TP n°1 - Processus et signaux

Ce premier TP aborde la notion de processus ainsi que les mécanismes de traitement de divers signaux qui vous ont été présentés récemment lors du cours de PRS.

Ce document reprend certains éléments de cours et d'exercices présentés dans le poly PRS de Samir El Khattabi, disponible sur le site Moodle. Les applications seront réalisées en langage C dans un environnement de type Unix.

Vous rendrez à la fin de chaque séance votre travail sur moodle. Vous rendrez à la fin du module un CR complet pour chaque sujet, comprenant :

- Les réponses aux diverses questions présentes dans cet énoncé,
- Les codes sources commentés des programmes réalisés,
- Les jeux d'essais obtenus,
- Une conclusion sur le travail réalisé, les difficultés rencontrées, les connaissances acquises ou restant à acquérir, …

Il sera tenu compte de la qualité de vos pratiques de production de code :

- Lisibilité du code : emploi de commentaires, indentation, constantes symboliques
- Automatisation : Fichiers makefile
- Robustesse : Vérification des valeurs de retours des appels systèmes

1 Découverte des processus

1.1 La commande `ps`

- Préciser son rôle et ses principales options ;
- Commenter le résultat de cette commande utilisée avec les options `aux` ;
- Commenter le résultat de l'option `-O` et citer les utilisations qui vous paraissent les plus intéressantes ;

1.2 Hiérarchie des processus

- Quel est l'intérêt de la commande `pstree` ?
- Produire un programme permettant d'afficher l'identificateur du processus courant et celui de son père.

2 Signaux

2.1 Commande et fonction kill

1. Afficher la liste des signaux pouvant être émis par la commande `kill`
2. Écrire un programme dont la fonction main commence par afficher son PID, et se poursuit par l'instruction `while(1) ;` exécutez-le et notez le comportement obtenu, lorsque vous lui envoyez les signaux suivants : `SIGINT, SIGCHLD, SIGUSR1`
3. Donner la commande man permettant d'obtenir de l'aide sur la fonction `kill()` (inclure dans le compte-rendu, le début de ces informations)
4. Même question sur les fonctions `alarm()` et `pause()`.
    - Comment peut-on qualifier l'attente induite par la commande `pause` ? S'agit-il d'un "appel bloquant" ou d'une "attente active" ?
5. Remplacer l'instruction `while(1) ;` par `alarm(20) ; pause() ;` Exécuter ce programme, et expliquer le comportement de cette application.

Ignorer un signal

1. Dans quel fichier sont contenues les définitions des noms symboliques identifiant les différentes sources de signaux ?
2. Quel fichier faut-il en fait inclure en tête de programme ?
3. Écrire et tester le programme de la page 82, qui désactive le traitement du contrôle-C

Traiter un signal

1. Modifier le programme précédent de manière à ce que le message suivant « Le contrôle-C est désactivé » s'affiche à chaque fois que l'utilisateur tape contrôle-C

3 Création de processus

3.1 L'appel système `fork()`

1. Tester, analyser et expliquer le comportement du programme dont la fonction main contient les instructions suivantes :

```
CHECK(fork(), "ne peut créer pas un nouveau processus");
CHECK(fork(), " ne peut créer pas un nouveau processus)");
printf("Je suis le process %d, fils de %d\n", getpid(), getppid());
```

2. Que réalise l'appel système `exit()` ?
3. Écrire un programme dont la fonction main se termine par une boucle infinie (`while(1) ;`) précédée de la création d'un processus fils, chargé d'afficher son pid, celui de son père puis qui se termine.
   - Comment, lors de l'exécution du programme, mettre alors en évidence la présence d'un processus zombie ?
4. Interrompre l'exécution du programme en lui envoyant un signal, puis exécutez à nouveau la commande `ps`. Que devient le zombie ?
5. Modifier le programme précédent de façon à ce qu'il n'y ait pas de création d'un tel processus (cf. fonctions de type `wait()`)

3.2 Exercice de synthèse

- Un processus qui ne peut être interrompu par un contrôle-C, crée un processus fils et arme une alarme.
- Le processus fils invite l'utilisateur à entrer un mot de passe dans un délai fixé et avec un nombre de tentatives également limité. Il affiche également un message signalant que le contrôle-C est désactivé lorsque l'utilisateur tape contrôle-C.
- Le processus fils réalise la lecture et le contrôle du mot de passe saisi par l'utilisateur et signale à son père la raison de sa terminaison : épuisement des tentatives ou saisie valide du mot de passe.
- Le processus père affichera un message de dépassement de délai si le processus fils ne se termine pas dans le délai imparti. Il affichera dans le cas contraire, si le mot de passe est correct ou non.
- La communication entre le processus fils et son père peut se fait par l'envoi de signal. Pour cela, vous pourrez procéder en définissant plusieurs fonctions de déroutement différentes, ou la même fonction de déroutement au début de laquelle le signal reçu sera testé.
- Il est également possible d'utiliser le statut de retour du fils, en utilisant les macros `WIFEXITED, WIFSIGNALED, WIFEXITSTATUS,…` Vous développerez une seconde version de votre programme utilisant cette solution
- Vous formaliserez votre conception de la solution à l'aide d'un diagramme de séquence pour chaque solution mise en œuvre, que vous ajouterez à votre CR.

4 Priorités des processus

4.1 Priorité « `nice` » et les commandes nice et renice

1. Qu'est-ce que la priorité « `nice` » d'un processus ?
2. Comment afficher la priorité « `nice` » d'un processus ?
3. Entre quelles valeurs peut être comprise cette priorité ?
4. Donner deux exemples de commande affectant la priorité `nice` par défaut d'un processus, le premier avec une priorité faible le second avec une priorité plus forte que d'ordinaire.
5. Quelle est la fonction de la commande `renice` ?

4.2 La commande `top`

1. Précisez son rôle et commentez le résultat de cette commande.

5 Politiques d'ordonnancement

Linux prend en charge plusieurs politiques d'ordonnancement des processus : une politique d'ordonnancement « temps partagé » et deux politiques « temps réel »

1. Donner les caractéristiques de ces trois politiques d'ordonnancement ;
2. Des noms symboliques ainsi que des valeurs numériques sont associées à chacune de ces méthodes, retrouvez leur définition : commencer votre recherche en examinant le fichier sched.h ;
3. Compléter le code source, présenté en annexe sur Moodle, en insérant sous forme de commentaire, les explications concernant les traitements réalisés, tester le programme correspondant et commenter les résultats obtenus.

Note : La compilation d'un programme source nommé `schedule.c` s'effectue à l'aide de la commande suivante :

```
gcc schedule.c -Wall -o schedule
```

L'exécution s'effectue à l'aide de : `./schedule`

4. Pour quelle raison se programme ne fonctionne pas ? Que faut-il faire pour qu'il n'y ait plus d'erreur à l'exécution.