Langage C et Programmation Système
TP n° 1 : Utilisation d'un système Unix

Exercice 1 : Editeur de texte

Lancez `emacs`, et créez un nouveau fichier. Rédigez un texte et sauvegardez le fichier, sous le nom `foo.txt`, dans le sous-répertoire `test` de votre répertoire personnel que vous aurez créé au préalable. Fermez le fichier. Ouvrez-le à nouveau et rajoutez-y un texte. Sauvegardez puis fermez l'éditeur de texte.

Exercice 2 : Bonjour le monde

1. Écrivez le programme suivant (`hello.c`), compilez-le avec la commande `gcc -Wall hello.c -o hello` puis exécutez-le.

```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

2. À quoi sert la fonction `printf` ? (cf. `man`)

Exercice 3 : Saisir/afficher des entiers

1. Écrivez le programme suivant (`sum.c`), compilez-le avec la commande `gcc -Wall sum.c -o sum` puis exécutez-le.

```
#include <stdio.h>

int main() {
    int x, y;
    printf("Le premier entier ?\n");
    scanf("%d", &x);
    printf("Le deuxieme entier ?\n");
    scanf("%d", &y);
    printf("La somme est %d \n",x+y);
    return 0;
}
```

2. À quoi sert la fonction `scanf` ? (cf. `man`)

Exercice 4 : Manipulation d'entiers

1. Écrivez un programme qui détermine si un entier saisi est pair ou impair.
2. Écrivez un programme qui affiche le plus grand de trois entiers saisis.

Exercice 5 : Boucles `while`

1. Écrivez le programme suivant (`while.c`), compilez-le avec la commande `gcc -Wall while.c -o while` puis exécutez-le.

```
#include <stdio.h>

int main() {
    int a = 1;
    while (a < 100) {
        printf("a is %d \n", a);
        a = a * 2;
    }
}
```

2. Qu'est-ce que représente les valeurs affichées par le programme ?

Exercice 6 : Manipulation de sequences d'entiers

1. Écrivez un programme qui affiche tous les diviseurs d'un nombre entier saisi, plus grand que 1.

2. Écrivez un programme qui affiche le plus grand et le plus petit d'une suite d'entiers saisis. Les nombres saisis ne sont pas conservés dans la mémoire. La suite se termine avec la valeur 0.

Exercice 7 : Fonctions

1. Écrivez un programme `hello2.c` en modifiant le programme `hello.c` du deuxième exercice pour que le `printf` ne soit pas directement dans le `main` mais dans une fonction `void bonjour()`.

2. Écrivez un programme `hello3.c` en modifiant le programme précédent pour qu'il demande, dans le `main`, le numéro de l'utilisateur, puis qu'il appelle une fonction `void bonjour2(int i)` qui affiche la phrase "Bonjour utilisateur numéro i !".

3. Écrivez un programme `hello4.c` en modifiant le programme précédent en rajoutant une fonction `int identification()`. Dans le `main` il y aura donc l'appel à la fonction `identification`, qui demande le numéro de l'utilisateur, puis à la fonction `bonjour`, qui affiche la phrase "Bonjour utilisateur numéro i !".

Exercice 8 : Makefile

Un exemple simple de Makefile

```
all : executable
executable : file1.o file2.o
	gcc -o executable file1.o file2.o
file1.o : file1.c file1.h
	gcc -c file1.c
file2.o : file2.c file1.h file2.h
```

```
	gcc -c file2.c
clean :
	rm file1.o file2.o executable core
```

On lance l'interpréteur de `Makefile` par la commande `make`. Par défaut, la première cible (*target*) rencontrée, `all`, sera traitée. Il s'agit ici d'une simple règle de dépendance, qui requiert de remettre à jour toutes les targets situées à droite du symbole ':', en l'occurence la target (`executable`). L'interpréteur `make` passe donc à la target suivante, `executable`. On voit que `executable` dépend des deux fichiers objets `file1.o` et `file2.o`. Récursivement, les fichiers dont dépendent `file1.o` et `file2.o` sont recherchés. Les fichiers `file1.h` et `file2.h` contiennent les entêtes des fonctions contenues dans `file1.o` et `file2.o`. Enfin chaque fois que les dépendances éventuelles d'une target ont été traitées récursivement, si l'une de ces dépendances a une date de dernière modification postérieure à la target en cours, la directive de compilation est appliquée. Par exemple lorsqu'on traite la target `file1.o`, si `file1.c` ou `file1.h` ont été modifiés plus récemment on exécute la commande `gcc -c file1.c`, ce qui modifie le fichier `file1.o` et provoque des compilations en chaîne dans les autres targets.

1. En vous inspirant du `Makefile` précédent, écrivez un `Makefile` pour le programme `hello3.c` dont la première ligne contient l'instruction suivante `all:hello3` et la deuxième ligne contient `hello3: bonjour.o main.o`

2. Écrivez un `Makefile` pour le programme `hello4.c`.

Exercice 9 : Compilation sous emacs

1. Dans l'éditeur de texte `emacs`, ouvrez le fichier `hello4.c`, puis dans le menu `tools` choisissez ensuite `compile` pour compiler. Testez.

2. Observez que vous pouvez faire la même chose en appuyant sur `alt x`, puis en écrivant `compile` dans la ligne de commandes de `emacs`. Testez.

3. On peut associer une touche à la commande de compilation sous `emacs`. Dans votre répertoire principal ouvrez le fichier `.emacs` s'il existe déjà ou créez le s'il n'existe pas. Rajoutez y la ligne de commande `(global-set-key [f11] 'compile)`. Relancez `emacs` et appuyez sur la touche `[f11]` du clavier. Regardez en bas dans la ligne de commandes de `emacs`. Que se passe-t-il ? Appuyez sur la touche `return`.

4. Dans le fichier `.emacs` rajoutez la commande `(global-set-key [f12] 'recompile)` où la commande `recompile` réexécute le dernier makefile exécuté même si on n'est plus dans le bon répertoire. Testez la touche `[f12]`.

Exercice 10 : Tableaux

Le but de cet exercise est de lire en entrée un tableau `tab` de `NB_ELEM`, où `NB_ELEM` est une constante de préprocesseur (définie par une directive `#define`), puis de lire une permutation `per` de 1 à `NB_ELEM`, et enfin d'afficher les éléments du tableau dans l'ordre donné par la permutation. Par exemple, avec `#define NB_ELEM 5`, la permutation 3 1 5 2 4 appliquée au tableau `20,40,-5,10,2` affichera le tableau `-5, 20, 2, 40, 10`.

1. Écrivez le programme.

2. Améliorez si nécessaire la lisibilité de votre programme en le découpant en fonctions, une fonction qui lit le tableau, une qui lit la permutation et qui vérifie que c'est bien une permutation, une qui affiche le résultat.
3. Séparez votre programme en plusieurs fichiers et utilisez un `makefile` pour compiler.

Exercice 11 : Fichiers/Répertoires

1. Quelle est la référence absolue de votre répertoire privé ?
2. Créez une copie de `foo.txt` appelée `.foo` (le point est voulu) dans votre répertoire personnel. Supprimez ensuite le fichier `foo.txt`. Affichez le contenu de votre répertoire personnel avec `ls`. Que constatez-vous ? Réessayez en ajoutant l'option `-a`. Concluez.
3. Quels sont les droits d'accès associés à votre répertoire privé ?
4. Qui peut consulter son contenu ?
5. Qui peut y déposer/créer un fichier ?
6. Sauriez-vous créer un répertoire dans lequel n'importe qui peut déposer un fichier mais personne ne peut consulter le contenu du répertoire ? Tester avec votre voisin.

Exercice 12 : Fichiers / regexp

1. Affichez la liste de tous les fichiers dans le répertoire `/usr/bin` dont le nom commence par `k` et contient exactement 6 caractères.
2. Affichez la liste de tous les fichiers dont l'extension est `.so` dans le répertoire `/usr/lib` (*note :* ces fichiers sont des bibliothèques).
3. Que fait la commande `find` ?
4. Utilisez-la pour retrouver les fichiers dont les noms contiennent la chaîne "conf" dans tout le système (on peut arrêter avec `^C` au bout d'un moment)
5. Recommencez de sorte que les messages d'erreurs soient "perdus" (`2>/dev/null`)
6. Créez un certain nombre de répertoires et des fichiers dont quelques-uns auront la chaîne "abc" dans leur nom. Pour cela, vous pouvez vous aider d'un script shell. Utilisez find pour les retrouver. Utilisez find pour afficher les informations les concernant (taille, dates, etc). Utilisez find pour les supprimer.

Exercice 13 : Redirection

1. Sauvegardez le résultat de la commande `ls -l /usr/lib` dans un fichier `liste`, puis affichez le contenu de ce fichier avec la commande `less` par exemple.
2. Redirigez l'entrée standard de la commande `cat` vers le fichier `liste` et observez ce qui se produit. Comme pour un certain nombre de commandes, le même résultat peut être obtenu en passant le fichier `liste` directement en argument de la commande `cat` (sans symbole de redirection). Essayez.
3. Ajoutez au fichier `liste` une ligne de texte.
4. Copiez `liste` dans `liste-bis` sans utiliser la commande `cp`.

Exercice 14 : Pipe

1. Si la commande `ls` fournit un résultat trop long, comment le consulter intégralement sans utiliser la souris ?
2. Ecrire dans un fichier le contenu de votre répertoire personnel trié par ordre alphabétique *inverse*.
3. En une seule commande composée, cherchez dans `/bin` tous les noms de fichier contenant la lettre `a` et triez-les par ordre alphabétique inverse.

Exercice 15 : Expansion

1. Que font les commandes `wc`, `wc -w`, `echo a | wc -w` ?
2. `echo * | wc -w` devrait vous fournir un résultat différent de 1 pourquoi ?