

Laboratoire
E.D.

jcj mba mwa nvs

septembre 2021

Table des matières

| | |
|-----------------------------------------------------------------|-----------|
| 1 Partitionnement et création de File System | 2 |
| 1.1 ED - LaboED 01-01 - Partitionnement DOS - fdisk | 2 |
| 1.2 ED - LaboED 01-02 - Partitionnement GPT - fdisk | 5 |
| 1.3 ED - LaboED 01-03 - Structure F.S. - mkfs | 8 |
| 1.4 ED - LaboED 01-04 - fdisk automatisé | 9 |
| 2 Ext - Inodes et blocs | 11 |
| 2.1 ED - LaboED 02-01 - Inodes - stat,lstat | 11 |
| 2.2 ED - LaboED 02-02 - Fichiers creux - lseek, ls, du -h | 13 |
| 2.3 ED - LaboED 02-03 - Structure F.S. - mkfs, debugfs | 15 |
| 3 Ext - Répertoires | 17 |
| 3.1 ED - LaboED 03-01 - Répertoires - opendir, readdir, lstat | 17 |
| 3.2 ED - LaboED 03-02 - Droits des fichiers - SUID | 19 |
| 4 Commandes filtre | 20 |
| 4.1 ED - LaboED 04-01 - Commande filtre, cat sans argument en c | 20 |
| 4.2 ED - LaboED 04-02 - cat avec arguments en c | 22 |
| 5 Exemples d'exercices | 24 |

Chapitre 1

Partitionnement et création de File System

1.1 ED - LaboED 01-01 - Partitionnement DOS - fdisk

| | |
|-----------|-------------------------------------------------|
| Titre : | ED - LaboED 01-01 - Partitionnement DOS - fdisk |
| Support : | OS 42.3 Leap |
| Date : | 08/2018 |

1.1.1 Énoncé

Être capable de partitionner un disque avec une table de type MBR. Être capable de décrire les partitions La démonstration de cette manipulation ne sera pas automatisée. Vous ne devrez pas écrire de script Demo.

1.1.2 Manipulation

Cette partie du laboratoire se fait en tant qu'administrateur. Soyez très prudent et réfléchissez aux commandes que vous tapez! N'utilisez le mode administrateur que quand c'est indispensable, c'est une mauvaise habitude de travailler en root à d'autres moments.

Demandez le mot de passe root au responsable du laboratoire.

1.1.2.1 Partitionnons un disque

fdisk permet de partitionner un disque avec une table de partitions MBR. Il faut lui donner comme argument le pilote associé au disque (/dev/sdb, /dev/sdc, ...) que l'on souhaite partitionner. Ici, nous utiliserons uniquement des stick usb de 32 GiB comme disque. Le partitionnement détruit toutes les données du support : il ne faut pas se tromper en donnant le nom du pilote (vous vous abstiendrez de partitionner /dev/sda donc)!!!. Pour travailler de façon sécurisée, vous procéderez comme suit :

1. N'utilisez que le stick usb que vous partitionnez, ôtez tous les autres
2. Placez le stick usb, attendez une seconde et tapez dmesg
3. Les dernières lignes affichées de cette commande vous donnent le nom du pilote associé au stick (sdb, sdc, sdd, ...), par exemple sdb.
4. Tapez fdisk /dev/sd? où ? est la lettre ci dessus
5. Affichez l'aide de la commande fdisk
6. Observez l'alignement de vos partitions
7. Quelle est la signification du champ Secteurs ?

1.1.2.2 Choix des partitions

Il suffit maintenant de dire ce que vous souhaitez à fdisk. Par exemple, après avoir créé une nouvelle table de partitions DOS, créez sur ce device 3 partitions : 2 primaires, et une logique. Vous choisirez des tailles de partition $\leq 2\text{GiB}$. Créez une partition primaire de 1000 secteurs. Affichez cette table dans fdisk et confirmez votre souhait en l'écrivant sur le stick.

Vérifiez l'état de votre table par la commande `fdisk -l /dev/? ?`

1.1.3 Lire la table des partitions via un programme c

1.1.3.1 Énoncé

Écrire un programme qui, comme fdisk, affiche la tables des partitions d'un disque. Ce programme utilisera un seul argument : le nom du pilote de ce disque. On se contentera d'afficher les partitions primaires uniquement.

1.1.3.2 Une solution

```
/*
NOM      : LectPart.c
CLASSE   : ED - LaboED 01-01
#OBJET   : réservé au makefile
AUTEUR   : J.C. Jaumain, 12/2012
BUGS    :

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

typedef struct ETP { // Entrée Table des Partitions
    unsigned char fill[8];
    int first; // LBA
    int sectors; // LBA
} __attribute__((packed)) ETP; //éviter l'alignement des données.

typedef struct MBR {
    unsigned char code[442];
    unsigned char signature[4];
    ETP TablePart[4];
    unsigned char bootable[2];
} __attribute__((packed)) MBR;

int main ( int argc, char * argv[] )
{ int fd,first, sectors;
    MBR struMBR;
    if ( argc != 2 )
        { printf("usage : LectPart <pilote>\n");
            exit(1);
        }
    fd=open(argv[1],O_RDONLY);
    if ( fd < 0 )
    {
        perror(argv[1]);
        exit(1);
    }
    printf("Partitions primaires de %s :\n",argv[1]);
    printf("===== \n");
    read(fd,&struMBR,512); // Lecture du premier secteur du disque
    for (int primaire=1;primaire<=4;primaire++)
    {
        first=struMBR.TablePart[primaire-1].first;
        sectors=struMBR.TablePart[primaire-1].sectors;
        if (first > 0) printf ("Partition %d , début %d - fin %d (%d secteurs) \n", primaire, first, first+sectors-1, sectors);
    }
    close(fd);
    exit(0);
}
```

1.1.4 Commentaires

- La table des partitions se trouve dans le MBR du disque ou du stick usb. Il convient de ne pas l'écraser lors d'une copie d'un programme de chargement au risque de perdre tout le contenu du disque ou du stick
- Un device est rattaché au système de fichiers via le répertoire /dev, il peut être lu comme si c'était un fichier. Les premiers 512 bytes lus seront le MBR du device.
- Toutefois, comme la mise au point du programme est fastidieuse si on souhaite privilégier la sécurité : modification du programme en tant que user, test en tant que root. Il est plus simple de créer une fois un fichier représentant le MBR et de tester le programme sur base de ce fichier. Vous pouvez créer ce fichier via la commande administrateur : dd if=/dev/... of=MBR bs=512 count=1*
- Dans le programme en C il faudra inhiber l'alignement sur les champs des structures. On utilise pour cela l'attribut packed.
- Les partitions sont alignées sur une frontière de cylindre
- Vous pouvez automatiser simplement fdisk en lui précisant les réponses dans un fichier et en utilisant la redirection de l'entrée standard < ou en utilisant la double redirection « dans le script Demo.

1.1.5 En roue libre

- Adaptez le programme précédent pour afficher le type des partitions.
- Que faudrait-il faire pour afficher les données de la partition logique ?

1.2 ED - LaboED 01-02 - Partitionnement GPT - fdisk

| | |
|-----------|-------------------------------------------------|
| Titre : | ED - LaboED 01-02 - Partitionnement GPT - fdisk |
| Support : | OS 42.3 Leap -installation Classique |
| Date : | 08/2018 |

1.2.1 Énoncé

Être capable de partitionner un disque avec une table de type GPT. Être capable de décrire les partitions. La démonstration de cette manipulation ne sera pas automatisée. Vous ne devrez pas écrire de script Demo.

1.2.2 Manipulation

Cette partie du laboratoire se fait en tant que administrateur. Soyez très prudent et réfléchissez aux commandes que vous tapez ! N'utilisez le mode administrateur que quand c'est indispensable, c'est une mauvaise habitude de travailler en root à d'autres moments.

1.2.2.1 Partitionnons un disque

fdisk permet de partitionner un disque avec une table de partitions GPT. Il faut lui donner comme argument le pilote associé au disque (`/dev/sdb`, `/dev/sdc`, ...) que l'on souhaite partitionner. Ici, nous utiliserons uniquement des stick usb de 32 Gib comme disque. Le partitionnement détruit toutes les données du support : il ne faut pas se tromper en donnant le nom du pilote (vous vous abstiendrez de partitionner `/dev/sda` donc)!!!. Pour travailler de façon sécurisée, vous procéderez comme suit :

1. N'utilisez que le stick usb que vous partitionnez, ôtez tous les autres
2. Placez le stick usb, attendez une seconde et tapez `dmesg` suivi de `lsblk`.
3. Les dernières lignes affichées par la commande `dmesg` vous donnent le nom du pilote associé au stick (`sdb`, `sdc`, `sdd`, ...), par exemple `sdb`.
4. Tapez `fdisk /dev/sd?` où ? est la lettre ci dessus
5. Affichez l'aide de la commande `fdisk`

1.2.2.2 Choix des partitions

Il suffit maintenant de dire ce que vous souhaitez à `fdisk`. Par exemple, après avoir créé une nouvelle table de partitions GPT, obtenez le partitionnement suivant :

- partition 1 de 1Gib
- partition 2 de 2Gib

Affichez cette table avec `fdisk` et confirmez votre souhait en l'écrivant sur le stick.

1.2.3 Lire la table des partitions via un programme c

1.2.3.1 Énoncé

Écrire un programme qui, comme `fdisk`, affiche la tables des partitions GPT. Ce programme lira également l'entête secondaire et le descripteur secondaire de la partition1. Ce programme utilisera un seul argument : le nom du pilote de ce disque.

1.2.3.2 Une solution

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <inttypes.h> // pout PRIu64
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

// le mbr du disque(LBA0)
typedef struct{
    char data [512];
}__attribute__((packed)) mbr_t;

// l'entete GPT du disque (LBA1 ou LBA-1)
typedef struct{
    char filler1 [32];
    uint64_t autreGPT_lba; // adresse de l'autre GPT
    char filler2 [472];
}__attribute__((packed)) headerGpt_t;

// le type d'un guid id ou du type de la partition
typedef struct{
    unsigned char guid[16];
}__attribute__((packed)) guid_t;

typedef struct{
    guid_t type; // type de la partition
    guid_t id; // son id unique dans la table GPT
    uint64_t first_lba; // adresse du 1er secteur
    uint64_t last_lba; // adresse du dernier secteur
    char revision[8]; //espace réservé
    char name[72]; // nom de la partition
}__attribute__((packed)) entryGpt_t;

void print72 (char name[72]){ // name est un tableau de 72 caractères
    printf ("name : ");
    for (int i=0; i< 72; i++)
        printf ("%c", name[i]);
    printf ("\n");
}

void printguid (guid_t guid) { // identifiant ou type
    printf ("guid : ");
    for (int car=0; car<16; car++) printf ("%1x ", guid.guid[car]);
    printf ("\n");
}

void printEntry(entryGpt_t entry, int num){ // entry est un descripteur de partition
    printf ("\nEntry %d\n", num);
    printf ("-----\n");
    // print72 (entry.name);
    // printguid (entry.id);

    // afficher des entiers non signés sur 64 bits
    // avec l'include inttypes.h
    printf ("first lba : %"PRIu64" byte(s) , last lba %"PRIu64", size=%"PRIu64" \n ",
           entry.first_lba, entry.last_lba, entry.last_lba-entry.first_lba+1);
}

void LireAfficherEntry (int fd, int num){

    entryGpt_t entry;
    read(fd, &entry,sizeof(entryGpt_t));
    if(entry.first_lba !=0) {
        // 0 signifie inutilisé
        printEntry(entry, num);
    }
}

int main(int argc, char * argv[]){

    int fd ;
    int i;
    headerGpt_t headerGPT;
    uint64_t position;
```

```

if (argc <2){
    printf ("usage %s <chemin device>", argv[0]);
    exit(2);
}

fd = open(argv[1], O_RDONLY);

if(fd <0){
    printf("device impossible à lire ou inexistant \n");
    exit (1); // pas le bon périphérique ou pas les droits
}

// lecture de LBA-1 : entête secondaire de la GPT
position=lseek(fd,-(1*512),SEEK_END); // revenir au secteur fin-1
read(fd, &headerGPT,sizeof(headerGpt_t));
printf ("Selon l'entête secondaire, l'entête primaire est au secteur %"PRIu64" \n", headerGPT.autreGPT_lba);

// lecture de LBA1 : entête primaire GPT
lseek (fd,512,SEEK_SET); // positionnement après MBR
read(fd, &headerGPT,sizeof(headerGpt_t));
printf ("Selon l'entête primaire, l'entête secondaire est au secteur %"PRIu64" \n", headerGPT.autreGPT_lba);

// afficher les entrées des partition présentes
for (i=0;i<128;i++){ // 128 entrées sur LBA3
    LireAfficherEntry(fd,i);
}

// afficher le descripteur secondaire de la partition1
position=lseek(fd,-(33*512),SEEK_END); // revenir aux descripteurs
printf ("\n\n\nLe Entry 1 secondaire se trouve au byte %"PRIu64" \n ", position);
// lire l'entrée 1 en LBA-33
LireAfficherEntry(fd, -1);

close(fd);

return 0;
}

```

1.2.4 Commentaires

- La table des partitions se trouve dans les secteurs suivants du disque ou du stick usb.
- L'attribut packed sert à forcer l'alignement des champs d'une structure.
- L'entête de partition secondaire est le dernier secteur du disque.

1.2.5 En roue libre

Dans le code ci fourni, réalisez la restauration du descripteur de la partition 2 à partir du descripteur secondaire qui se trouve en fin de disque. Soyez très prudent dans la spécification du nom de device pour cette opération d'écriture, une erreur peut endommager votre installation. Affichez également en hexadécimal (chaîne de format %x de printf), le champ contenant le type de la partition primaire décrite dans le MBR. Vérifiez le résultat obtenu (commande l de fdisk).

1.3 ED - LaboED 01-03 - Structure F.S. - mkfs

| | |
|-----------|-------------------------------------------|
| Titre : | ED - LaboED 01-03 - Structure F.S. - mkfs |
| Support : | OS 43.2 Leap - Installation Classique |
| Date : | 08/2018 |

1.3.1 Énoncé

Être capable de créer un F.S. et de l'utiliser.

Cet exercice de laboratoire ne sera pas automatisé. Vous ne devez pas créer de script Demo.

1.3.2 Manipulation

Cette partie du laboratoire se fait en tant que administrateur. Soyez très prudent et réfléchissez aux commandes que vous tapez ! N'utilisez le mode administrateur que quand c'est indispensable.

Demandez le mot de passe root au responsable du laboratoire.

1.3.2.1 Partitionner un disque et formater les partitions

Vous savez partitionner un disque. Pour que ces partitions soient utilisables, il faut y inscrire la structure d'un F.S. de votre choix. (=formater les partitions)

Adaptez sdb, sdc, sdd, ... suivant votre situation.

A l'aide de fdisk, obtenez deux partitions primaires et une logique, toutes de taille <= 2Gib
vérifiez le partitionnement et formatez quelques partitions en adaptant la lettre ? :

```
fdisk -l
mkfs.vfat /dev/sd?1 # fat 16
mkfs.ext2 /dev/sd?2
mkfs.ext2 /dev/sd?5
fdisk -l
```

La table des partitions tient-elle compte de vos changements de type ?

1.3.3 Commentaires

- fdisk permet de partitionner, mkfs permet de formater.
- ces commandes nécessitent les droits administrateur

1.4 ED - LaboED 01-04 - fdisk automatisé

| | |
|-----------|---------------------------------------|
| Titre : | ED - LaboED 01-04 - fdisk automatisé |
| Support : | OS 43.2 Leap - Installation Classique |
| Date : | 08/2018 |

1.4.1 Énoncé

Automatiser le partitionnement d'un disque.

N'utilisez que le stick usb que vous partitionnez, ôtez tous les autres

1.4.2 Commentaires

- Vous pouvez automatiser simplement fdisk dans un script en lui précisant les réponses avec la double direction d'entrée «<». En ligne de commande cela peut également être fait par la redirection de l'entrée standard <. En effet c'est sur stdin que fdisk lit vos commandes.
- Soyez toujours très prudents quand vous effectuez des manipulations en administrateur. Le script Demo devra être exécuté en administrateur. Il faudra vérifier cela, de plus le nom du pilote/device n'est pas identique sur chacune des machines. Une erreur de nom peut compromettre votre système d'exploitation !
- Demandez le mot de passe root au responsable du laboratoire.

1.4.3 Script Demo

```
#!/bin/bash
#NOM      : Demo
#CLASSE   : ED - LaboED 01-04
#OBJET    : réservé au makefile
#AUTEUR   : J.C. Jaumain, 07/2011 M. Bastreghi 09/2020

clear
C='\033[44m'
E='\033[32m\033[1m'
W='\033[31m\033[1m'
N='\033[0m'
clear
#Il faudra être root (0) pour exécuter ces commandes
if [ $UID -ne 0 ] ; then
    echo
    echo "Pour ce script vous avez besoin des droits d'administration"
    echo "veuillez vous identifier en root et reessayer"
    echo
    echo -e "${C}          --> Enter pour continuer${N}"
    read
    exit 1
fi
echo "Automatiser le partitionnement une opération dangereuse !!"
echo -----
echo
# partitions selon fdisk
fdisk -l
echo
echo "Si vous le souhaitez, insérez le device dont vous voulez afficher la table des partitions"
echo -e "${C}          --> Enter pour continuer${N}"
read
#les derniers avertissements du noyau
dmesg | tail
echo
#choix du device
while [ "$device" == "" ]
do
    echo -n "quel est le chemin correspondant au device (/dev/sdc, ...) ? "
    #lecture du clavier
    read device
done
echo
```

```

while [ "$rep" != "y" -a "$rep" != "n" ]
do
    echo -n "Nous allons afficher les partitions de $device [y-n] ?"
    read rep
done

#fdisk : une partie de ce script fait office de stdin (remplace le clavier)
if [ $rep == "y" ]
then
    #NB Les commandes de fdisk suivent l'exécution de fdisk jusque au mot choisi (FIN pour notre exemple).
    #en spécifiant <<FIN après une commande on remplace le clavier par le texte fourni dans le script.
    #FIN est un mot au choix de maximum 8 caractères

    fdisk "$device" << FIN
p
q
FIN
#Cette séquence isimple ne fait qu'une chose : elle affiche la table des partitions avant de quitter fdisk
##le deuxième mot FIN marque la fin de stdin pour la commande fdisk
##pour réaliser des actions plus complexes il faut reproduire dans le fichier ce qui est tapé au clavier y compris les lignes blanches
la ligne (cas d'une valeur par défaut)

else
    echo vous avez abandonné ...
fi
echo
echo -e "${C}"           --> Enter pour continuer${N}"
read

```

1.4.4 En roue libre

Adaptez le script Demo pour automatiser le partitionnement suivant

- création d'un nouvelle table de partitions DOS contenant :
- une partition primaire 1 < 1Gib
- une partition primaire 2 étendue de tout le reste du stick usb
- une partition logique 5 < 1 Gib

Les valeurs par défaut obtenues par la touche <CR> correspondent à une ligne blanche dans le fichier de commandes

Chapitre 2

Ext - Inodes et blocs

2.1 ED - LaboED 02-01 - Inodes - stat,lstat

| | |
|-----------|-----------------------------------------|
| Titre : | ED - LaboED 02-01 - Inodes - stat,lstat |
| Support : | OS 42.3 Leap - Installation Classique |
| Date : | 08/2018 |

2.1.1 Énoncé

Écrivez un programme qui affiche le contenu de l'inode d'un fichier passé en argument.

2.1.2 Une solution

```
/*
NOM      : ContenuInode.c
CLASSE   : FS - LaboFS 02-01
#OBJET   : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011 revu mba 1/2015
*/
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
int main ( int argc, char * argv[] )
{ int r;
  struct stat inode;
  if (argc!=2)
  { printf("Vous devez donner un nom de fichier \n");
    exit(1);
  }
  r=stat(argv[1],&inode);
  if (r<0)
  { perror(argv[1]);
    exit(1);
  }
  if (S_ISREG(inode.st_mode))
  { printf("Le fichier %s est un fichier régulier\n",argv[1]);
    printf("Numéro d'inode : %d\n",(int)inode.st_ino);
    printf("Nombre de liens hardware : %d\n",(int)inode.st_nlink);
    printf("Propriétaire du fichier : %d\n",(int)inode.st_uid);
    printf("Groupe du propriétaire du fichier : %d\n",(int)inode.st_gid);
    printf("Taille (en bytes) : %d\n",(int)inode.st_size);
    printf("Nombre de blocs (en secteurs) : %d\n",(int)inode.st_blocks);
  }
  else
  { printf("Le fichier %s n'est pas un fichier régulier)\n",argv[1]);
    exit(1);
  }
  exit(0);
}
```

2.1.3 Commentaires

- man 2 stat donne toutes les informations nécessaires pour l'écriture de ce programme.
- Le lien software lienS est déréférencé par l'appel système fstat.
- lstat donne des informations sur le fichier lien.
- Le numéro du propriétaire est mémorisé dans l'inode du fichier. L'association numéro - nom du propriétaire se fait à l'aide du fichier /etc/passwd. Si on copie un fichier d'un PC vers un autre, il peut changer de propriétaire!!!
- Le nombre de blocs est souvent exprimé en secteurs de 512 bytes. du -h (alias pour du) donne cette information en K (Kib=1024 bytes)

2.1.4 En roue libre

Adaptez l'exercice :

- en y ajoutant un lien hardware
- en y ajoutant le détail pour le lien software.

Créez ensuite le fichier archive tar pour cet exercice uniquement en adaptant la structure et les fichiers latex fournis.

Montrez le fichier tar à votre professeur.

2.2 ED - LaboED 02-02 - Fichiers creux - lseek, ls, du -h

| | |
|-----------|-------------------------------------------------------|
| Titre : | ED - LaboED 02-02 - Fichiers creux - lseek, ls, du -h |
| Support : | OS 42.3 Leap - Installation Classique |
| Date : | 08/2018 |

2.2.1 Énoncé

Écrivez un programme qui crée un fichier contenant 'Hello world' au début du fichier et 'Bye world' en position 100000 grâce à l'appel système lseek.

2.2.2 Une solution

```
/*
NOM      : FichierCreux.c
CLASSE   : FS - LaboFS 02-02
#OBJET   : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
int main ()
{
    char bufA []="Hello world";
    char bufB []="Bye world";
    int handle;
    handle=open("FichCreux.dat",O_WRONLY|O_CREAT,0644);
    if (handle < 0){
        perror ("FichierCreux.dat");
        exit(1);
    }

    write(handle,bufA,strlen(bufA));
    lseek(handle,100000,SEEK_SET);
    write(handle,bufB,strlen(bufB));
    close(handle);
    exit(0);
}
```

2.2.3 Commentaires

- Ce fichier a une taille de 100009 bytes et occupe 3 blocs de 1K sur le disque. Le premier bloc référencé par le 1^{er} pointeur contient Hello world, le deuxième bloc, référencé par le 13^{eme} pointeur, contient 256 pointeurs. Un de ceux-ci référence le bloc qui contient 'Bye world'. Les pointeurs 2-12,14 et 15 contiennent 0 car il ne pointent vers aucun bloc.
- Sur un F.S. de type FAT, le fichier occupera réellement 100K.
- Pour connaître la taille des blocs de votre FS, il suffit de créer un fichier de un byte et de regarder son occupation sur le disque. Elle est forcément de un bloc.
- Ce programme peut être utilisé pour créer des fichiers de grande taille (jusqu'à 2 Gib sur un système 32 bits) qui occupe très peu de place sur le disque. Mais ne copiez pas un tel fichier sur un stick usb en FAT!!!
- Ce programme, exécuté sur un F.S. de type ext2, ne prendra que quelques secondes car il écrit peu sur le disque. Inversement, ce même programme, exécuté sur un F.S. de type FAT, prendra plusieurs minutes car il écrit beaucoup sur le disque.

2.2.4 En roue libre

Vérifiez et justifiez le comportement si on ajoute le flag O_APPEND dans le paramètre flags du open. L'appel système lseek retourne-t-il une erreur dans ce cas ?

2.3 ED - LaboED 02-03 - Structure F.S. - mkfs, debugfs

| | |
|-----------|----------------------------------------------------|
| Titre : | ED - LaboED 02-03 - Structure F.S. - mkfs, debugfs |
| Support : | OS 42.3 Leap - Installation Classique |
| Date : | 08/2018 |

2.3.1 Énoncé

Être capable de créer un F.S. et de l'utiliser.

Cet exercice de laboratoire ne sera pas automatisé. Vous ne devez pas créer de script Demo.

2.3.2 Manipulation

Cette partie du laboratoire se fait en tant que administrateur. Soyez très prudent et réfléchissez aux commandes que vous tapez ! N'utilisez le mode administrateur que quand c'est indispensable.
Demandez le mot de passe root au responsable du laboratoire.

2.3.2.1 Partitionner un disque et formater les partitions

Vous savez partitionner un disque. Pour que ces partitions soient utilisables, il faut y inscrire la structure d'un F.S. de votre choix. (=formater les partitions)

Adaptez sdb, sdc, sdd, ... suivant votre situation.

A l'aide de fdisk, obtenez deux partition de même taille <= 2Gib
formatez-la première en ext2.

```
fdisk -l
mkfs.ext2 /dev/sd??
```

En lisant l'output fourni avec la commande mkfs.ext2, répondez aux questions suivantes :

1. Combien de fichiers pourriez-vous avoir au maximum sur ce F.S. ?
2. Quelle est la taille par défaut d'un bloc ?
3. À combien de blocs avez-vous droit ?
4. Formatez en ext2 l'autre partition en fixant la taille de bloc à 1Kib (man mkfs.ext2) et comparez les valeurs obtenues

Copier dans ce deuxième F.S. un petit(<1K), un gros(<10K) et un très gros(>7M) fichiers. Pour ce dernier, les commandes yes (attention, ça va très vite ...) ls -lR ... redirigées vous permet de créer cela facilement.

```
mkdir disk      # créer un point de montage
mount /dev/sdb2 disk # monter la partition choisie
cp petit disk/
cp gros disk/
cp tresgros disk/
```

Utilisez debugfs pour décrire le F.S.

```
/sbin/debugfs /dev/sdb2
help # donne la liste des commandes
quit # sortir de debugfs
```

1. (debugfs)ls -l : quels sont les numéros d'inode de petit et de gros ?
2. (debugfs)stat petit : Combien de blocs petit utilise-t'il ? et gros et tresgros ?
3. Quels sont les numéros de blocs utilisés par petit et gros et tresgros ?
4. (debugfs)icheck : vérifier qu'un bloc appartient bien à l'inode supposé
5. (debugfs)ncheck : vérifier qu'un inode représente bien le fichier supposé

6. (debugfs)ffb : le premier bloc libre
7. (debugfs)ffi : le premier inode libre
8. (debugfs)undel : restaure un fichier
9. (debugfs)stats : Le contenu du superbloc, combien d'inodes y a-t-il par groupe ?
10. (debugfs)quit : pour terminer
11. Pourriez-vous dessiner ce F.S. avec les bons numéros d'inodes et de blocs ?

2.3.3 Commentaires

- debugfs permet de lire et écrire la structure d'un F.S. de type ext.
- debugfs permet de voir le chaînage des blocs dans le détail

Chapitre 3

Ext - Répertoires

3.1 ED - LaboED 03-01 - Répertoires - opendir, readdir, lstat

| | |
|-----------|-----------------------------------------------------------|
| Titre : | ED - LaboED 03-01 - Répertoires - opendir, readdir, lstat |
| Support : | OS 42.3 Leap - Installation Classique |
| Date : | 08/2018 |

3.1.1 Énoncé

Écrivez un programme qui affiche les noms et les numéros d'inode des objets de votre répertoire.

3.1.2 Une solution

```
/*
NOM      : ContenuRepertoire.c
CLASSE   : FS - LaboFS 03-01
#OBJET   : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
*/
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#include <dirent.h>
int main ()
{ struct dirent *dirp;
DIR *dp;
dp=opendir(".");
printf(" Inode - Nom\n");
printf("-----\n");
while ((dirp=readdir(dp)) != NULL) printf("%8d - %s\n", (int)dirp->d_ino, dirp->d_name);
closedir(dp);
exit(0);
}
```

3.1.3 Commentaires

- En unix, un fichier caché est un fichier dont le nom commence par un .
- Pour obtenir la liste des fichiers réguliers, il faut ouvrir chaque fichier et vérifier son mode (soit via stat, soit via open et fstat : voir man 2 stat)
- Pour obtenir une liste récursive d'un répertoire, il faut écrire un programme récursif sans oublier d'exclure les répertoires . et .. (sinon boucle infinie !)
- Il est très difficile d'estimer la place libre d'un minidisque. L'espace total - la somme des occupations des différents fichiers (du) donne un nombre. Il reste possible d'écrire un fichier qui aura une taille supérieure à ce nombre (fichier creux). df donne un résumé des minidisques montés et de la place qu'il y reste.

3.1.4 En roue libre

Dans un dossier contenant des sous-dossiers non vides, écrivez le code en c qui réalise la même chose que la commande ls *.

3.2 ED - LaboED 03-02 - Droits des fichiers - SUID

| | |
|-----------|------------------------------------------------|
| Titre : | ED - LaboED 03-02 - Droits des fichiers - SUID |
| Support : | OS 42.3 Leap - Installation Classique |
| Date : | 08/2018 |

3.2.1 Énoncé

Comme administrateur, créez un fichier appelé Confidential qui contient le texte 'LE SECRET'. Écrivez un programme Conf qui affiche "Contenu du fichier Confidential : ", suivi du contenu du fichier appelé Confidential. Modifiez les droits de telle façon que personne d'autre que l'utilisateur root ne puisse afficher le contenu du fichier appelé Confidential mais que tout le monde puisse exécuter le programme Conf et afficher ainsi le contenu du fichier Confidential.

3.2.2 Une solution

```
/*
NOM      : Conf.c
CLASSE   : FS - LaboFS 03-02
#OBJET   : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
int main ()
{ char Texte []="Contenu du fichier Confidential : ";
char c;
int handle;
printf ("UID = %d - EUID = %d\n", getuid(), geteuid());
handle=open("Confidential",O_RDONLY);
if (handle<0)
{ perror ("Confidential");
exit(1);
}

write(1,Texte,strlen(Texte));
while(read(handle,&c,1)>0) write(1,&c,1);
close(handle);
exit(0);
}
```

3.2.3 Commentaires

- La commande `chmod 4755 Conf` positionne le bit SUID du programme Conf. Le *effective uid (euid)* de l'utilisateur qui exécute Conf sera modifié en le uid du propriétaire de Conf (root). Tout utilisateur aura le droit de voir le contenu de votre fichier Confidential uniquement en exécutant le programme Conf. Ce droit lui sera refusé si il veut visionner le fichier par d'autres moyens (par exemple par `cat Confidential`).
- Quels droits a l'exécutable `/bin/cat` ?
- Quelle conséquence aurait la commande `chmod 4755 /bin/cat`, exécutée par un administrateur ?
- ATTENTION!!! Veillez à remettre les bons droits à `cat` si vous les avez modifiés !
- Quittez le login administrateur.

Chapitre 4

Commandes filtre

4.1 ED - LaboED 04-01 - Commande filtre, cat sans argument en c

| | |
|-----------|-------------------------------------------------------------|
| Titre : | ED - LaboED 04-01 - Commande filtre, cat sans argument en c |
| Support : | OS 42.3 Leap - Installation Classique |
| Date : | 08/2018 |

4.1.1 Énoncé

Une commande filtre en Unix est une commande qui lit des données sur stdin et les restitue transformées sur stdout. *cat*, utilisée sans arguments est la plus simple des "commandes filtre", elle n'effectue aucune transformation de données. Une commande filtre peut être utilisée à gauche et à droite d'un pipe.

Testez la commande *cat* sans arguments.

Ce programme très simple ne fait qu'écrire sur la sortie standard (stdout) ce qu'il lit sur l'entrée standard (stdin) jusqu'à la fin de celle-ci. Dans le cas du clavier, la fin de stdin est provoquée par la combinaison de touches **CTRL-D** (très différent de CTRL-C) qui tue les programmes liés au terminal.

cat et Mcat sans argument permettent d'afficher le contenu d'un fichier (<) ou de créer un fichier avec un contenu introduit au clavier (>).

Réécrivez la commande *cat* sans arguments en c et baptisez-la Mcat. Votre Mcat doit pouvoir réaliser les fonctions équivalentes de *cat* dans les situations suivantes : Mcat, ls | Mcat, Mcat > fichier, ls | Mcat » fichier , Mcat < fichier

4.1.2 Une solution

```
/*
NOM      : Mcat.c
CLASSE   : FS - LaboFS 04-01
#OBJET   : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011
BUGS    :
*/
#include <stdlib.h>
#include <unistd.h>

int main ()
{ int n;
char c;
while ((n=read(0,&c,1)) > 0) write(1,&c,1);
exit(0);
}
```

4.1.3 Commentaires

- Nous avons découvert que les redirections et les pipes ne sont pas traités comme des arguments, ils sont donc gérés au préalable par le shell. Dans l'énoncé, on fait donc toujours appel à Mcat sans argument.
- La logique du programme Mcat est de recopier sur stdout tout ce qu'on lit sur stdin jusqu'à la fin du fichier stdin.
- **La fin du fichier stdin "clavier" se fait par CTRL-D.**
- La lecture et l'écriture se font par appel au S.E. Ceci est beaucoup plus visible en assembleur.
- C'est le S.E. qui gère les buffers d'E/S. Le programme ne sera pas plus rapide en utilisant un buffer plus grand.
- Une commande qui utilise stdin et stdout est appelée un filtre. cat est un filtre qui 'laisse tout passer'.
- Chaque commande que vous utilisez peut être parfois vue comme un filtre. Cela dépend de la commande et de la façon de l'utiliser. Seuls les filtres peuvent être placés entre deux pipes.

4.1.4 En roue libre

Écrivez le script Demo qui montre que le code fait bien ce qu'on attend de lui. Référez-vous à l'énoncé.

4.2 ED - LaboED 04-02 - cat avec arguments en c

| | |
|-----------|---------------------------------------------|
| Titre : | ED - LaboED 04-02 - cat avec arguments en c |
| Support : | OS 42.3 Leap - Installation Classique |
| Date : | 08/2018 |

4.2.1 Énoncé

Réécrivez la commande cat en c et baptisez la McatArgs. Votre Mcat doit pouvoir réaliser les fonctions équivalentes de cat sans option.

4.2.2 Une solution

```
/*
NOM      : Mcat.c
CLASSE   : FS - LaboFS 04-02
#OBJET   : réservé au makefile
AUTEUR   : J.C. Jaumain, 07/2011 - revu mba 1/2015
BUGS    :
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

void traitement(int h)
{ int n;
  char c;
  while ((n=read(h,&c,1)) > 0)
    write(1,&c,1);
}

int main ( int argc, char * argv[] )
{ int i,h;
  if ( argc==1) traitement(0);
  else
    for (i=1; i<argc; i++)
      { h=open(argv[i],O_RDONLY);
        if (h<0){ perror (argv[i]);
        }
        else {
          traitement(h);
          close(h);
        }
      }
  exit(0);
}
```

4.2.3 Commentaires

- Les commandes telles que cat ne font pas partie du S.E. Elles sont fournies pour éviter que chacun reprogramme de telles commandes. Elles ne sont pas indispensables au fonctionnement du S.E.
- L'écriture d'une **commande filtre** suit quasi toujours le même canevas. Un main qui appelle un traitement pour chaque argument passé. Si il n'y a pas d'argument, le seul fichier traité est stdin.
- Cette commande fonctionne aussi avec des wildcards. C'est le shell qui interprète les wildcards et transforme le résultat en arguments. Pour cat, il n'y a pas de différence entre cat * et cat f1 f2 f3 si ces trois fichiers sont les seuls du répertoire courant.
- Vous pouvez obtenir une liste des commandes existantes en regardant le contenu des répertoires /bin, /usr/bin, /sbin, /usr/sbin, ... et leur rôle à l'aide des pages de manuel, chapitre 1.

4.2.4 En roue libre

Adaptez cet exercice pour réécrire la commande filtre head semblable à cat mais se limite aux 10 premières lignes du fichier(man head). Une ligne se termine par le caractère fin de ligne.

Chapitre 5

Exemples d'exercices

- + **ED005** : Réécrivez la commande head en c et baptisez-le Mhead. Votre Mhead doit pouvoir réaliser les fonctions équivalentes de head dans les situations suivantes : Mhead, ls | Mhead, Mhead > fichier, ps | Mhead »fichier. Utilisez man pour comprendre ce que fait la commande head.
- + **ED009** : Réécrivez la commande cat en c et baptisez-le Mcat. Donnez ensuite toutes les étapes que vous devez effectuer pour créer un fichier confidentiel contenant 'phrase confidentielle' que VOUS SEUL pouvez lire / écrire. Donner les droits au programme Mcat de telle façon que tous puissent exécuter Mcat qui permettra de visualiser TOUS les fichiers que VOUS pouvez lire notamment le fichier confidentiel.
- + **ED010** : Imaginez un exemple qui prouve que les droits d'accès sont vérifiés uniquement à l'open d'un fichier.
- + **ED011** : Écrivez un programme qui crée un nouveau fichier. Ce fichier contient seulement un 'a' en position 0, un 'b' en position 1000 et un 'c' en position 10000. Imaginez une technique pour connaître la taille des blocs de votre système de fichiers et déduisez la place occupée par le fichier que vous venez de créer. Vérifiez avec la commande du. Même question si vous écrivez 'a' en 0, 'b' en 70000 ? Même question si vous écrivez 'a' en 0, 'b' en 500000 ? Même question si vous écrivez 'a' en 0, 'b' en 1000000000 ? Expliquez en détail la réponse obtenue.
- + **ED013** : Écrivez un programme qui donne toute l'arborescence qui se trouve en dessous d'un répertoire donné en paramètre. Vous le baptiserez Mtree. Vous afficherez uniquement les noms des répertoires, pas les noms des fichiers. Un décalage d'un espace vers la droite symbolisera qu'un répertoire est compris dans un autre répertoire. Par exemple :

```
\  
home  
g12345  
    projet  
    systeme  
g23456  
    projet  
...
```

- + **ED020** : Un processus exécute le code correspondant à ces 3 appels :

```
fd1 = open(Nomf, OFlags);  
fd2 = dup (fd1);  
fd3 = open(Nomf, OFlags);
```

Dessinez la table des descripteurs de fichier correspondant à ce processus après l'exécution des lignes ci-dessus. Écrivez un programme qui montre que votre réponse est correcte.

- + **ED023** : LienFich est un lien vers le fichier Fich. Comment faites-vous pour savoir s'il s'agit d'un lien soft ou d'un lien hard ?

- + **ED027** : Décrivez le comportement de ce programme :

```
main()
{
    int f;
    f=open("fichier",O_WRONLY|O_CREAT,0777);
    dup2(f,1);
    close(f);
    printf("Hello world\n");
    exit(0);
}
```

- + **ED037** : Voici une série de commandes effectuées par l'utilisateur root

```
mkfs.ext /dev/fd0
mount /dev/fd0 flop
./CreeFich
cd flop
ln -s fich fs
ln fich fh
```

Où le programme CreeFich est le résultat de la compilation du programme CreeFich.c :

```
int main()
{
    char * buffer = "Hello";
    int h;
    h=open("flop/fich",O_WRONLY|O_CREAT,0644);
    write(h,buffer,5);
    lseek(h,12000,SEEK_SET);
    write(h,buffer,5);
    close(h);
    exit(0);
}
```

Réalisez une description complète et précise du F.S. ext qui résulte de l'exécution de ces commandes sachant que la taille des blocs est de 1K.

- + **ED048** : Créer un minidisk contenant un FS de type ext2. Copiez sur ce minidisk un répertoire qui contient deux fichiers simples. Décrivez le minidisk correspondant (tableau d'inodes tableau de blocs) en utilisant les résultats donnés par debugfs.
- + **ED057** : Écrivez un programme qui affiche les caractéristiques des partitions primaires d'un disque en précisant si ces partitions sont bootables.
- + **ED064** : Écrivez un programme qui construit une table contenant deux champs. Le premier champ est la taille d'un fichier contenant uniquement des caractères 'y'. Le deuxième champ est le nombre de blocs utilisés pour mémoriser ce fichier sur le disque. Exécutez ce programme avec 10, 100, 200, 300, 1000, 2000, 3000, 10000, 20000, 30000, 100000, 200000, 300000,... caractères. Expliquez le résultat obtenu.
- + **ED073** : Réécrivez la commande cp f1 f2 en c et baptisez-le Mcp. Votre Mcp ne doit réaliser que la copie d'un seul fichier vers un seul fichier. Les noms sont donnés en paramètre.
- + **ED079** : Réécrivez la commande rm en c et baptisez-le Mrm. Votre Mrm doit fonctionner avec un ou plusieurs fichiers donnés en paramètre.
- + **ED086** : Écrivez, en c, l'équivalent de la commande find /usr -perm 4755.