Lucas Da Silveira

Dans les codes sources chaque question est représenté par prog1 = question 1, prog2 = question 2 …

TP4
Q1)
La fonction affiche_car prend en paramètre une chaîne de caractères prefixe et affiche le numéro du processus courant (getpid()) ainsi que le numéro du processus parent (getppid()).

La fonction main utilise une boucle for pour créer 100 processus fils en utilisant la fonction fork().
Si fork() renvoie 0, cela signifie que c'est le processus fils, qui appelle alors affiche_car pour afficher ses informations et se termine avec un code d'erreur i en utilisant exit(i).
Si fork() renvoie une valeur négative, cela signifie qu'il y a eu une erreur dans la création du processus fils.
Le processus parent attend la terminaison des fils en utilisant wait(), puis affiche si le fils s'est terminé normalement ou anormalement en utilisant WIFEXITED et WEXITSTATUS.

Q2)
La fonction main utilise une boucle for pour créer 100 processus fils. Dans chaque itération de la boucle, le programme fait :
Il crée un nouveau processus fils en appelant fork(), et le PID du fils est stocké dans la variable pid.

- Si pid est égal à 0, cela signifie que le code suivant est exécuté par le processus fils :
  - Le fils affiche un message indiquant son numéro et son PID à l'aide de printf().
  - Ensuite, il se termine avec un code de retour 0 en utilisant exit(0). Cela signifie que le fils se termine normalement.
- Si pid est inférieur à 0, cela signifie qu'il y a eu une erreur dans la création du processus fils. Le programme affiche un message d'erreur et retourne 1 pour indiquer une erreur.

Après avoir créé les 100 processus fils, le processus parent entre dans une deuxième boucle for pour attendre la terminaison de tous les fils en utilisant la fonction wait(). La fonction wait(NULL) attend la terminaison de n'importe quel fils sans collecter son code de retour.
Une fois que tous les fils se sont terminés et que le processus parent a attendu leur terminaison, le programme retourne 0 pour indiquer qu'il s'est exécuté avec succès.

Q3)
La fonction creer_processus_hierarchie est une fonction récursive qui prend deux paramètres : niveau et max_niveau. Elle est utilisée pour créer une hiérarchie de processus avec une profondeur maximale spécifiée.

Si le niveau actuel atteint ou dépasse la max_niveau, la fonction retourne, mettant fin à la récursion. Cela permet de contrôler la profondeur de la hiérarchie.

Dans la première exécution de la fonction (niveau = 0), un processus fils est créé à l'aide de fork(). Si pid est égal à 0, cela signifie que le code suivant est exécuté par le processus fils :

- Il appelle la fonction affiche_car("Fils:") pour afficher des informations sur le fils.
- Ensuite, il appelle récursivement creer_processus_hierarchie en augmentant le niveau de 1.
- Finalement, le fils se termine avec un code de retour 0 en utilisant exit(0).

Si fork() échoue (retourne une valeur négative), le code affiche un message d'erreur et se termine avec un code de retour 1.

Dans le cas du processus parent, il attend que le fils se termine en utilisant wait(&status).

La fonction main appelle creer_processus_hierarchie avec un niveau initial de 0 et une profondeur maximale de 100. Cela démarre la création de la hiérarchie de processus.

Q4)
La fonction main commence par définir une variable num_processes à 100, ce qui indique le nombre de processus fils à créer.

Ensuite, le programme entre dans une boucle for qui va créer 100 processus fils. Dans chaque itération de la boucle :

- Il crée un processus fils en utilisant fork() et stocke son PID dans la variable pid.
- Si pid est égal à 0, cela signifie que le code suivant est exécuté par le processus fils :
- Le fils appelle la fonction affiche_car("Fils:") pour afficher des informations sur le fils.
- Ensuite, il crée un processus petit-fils en utilisant une deuxième utilisation de fork(). Le PID du petit-fils est stocké dans la variable petit_fils_pid.
- Si petit_fils_pid est égal à 0, le code suivant est exécuté par le petit-fils, qui appelle également affiche_car("Petit-fils:") et se termine avec un code de retour 0 en utilisant exit(0).
- Si la création du petit-fils échoue, le fils affiche un message d'erreur et se termine avec un code de retour 1.
- Le processus fils attend la terminaison de son petit-fils en utilisant wait(NULL) et se termine ensuite avec un code de retour 0.

Si pid est négatif, cela signifie qu'il y a eu une erreur dans la création du processus fils, et le code affiche un message d'erreur et se termine avec un code de retour 1.

Après avoir créé tous les fils et petits-fils, le processus parent entre dans une deuxième boucle for pour attendre la terminaison de tous les fils en utilisant wait(&status).

Une fois que tous les processus se sont terminés, le programme affiche un message indiquant que tous les fils et petits-fils sont terminés, puis le processus initial se termine avec un code de retour 0.

Q5)
La fonction main commence par une boucle for qui s'exécute 100 fois. Dans chaque itération, il crée un processus fils en utilisant fork().

Si la création du processus fils échoue (si pid < 0), le programme affiche un message d'erreur et se termine avec un code de retour 1.

Si pid est égal à 0, cela signifie que le code suivant est exécuté par le processus fils. Le processus fils crée ensuite un petit-fils en utilisant une deuxième utilisation de fork().

Si la création du petit-fils échoue (si petit_fils_pid < 0), le fils affiche un message d'erreur et se termine avec un code de retour 1.

Si petit_fils_pid est égal à 0, cela signifie que le code suivant est exécuté par le petit-fils. Le petit-fils affiche "Petit-fils" à l'écran, puis se termine avec un code de retour 1.

Dans le cas où petit_fils_pid n'est ni < 0 ni = 0, il s'agit du processus fils. Le processus fils attend que le petit-fils se termine en utilisant wait(), puis affiche si le petit-fils s'est terminé normalement ou anormalement en fonction du code de retour du petit-fils.

Si le PID != 0 (processus parent), il attend que le fils se termine, puis affiche également si le fils s'est terminé normalement ou anormalement en fonction du code de retour du fils.

Une fois que les deux processus fils et leurs petits-fils ont terminé, le programme principal se termine avec un code de retour 0.

TP5

Q1)
Dans la fonction main, un tableau de chaînes de caractères args est déclaré. Ce tableau contient les arguments passés au programme que vous souhaitez exécuter. Dans ce cas, un seul argument est spécifié, "xterm", qui est le nom du programme que l'on veut exécuter. Le dernier élément du tableau doit être NULL, indiquant la fin des arguments.

La fonction execvp est appelée avec deux arguments : le nom du programme à exécuter ("xterm") et le tableau d'arguments args. execvp cherche le programme spécifié dans les répertoires du chemin d'accès (PATH) et le remplace par le programme actuel.

Si execvp réussit à exécuter le programme spécifié, il ne retourne pas, car il remplace le processus courant par le nouveau processus. Par conséquent, si l'exécution atteint ce point, cela signifie que execvp a échoué. Dans ce cas, la fonction perror est appelée pour afficher un message d'erreur lié à execvp.

En fin de compte, la fonction main retourne 1 pour indiquer une erreur.

Q2)
La fonction main prend en entrée les arguments de la ligne de commande, qui sont représentés par argc (le nombre d'arguments) et argv (un tableau de chaînes de caractères contenant les arguments).

La première instruction if (argc < 2) vérifie si l'utilisateur a fourni au moins un argument supplémentaire en plus du nom du programme lui-même. Si ce n'est pas le cas, le programme affiche un message d'utilisation sur la sortie d'erreur standard (stderr) indiquant comment utiliser le programme et renvoie 1 pour indiquer une erreur.

Si au moins un argument supplémentaire a été fourni, le programme utilise la fonction execlp pour exécuter la commande spécifiée par l'utilisateur. La fonction execlp prend au moins trois arguments :

- Le premier argument (argv[1]) est le nom de la commande que l'utilisateur souhaite exécuter.
- Le deuxième argument (argv[1]) est également le nom de la commande. Il est commun d'utiliser le nom de la commande elle-même comme premier argument.
- Le troisième argument (NULL) indique la fin de la liste des arguments à passer à la commande.

Si execlp réussit à exécuter la commande, le processus courant est remplacé par la commande spécifiée, et le code qui suit l'appel à execlp ne sera pas exécuté.

Si execlp échoue, il renverra -1, et la fonction perror est utilisée pour afficher un message d'erreur associé à execlp. Ensuite, le programme renvoie 1 pour signaler une erreur.

Q3)
La fonction main utilise la fonction execlp pour tenter d'exécuter la commande "sudo bash".

La fonction execlp prend plusieurs arguments :

- Le premier argument ("sudo") est le nom du programme que l'on veut exécuter.
- Les arguments suivants ("sudo", "bash") sont les arguments passés au programme. Dans ce cas, "sudo" est le nom du programme, et "bash" est un argument indiquant au shell de lancer un shell Bash.
- Le dernier argument (NULL) indique la fin de la liste des arguments à passer à la commande.

Si execlp réussit à exécuter la commande "sudo bash", le processus courant est remplacé par le shell Bash avec les privilèges root, et le code qui suit l'appel à execlp ne sera pas exécuté.

Si execlp échoue, il renverra -1, et la fonction perror est utilisée pour afficher un message d'erreur associé à execlp. Ensuite, le programme renvoie 1 pour signaler une erreur.

TP6

J’ai pas eu le temps d’expliquer mon code, mais j’ai commencé la question 1