TD de Système n° 8 : fork, wait, exit

On présente les fonctions suivantes :

`pid_t fork(void);`

- crée un clone du processus appelant (duplique tout l'espace d'adressage) ;
- renvoie : le PID du fils dans le processus père, 0 dans le processus fils ;
- renvoie -1 en cas d'erreur.

`pid_t wait(int *status);`

- attend la terminaison d'un processus fils ;
- renvoie le PID du fils qui s'est terminé ou -1 en cas d'erreur ;
- quand `wait` se débloque `status` contient la valeur du code de retour du fils.

`void exit(int status);`

- termine l'exécution d'un processus et demande au système de le supprimer ;
- la valeur `status` est passée au père s'il est en attente sur un `wait`.

Exercice 1 :

Que fait le programme suivant ?

```
#define N 10
int main() {
  int i = 1;
  while (fork() == 0 && i <= N) i++;
  printf("%d\n",i);
  exit(0);
}
```

Exercice 2 :

Combien de « meuh ! » et de « coin ! » affiche le programme suivant ?

```
int main() {
  int i;
  for (i=0 ; i<4 ; i++) {
    int pid = fork() ;
    if (pid != 0)
      printf("meuh !\n") ;
    printf("coin !\n") ;
  }
  exit(0);
}
```

Exercice 3 : processus en cascade

Écrire deux programmes qui créent n processus p_i tels que :

1. pour tout i entre 2 et n, p_i soit le fils de p_{i-1} ;
2. pour tout i entre 2 et n, p_i soit le fils de p_1.

Chaque processus devra afficher son identifiant et celui de son père.

Exercice 4 : Exponentielle

Écrire un programme qui utilise plusieurs processus pour calculer une valeur approchée de $exp(x)$, pour x donné.

Exercice 5 : « `find` » parallélisé

Écrire une commande « `find_parallele` » telle que « `find_parallele` *fic rep* » effectue la recherche d'un fichier de nom *fic* dans le répertoire *rep*. Pour cela, chaque processus parcourt le contenu du répertoire dont la référence lui est fournie ; s'il trouve un fichier de nom *fic*, il en affiche la référence complète ; et pour chaque sous-répertoire, il crée un processus fils chargé d'y poursuivre la recherche.