Responsable pédagogique	AF	AM	PB	**PM**
Période	Sem1	Sem2	**Sem 3**	Sem4
Volume horaire	Cours/TD		TP	
			8	

PSYST

TP 1 commandes unix

Indicateur temporel (hors rédaction du compte-rendu) :

questions	1h	2h	3h	4h	5h	6h-8h
1						
2						
3						

Documents à rendre : Compte-rendu contenant à minima des explications concises des commandes développées ; les sources (avec entête standard) et les résultats obtenus.

Les commandes ci-dessous sont celles qui opèrent des redirections en entrée et /ou en sortie (*cat* par exemple), et celles qui gèrent les informations sur les fichiers réguliers (*ls* par exemple).

1. La commande cat et mycat

Cat est une commande standard de l'OS unix, faites un man sur cette commande (lire la documentation) ; quelques essais puis à vous de coder en C un petit programme nommé mycat qui fera le minimum de ce que fait cat !

Cf Annexe 1 Fig.1

Pour information il est possible de rediriger le contenu d'un fichier, en entrée de cat (ou mycat) afin d'en afficher le contenu ! Faites l'essai avec votre source !! A tester ./mycat < mycat.c

Cf Annexe 1 Fig.2

2. Evolution de mycat

mycat2

Modifier votre programme *mycat.c* précédent, pour gérer maintenant l'ouverture en Lecture seule d'un fichier dont le nom sera passé en argument sur la ligne de commande.

Pour cela, créer une fonction de prototype *int lire_fichier(char *nom,* , `char` *buffer) ;*

fonction qui sera appelée depuis le main().

Cette fonction lit le fichier dont le nom est récupérer sur ligne de commande, et stocke dans buffer le texte à afficher sur la sortie standard.

Le code de retour de cette fonction pourra être le nombre d'octets lus.

A tester par exemple (pour afficher le listing de mycat2.c lui-même !) :

./mycat2 mycat2.c ou ./mycat2 mycat.c pour le 1er listing

mycat3

Partez de l'algorithme indicatif en Annexe 2. Créez donc un nouveau source mycat3.c

Faites en sorte de gérer une boucle sur les arguments de la ligne de commande, de façon à traiter plusieurs fichiers , quel que soit leur nombre.

Dans le cas ou leurs noms respectifs seront séparés par des - ; Un EOF (CTRL-D) devra être saisi au fur et à mesure des affichages comme avec *cat* d'unix. Considérez (et codez en conséquence) que la fonction read(...) renvoie 0 sur lecture du EOF (CF Annexe 1 Fig.3)

A tester par exemple (pour afficher tous vos listings) :

./mycat3 mycat.c – mycat2.c – mycat3.c

3. La commande ls et myls

Partir du source fourni en annexe 3 : on donne le source myls.c dans l'archive du TP1.

Compléter les parties adéquates afin notamment :

- **d'ouvrir le répertoire courant** si il n'y aucune option au lancement du programme
  (Cf figure 4 et 5).
- **de tester si le nom passé en argument est un répertoire !** (masquez le champs *st_mode* de la variable de type *stat ; par le masque S_IFMT*) .**Faire un man 2 stat (qq extraits en Annexe 4.)**
- si il est lisible (pensez à faire un open() sur le nom passé en argument)

- En plus l'option : -l : il faudra accéder aux informations contenues d'une variable de type stat.

Vous compléterez la fonction mode() partiellement fournie, qui interprète les codes octaux rwx , récupérés dans le champ st_mode de la variable de type stat

Le but est d'ouvrir un répertoire et de le gérer au moyen des fonctions dédiées.

Rappel : Une entrée de répertoire est décrite entre autres par la structure

```
struct direct
{
        ino_t d_ino ;           /* inode */
        char d_name[DIRSIZ] ;   /* DIRSIZ est une pseudo-cste définie à 14
                                   en général */
}
```

Il existe des fonctions dont les prototypes systèmes sont dans <dirent.h> ; dont le but est de rendre le listage d'un répertoire indépendant de la version du système.

Cf le lien : http://man7.org/linux/man-pages/man2/stat.2.html

opendir : ouverture d'un flot en lecture sur un répertoire avec positionnement au début
Elle retourne un DIR* , pointeur de directory servant en argument des autres fonctions ci-dessous.
seekdir : positionnement sur la ième entrée

readdir : lecture séquentielle d'une entrée à partir de la position courante.
closedir : fermeture du flot

Pour parcourir les données d'un répertoire ; on peut utiliser le code suivant :

```
    DIR *fdrep ;   //pointeur de type DIR
    char *nom = argv[1] ; //par exemple ….
    fdrep = opendir(nom) ;    //ouverture du repertoire
for(;;)
{   struct direct *entree  =readdir(fdrep);
  if(entree == 0)                break ;
  if(entree->d_ino == 0)     //entrée inode non utilisée
     continue ;
   …..    /* et enfin Afficher les informations liées à  entree-> d_name ;  */
  }
```

Annexe 1 : exemples d'exécutions

```
patrickMAC:correc maylaenderpatrick$ ./mycat
le fichier toto3.txt contient
le fichier toto3.txt contient
voila le tout
voila le tout
de ces commandes unix
de ces commandes unix
de base !!!
de base !!!
^C
patrickMAC:correc maylaenderpatrick$ cat
le fichier
le fichier
toto.txt contient azerty puis QWERTY puis hello
toto.txt contient azerty puis QWERTY puis hello
patrickMAC:correc maylaenderpatrick$
```

Figure 1 : exemples comparés de cat et mycat

```
patrickMAC:correc maylaenderpatrick$ ./mycat < toto.txt
azerty
QWERTY
hello
patrickMAC:correc maylaenderpatrick$ cat < toto.txt
azerty
QWERTY
hello
patrickMAC:correc maylaenderpatrick$ ./mycat < toto2.txt
AZERTY
qwerty
HELLO
vive unix...
patrickMAC:correc maylaenderpatrick$ cat < toto2.txt
AZERTY
qwerty
HELLO
vive unix...
patrickMAC:correc maylaenderpatrick$
```

Figure 2 : exemples comparés de cat et mycat
avec une redirection de fichier en entrée

```
patrickMAC:~/Documents/TS2/PSYST/TP/correc$ ./mycat2 toto.txt
azerty
QWERTY
hello
```

Figure 2 bis : exemple de mycat2
avec saisie du nom de fichier

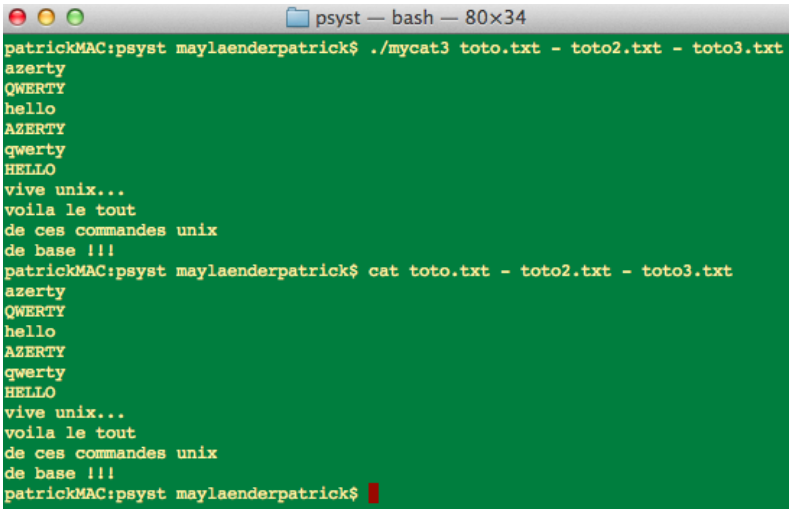

Figure 3 : exemples comparés de cat et mycat3 avec plusieurs arguments

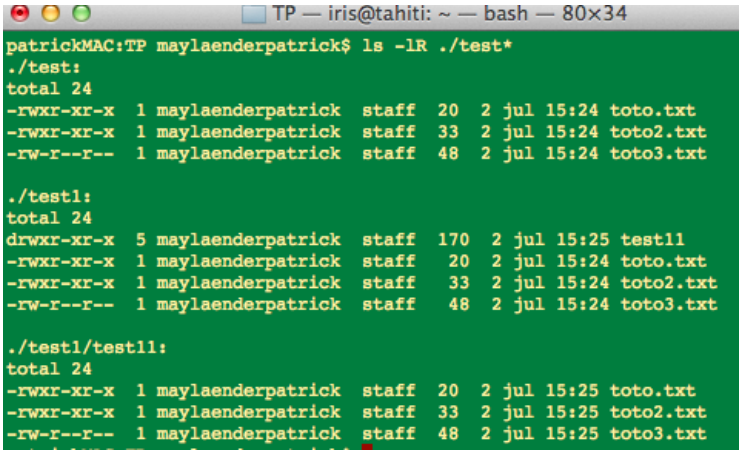

Figure 4 : listage de répertoires

pour valider myls

```
patrickMAC:correc maylaenderpatrick$ ./myls -l .
89 501  20 3026
 drwxr-xr-x89 501  20 3026 .
 drwxr-xr-x6 501  20 204 ..
 -rwxr-xr-x1 501  20 6148 .DS_Store
 -rwxr-xr-x1 501  20 0 777
 -rwxr-xr-x1 501  20 8752 autokill
 -rwxr-xr-x1 501  20 356 autokill.c
 -rwxr-xr-x1 501  20 8544 autokill1
 -rwxr-xr-x1 501  20 126 autokill1.c
 -rwxr-xr-x1 501  20 8752 autokill2
 -rwxr-xr-x1 501  20 508 autokill2.c
 -rwxr-xr-x1 501  20 8584 av
 -rwxr-xr-x1 501  20 272 av.c
 -rwxr-xr-x1 501  20 8872 avecpause
 -rwxr-xr-x1 501  20 953 avecpause.c
 -rwxr-xr-x1 501  20 307 exec.c~
 -rwxr-xr-x1 501  20 8488 execle
 -rwxr-xr-x1 501  20 243 execle.c
 -rw-rw----1 501  20 0 F1
 -rw-rw-rw-1 501  20 0 F2
 -rwxr-xr-x1 501  20 4304 gros
```

```
patrickMAC:correc maylaenderpatrick$ ./myls
89 501  20 3026
.
..
.DS_Store
777
autokill
autokill.c
autokill1
autokill1.c
autokill2
autokill2.c
av
av.c
avecpause
avecpause.c
exec.c~
execle
execle.c
F1
F2
gros
```

Figure 5 : exécutions comparées de myls avec et sans option -l

Annexe 2 : Algorithme proposé pour mycat3

```
Debut
    appeler fonction lister( argument 1 ) ;     { fonction faite dans mycat2 }
    Tant que tous arguments non traités  FAIRE
        SI argument n == '-' ALORS    { tiret d'option en argv[n]  }
            Attendre lecture d'un EOF        /* lecture CTRL-D*/
            appeler fonction lister( argument n+1 ) ;
            passage argument suivant ;
        Sinon   {argv[n] est un nom de fichier }
            { concatene en affichage les fichiers }
            appeler fonction lister( argument n ) ;
        Finsi

      passage argument suivant ;
    Fin tant que
Fin
```

Annexe 3 : source pour myls

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

#include <sys/dir.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <dirent.h>
#include <errno.h>

#define TBUF 1024
#define HEADSIZE    (sizeof(struct direct) - (MAXNAMLEN + 1))
/*****************************************************
* Role : analyse les types de fichiers et les droits *
*       E : 1 : nom du fichier courant              *
*       Sortie : int 0 = NOK ; 1 = OK               *
*****************************************************/
int mode(char *nom)
   {
      struct stat bufstat2;

      char t[11] , *p;  //pointe chaine des droits
      p = t;

      strcpy(p,"----------");   //remet à 0  les droits pour nouveau fichier courant

      if(stat(nom,&bufstat2) == -1)    //recupere info fichier courant
      {
         perror("stat in mode");
         return 0;
      }
      if ( (bufstat2.st_mode & S_IFMT) == S_IFDIR )
      {
         *p = 'd';
      }
      else if ( (bufstat2.st_mode & S_IFMT) == S_IFCHR )  //fichier special mode caractere
      {
```

```
            *p = 'c';
        }
        else if ( (bufstat2.st_mode & S_IFMT) == S_IFBLK )  //fichier special mode bloc
        {
            *p = 'b';
        }
        else if ( (bufstat2.st_mode & S_ISUID ) == S_ISUID )  //euid  (sticky bit positionne)
        {
            p +=3;  //se place sur le champ x du owner
            *p = 's';
            p -=3; //se replace sur champ initial
        }

        //extrait les champs ugo A FINIR


        if ( (bufstat2.st_mode & S_IRWXU) == S_IRWXU)
        {   //OWNER

            //printf("acces total user");
            p += 1;
            *p++ = 'r';
            *p++ = 'w';
            *p = 'x';
            p -= 3;

        }

        //extrait les champs ugo A FINIR

    //GROUP

    //OTHER


    /* affiche les flags type de fichier, et droits */
    printf(" %s", t);
    return 1;
}
```

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

    struct stat bufstat ;
    int fd;
    DIR *fdrep ;

    /* par defaut listage repertoire courant */
    //TO DO ELEVE   Q3
    char nom[25] ;

    if(argc == 1)
        strcpy(nom,".");
    else if(argc == 2)
        strcpy(nom, argv[1]) ;
    else  //option -l à priori ...
    {
        strcpy(nom, argv[2]) ;
    }
        /* fichier ne peut etre ouvert */
    if (stat(nom,&bufstat) == -1 )
        printf("%s:cannot stat %s\n",argv[0] , nom);
    /* n'est pas un repertoire */
    if ( (bufstat.st_mode & S_IFMT) != S_IFDIR )
        printf("%s: %s is not a directory \n",argv[0] , nom);
    /* n'est pas lisible  */
    if ((fd = open(nom,O_RDONLY) ) == -1 )
        printf("%s: cannot read \n",argv[0] );

    fdrep = opendir(nom);
    if(fdrep == 0)
    {
        perror("pb open dir");
        exit(0);
    }
```

```
    /**** Boucle de parcours des informations repertoires */
  for(;;)
  {

     struct dirent *entree = readdir(fdrep);
     if(entree == 0)
     break;

     if(entree->d_ino == 0)     //entrée inode non utilisée
        continue;

        if (stat(entree->d_name,&bufstat) == -1 )
        {
           printf("cannot stat %s\n",entree->d_name);
           return -1;
        }

        //TO Q3 APPELER cette fonction si option -l
       //TESTER si avec option -l sur ligne de commande

       //if( mode(          ) == 0)    //fonction analyse des droits
       //          return 0;

     //affiche le nom du fichier courant
     printf("%s \n", entree->d_name);

  }
  closedir(fdrep);
  close(fd);
  /**********  fin boucle parcours repertoire ****/


}
```

Annexe 4 : interface système stat.h

```
struct stat {
               dev_t     st_dev;         /* ID of device containing file */
               ino_t     st_ino;         /* inode number */
               mode_t    st_mode;        /* protection */
               nlink_t   st_nlink;       /* number of hard links */
               uid_t     st_uid;         /* user ID of owner */
               gid_t     st_gid;         /* group ID of owner */
               dev_t     st_rdev;        /* device ID (if special file) */
               off_t     st_size;        /* total size, in bytes */
               blksize_t st_blksize;     /* blocksize for filesystem I/O */
               blkcnt_t  st_blocks;      /* number of 512B blocks allocated */

               /* Since Linux 2.6, the kernel supports nanosecond
                  precision for the following timestamp fields.
                  For the details before Linux 2.6, see NOTES. */

               struct timespec st_atim;  /* time of last access */
               struct timespec st_mtim;  /* time of last modification */
               struct timespec st_ctim;  /* time of last status change */

           #define st_atime st_atim.tv_sec      /* Backward compatibility */
           #define st_mtime st_mtim.tv_sec
           #define st_ctime st_ctim.tv_sec
           };

…
The following mask values are defined for the file type component of
       the st_mode field:

           S_IFMT     0170000   bit mask for the file type bit fields

           S_IFSOCK   0140000   socket
           S_IFLNK    0120000   symbolic link
           S_IFREG    0100000   regular file
           S_IFBLK    0060000   block device
           S_IFDIR    0040000   directory
           S_IFCHR    0020000   character device
           S_IFIFO    0010000   FIFO

       Thus, to test for a regular file (for example), one could write:

           stat(pathname, &sb);
           if ((sb.st_mode & S_IFMT) == S_IFREG) {
               /* Handle regular file */
        }
```