# IFT2245 - Systèmes d'exploitation - TP1 (5%)

David Quiroz Marin et Alexandre St-Louis Fortier
DIRO - Université de Montréal

Disponible : 29/01/2015 - Remise : 12/02/2015 avant 23h55

LISEZ TOUT LE DOCUMENT AVANT DE COMMENCER LE TRAVAIL.

## 1 Introduction

Le but de ce TP est de se familiariser avec l'environnement de développement d'un système d'exploitation de type Linux (Debian 2.6). Plus précisément, on vous demande de travailler avec bash, un interpréteur de ligne de commande pour les systèmes UNIX, afin de scripter quelques tâches relatives à la gestion d'un système d'exploitation. On vous demande aussi d'écrire une petite application système dans le but de vous familiariser avec le langage C/C++ qui sera utilisé tout au long de la session.

## 2 Mise en place

Vous devez utiliser une Machine Virtuelle (MV), i.e., VirtualBox (comme on a vu pendant la session de démonstration). VirtualBox fonctionne très bien sous Windows, Linux et Mac OS, et est facile à utiliser en même temps que votre système. Toutefois, faites bien attention de sauvegarder votre travail régulièrement sur une autre partition que celle de votre MV. Vous pouvez utiliser une clé USB, le système de synchronisation des fichiers Dropbox, ou bien installer les Guest Additions (tel que vu lors de la démonstration).

Rappelez-vous qu'on utilisera une machine virtuelle pré-installée disponible ici, dont l'usager par défaut est `os` avec mot de passe `os` et l'usager administrateur ou `root` a le mot de passe `cmpt351`.

## 3 Travail à faire

Le travail pour ce TP peut se faire seul ou en équipe de deux, par contre le travail à faire reste le même tout comme le barème de correction. Il y a deux exercices à faire. Le premier exercice consiste à écrire un programme permettant d'imprimer la hiérarchie des processus. Le deuxième vous demande d'écrire un script bash permettant d'automatiser certaines tâches.

### 3.1 ptree.cpp (3% [1])

Vous devez écrire une application qui imprime l'arborescense des processus qui roulent sur votre système. La tâche peut sembler difficile, mais Linux, comme beaucoup d'autres systèmes de la famille UNIX, hérite d'un pseudo-système de fichiers `procfs` (monté sur le dossier `/proc`), une interface pour les processus initiée dans la huitième édition de UNIX et rafinée par le système d'exploitation *Plan 9 from Bell Labs*. `Procfs` est en fait une interface permettant d'obtenir (ainsi que de modifier) diverses informations relatives aux processus à travers des opérations de lecture et d'écriture de fichiers.

À l'exécution de la commande `ls /proc` apparait une liste de dossier avec des noms constitués uniquement de chiffres. Ces dossiers correspondent aux *pids* (*Process ID*) des processus associés. Un fichier particulièrement intéressant pour le travail est `/proc/<pid>/stat`. Il contient entre autres le nom du processus ainsi que le *pid* de son parent (*ppid*). Le nom du processus est en fait le deuxième élément du fichier (celui entouré de parenthèses) et le *ppid*

1. 3% de la note finale du cours.

correspond au quatrième élément. Le fichier `/proc/<pid>/stat` n'est malheureusement pas très compréhensible pour les humains. Vous pouvez regarder le fichier `/proc/<pid>/status` afin de vous donner une intuition de `/proc/<pid>/stat`. Notez par contre que les champs ne sont pas nécessairement dans le même ordre.

Avec le *pid*, le nom du processus et le *ppid*, vous avez toutes les informations pour reconstruire la hiérarchie des processus.

### 3.1.1 Syntaxe et affichage

Votre programme doit accepter la syntaxe suivante `ptree [pid]`. Si aucun *pid* n'est fourni, le programme affichera à partir du processus 0. Pour tout *pid* fourni en paramètre le programme doit afficher l'arborescence pour ce *pid*. Un *pid* invalide est simplement ignoré.

On vous demande d'afficher l'arborescence de telle sorte que les enfants d'un processus se trouvent sous leur parent avec une indentation relative à leur parent. Chaque ligne devrait comporter un seul processus identifié par son *pid* suivi de son nom. Il est très important de vous assurer d'utiliser une indentation de deux espaces par niveau d'indentation. Assurez-vous aussi qu'il n'y ait qu'un seul espace entre le *pid* et le nom du processus. Une pénalité pourra être considérée si ce n'est pas respecté.

Voici deux exemples d'affichage :

```
> ptree
0 [SCHED]
  1 init
    273 upstart-udev-br
    ...
  2 kthreadd
    3 ksoftirqd/0
    ...

> ptree 1729 2532
1729 gnome-session
  1913 gnome-shell
    2301 gnome-terminal
      ...
    2532 firefox
      ...
2532 firefox
  ...
```

Remarquez que le parent de `init` et de `kthread` est `[SCHED]` ayant 0 comme *pid*. Ce processus ne se trouve pas dans le dossier `/proc`. En fait, 0 correspond d'une certaine façon à l'ordonnanceur à l'intérieur du noyau. Comme plusieurs processus partagent le processus 0 comme parent, il sera nécessaire de le simuler pour s'assurer que notre arborescence n'a qu'une seule tête. On vous demande donc de construire l'arbre de tel sorte que sa tête soit le processus au nom de `[SCHED]` avec comme *pid* 0.

### 3.1.2 Code et librairie

Pour réaliser ce TP, vous pouvez utiliser le langage C ou C++. Vous pouvez utiliser toute librairie disponible sur la machine virtuelle. Celle-ci doit être en mesure de compiler votre programme. Votre programme doit cependant faire la lecture de fichiers en C/C++ (donc pas d'appel à `exec` par exemple).

Comme point de départ, le fichier `ls.cpp` vous montre comment parcourir des dossiers en C/C++. Bien que l'extension soit `.cpp`, le code est du C valide et peut donc servir de point de départ autant pour ceux qui veulent

programmer en C qu'en C++.

Pour compiler un programme C++, vous devez entrer la commande `g++ <nom du fichier>`. Un fichier `a.out` est écrit et vous pouvez l'exécuter avec la commande `./a.out <liste des paramètres>`

Notez que notre version de `ptree.cpp` importe les mêmes en-têtes que ceux dans le fichier fournis (libre à vous d'utiliser cet ensemble d'en-têtes ou bien d'en diverger).

## 3.2 setup.sh (2% [2])

Écrivez un script `setup.sh` qui effectuera les tâches suivantes :

— Prend en paramètre un nom d'utilisateur et un mot de passe
— Crée un nouvel utilisateur avec le nom ainsi que le mot de passe passé en paramètre. Le nouvel utilisateur doit avoir un répertoire personnel (*home directory*) dans le dossier `/home`
— Créer la hiérarchie de dossiers suivante :
`/home/<username>/tp1`
`/home/<username>/tp1/src`
`/home/<username>/tp1/bin`
`/home/<username>/tp1/result`
Les droits et propriétaire de ces dossiers devraient être les mêmes que si l'utilisateur les avaient créés lui-même.
— Vous devez placer vos fichiers `setup.sh` et `ptree.cpp` à l'intérieur du dossier `/home/<username>/tp1/src`
— Compilez le fichier `ptree.cpp` et placez l'exécutable à l'intérieur de `/home/<username>/tp1/bin` L'exécutable devrait être au nom de `ptree` (pas `a.out`) et devrait avoir les droits nécessaires pour l'exécuter avec l'utilisateur créé.
— Ajoutez `/home/<username>/tp1/bin` à l'intérieur de la variable d'environnement `PATH` de l'usager créé afin qu'il puisse exécuter la commande `ptree`
— Mettez le résultat de l'exécution de `ptree` à l'intérieur du fichier `/home/<username>/tp1/result/zero` et le résultat de l'exécution de `ptree 1 2` à l'intérieur du fichier `/home/<username>/tp1/result/one_two`
— Pour finir, ajoutez un fichier texte `auteur` contenant vos noms à l'intérieur du dossier `/home/<username>/tp1`

Le script doit pouvoir être (et sera lors de la correction) lancé à partir de l'utilisateur `root`

### 3.2.1 Quelques astuces

Vous pouvez explorer comment effectuer ces actions par l'utilisation du terminal de votre machine virtuelle. Essayez d'abord d'exécuter les commandes une à une en essayant d'obtenir le résultat souhaité. Il vous sera plus facile d'écrire le script par la suite.

Pour apprendre les options d'une commande, vous pouvez utiliser la commande `man <nom de la commande>` Cela vous permet de lire le manuel d'une commande. Vous pouvez, par exemple, essayer la commande `man man`

Considérez particulièrement les commandes `cd`, `ls`, `mv`, `cp`, `install`, `mkdir`, `chmod`, `chown`, `useradd`, `gcc`, `g++`, `sudo` et `su` Elles ne sont pas toutes nécessaires car certaines options de l'une correspondent à la composition de quelques autres.

Trouvez comment vous pouvez modifier le fichier `.bash_profile` d'un utilisateur afin de modifier ses variables d'environnement.

Pour vous assurer que vous n'avez pas d'erreurs majeures, connectez-vous avec l'utilisateur créé et assurez-vous que vous pouvez ouvrir et modifier chacun des fichiers créés. Vous devriez aussi être en mesure d'exécuter la com-

2. 2% de la note finale du cours.

mande `ptree` à partir de n'importe quel endroit.

Pour que les droits d'accès et propriétaire soient corrects, assurez-vous qu'ils correspondent avec les droits d'un dossier et d'un fichier créé par un utilisateur autre que root. Pour afficher les droits et propriétaire, utilisez la commande `ls -l <nom de fichier>`.

# 4 Format des documents à remettre

Il vous suffit de remettre vos fichier `setup.sh` et `ptree.cpp` (ou `ptree.c` si vous avez seulement utilisé du C) ainsi que tout autre fichier nécessaire pour l'exécution de votre script. Ces fichiers devront se retrouver dans une archive compressée **PRENOM_NOM_TP1.zip** (ou .tar.gz) et déposée sur StudiUM. Notez que le script setup.sh doit fonctionner après la décompression.

# 5 Liens utiles

— *Linux Shell Scripting Tutorial* - `http://bash.cyberciti.biz/guide/Main_Page`
— Modification de la variable PATH - `http://www.troubleshooters.com/linux/prepostpath.htm`
— Référence C++ - `http://en.cppreference.com/w/`
— Référence C++ - `http://www.cplusplus.com/reference/`
— `dirent.h` - `http://pubs.opengroup.org/onlinepubs/7908799/xsh/dirent.h.html`