Séance de TP N°1

Entrées-sorties – Traitement des erreurs et entrées-sorties

Une commande UNIX est une "boîte noire" à laquelle on fournit des arguments sur un fichier logique nommé entrée standard et qui renvoie deux types de réponses éventuelles sur des fichiers logiques appelés respectivement sortie standard et sortie erreur standard, comme illustré à la figure 1.

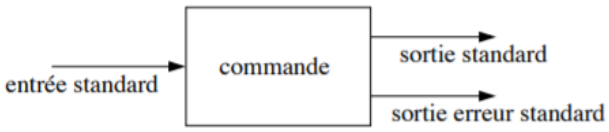

FIG. 1 – commande UNIX

Quelques exemples :

- La commande **cat** écrit sur la sortie standard ce qu'elle reçoit sur l'entrée standard.
- La commande ***date*** ne prend rien sur le flot d'entrée et renvoie sur le flot de sortie.
- La commande ***cd*** ne prend rien en entrée et ne renvoie rien en sortie. Toutes ces commandes peuvent évidemment renvoyer quelques choses sur la sortie erreur standard pour signaler un problème quelconque (mauvaise utilisation de la commande, problèmes de droits, etc.).

Exercice I - Premier pas avec *ls* et *grep*

I. L'objectif de cette exercice est de faire une recherche avec ls et grep

A. Dans votre répertoire courant:

1. *créez en une commande **avec touch** les fichiers suivants:*
   test1 test2.c testt44.pdf test455 testt41.c test510.c test_all.pdf testmaker
2. *Créer le répertoire **Exercice1** dans votre répertoire courant, en une commande. Déplacez les fichiers précédemment créés dans le répertoire **Exercice1***
3. *Listez tous les fichiers*
   a) *se terminant par '5'*
   b) *commençant par 'testt'*
   c) *du format pdf*
   d) *ayant comme 1ère et 5ème lettres la lettre 't'*

*B. Dans le répertoire **Exercice1** que vous avez créé:*

1. *Affichez avec grep:*
   a) *tous les fichiers sauf le fichier "testmaker"*
   b) *le numéro de la ligne des fichiers contenant "455"*
   c) *les programmes en C*

Exercice II - Redirection de la sortie : les symboles ‘>’ et ‘>>’

II. L’objectif de cette exercice est de faire une redirection de la sortie avec **‘>’ et ‘>>’**

A. Tapez **who** dans une fenêtre de terminal.

B. Tapez ensuite **who > fich**. Un nouveau fichier nommé fich a été créé.

C. Regardez son contenu. – Maintenant faites **date > fich**. Regardez à nouveau le contenu du fichier fich.

D. Concluez sur le rôle de **‘ >’.**

E. Faites de nouveau **who>>fich** dans le terminal. Regarder à nouveau le contenu de fich,

F. Concluez sur le rôle de **‘ >>’.**

Exercice III - Redirection de l'entrée : les symboles ‘<’

III. L’objectif de cette exercice est de faire une redirection de l'entrée avec **‘<’.**

A. La commande ***cat*** prend ce qui lui arrive dans le flux d’entrée et le réécrit en sortie.

1. Lancez cat sans argument. L’invite change de forme pour vous permettre de fournir des données.
2. Tapez quelques caractères puis frappez la touche entrée. Observez le résultat.
3. Tapez encore quelques lignes, puis pour indiquer au processus la fin des données à traiter, pressez la combinaison de touches Ctrl-D (aussi appelé EOF, ou end of file).
4. Que se passe-t-il si à la fin d’une ligne contenant des caractères, vous pressez Ctrl-C au lieu de Ctrl-D ? Expliquez.
5. Lancez maintenant la commande **cat < fich**. Comment expliquez-vous ce qui se passe ?

B. La commande ***wc*** sert à compter un nombre de caractères, de mots et/ou de lignes. On l’utilise soit en lui fournissant comme paramètre le nom d’un fichier, soit avec le flux de l’entrée standard.

1. Appliquez-la au fichier fic précédemment créé en utilisant ces 2 méthodes.

Exercice IV - Redirection de la sortie erreur : les symboles ‘2>’ et ‘2>>’

IV. L’objectif de cette exercice est de faire une redirection de la sortie erreur avec **‘2>’ et ‘2>>’**

A. Tapez dans votre terminal la commande **dater** (qui n’existe pas) : vous obtenez un message d’erreur:

1. Rediriger la sortie standard de cette commande vers un fichier. Regardez le contenu de ce fichier. Que constatez-vous ?

2. En fait, les messages d'erreur ne sortent pas sur le même canal que le résultat des commandes. Recommencez l'opération en redirigeant avec le symbole **2>.**
3. Faites des tests pour comprendre la signification du symbole **2>>**

B. Le fichier **/dev/null** est un "puits sans fond" : on peut écrire dedans tant qu'on le veut et les données sont alors perdues. Il peut servir à jeter par exemple la sortie erreur quand on en n'a pas besoin.

1. Lancez la commande **ls -R /home**. Vous pouvez constater que vous n'avez pas accès à un certain nombre de répertoires et fichiers. Relancez cette commande en dirigeant la sortie erreur sur /dev/null.

Exercice V - Ouverture et fermeture d'un fichier

V. Pour toutes les programmes, il faut inclure le header (**#include<stdio.h>** au début du programme). Pour ouvrir un fichier et obtenir un descripteur de fichier pouvant y accéder, utilisez l'appel **open**. Il prend le chemin du fichier à ouvrir sous forme d'une chaîne de caractères et des indicateurs spécifiant comment il doit l'être. Vous pouvez utiliser open pour créer un nouveau fichier ; pour cela, passez un troisième argument décrivant les droits d'accès à appliquer au nouveau fichier. Si le second argument est **O_RDONLY**, le fichier est ouvert en lecture seul ; un erreur sera signalée si vous essayez d'y écrire. De même, **O_WRONLY** ouvre le descripteur de fichier en écriture seule. Passer **O_RDWR** crée un descripteur de fichier pouvant être utilisé à la fois en lecture et en écriture. Notez que tous les fichiers ne peuvent pas être ouverts dans les trois modes. Vous pouvez passer des options supplémentaires en utilisant un OU binaire de ces valeurs avec d'autres indicateurs. Voici les valeurs les plus courantes :

- Passez **O_TRUNC** pour tronquer le fichier ouvert, s'il existait auparavant. Les données écrites remplaceront le contenu du fichier ;
- Passez **O_APPEND** pour ajouter les données au contenu d'un fichier existant. Elles sont écrites à la fin du fichier ;
- Passez **O_CREAT** pour créer un nouveau fichier. Si le nom de fichier que vous passez à open correspond à un fichier inexistant, un nouveau fichier sera créé, si tant est que le répertoire le contenant existe et que le processus a les permissions nécessaires pour créer des fichiers dans ce répertoire. Si le fichier existe déjà, il est ouvert ;
- Passez **O_EXCL** et **O_CREATE** pour forcer la création d'un nouveau fichier. Si le fichier existe déjà, l'appel à open échouera.

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
```

```
int main(int argc, char *argv[])
{
    /* Chemin vers le nouveau fichier */
    char* path = argv[1];
    /* Permissions du nouveau fichier */
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRO | S_ IWGRP | S_UROTH;
    /* Crée le fichier */
    int fd = open (path, …|……|……|…..., mode);

    if (fd == -1){
     /* Une erreur est survenue, affiche un message et quitte. */
    perror(“open”);
    return 1;
    }
  return 0;
}
```

A. *Rédigez le programme ci-dessus **creation_fichier.c** qui va créer un fichier vide inexistant en remplissant les options nécessaires dans la commande **open**. Justifier votre choix.*
B. *Apres la creation du fichier, lancez le même programme une autre fois. Que constatez vous?*
C. *Fermez le fichier avec la commande **close** et prenez une capture d'écran du résultat.*

Exercice VI - Lecture d’un fichier

VI. Le fichier source C **read.c** fournit un exemple de manipulation de fichier en lecture :

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

const int N=20; //taille du buffer

int main() {
  char afilename[12]="test1.txt";
  char buff[N];

 //tentative d’ouverture du fichier.
 //ouverture en lecture seulement
 int f2=open(afilename, O_RDONLY);

 if (f2 == -1) { printf("open for write: failed\n"); exit(1) ; }
 else {
 int nbreal=read(f2,buff,N); // lecture de 100 chars au max
```

```
 printf("j'ai lu %d chars\n",nbreal);
 printf("Voici les chars lus : %s\n", buff);
 close(f2);// bien fermer proprement.
}
 return 0;
}
```

A. *Compiler, exécuter.*

B. *Expliquez ce que fait ce programme.*

C. *Que se passe-t-il si le fichier test1.txt n'existe pas ?*

D. *Que se passe-t-il si le fichier contient plus de N caractères ?*

Exercice VII - Ecriture d'un fichier

VII. Le fichier source C **write.c** fournit un exemple de manipulation de fichier en écriture:

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

const int N=20; //taille du buffer

int main() {

 //tentative d'ouverture du fichier.
 //ouverture en écriture seulement
 // en supposant que le fichier existe.
 int f2=open("test1.txt", O_WRONLY);

 if (f2 == -1) { printf("open for write: failed\n"); exit(1) ; }
 else {
   char* mon_msg="oh le joli fichier";
   write(f2,mon_msg,strlen(mon_msg));
}
 return 0;
}
```

E. *Exécuter dans un terminal la commande **file test1.txt**.*

F. *Compiler le fichier, exécuter le binaire.*

G. *Regarder le contenu du fichier **test1.txt** après l'exécution du programme C dans lequel on a modifié le nombre d'octets écrits (dernier paramètre de la fonction write) et ce que retourne alors la commande **file**.*

H. *Ecrire un nouveau message **"j'aime linux"** sans écraser le contenu de l'ancien fichier. Prenez les captures d'écran nécessaires pour valider le résultat.*

Exercice VIII - Lecture et écriture d'un fichier

VIII. Le fichier source C **read_write.c** fournit un exemple de manipulation de 2 fichiers :

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

int main(int argc, char *argv[]) {
const int MAX=1000;
int f1, f2, n;
char buf [MAX];

//vérification du nombre
 if (argc != 3){
     fprintf(stderr, "erreur parametres\n"); exit(1);
}

//ouverture readOnly du premier argument
f1=open(argv[1], O_RDONLY);

if( f1 == -1){ /* mode lecture */
 perror("ouverture fichier source impossible"); exit(2);
}

 //ouverture du deuxième argument en écriture et ?
 f2=open(argv[2], O_WRONLY|O_CREAT|O_APPEND, S_IRWXU);

 if( f2 ==-1 ){//700
 perror("creation fichier destinataire impossible"); exit(3);
}

//lecture, ecriture.
while ((n == read(f1, buf, MAX)) > 0){
 write(f2, buf, n);
}
 return 0;
}
```

A. *Compiler, et tester comme ceci : **./nomdubinaire test1.txt nomquinexistepas.txt***

B. *Vérifier qu'une copie du fichier **nomquinexistepas.txt** est réalisée, et expliquer pourquoi.*

C. *Que signifient les options passées à **open** pour l'ouverture du deuxième argument de la ligne de commande (argv[2]). On pourra remarquer l'utilisation du "ou" bit à bit pour "ajouter" les flags entre eux. Tester (avec un fichier existant comme deuxième argument, par exemple).*

Exercice IX

IX. Rédiger un programme C avec read and write qui :

A. *Créer et ouvrir un nouveau fichier **"name.txt"** et écrire dedans votre nom et prénom*

B. *Lire le fichier et compter le nombre de caractères dans ce fichier (avec **read**).*

C. *Écrire le résultat dans un autre fichier "result.txt" (avec **write**)*

D. *Vérifier le résultat et prenez les captures d'écran nécessaires*