Carnet de Travaux Pratiques
Système et programmation système
Licence 2 Informatique

Julien Bernard

Introduction

Ces travaux pratiques vous permettent de vous entraîner à manipuler toutes les notions vues en cours. Si vous ne parvenez pas à finir le TP dans le temps encadré imparti, il sera obligatoire de le finir pour la semaine suivante.

Tout le travail demandé ne nécessite qu'un éditeur de texte (comme `Kate`) et un terminal pour le shell (comme `Konsole`). En particulier, il est expressément interdit d'utiliser un gestionnaire de fichier graphique (comme `Konqueror`).

La langue de la programmation est l'anglais. Vous devrez donc écrire tout votre code (commentaires inclus) en utilisant l'anglais. Cela concerne aussi bien les noms (variables, fonctions, etc) dans votre code que l'affichage.

Table des matières

Travaux Pratiques de Système n°1

Exercice 1 : Prise en main de l'environnement

Cet exercice doit vous permettre de prendre en main votre environnement de travail. Pour cela, connectez-vous d'abord sur votre compte avec votre nom d'utilisateur. Puis, ouvrez une console (`Konsole` par exemple) qui vous donne accès à un interpréteur de commande.

→ Connexion à votre compte

Avant de vous connecter sur votre compte, tapez CTRL+ALT+F1. Entrez votre nom d'utilisateur et votre mot de passe.

Question 1.1 Dans quel répertoire vous trouvez-vous ?

Question 1.2 À l'aide de la commande `ls(1)`, afficher le contenu de votre répertoire. Combien de fichiers et répertoires cachés avez-vous ?

Question 1.3 Effacer l'écran à l'aide de la commande `clear` ou avec la combinaison de touches CTRL+L.

Question 1.4 Fermer la session à l'aide de la commande `exit` ou avec la combinaison de touche CTRL+D.

Pour revenir à l'écran de login graphique, tapez CTRL+ALT+F7.

→ Commandes de base

Question 1.5 Tester les commandes vues en cours : `whoami(1)`, `uname(1)`, `uptime(1)`, `date(1)`, `cal(1)`, `echo(1)`, `man(1)`, `whatis(1)`, `apropos(1)`. En particulier, en cas d'options multiples, vous pouvez utiliser deux écritures : `-a -b -c` ou `-abc`. Le vérifier à l'aide de la commande `uname` par exemple.

Question 1.6 Comment obtenir une commande équivalente à `whoami` avec la commande `id` ?

Question 1.7 Dans quels sections pouvez-vous trouver une manpage appelée `time` ?

→ Système de fichier

Pour créer un fichier, il existe la commande `touch(1)`. En fait, cette commande met à jour le `atime` et le `mtime` d'un fichier (en le touchant), mais si le fichier n'existe pas, il est créé. Vous utiliserez cette commande pour créer des fichiers vide. Pour rappel, la commande pour créer un répertoire est `mkdir`.

En outre, pour les commandes `cp(1)`, `rm(1)`, `mv(1)`, vous testerez l'option `-i` qui permet de demander une confirmation.

Question 1.8 Créer un répertoire `SYS` dans votre répertoire utilisateur. Entrer dans ce répertoire puis créer un répertoire `exemple`. Entrer dans le répertoire `exemple`.

Question 1.9 Dans le répertoire `exemple` que vous venez de créer, créer les répertoires `skywalker/luke` en une seule commande. Puis créer un répertoire `skywalker/anakin` et un fichier `skywalker/anakin/README`.

Question 1.10 Supprimer le répertoire `skywalker/luke`. Que se passe-t-il si vous essayez de supprimer `skywalker/anakin` ?

Question 1.11 Renommer (en fait, déplacer) le répertoire `skywalker/anakin` en `darth_vader`.

Question 1.12 Créer un répertoire `yoda` et un fichier `yoda/README`. Copier le fichier `yoda/README` dans le fichier `yoda/README.old`, puis supprimer le fichier `yoda/README`.

Question 1.13 Supprimer récursivement le répertoire `yoda` (c'est-à-dire le répertoire et tout ce qu'il contient) à l'aide de l'option `-r` de `rm`.

Continuez à créer des fichiers et des répertoires et à les déplacer, copier, supprimer.

Exercice 2 : Archives

Cet exercice consiste à manipuler des archives (compressées ou non) à l'aide de la commande `tar(1)` (*Tape ARchive*). À l'origine, la commande `tar(1)` servait à créer des sauvegardes sur des bandes magnétiques (*tape*). Les bandes magnétiques avaient une plus grosse capacité de stockage que les disques durs mais avait un accès linéaire (c'est-à-dire que le temps pour accéder à une donnée était fonction de sa position sur la bande). La commande `tar(1)` a évolué pour créer des archives dans des fichiers.

La commande `tar(1)` prend en option :

- Une option de compression parmi :
  - aucune option si on ne veut pas de compression
  - `z` pour compresser avec `gzip`
  - `j` pour compresser avec `bzip2`
- Une action parmi :
  - `c` pour créer une archive avec des fichiers
  - `x` pour extraire les fichiers d'une archive
  - `t` pour lister le contenu d'une archive
  - `r` pour ajouter des fichiers dans une archive
  - `u` pour mettre à jour des fichiers dans une archive
- L'option `f` qui indique qu'on utilise un fichier dont le nom est indiqué
- L'option `v` (non-obligatoire) si vous voulez afficher le déroulement des opérations

Par exemple :

```
tar cf archive.tar repertoire
tar cf archive.tar fichier1 fichier2
```

Question 2.1 Créez une archive `exemple.tar` avec le répertoire `exemple` de l'exercice de prise en main.

Question 2.2 Allez chercher le fichier des figures sur MOODLE. Quels sont les fichiers contenus dans cette archive ?

Question 2.3 Extrayez l'archive puis créez une archive `inodes.tar.bz2` avec les deux figures concernant les inodes.

Question 2.4 Les options `z` et `j` sont en fait équivalentes à appeler directement `gzip(1)` et `bzip2(1)` (pour la compression) ou `gunzip(1)` et `bunzip2(1)` (pour la décompression). Décompressez l'archive `inodes.tar.bz2` sans en extraite les fichiers. Vous obtenez l'archive `inodes.tar`.

Question 2.5 Ajoutez les figures concernant des liens (symboliques et durs) à l'archive `inodes.tar` obtenue à la question précédente.

Travaux Pratiques de Système n°2

Exercice 3 : Arguments de la ligne de commande

Jusqu'à présent, nous n'avons pas utilisé les paramètres `argc` (*ARGument Count*) et `argv` (*ARGument Vector*) de la fonction `main`. Nous allons nous y intéresser. Pour rappel, `argv` est un tableau de `argc` chaînes de caractères dont la première indique le nom du programme.

Question 3.1 Écrire un programme `printargs` qui affiche l'ensemble des arguments passés sur la ligne de commande. On testera notamment l'utilisation des guillemets pour fournir des paramètres incluant des espaces.

Question 3.2 Écrire un programme `ints` qui prend en argument des entiers et qui en fait la somme et le produit et les affiche sur la sortie standard. On utilisera `atoi(3)` pour convertir une chaîne de caractères en entier. On utilisera le type `long` pour stocker la somme et le produit (pour éviter de boucler), et la séquence de contrôle `%ld` pour afficher un `long` (voir `printf(3)`).

Exercice 4 : Gestion avancée des arguments

Question 4.1 Écrire un programme `mycc` qui prend un ensemble de noms de fichier (au maximum 64) et trois options :

- `-h` qui affiche l'aide et arrête ;
- `-c` qui permet de dire qu'on veut uniquement compiler ;
- `-o output` qui permet de préciser le nom du fichier de sortie (`a.out` par défaut).

On affichera uniquement le résultat du traitement des options. Voici ce qu'on doit obtenir au final :

```
$ ./mycc
        compile: no
        output: a.out
        inputs:
                standard input
$ ./mycc -h
Usage: mycc [-h] [-c] [-o output] [files...]
$ ./mycc -o
Error: No output given after -o
Usage: mycc [-h] [-c] [-o output] [files...]
$ ./mycc -c toto.c
        compile: yes
        output: a.out
        inputs:
                file: toto.c
$ ./mycc -o toto.o toto.c -c titi.c
        compile: yes
        output: toto.o
        inputs:
                file: toto.c
                file: titi.c
```

Exercice 5 : Environnement

L'environnement (voir `environ(7)`) est l'ensemble des variables d'environnement. Dans un programme en C, il est possible d'accéder aux variables d'environnement via la fonction `getenv(1)`.

Question 5.1 Écrire un programme `readenv` qui lit et affiche les variables d'environnement suivantes : `HOME`, `LANG`, `PATH`, `PWD`, `SHELL`, `USER`. On pourra, par exemple, faire une fonction `void printenv(const char *name);` qui regarde si la variable d'environnement `name` existe et affiche son nom et sa valeur le cas échéant.

Certains systèmes d'exploitation (dont Linux et Windows) permettent de définir une fonction `main` avec un troisième argument contenant l'ensemble des variables d'environnement. Cette fonction `main` a pour prototype :

```
int main(int argc, char **argv, char **envp);
```

La variable `envp` est un tableau de chaînes de caractères. Le dernier pointeur du tableau est `NULL`, ce qui permet de connaître la fin du tableau.

Question 5.2 Écrire un programme `env` qui lit l'ensemble des variables d'environnement et les affiche. Comparer la sortie du programme `env` avec celle de la commande `env(1)` qui fait la même chose quand on l'appelle sans argument.

Question 5.3 Définir une variable `FOOBAR` dans le shell et lui donner la valeur `Hello World!`. Vérifier qu'elle n'apparaît pas à l'appel de `env`. Exporter la variable `FOOBAR` dans l'environnement via la command `export`. Vérifier qu'elle apparaît à l'appel de `env`.

Travaux Pratiques de Système n°3

Exercice 6 : Jeu du «Plus petit / Plus grand»

Le but est de faire un script `plus_moins.sh` pour jouer au jeu «Plus petit / Plus grand». Ce jeu consiste à deviner un nombre entre 1 et 100 choisi au hasard par l'ordinateur. Pour cela, le joueur peut faire plusieurs propositions et l'ordinateur dit si la valeur qu'il a choisie est plus petite ou plus grande que la proposition du joueur.

Voici un exemple d'utilisation du jeu :

```
L'ordinateur a choisi une valeur entre 1 et 100.
Quelle est votre proposition ? 50
La valeur est plus petite
Quelle est votre proposition ? 20
La valeur est plus grande
Quelle est votre proposition ? 35
Gagné ! Vous avez trouvé en 3 essais
```

Question 6.1 Définir une variable appelée `MAX` avec la valeur 100. Faire tirer à l'ordinateur une valeur au hasard entre 1 et `MAX` en utilisant `$RANDOM`

Question 6.2 Demander à l'utilisateur sa proposition tant qu'il n'a pas trouvé la bonne réponse et afficher le résultat par rapport à la valeur de l'ordinateur. Indice : `test(1)`.

Question 6.3 Afficher le nombre d'essais nécessaires pour trouver la réponse

On veut limiter le nombre d'essais possibles pour le joueur à 6. Si le joueur échoue 6 fois de suite, l'ordinateur affiche un message au joueur. Par exemple :

```
Perdu ! La bonne réponse était : 32
```

Question 6.4 Modifier le programme pour s'arrêter si le joueur fait 6 propositions perdantes et afficher un message adéquat

Exercice 7 : Lister les fichiers d'un répertoire en shell

Le but est de faire un script `ls.sh` qui est une version très simple de `ls(1)` en shell. Le script prendra éventuellement un paramètre, le nom du répertoire à lister (le répertoire courant par défaut).

Question 7.1 Déterminer le répertoire à lister.

Question 7.2 Parcourir tous les fichiers du répertoire courant et afficher leur nom. Indice : `for`

Question 7.3 Déterminer le type de chaque fichier et afficher cette information dans le cas où c'est un fichier régulier ou un répertoire. Indice : `test(1)`

Exercice 8 : Destruction récursive des répertoires vides

Le but est de faire un script `rec_rmdir.sh` qui efface récursivement les répertoires vides. Pour cela, on peut définir une fonction nommée `rec_rmdir` qui fera tout le travail et uniquement appeler cette fonction avec le répertoire courant :

```
rec_rmdir .
```

Question 8.1 Dans la fonction `rec_rmdir`, vérifier qu'il y a un seul paramètre et l'afficher, sinon sortir de la fonction (avec la commande `return`). Indice : `$#`

Question 8.2 Vérifier que le paramètre est bien un répertoire, puis, si c'est le cas, entrer dans ce répertoire. Indice : `test(1)`.

Question 8.3 Parcourir tous les fichiers de ce répertoire et pour chaque fichier qui est lui-même un répertoire, afficher son nom. Indice : `for`

Arrivé à ce stade, il ne reste plus qu'à appeler récursivement la fonction `rec_rmdir` sur le répertoire trouvé puis de le supprimer(avec `rmdir(1)`). Seulement, si la fonction s'appelle récursivement sans précaution, la variable utilisée pour itérer sur les fichiers sera écrasée : en effet, toutes les variables définies dans une fonction sont visible globalement (pour le processus en cours) et donc ne sont pas uniquement locale à la fonction. Pour résoudre ce problème, il est nécessaire de créer un nouveau processus puisque les variables qui n'appartiennent pas à l'environnement ne sont pas visibles pour les processus créés. Pour créer un nouveau processus, on peut entourer le groupe de commandes concerné par des parenthèses (semblables aux accolades). Deux manières de faire sont possibles :

- soit placer les parenthèses autour de l'appel récursif uniquement
- soit placer les parenthèses à la place des accolades qui entoure la fonction de manière à ce que toute la fonction s'éxécute dans un nouveau processus

Question 8.4 Terminer le script avec les indications données. On veillera notamment à supprimer le message d'erreur de `rmdir(1)` quand le répertoire n'est pas vide.

Question 8.5 Bonus : supprimer les fichiers vides

Travaux Pratiques de Système n°4

Exercice 9 : La commande `cat(1)`

La commande `cat(1)` permet de concaténer des fichiers textes et d'envoyer le résultat sur la sortie standard. On ne va pas stocker le résultat, on va directement l'envoyer sur la sortie standard.

Question 9.1 Implémenter la commande `cat(1)`. Pour chaque fichier passé en ligne de commande :

1. ouvrir le fichier (en lecture seule) ;
2. lire les données dans un buffer statique (d'une taille de 1024 octets) ;
3. écrire les données du buffer sur la sortie standard ;
4. tant qu'il y a des données dans le fichier, aller à 2. ;
5. fermer le fichier.

Exercice 10 : La commande `tee(1)`

La commande `tee(1)` lit son entrée standard et écrit sur la sortie standard et sur un fichier.

Question 10.1 Implémenter la commande `tee(1)`.

Question 10.2 Améliorer votre code en considérant qu'il peut y avoir plusieurs fichiers à écrire sur la ligne de commande.

Exercice 11 : La commande `wc(1)`

Il s'agit implémenter la commande `wc(1)` avec deux options : `-c` (pour compter les octets) et `-l` (pour compter les lignes). Si aucun fichier n'est précisé, on utilisera l'entrée standard.

Question 11.1 Analyser la ligne de commande pour vérifier la présence des différentes options et du nom du fichier. S'il n'y a aucune option, on affichera les deux nombres (d'octets et de lignes) par défaut. Indice : `strcmp(3)`.

Question 11.2 Implémenter le comptage des octets.

Question 11.3 Implémenter le comptage des lignes. On supposera que les lignes sont terminées par `"\n"`.

Question 11.4 Comparer le résultat de votre commande avec le résultat de `wc(1)`.

Travaux Pratiques de Système n°5

Exercice 12 : La commande `head(1)`

Le but de cet exercice est de réimplémenter la commande `head(1)` :

```
head [-n N] [file]
```

La commande peut être appelée avec l'option `-n` suivi du nombre de ligne à afficher, 10 par défaut. Si le nom du fichier n'est pas précisé, on traitera l'entrée standard.

Question 12.1 Déterminer le nombre de lignes à afficher et le fichier à lire (qu'on ouvrira si besoin). Indice : `atoi(3)`.

Question 12.2 Lire le fichier jusqu'à la ligne indiquée. On supposera que les lignes sont terminées par `\n`.

Question 12.3 Écrire le résultat sur la sortie standard.

Question 12.4 Fermer le fichier

Exercice 13 : La commande `cut(1)`

Le but de cet exercice est de réimplémenter une version simplifiée de la commande `cut(1)` en langage C.

La commande à implémenter prendra 2 arguments obligatoires et un argument optionnel. Les deux arguments obligatoires sont, dans cet ordre, le séparateur et le numéro du champs à sélectionner (le premier champs étant numéroté 1). L'argument optionnel sera le nom du fichier à lire. S'il n'y a pas de troisième argument, on lira l'entrée standard. Voici un exemple d'utilisation :

```
cut ':' 1 /etc/passwd
```

Cette commande sélectionne le premier champ du fichier `/etc/passwd` avec `:` comme séparateur. Vous noterez que, contrairement à l'outil que vous avez utilisé, le nom des options n'apparaît pas.

Question 13.1 Déterminer le séparateur et le numéro du champs. On affichera un message d'erreur si le séparateur indiqué contient plus d'un caractère, ou si le numéro du champ est strictement inférieur à 1.

Question 13.2 Déterminer le fichier à lire. On affichera un message d'erreur si le fichier donné en paramètre ne peut pas être ouvert.

Question 13.3 Afficher le champs sélectionné sur chaque ligne. On veillera à ne pas afficher le séparateur et à prendre en compte les passages à la ligne correctement.

Question 13.4 Fermer le fichier.

Exercice 14 : La commande `ls(1)`

Question 14.1 Écrire un programme `ls(1)` qui liste les fichiers du répertoire courant.

Question 14.2 Gérer l'option `-l`. Indice : `stat(2)`

Travaux Pratiques de Système n°6

Exercice 15 : Sauvegarde du PID

Certains programmes (les daemons en particulier) sauvegarde leur PID dans un fichier, ce qui permet à des commandes extérieures de leur envoyer un signal plus facilement. Ces fichiers sont généralement sauvegardés dans le répertoire `/var/run` ou `/run`.

Question 15.1 Écrire un programme `myself` qui écrit son pid dans un fichier `myself.pid`.

Exercice 16 : Lanceur de commande

Nous allons réaliser un lanceur de commande `launch` chargé de lancer la commande passée en paramètre et de calculer le temps réel que la commande a mis pour s'exécuter.

Question 16.1 Récupérer l'heure courante à l'aide de `gettimeofday(2)`. La fonction `gettimeofday(2)` permet de récupérer l'heure courante à l'aide d'une structure de type `timeval` :

```
struct timeval {
  time_t      tv_sec;     /* secondes */
  suseconds_t tv_usec;    /* microsecondes */
};

int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Question 16.2 Forker et exécuter la commande passée en paramètre dans le fils, le père attendant la terminaison du fils. On utilisera la variante suivante de la famille `exec(3)` :

```
int execvp(const char *file, char *const argv[]);
```

Question 16.3 Récupérer l'heure courante puis calculer la durée (très approximative) du processus fils.

Exercice 17 : La commande `kill(1)`

La commande `kill(1)` permet d'envoyer un signal à un processus, `SIGTERM` par défaut. Le signal est précisé avec l'option `-s` suivi du numéro du signal voulu. Les processus sont désignés par leur PID.

Question 17.1 Déterminer le signal à envoyer. On supposera que l'option `-s` ne peut apparaître qu'en première position.

Question 17.2 Lancer le signal à chacun des processus donné en paramètre. Indice : `kill(2)`

Exercice 18 : La fonction `system(3)`

Le but de cet exercice est d'implémenter la fonction `system(3)` dont le prototype est :

```
int system(const char *command);
```

La fonction `system()` exécute la commande indiquée dans `command` en appelant `/bin/sh -c command`, et revient après l'exécution complète de la commande. Durant cette exécution, les signaux `SIGINT` et `SIGQUIT` sont ignorés dans le père. la valeur renvoyée est le code de retour du processus.

Question 18.1 Implémenter cette commande. On justifiera précisément la version de la famille `exec` utilisée à l'aide d'un commentaire dans le code.

Travaux Pratiques de Système n°7

Exercice 19 : Signaux

Question 19.1 Écrire un programme `invincible` qui dit «boom !» quand il reçoit SIGINT et qui continue de s'exécuter. Indice : `signal(2)`, `pause(2)`, `signal(7)`.

Question 19.2 Faites en sorte qu'il abandonne au bout de n tentatives (valeur donnée en paramètre de la commande).

```
$ ./invincible 5
^Cboom!
^Cboom!
^Cboom!
^Cboom!
^CKABOOM!
$
```

Question 19.3 Modifiez-le pour qu'il compte des moutons (1 par seconde) lorsqu'il n'est pas embêté. Mais faites en sorte qu'il reste inerte pendant les cinq secondes qui suivent un «boom !», bloqué dans sa procédure de récupération. Indice : `alarm(2)`.

```
$ ./invincible 2
1
2
^Cboom!
3
4
5
^CKABOOM!
$
```

Exercice 20 : Copie de fichiers

Le but de cet exercice est de réaliser une copie de fichier à l'aide de deux processus : un qui va lire le fichier à copier et un autre qui va écrire le nouveau fichier. Les deux processus vont communiquer à l'aide d'un tube. La commande prendra donc en paramètre deux noms de fichier : la source et la destination.

Question 20.1 Vérifier que la commande a bien deux paramètres et écrire un message sur l'erreur standard sinon.

Question 20.2 Créer les deux processus et le tube et faire la copie comme indiqué. On veillera à ce que tous les descripteurs ouverts soient correctement fermés dans tous les processus.

Question 20.3 À votre avis, est-ce que cette méthode permet d'accélérer la copie par rapport à un seul processus ? Pourquoi ?

Question 20.4 On veut à présent pouvoir arrêter la copie en cours à l'aide de `CTRL+C`. Il faut alors interrompre le second processus et supprimer le fichier destination. Modifier le programme pour mettre en place ce comportement.

Exercice 21 : Calcul de π

Une méthode pour obtenir une approximation de π est de tirer aléatoirement n points (x, y) avec $x, y \in [0, 1]$. Parmi ces n points, p points appartiennent au disque unité (de centre O et de rayon 1), c'est-à-dire que $x^2 + y^2 < 1$. Or, la probabilité de tomber dans le disque unité est de $\frac{\pi}{4}$. Une approximation de π est donc $4 * \frac{p}{n}$. On utilisera le type `long` pour n et p.

Question 21.1 En utilisant `rand(3)` et `RAND_MAX`, écrire une fonction C qui renvoie un `double` compris dans l'intervalle $[0, 1]$. On prendra garde à initialiser le générateur aléatoire en utilisant `srand(3)` et `getpid(2)` (de manière à ce que chaque processus ait une graine différente).

Correction de la question 21.1

```
double get_double() {
  return (double) rand() / (double) RAND_MAX;
}
```

Question 21.2 Écrire une commande `seq_pi` qui fait le calcul de π suivant la méthode indiquée en utilisant un seul processus. On fixera n à 10^7.

Correction de la question 21.2

```
void compute_pi() {
  srand(getpid());
  long i;
  for (i = 0; i < N; ++i) {
    double x = get_double();
    double y = get_double();
    if (x*x + y*y < 1.0) {
      p++;
    }
  }
}
```

À partir de maintenant, on veut lancer plusieurs processus en même temps pour accélérer le calcul. Pour faire le calcul final de π, on va propager les résultats (n et p) avec un tube entre chacun des processus, le dernier processus réalisant le calcul final de π.

Question 21.3 Écrire une commande `biprocess_pi` qui utilise deux processus et un tube. On fixera n à 10^7.

Question 21.4 Écrire une commande `interrupted_pi` qui utilise deux processus et qui calcule tant qu'il n'est pas interrompu (par un `CTRL+C`).

Question 21.5 Écrire une commande `parallel_pi` qui utilise q processus, q étant passé en paramètre de la commande et qui calcule tant qu'il n'est pas interrompu (par un `CTRL+C`). Chaque processus envoie ses données à son fils et interromp alors son fils en envoyant le signal `SIGINT`.