Univérsité Cheikh Anta Diop

Faculté des Sciences et Techniques
Département Mathématique et Informatique
Année Universitaire 2021-2022
MS2E
Travaux Pratiques de Système d'Exploitation

Atelier : *Les Processus sous UNIX/Linux*

Dr Mandicou BA

Exercice 1 : Analyse et interprétation de code source

Après avoir complété les programmes suivants, dites ce qu'ils réalisent. Combien de processus sont créés ? Indiquez le résultat à l'écran.

```c
//Programme 1
int main(int argc, char *argv []) {
    int p1, p2, p3;
    p1 = fork();
    p2 = getpid();
    p3 = getppid();
    printf("p1=%d - p2=%d - p3=%d\n",p1,p2,p3);
    return 0;
}

//Programme 2
#define N 10
int main(int argc, char *argv []) {
    int i = 1;
    while (fork() == 0 && i <= N) i++;
    printf("%d\n",i);
    exit(0);
}

//Programme 3
int main(int argc, char *argv []) {
    int pid[3], i;
    for (i = 0; i < 3; ++i) {
        pid[i] = fork();
    }
    printf("%d %d %d\n", pid[0], pid[1], pid[2]);
    return 0;
}

//Programme 4
int main ()
{
    pid_t valeur, valeur1;
    printf("Affichage 1 --> Processus pere [%d] : mon père à moi est [%d] \n",getpid(), getppid());
    valeur = fork();
    printf("Affichage 2 --> retour fork [%d] - Processus fils [%d]; mon père est [%d] \n",valeur, getpid(), getppid());
    valeur1 = fork();
    printf("Affichage 3 --> retour fork [%d] - Processus fils [%d]; mon pere est [%d] \n",valeur1, getpid(),getppid());

    return 0;
}
```

Exercice 2 : Création de processus

1. Écrivez un programme qui illustre la création de 5 processus fils qui se contentent d'afficher leur numéro (compris de 0 à 4), ainsi que leur PID.
2. Écrivez maintenant un programme qui permet la création de N processus fils, avec N passé en paramètre :
   (a) Les N fils crées doivent afficher leur numéro i (compris de 0 à $N-1$), leur PID, ainsi que leur PPID. L'affichage est réalisé par l'appel d'une procédure « **void parler(int i)** ». Après l'affichage des informations, le fils i est tué.
      — La procédure « **void parler(int i)** » est à écrire en dehors de la méthode **main** et est appelée en cas de succès de création de chaque fils i.
   (b) Quant au père, il stocke dans un tableau de taille N les PID de tous ses fils créés. Puis, il affiche tableau après la mort de fils.
   (c) En outre, le nombre d'arguments passé au programme doit être contrôlé. Le programme doit se terminé immédiatement si le nombre de paramètres qui lui est passé est supérieur à 1.
   (d) Tester le bon fonctionnement du programme

Exercice 3 : Terminaison de processus

Écrivez un programme illustrant l'utilisation de l'appel système **wait** : un processus fils est crée et le processus père attend la fin de l'exécution de son fils.

1. Il faut reprend le principe de l'exercice 2, question 2 : le fils crée affiche son PID, ainsi que son PPID. L'affichage est fait par l'appel d'une procédure « **void parler()** » écrite en dehors de la méthode **main** et appelée en cas de succès de création du fils.
2. La procédure « **void parler()** », après l'affichage, doit faire dormir le processus fils pendant 5 secondes. Puis, elle tue le processus fils (avec l'appel système **exit**) en spécifiant une valeur à retourner.
3. Le père attend la mort du fils avec le macro WIFEXITED, tout en gérant les éventuelles erreurs qui peuvent se produire.
4. Testez tous les autres macros vu en cours.

Exercice 4 : Pour aller plus loin : étude de l'appel système waitpid !

L'appel système waitpid() {**pid_t waitpid(pid_t pid, int *status, int options);**} suspend l'exécution du processus appelant jusqu'à ce que le fils spécifié par son **pid** ait changé d'état. Par défaut, waitpid() n'attend que les fils terminés, mais ce comportement est modifiable avec l'argument **options** (*voir la page man pour plus de détails*).

La valeur de **pid** peut être l'une des suivantes :

1. < -1 : attendre la fin de n'importe lequel des processus fils dont le GID du processus est égal à la valeur absolue de pid.
2. -1 : attendre n'importe lequel des processus fils.
3. 0 : attendre n'importe lequel des processus fils dont le GID du processus est égal à celui du processus appelant.
4. > 0 : attendre n'importe lequel des processus fils dont le PID est égal à pid.

Question : *Écrivez un programme qui illustre le fonctionnement de **waitpid**. Par exemple, cinq fils sont crées, ils se mettent en pause (avec sleep) pendant un temps aléatoire. Le père attend la fin de chaque fils, dans l'ordre de création.*