ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

2e année – Majeure Informatique

TDm 1 : Fork & co...

C. BARÈS

Les manipulations proposées dans ce TDm sont à réaliser sous GNU/Linux. Référez-vous en permanence aux pages de manuel des fonctions utilisées, section 2 ou 3. Pour accéder à ces pages vous pouvez :

- les rechercher sur internet, par exemple sur `http://man7.org/linux/man-pages/`
- taper dans un terminal : `man 2 fork` pour la 2e section du manuel de `fork`.

Les sections qui vont nous intéresser en programmation système sont :

2 Appels systèmes
3 Fonctions de la bibliothèque C standard

Remarque : Évitez les pages de manuel en français, elles ne sont pas à jour...

1 – À table !

1.1 PCB

On considère le programme suivant :

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

void print_PCB()
{
    // TODO: complete this function
}

int main()
{
    pid_t ret = fork();

    printf("fork() has returned: %d\n", ret);
    print_PCB();
    exit(EXIT_SUCCESS);
}
```

Complétez la fonction `print_PCB`, de façon à ce qu'elle affiche pour le processus courant les informations suivantes : PID, PPID, UID, GID ; un exemple de sortie pour un processus unique pourrait être :

```
PCB | PPID: 25934
    | PID:  26451
    | UID:   1000
    | GID:   1000
```

Compilez et exécutez ce programme une fois complété. Le résultat était-il prévisible ? Toutes les valeurs affichées par `print_PCB` sont-elles cohérentes ?

1.2 Père et fils

Modifiez la fonction `main` de la question précédente de manière à ce que le père et le fils affiche une phrase différente à la place de « `fork() has returned:...` ».

1.3 fork ! fork ! fork !

Pouvez-vous prédire combien de processus vont être créés si on exécute le programme suivant ?

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

int main()
{
    fork();fork();fork();
    printf("fork!\n");
    exit(EXIT_SUCCESS);
}
```

Faites un arbre des différents processus créés à chaque `fork()`.

---

2 – Lapins ?

2.1 Du bon usage de la fourchette

Testez le programme suivant. Comment votre système réagit-il à ce programme ? Vous pouvez l'interrompre avec `<ctrl>+c`.

```c
#include <unistd.h>

int main()
{
    while(1) fork();
}
```

Que fait donc ce programme ? (faites un graphe...)

2.2 Savoir tester ses limites

Écrivez un programme qui génère un nombre infini de processus (que des fils, pas de petit-fils) ; utilisez la fonction `sleep()` (`man 3 sleep`) pour que les fils ne meurent pas tout de suite et testez la valeur de retour du `fork()` pour déterminer si tout se passe bien.

Existe-t-il une limite ? Si oui que vaut-elle ?

3 – Wait & sig (si il reste du temps)

3.1

Créez un processus fils qui s'endort pendant 10 secondes avant de se terminer sur un `exit(123)`. Pendant ce temps, le père attend la terminaison de son fils avec `wait`.

a) Quel est le statut retourné par le `wait` quand tout se passe normalement ? (utilisez les macros `WIFEXITED` et `WEXITSTATUS`)
b) Quel est le statut retourné par le `wait` quand vous tuez le fils prématurément par le biais d'un signal de votre choix ? (utilisez les macros `WIFSIGNALED` et `WTERMSIG`)
c) Observez ce qui se passe lorsque le père ne se met pas en attente sur le fils alors que ce dernier est mort.
d) Que se passe-t-il lorsque c'est le père qui est tué prématurément.

Pour toutes ces questions, vous pouvez utiliser les commandes suivantes depuis un 2e terminal :

- `ps -f xxx` pour obtenir des informations sur le processus de pid `xxx`
- `kill -SIGUSR1 xxx` pour envoyer le signal `USR1` au processus de pid `xxx`

3.2

Générez par une boucle *n* processus issus du même père. Mettez le père en attente de tous ses fils. Affichez les statuts des fils au fur et à mesure de leur disparition. Les fils font un `sleep(i)` avec des valeurs de `i` différentes.

4 – Exec + argv = <3

4.1

Créez un programme « padawan » capable de lancer différentes commandes unix quand on lui donne la chaîne de caractères correspondante sur la ligne de commande :

```
$ whoami
ensea
$ ./padawan whoami
Yes, my master:
ensea
$ ./padawan date +%F
Yes, my master:
2020-12-08
```

4.2

En utilisant les arguments `argc` et `argv` de la fonction `main()`, créez un programme qui s'auto-exécute 5 fois de suite (sans utiliser `fork`).

Refaites la même chose mais en utilisant la variable `environ` à la place de `argv`.