Système d'exploitation II

TP 4 : Processus zombies

Objectifs du TP

1. Comprendre ce qu'est un processus zombie.
2. Identifier les situations qui conduisent à la création de processus zombies.
3. Utiliser wait() et waitpid() pour éliminer les processus zombies.

Un processus Zombie est l'état d'un processus dont le père n'a pas encore pris connaissance de sa terminaison. Il se caractérise dans le résultat affiché par la commande ps par la valeur Z de l'état, 0 pour l'espace mémoire et sur de nombreuses versions 'defunct' dans le champ commande. Pour éviter l'état zombies on utilise les primitives wait et waitpid

Ce sont ces primitives et elles seules qui permettent l'élimination des processus Zombies et permettent la synchronisation d'un processus sur la terminaison de ses descendants avec récupération des informations relatives à cette terminaison.

L'appel système wait() permet à un processus appelant de suspendre son exécution en attente de recevoir un signal de fin de l'un de ses fils.

Syntaxes :

```
int *status;
int wait(status);
```

```
int status;
int wait(&status);
```

Les principales actions de wait sont :

- Recherche de processus fils dans la table des processus : si le processus n'a pas de fils, le wait renvoie une erreur, sinon elle incrémente un compteur.
- S'il y a un zombie, il récupère les paramètres nécessaires et libère l'entrée correspondante de la table des processus.
- S'il y a un fils mais pas de zombie, alors le processus se suspend (état endormi) en attente d'un signal.
- Lorsque le signal est reçu, la cause du décès est stockée dans la variable "status". 3 cas à distingués : processus stoppé, processus terminé volontairement par exit, processus terminé à la suite d'un signal.
- Enfin le wait permet de récupérer le PID du processus fils : PID=wait(&status).

La primitive **waitpid** permet de sélectionner parmi les fils du processus appelant un processus particulier.

Exercice 1 :

- Exécuter le programme suivant :

```
#include <unistd.h>
#include <stdio.h>
```

```c
#include <stdlib.h>

int main() {
    if (fork()==0) {
        printf("processus fils %d\n", getpid());
        exit(10);
    }
    printf("processus père %d\n", getpid());
    for (;;);   /* le processus père boucle */
    return 0;
}
```

1) Taper la commande ps –ef, qu'est-ce que vous remarquez ?

```
ouassit@Youssef:~$ ps -eo uid,pid,ppid,stat,cmd
  UID     PID    PPID STAT CMD
    0       1       0 Ss   /sbin/init
 1000     541     367 Sl   /usr/libexec/at-spi2-registryd --use-gnome-session
 1000     547     367 Ssl  /usr/libexec/dconf-service
 1000   25736     309 S    -bash
 1000   25743   25736 R    ./test
 1000   25744   25743 Z    [test] <defunct>
 1000   27130     309 R+   ps -eo uid,pid,ppid,stat,cmd
```

Deux processus ont été créés : père pid = 25743 et fild pid = 25744

2) Dans ce cas c'est qui le processus zombie ?

Le processus zombie est le processus fils de pid = 25744 dans cet exemple. Car il a terminé son exécution (appelé exit) mais le processus parent ne l'a pas attendu (wait).

3) Généralement qu'elles sont les causes de la création des processus zombies ?

Principalement, un processus devient zombie parce que le processus parent n'a pas récupéré le statut de terminaison de l'enfant (via wait ou waitpid).

4) Modifier le code en ajoutant la function wait().

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
#include <stdlib.h>

int main() {

    if (fork()==0) {
        printf("processus fils %d\n", getpid());
```

```
        exit(10);
    }
    if(wait(NULL)!=-1){
        printf("Prcessus fils est terminé!");
    }
    printf("processus père %d\n", getpid());
    for (;;);   /* le processus père boucle */
    return 0;
}
```

Exercice 2 :

1) Créer l'arborescence suivante :

Main crée trois processus, PF1, PF2 et PF3, ce dernier crée deux processus PF4 et PF5. Simuler un travail de 20 second à exécuter par PF3 avec la fonction sleep(20). Et de 10 second les deux fils PF4 et PF5.

```
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
#include <stdlib.h>

int main() {
    int pf1, pf2, pf3, pf4, pf5;

    if ((pf1 = fork()) == 0) {
        printf("Processus PF1 %d\n", getpid());
        exit(0);
    }

    if ((pf2 = fork()) == 0) {
        printf("Processus PF2 %d\n", getpid());
        exit(0);
    }

    if ((pf3 = fork()) == 0) {
        if ((pf4 = fork()) == 0) {
            printf("Processus PF4 %d\n", getpid());
            sleep(10);
            exit(0);
        }
```

```
        if ((pf5 = fork()) == 0) {
            printf("Processus PF5 %d\n", getpid());
            sleep(10);
            exit(0);
        }

        printf("Processus PF3 %d\n", getpid());
        sleep(20);
    }

    return 0;
}
```

2) Si on tue le PF3 avant que les files PF4 et PF5 se termine. Qu'est ce qui arrive ?

Si PF3 est tué alors que ses fils sont encore en cours d'exécution, ses processus enfants (PF4 et PF5) deviendront des orphelins et seront adoptés par init ou systemd, qui récupérera leur statut quand ils termineront.

3) Réexécutez le programme sans tuer le PF3. Qu'est-ce que vous remarquez après 10 secondes d'exécution de votre programme ?

Après 10 secondes les deux files deviennent des processus zombies.

```
ouassit@Youssef:~$ ps -o uid,pid,ppid,stat,cmd
  UID     PID    PPID STAT CMD
 1000     297     296 Ss   -bash
 1000    1269     297 S    -bash
 1000    1276    1269 S    ./prog
 1000    1277    1276 S    ./prog
 1000    1278    1276 S    ./prog
 1000    1279    1276 S    ./prog
 1000    1280    1279 Z    [prog] <defunct>
 1000    1281    1279 Z    [prog] <defunct>
 1000    1294     297 R+   ps -o uid,pid,ppid,stat,cmd
```

4) Expliquer la valeur de la variable **status** utilisée dans la fonction wait.

La variable status utilisée dans les appels aux fonctions wait() ou waitpid() dans un programme C sous Unix ou Linux joue un rôle crucial dans la gestion des processus. Elle sert à récupérer et à interpréter le statut de terminaison d'un processus enfant.

5) Modifier le code en ajoutant une solution pour traiter tous les processus fils.

6) `#include <stdio.h>`

```c
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int pf1, pf2, pf3, pf4, pf5, pid, status;

    printf("Processus MAIN %d\n", getpid());

    if ((pf1 = fork()) == 0) {
        printf("Processus PF1 %d\n", getpid());
        exit(0);
    }

    if ((pf2 = fork()) == 0) {
        printf("Processus PF2 %d\n", getpid());
        exit(0);
    }

    if ((pf3 = fork()) == 0) {
        if ((pf4 = fork()) == 0) {
            printf("Processus PF4 %d\n", getpid());
            sleep(10);
            exit(0);
        }

        if ((pf5 = fork()) == 0) {
            printf("Processus PF5 %d\n", getpid());
            sleep(10);
            exit(0);
        }
        while((pid=wait(&status))!=-1){
            printf("Prcessus fils %d est terminé avec le status %d\n", pid,
WEXITSTATUS(status));
        }
        printf("Processus PF3 %d\n", getpid());
        sleep(20);
    }

    while((pid=wait(&status))!=-1){
        printf("Prcessus fils %d est terminé avec le status %d\n", pid,
WEXITSTATUS(status));
    }
    printf("FIN Processus MAIN %d\n", getpid());
```

```
    return 0;
}
```