, pidFils vaut 0

Le père de ce processus est le terminal (bash).

Q3)

Le processus père affiche "je lance un terminal\n"

Le processus fils lance un terminal avec la fonction execl() : (execl("/usr/bin/xterm", "mon terminal",NULL);)

Q4)

ps -a

PID TTY TIME CMD

2600 tty2 00:05:20 Xorg

2618 tty2 00:00:00 gnome-session-b

37714 pts/0 00:02:11 exec2

37715 pts/0 00:00:00 exec2 <defunct>

37783 pts/1 00:00:00 ps

37714 pts/0 00:02:11 exec2

processus qui exécute exec2

37715 pts/0 00:00:00 exec2 <defunct>

processus qui exécute le fils créer par exec2 ( qui est mort )

Q5)

A la première exécution, le programme est séquentiel ( 0 puis 1 ). C’est parce qu’on lance la commande mt.run()

On la remplace par mt.start(), ce qui permet d'appeler mt.run(). On n’appelle généralement jamais mt.run() mais toujours mt.start().

Il y a alors 2 threads en parallèle, on perd la séquence de 0 et de 1 pour une alternance plus aléatoire.

Q6)

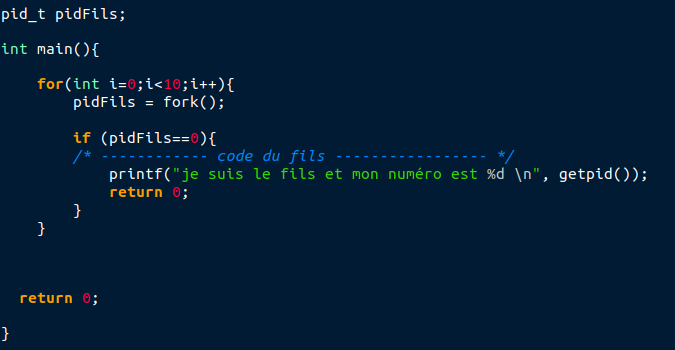
A l'exécution, le père à la main. Ensuite père et fils s’alterne, puis une fois que le père à fini sa boucle il ne reste que le processus fils.

Q7)

En ajoutant les sleeps, on se rend compte que père et fils se lancent en parallèle, le fils à un léger retard sur le père au début ( temps de création ), mais les deux sont en parallèle. On se rend également compte qu’il n’y pas de priorité entre les deux processus.

FIFO ?

Q8)



exit(0) permet de faire mourir le fils tout de suite, ainsi il n’exécute pas le code à nouveau.

( ~ return 0 dans le main )

Sinon un fils exécute tout le code à nouveau donc créer un nouveau fils -> boucle infini.

Q9)

Entrelace.java

Pour chaque thread, j’appelle la fonction strat() qui appelle la fonction run(). Ainsi, pour i=10 on affiche 100\*10 fois l’id. On se rend compte que cet affichage est très aléatoire. Mais semble être séquentiel ( suite de plusieurs répétition importante )

FIFO ?

Avec un sleep(100), on perd tout séquencement -> aléatoire, pas de priorité.

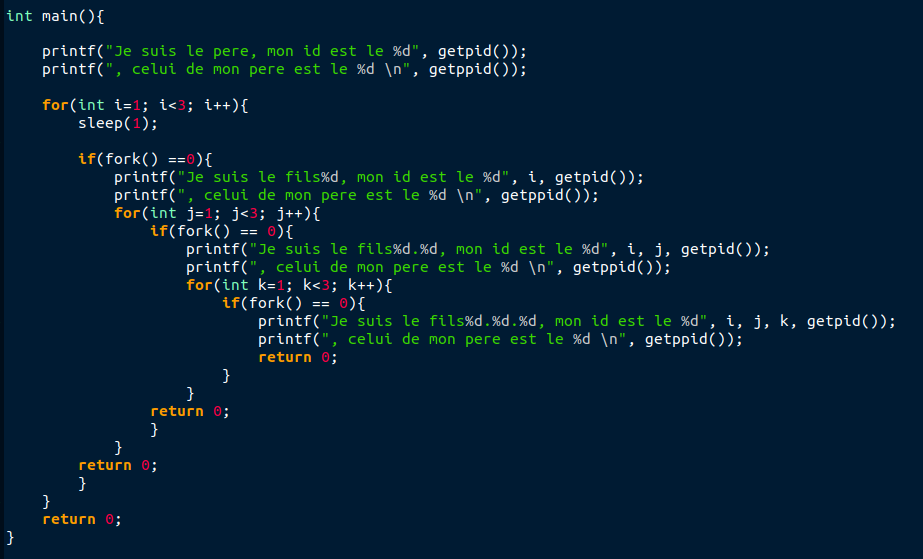
Q10)

L’espace d’adressage du processus père est dupliqué à chaque création d’un fils. Une fois le fils créé, les variables dupliquées sont indépendantes des variables d’origine. Il n’y a pas de synchronisation.

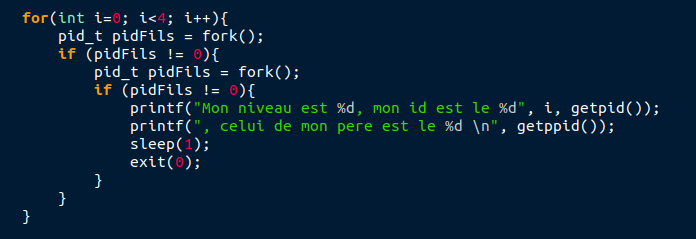
Q11)

En java les types primitifs ne sont pas partagés car ce ne sont pas des objets, mais les objets créés dans le tas sont partagés par les threads par référence.

Q12)



En mieux :



Q13)

Stocker les ancêtres dans une table… chiant à faire en C