	Matière : Système Exploitation 2 Travaux pratiques	Niveau : LF1-INFO
	TP 2 Les commandes wait(), waitpid() et exec()	AU. 2024-2025

Exercice 1 : La chasse aux zombies

1-Interpreter le résultat en exécutant le code et en lançant la commande suivante **dans un autre terminal (il faut avoir deux terminaux ouverts)**

ps aux | grep pid_fils | grep Z+

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

 int main(void) {

   pid_t val;
   val = fork();

    if (val == 0)  {

      printf("Fils : Je suis le fils, la valeur de retour du fork dans mon contexte est %d.\n", val);
      printf("Fils : Termine !\n");
                  }
   else if (val > 0)
   {
    printf("Pere : Je suis le pere, la valeur de retour du fork dans mon contexte est %d.\n", val);
    while (1) // Boucle infinie, le père ne termine jamais !
    usleep(1);
   }
 return (0);
}
```

2- Ajouter d'autres processus zombies avec leur pid.

3- Modifier le code pour ne pas avoir un processus zombie.

Exercice 2 : Récupérer les orphelins

1- tester ce code et lancer dans un autre terminal la commande suivante en interprétant le résultat trouvé :

ps –aux

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

int main(void)
{
   pid_t val;

   val = fork();

   if (val == 0) // Code du fils
   {
      for (int i = 0; i < 10; i++)
      {
         printf("Fils : Je suis le fils, la valeur de retour du fork dans mon contexte est %d, mon PID est %d, mon père était %d.\n", val, getpid(), getppid());
         sleep(1); // Le fils attend un peu pour simuler son exécution
      }
   }
   else if (val > 0) // Code du père
   {
      printf("Père : Je suis le père, la valeur de retour du fork dans mon contexte est  %d, mon PID est %d.\n", val, getpid());

      printf("Je vais mourir maintenant, au revoir beau monde !\n");

      // Attendre un peu avant de terminer pour permettre au fils de devenir orphelin
      sleep(3);

      exit(0); // Le père se termine
   }
   return 0;
}
```

2- Modifier le code pour ne pas avoir un fils orphelin

Exercice 3 :

1- En exécutant ce code, quels sont les statuts de sortie et les états pour chaque processus.
2- Lancer un SIGKILL pour le processus 4 et un SIGTERM pour le processus 5. Vérifier le résultat obtenu.

```
int main() {
   pid_t child_pid;
   int status;

   // Création de 5 processus fils
   for (int i = 0; i < 5; ++i) {
      child_pid = fork();

      if (child_pid < 0) {
         // Erreur lors de la création du processus fils
         perror("fork");
         exit(EXIT_FAILURE);
      } else if (child_pid == 0) {
         // Code exécuté par le processus fils
         printf("Je suis le processus fils %d (PID : %d, PPID : %d)\n", i+1, getpid(), getppid());

         // Chaque processus fils attend un certain temps différent
         sleep(20);
         sleep(i+1);
         // Chaque processus fils termine avec un code de sortie différent
         exit(i+1);}
```

```
      }
// Le processus parent attend la terminaison de tous les processus fils
   while ((child_pid = wait(&status)) > 0) {
      if (WIFEXITED(status)) {
         printf("Le processus fils %d (PID : %d) s'est terminé avec le code de sortie : %d\n",
               child_pid, child_pid, WEXITSTATUS(status));
      } else if (WIFSIGNALED(status)) {
         printf("Le processus fils %d (PID : %d) a été tué par un signal : %d\n",
               child_pid, child_pid, WTERMSIG(status));
      }
   }

   return 0;  }
}
```

Exercice 4 :

Qu'affiche ce code ? déterminer la syntaxe de chaque commande exec.

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

int main() {
   pid_t pid;
   int status;

   for (int i = 1; i <= 6; ++i) {
      pid = fork();
      if (pid < 0) {
         perror("fork");
```

Par exemple : ./ex5 ls -Ral

Exercice 6 :

Écrire un programme réalisant l'exécution d'une commande passée en argument au maximum 7 fois (7 processus fils). Vous pouvez utiliser les macros WIFEXITED et WEXITSTATUS pour renvoyer l'état de terminaison de chaque processus fils.

Annexe :

1- La commande wait() et waitpid():

Wait() renvoie soit :

-le PID du processus fils terminé
- En cas d'erreur -1
Le paramètre status correspond au code de retour du processus fils qui va se terminer

La primitive **waitpid()** permet au processus père d'attendre un **fils particulier**

pid_t waitpid (pid_t pid, int *status, int options)

2- Les macros WIFEXITED() et WEXITSTATUS() :

Ces macros sont utilises pour vérifier si le processus fils s'est terminé normalement et récupérer son code de sortie
Les macros WIFSIGNALED() et WTERMSIG() pour vérifier si le processus fils a été tué par un signal et récupérer le numéro de ce signal.

3- La commande exec:

Lorsqu'un appel exec est effectué, le noyau remplace le code du processus actuel par le code du programme exécutable spécifié, et il réinitialise les registres et les autres données du processus pour correspondre aux nouvelles données de l'image du programme exécutable.

Cela signifie que l'image mémoire, les variables et les structures de données du processus sont entièrement remplacées par celles du nouveau programme.

La famille exec comporte : execl, execle, execlp, execv, execve et execvp

« l » (resp. « v ») indique que les mots composant la ligne de commande du nouveau code exécutable sont passés séparément l'un à la suite de l'autre (rep. Sont passés via un vecteur de mots).

« p » : indique que le chemin d'accès au nouveau code exécutable est sauvegardé dans la variable PATH.

« e » signifie qu'il est possible d'ajouter un vecteur de variables d'environnement.

Par exemple : ./ex5 ls -Ral

Exercice 6 :

Écrire un programme réalisant l'exécution d'une commande passée en argument au maximum 7 fois (7 processus fils). Vous pouvez utiliser les macros WIFEXITED et WEXITSTATUS pour renvoyer l'état de terminaison de chaque processus fils.

Annexe :

1- La commande wait() et waitpid():

Wait() renvoie soit :

-le PID du processus fils terminé
- En cas d'erreur -1
Le paramètre status correspond au code de retour du processus fils qui va se terminer

La primitive **waitpid()** permet au processus père d'attendre un **fils particulier**

pid_t waitpid (pid_t pid, int *status, int options)

2- Les macros WIFEXITED() et WEXITSTATUS() :

Ces macros sont utilises pour vérifier si le processus fils s'est terminé normalement et récupérer son code de sortie
Les macros WIFSIGNALED() et WTERMSIG() pour vérifier si le processus fils a été tué par un signal et récupérer le numéro de ce signal.

3- La commande exec:

Lorsqu'un appel exec est effectué, le noyau remplace le code du processus actuel par le code du programme exécutable spécifié, et il réinitialise les registres et les autres données du processus pour correspondre aux nouvelles données de l'image du programme exécutable.

Cela signifie que l'image mémoire, les variables et les structures de données du processus sont entièrement remplacées par celles du nouveau programme.

La famille exec comporte : execl, execle, execlp, execv, execve et execvp

« l » (resp. « v ») indique que les mots composant la ligne de commande du nouveau code exécutable sont passés séparément l'un à la suite de l'autre (rep. Sont passés via un vecteur de mots).

« p » : indique que le chemin d'accès au nouveau code exécutable est sauvegardé dans la variable PATH.

« e » signifie qu'il est possible d'ajouter un vecteur de variables d'environnement.