TD/TP : Gestion des processus

- Tous les programmes sont à implanter en C.
- Lorsqu'il est demandé d'utiliser un éditeur de texte, vérifiez que l'éditeur n'ajoute pas systématiquement un retour à la ligne lorsque vous avez terminé votre saisie : cela peut influencer le déroulement de votre programme suivant ce que vous faites.

Exercice 1 Arborescence de processus

À l'aide de la commande `ps(1)`, affichez la liste des processus.

1. Quels sont les états possibles pour les processus ?
2. Cherchez à quoi correspondent les processus dont le père a le PID 0.
3. Que remarquez-vous sur l'arborescence des processus par rapport au processus `init` ?
4. Que fait la commande `top(1)` ?

Exercice 2 `kill(1)`, oups !

1. Ouvrir un nouveau terminal.
2. À l'aide de la commande `echo $$`, afficher le PID de ce terminal.
3. Dans le premier terminal, arrêtez le nouveau terminal créé à l'aide de la commande `kill(1)`.

Attention, cette commande est à utiliser uniquement s'il n'est pas possible de terminer normalement le processus !

Exercice 3 UNIX to Windows

Recherchez les informations suivantes :

— Quel est l'appel système sous Windows qui correspondrait à l'appel système `fork(2)` de UNIX ?

— Quels paramètres prend t-il ?

Exercice 4 UID

1. Écrivez un programme qui affiche ses UID réel et effectif.
2. Exécutez votre programme.
3. Utilisez (dans le shell) la commande `chown(1)` pour déclarer `root` comme propriétaire de ce programme.
   N'oubliez pas de lui accorder les privilèges du propriétaire du fichier.
4. Exécutez votre programme.
   Que remarquez-vous ?

Exercice 5 Appel système `fork(2)`

Écrivez un programme qui crée un processus à l'aide de l'appel système `fork(2)`. Le processus père affichera « Je suis le processus père, mon PID est : ... et le PID de mon père est : ... » et le processus fils : « Je suis le processus fils, mon PID est : ... et le PID de mon père est : ... ». Vous ajouterez aussi un message en dehors de la structure conditionnelle.

1. Utilisez la structure conditionnelle `switch`. À la fin du processus fils, terminez par un `break`.
2. Créez une copie du script fait précédemment. Modifiez-le à présent pour pouvoir tester l'envoi de code de retour : utilisez ici `exit(3)` à la fin du processus fils (à la place de `break`). Cette fonction enverra un code de retour qui sera lu dans l'appel système `wait(2)`.
   Pour afficher le code de sortie du processus fils dans le processus père, utilisez la macro `WEXITSTATUS(status)`.
3. Modifiez le code précédent afin d'ajouter une fonction qui sera appelée à la terminaison du script principal (`atexit(3)`).

Lors de l'exécution de votre programme, vérifiez que le PID du père du processus père correspond au PID du shell.

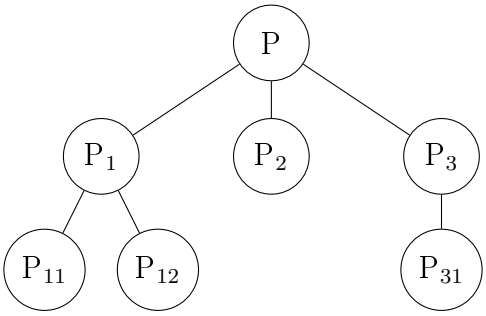
Exercice 6 Appel système `fork(2)` - erreurs

Dans le manuel de l'appel système `fork(2)`, regardez à quoi correspondent les erreurs `ENOMEM` et `EAGAIN`.

Écrivez un programme qui effectue les mêmes tâches que le programme précédent (avec la structure conditionnelle `if`) mais en gérant les erreurs `ENOMEM` et `EAGAIN`. En particulier, créez une boucle pour vous assurer de continuer à tenter la création du processus jusqu'à ce que le noyau ait de la place dans sa table des processus.

Exercice 7 Filiation

Écrivez un algorithme puis un programme permettant de créer l'arborescence de processus suivante.



Exercice 8 Arguments en ligne de commande

Écrivez un algorithme puis un programme qui permet de lister les arguments ayant l'extension txt.

```
etudiant@LinuxVM:~/ProgSys/TP$ ./verif_extension_txt
Pas d'argument saisi.
etudiant@LinuxVM:~/ProgSys/TP$ ./verif_extension_txt test.txt a b c.txt t
Extension cherchée : txt
test.txt : extension OK
c.txt : extension OK
```

FIGURE 1 – Exemple de résultat d'affichage des arguments ayant l'extension txt

.

Quelques fonctions utiles : `strcmp(3)`, `strlen(3)`, `sprintf(3)`, `strstr(3)`.

Pour aller plus loin : proposez une nouvelle version du programme qui cherche dans la liste d'arguments donnée ceux dont l'extension est donnée en dernier argument (voir exemple ci-dessous).

```
etudiant@LinuxVM:~/ProgSys/TP$ ./verif_extension test.c a b.c c
Extension cherchée : c
test.c : extension OK
b.c : extension OK
```

FIGURE 2 – Exemple de résultat d'affichage des arguments ayant l'extension c

.

Exercice 9 Lancement d'un programme

Créez un programme nommé `main_sec.c` qui affiche le nom avec lequel il a été appelé.

Créez un autre programme nommé `test_exec.c` qui lance le programme précédent avec le nom d'exécutable choisi puis avec un autre nom.

Que remarquez-vous ?

Exercice 10 Lancement d'un programme

Écrivez un algorithme puis un programme programme qui liste tous les fichiers des noms de répertoire donnés en arguments avec la commande `ls`. Si aucun argument n'est donné, afficher un message d'erreur.

Après l'appel de la fonction `exec(3)` choisie, placez un `printf(3)` pour afficher un message sur la sortie standard. Que constatez-vous ? Quelle solution proposez-vous pour y remédier ?

Exercice 11 Hiérarchie

Écrire un algorithme puis un programme qui permet de créer une hiérarchie de processus suivant les valeurs données en argument.

La hiérarchie est constituée du père, des fils et des petit-fils.

Le programme prend en argument dans l'ordre : le nombre de fils puis le nombre de petit-fils pour chaque fils.

Par exemple la commande suivante permet de créer trois fils ayant dans l'ordre 1, 3 et 0 fils (petit-fils).

```
./hierarchie 3 1 3 0
```

Il faudra s'assurer que le nombre d'arguments est correct.

On affichera bien-sûr les messages permettant d'identifier la hiérarchie.

Vous commenterez vos résultats en montrant comment vous vérifiez que la hiérarchie demandée est respectée.