Ecole Nationale des Sciences de l'Informatique *A. U. : 2017/2018*

Systèmes d'Exploitation& Programmation Concurrente *II2*

TD/TP

Gestion des Processus/Threads sous Linux

I. Gestion des Processus

TP. 1

Afin de vous familiariser avec les différentes commandes du shell Unix n'hésitez pas à utiliser le `man`.

1) Quel est le processus de `pid 1` ? justifier

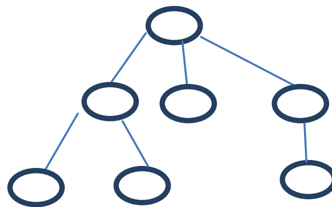
2) Quelle est la différence entre les commandes `ps`et `top`?

3) Que fait la commande `pstree`?

4) Que fait la commande `stracels` (resp. `processus`)?

5) Comment utiliser la commande `ps` pour obtenir la liste des processus en première colonne et leur état en 2ème colonne ? Quels sont les états possibles ?

6) Ecrire un programme C qui engendre 6 processus liés au ancêtre de la manière suivante:



En affichez l'identité de chaque processus ainsi que celle du parent.

Exercice 0 (QCM --Exam. Rat. 04/2017)

Supposons l'entier n initialisé à 0, le nombre de processus créés par l'instruction, ci-dessous, est:

```
while (pid=fork()) if (n >= 5) break ; else n=n+1;
```

a) 4

b) 5

c) 6

d) >10

e) aucune des réponses ci-dessus.

Sélectionnez une ou plusieurs réponses

Exercice 1 (Recueil examens)

1) (Exam 3/01/2017)

Combien de processus sont créés par le code C suivant (où n est un entier initialisé à 0)

```
while (pid=fork()) if (n >= 5) break; else n++;
```

En dessinez le graphe des processus.

2) (DS 14/11/2013)

Soit le programme suivant :

#include <unistd.h> *#include <stdio.h>* *int main ( ) {* *inti=0 ;*	*while (fork() !=0 && i<2)* *i=i+1;* *printf(" Process %d termine avec i=%d \n", getpid(), i);* *return 0;* *}*

Supposez que les appels à la fonction *fork* ne retournent pas d'erreur.

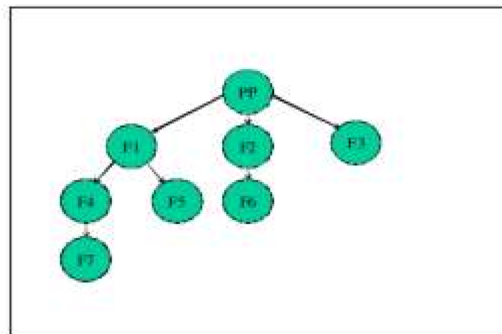
1) Donnez l'arborescence des processus engendrés par ce programme.

2) Peut-on risquer d'engendrer un/des processus orphelin(s) et/ou zombie(s) ? si oui justifiez vos réponses et dire comment peut-on le vérifier ?

3) Modifiez le code de la fonction main de manière à

a) éviter la présence éventuelle d'orphelins et zombies, et

b) créer la nouvelle arborescencesuivante, où PP est le processus principal :



Exercice 2 (DS 16/11/2012)

Combien de processus engendre l'exécution du programme C suivant et en donner l'arborescence.

```
# include <unistd .h>
int main ( void )
{
        fork () && ( fork () || fork () );
        sleep (2);
        return 0; }
```

Exercice 3. (TP)

1) Lancer le programme ci-dessous avec les arguments 10 20. Tapez ps -la dans un autre terminal avant la fin du père, avant la fin du fils. Quels sont les ppid du père et du fils ? Donnez une explication.
2) Lancer le programme ci-dessous avec les arguments 10 0. Tapez ps -la dans un autre terminal avant la fin du père. Que constatez-vous?

```
/**********************************************************/
    #include<unistd.h> /* necessaire pour les fonctions exec */
    #include <sys/types.h>
    #include <unistd.h>
    #include <stdio.h>
    int main(intargc, char * argv[]) {
     pid_tpid;
     intattente_fils,attente_pere;
     if(argc != 3)
          perror("usage: ex1 n m\n");
     attente_pere = atoi(argv[1]);
     attente_fils = atoi(argv[2]);
     switch(pid=fork()) {
          case -1:
               perror("fork error");
          break;
          case 0:
               sleep(attente_fils);
               printf("fils attente finie\n");
          break;
          default:
               sleep(attente_pere);
               printf("pere attente finie\n");
               break;
     }
     return 0 ;
}
```

Exercice 4. (→TP)

Écrire un programme modulaire qui va créer un deuxième processus. Le père va afficher les majuscules à l'écran et le fils les minuscules. Ici, le travail effectué par les 2 processus est trop court et il n'y a pas entrelacement des exécutions. Pensez à mettre un "\n" à la fin de chaque écriture afin de vider le buffer !

Exercice 5. (→TP)

Écrire un programme modulaire qui va créer un deuxième processus. Le père et le fils comptent de 0 à 100000 et l'affiche à l'écran. Le père place un P devant son comptage et le fils un F. Analysez le travail de l'ordonnanceur.

II. TP.Threads

La création et le lancement du thread se fait par :

```
pthread_tth1 ;
int ret ;

pthread_create (&th1, NULL, runDuThread, "1");
if (th1 == NULL) {
      fprintf (stderr, "pthread_create error 1\n") ; exit(0) ;
}
```

Le thread exécutera alors la fonction `runDuThread`dont le prototype est :

```
void* runDuthread (void *param) ;
```

Cette fonction est à écrire par le programmeur pour décrire le comportement du thread. Le paramètre *param*est un pointeurdont la valeur est celle passée en argument (le 4ème) de la fonction *pthread_create*. Il permet de passer des données authread.

Si la fonction main se termine, le programme et tous les threads lancés se terminent aussi. Il faut donc s'assurer avant determiner le programme que tous les threads ont fini leur travail. L'attente de la terminaison d'un thread se fait comme ceci :

```
(void) pthread_join (th1, (void *)&ret) ;
```

Le paramètre *ret*contiendra la valeur retournée par la fonction `pthread_exit (int val)`à exécuter avant de terminer un thread.

1. Écrire un programme qui lance 2 threads. L'un écrira les 26 minuscules à l'écran et l'autre les 26 majuscules.
2. Écrire un programme qui initialise une variable globale à 0 et crée 2 threads. Chacun des threads va incrémenter la variable N fois. Afficher la valeur de la variable à la fin de l'exécution de chacun des threads.