

Les fichiers UNIX Rappels de L2

❑ Périphériques

- Un périphérique est un matériel connecté à l'ordinateur
 - Exemples : disques, lecteur CD, écrans, claviers, imprimantes...
- Des logiciels particuliers assurent le contrôle des entrées-sorties sur les périphériques : les pilotes (ou « drivers »)
 - Le pilote cache les détails des opérations sur le périphérique et assure la sécurité dans son utilisation
- UNIX a simplifié l'utilisation des périphériques en introduisant cinq opérations pour la plupart des périphériques
 - open, close, read, write et ioctl

Catégories de fichiers

- ❑ Les périphériques sont appelés fichiers spéciaux
- ❑ Les fichiers spéciaux comprennent
 - Les fichiers spéciaux blocs (tels que des disques) : le transfert de données est assuré par blocs entiers; un bloc particulier peut être retrouvé sur le périphérique
 - Les fichiers spéciaux caractères (tels que des claviers) : le transfert de données est géré comme un flot de caractères séquentiel
- ❑ Les autres fichiers comprennent
 - Les fichiers de données
 - Les fichiers répertoires

Gestion des fichiers par le système

- ❑ Plutôt que de désigner l'emplacement exact du fichier sur le disque, l'utilisateur utilise un nom symbolique (le nom externe)
 - Le système assure la correspondance entre ce nom et l'emplacement physique du fichier
- ❑ Les noms de fichiers sont conservés dans les répertoires (ou catalogues ou nœuds). Les fichiers sont donc organisés de manière hiérarchique.
- ❑ Les parties du système qui assurent la gestion d'une hiérarchie de fichiers sont appelées les SGF (Système de Gestion de Fichiers)
- ❑ Un SGF peut être « monté » sur un nœud d'un autre SGF
- ❑ Le nœud le plus élevé d'un SGF est appelé la racine (« root »)
- ❑ Un fichier peut être désigné
 - par une référence absolue à partir de la racine de la hiérarchie
 - par une référence relative à partir d'un catalogue particulier (« working directory »)

- ❑ Chaque utilisateur possède un **nom** d'utilisateur et un **mot de passe** nécessaires lors de la connexion
- ❑ Chaque utilisateur est répertorié dans les tables du système et possède un **numéro** (« uid »)
- ❑ Chaque utilisateur appartient à un **groupe** identifié par un numéro (« gid »)
- ❑ Le système définit trois classes d'utilisateurs
 - La classe réduite au seul **propriétaire** (« user »)
 - La classe des utilisateurs appartenant au **groupe** du propriétaire (« group »)
 - La classe des **autres** utilisateurs (« others »)
- ❑ Des droits peuvent être attribués à chacune de ces classes d'utilisateurs

- ❑ Configurations de protection
 - A chaque fichier est associé un descripteur de **droits d'accès**
 - Les droits sont définis pour chacune des trois **classes**
 - Les droits sont définis en termes de **lecture** ("r"), **écriture** ("w") et **exécution** ("x")
 - 9 combinaisons sont ainsi possibles
- ❑ Autres protections
 - Les bits "SETUID" et "SETGID" permettent d'attribuer temporairement les pouvoirs du propriétaire à un utilisateur
 - Exemple de la commande passwd et du fichier des mots de passe en accès interdit en écriture à tout utilisateur à l'exception du super-utilisateur
 - Le bit "STICKY" indique que le code doit être conservé en mémoire centrale en fin d'exécution afin d'éviter d'avoir à le recharger ultérieurement

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				SETUID	SETGID	STICKY	r	w	x	r	w	x	r	w	x
							user			group			others		

- ❑ Les fonctions **standard** du langage C
 - les fichiers **stdin**, **stdout**, **stderr** sont définis dans <stdio.h> comme des structures
 - les opérations fopen, fscanf, fprintf, fread, fwrite, fclose... utilisent des **pointeurs** sur ces structures (FILE *)
- ❑ Les primitives **UNIX**
 - Elles utilisent des **entiers**, entrées dans une table de descripteurs de fichiers

- L'utilisation d'un fichier passe TOUJOURS par les étapes suivantes
 - Ouverture --> Accès --> Fermeture
- Désignation d'un fichier
 - Un fichier possède un **nom externe** (chaîne de caractères) permettant de le désigner dans la hiérarchie des fichiers
 - Après ouverture, le fichier possède un **nom interne** (entier) permettant de le désigner lors de l'utilisation des primitives d'accès et de fermeture
- Fichiers ouverts
 - A tout fichier ouvert est associé une **position courante** désignant un déplacement en octet par rapport au début du fichier
 - Les fichiers ouverts par un processus sont décrits dans une table spécifique du PCB (FDT: File Descriptor Table)
 - Le nombre de fichiers ouverts par un processus, à un instant donné, est **limité**
 - ➔ Les fichiers de noms internes 0, 1 et 2 (correspondant aux entrées 0, 1 et 2 de la table) sont pré-ouverts (ouverts par le système au démarrage du processus)

- ❑ La primitive fork crée un processus fils et lui attribue une FDT **identique** à celle du processus père au moment de la création
- ❑ Processus père et processus fils partagent les mêmes fichiers à la suite de la création du fils
 - La position initiale dans les fichiers du processus fils est celle du père au moment du fork
 - A partir du fork, les opérations dans les fichiers sont indépendantes chez le processus père et chez le processus fils
- ❑ La fermeture d'un fichier dans un des deux processus n'a aucun effet dans l'autre processus

- ❑ A tout processus est associé un masque de création précisant les **interdictions** qui seront systématiquement imposées lors de la création d'un fichier
- ❑ Ce masque s'exprime en termes r, w et x sur les trois classes d'utilisateurs
 - Exemple : Le masque 037 signifie "0-011-111"
 - Interdiction en "w" et "x" sur la classe "group"
 - Interdiction en "r", "w" et "x" sur la classe "others"
- ❑ Rappel
 - 037 est un nombre exprimé en octal (base 8, où chaque chiffre est codé sur 3 bits)
- ❑ Ces interdictions se **combineront** avec les **autorisations** spécifiées lors de la **création** de tout fichier
- ❑ La primitive umask permet de préciser ce masque d'interdiction
 - L'appel à cette primitive retourne l'ancienne valeur de masque
 - Voir création de fichier

```
int umask(int nouveauMasque);
```

- ❑ La primitive creat permet de créer un fichier spécifié par son nom externe.
 - Après création, le fichier est ouvert en **écriture**
 - creat retourne
 - le nom interne du fichier
 - -1 en cas d'erreur
- ❑ L'appelant doit posséder les **droits** permettant cette création
- ❑ Si le fichier existe déjà, le contenu est effacé, mais les attributs restent inchangés
- ❑ Sinon, le fichier est créé avec les attributs suivants
 - les droits d'accès sont ceux précisés par l'appelant, modifiés en fonction de la valeur du masque de création
 - le bit STICKY est mis à 0
 - le propriétaire est défini par le numéro effectif de l'appelant

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

int creat
(const char *cheminAcces,
 mode_t droits);
```

- ❑ Masque de saisie : **0077**
 - toutes interdictions à la classe "group" et la classe "others"
- ❑ Création d'un fichier par un éditeur de texte : **0666**
 - autorisations en lecture et écriture pour tous
- ❑ Résultat : **0600**
 - autorisations en lecture et écriture pour le propriétaire uniquement

r	w	x	r	w	x	r	w	x
1	1	0	0	0	0	0	0	0
user	group			others				

r	w	x	r	w	x	r	w	x
1	1	0	1	1	0	1	1	0
user	group			others				

r	w	x	r	w	x	r	w	x
1	1	0	0	0	0	0	0	0
user	group			others				

- Dans un but de portabilité, les droits, de type `mode_t`, sont définis par des constantes en POSIX

➤ Les droits doivent être combinés avec des opérateurs `or` (caractère `'|'`)

S_IRUSR S_IWUSR S_IXUSR S_IRWXU	Droits du propriétaires
S_IRGRP S_IWGRP S_IXGRP S_IRWXG	Droits du groupe du propriétaire
S_IROTH S_IWOYH S_IXOTH S_IRWXO	Droits des autres
S_ISUID S_ISGID	Bits d'exécution

- La primitive `open` permet d'ouvrir un fichier spécifié par son nom externe

- Elle retourne

➔ le nom interne du fichier

➤ -1 en cas d'erreur

- **Attention** : L'ouverture d'un fichier peut entraîner la création du fichier lorsque celui-ci n'existe pas et que le programmeur ne l'a pas interdit

➤ Dans ce cas, le paramètre `droits` sera pris en compte et doit être **spécifié**

➤ Dans le cas contraire, ce paramètre peut être omis

- Le fichier est ouvert avec un mode d'utilisation fonction du paramètre `mode`
- L'ouverture d'un fichier réserve une entrée dans la table des fichiers ouverts (FDT), définie dans le PCB du processus

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

int open (const char *cheminAcces,
          int mode,
          mode_t droits);
```

- Le mode est défini par une combinaison de bits entre

➤ un des 3 bits

- O_RDONLY Lecture seule
- O_WRONLY Écriture seule
- O_RDWR Lecture et écriture

➤ et un ou plusieurs bits parmi

- O_APPEND Écriture en fin de fichier
- O_CREAT Si le fichier est inexistant : création du fichier
Sinon : sans effet (sauf si O_EXCL est aussi positionné)
- O_EXCL Si O_CREAT est aussi positionné : Interdit la création du fichier
Sinon, l'effet est indéterminé
- O_TRUNC Si le fichier est ouvert avec O_WRONLY ou O_RDWR, le fichier est effacé
(curseur remis à 0, droits et propriétaire sont inchangés)
Sinon, l'effet est indéterminé

- Ces constantes sont définies dans le fichier `fcntl.h`

- La primitive `close` permet de fermer un fichier spécifié par son nom interne
 - L'entrée correspondante de la table des fichiers ouverts est alors libérée
 - Lorsqu'un processus se termine, cette primitive est automatiquement appelée
- ➔ pour tous les fichiers encore ouverts

```
int close(int nomInterne);
```

- La primitive `read` permet de lire une **suite d'octets** dans un fichier spécifié par son nom interne

- Elle retourne

- le nombre d'octets **effectivement** lus
 - 1 en cas d'erreur

```
#include <unistd.h>

int read(int nomInterne,
        void *adresseInfoEcrire,
        unsigned nombreOctetsDemande);
```

- Le nombre d'octets lus par la primitive peut être différent (inférieur) au nombre d'octets demandés si la fin de fichier a été rencontrée lors de la lecture
- Une lecture demandée en **fin** de fichier retourne la **valeur 0**
 - La notion de EOF n'existe pas !!!

- La primitive `write` permet d'écrire une **suite d'octets** dans un fichier spécifié par son nom interne

- Elle retourne

- Le nombre d'octets **effectivement** écrits
 - 1 en cas d'erreur

- L'écriture est effectuée à partir de la **position courante**

- Si le fichier a été ouvert avec le mode `O_APPEND`, la position courante est située en fin de fichier

```
#include <unistd.h>

int write (int nomInterne,
          void *adresseInfoLue,
          unsigned nombreOctetsTranmis);
```

- Écrire en langage C, la fonction `int afficher(char* nomFichier);` qui permet de copier le contenu d'un fichier, désigné par son nom externe dans le système, sur la sortie standard de l'application (entrée numéro 1 de la table des fichiers ouverts i.e. l'écran)

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#define TAILLE 100

int main(){
    int fic;
    int nb;
    char tmp[TAILLE];

    printf("\n *** DEBUT DU PROGRAMME\n");
    /* Ouverture du fichier a afficher */
    fic = open("ex_fichier.c", O_RDONLY);
    if (fic == -1) {
        perror("Probleme d'ouverture ");
        exit(2);
    }

    /* Affichage du fichier */
    while ((nb = read(fic, tmp, TAILLE)) > 0)
        write(1, tmp, nb);

    printf("\n\n *** FIN DU PROGRAMME\n");
}
```

Demande d'ouverture du fichier de nom « ex_fichier.c »
L'ouverture est en lecture seule

Test effectué suite à la demande d'ouverture

Lecture d'un bloc de TAILLE octets dans le fichier ouvert
Le nombre d'octets effectivement lus peut être inférieur à TAILLE, voire être nul

Écriture dans le fichier de nom interne 1 (probablement écran, cf. dup) du bloc de nb octets lus

De nombreux programmes UNIX sont appelés des **filtres**

- Un filtre lit sur l'entrée standard, effectue une transformation puis écrit sur la sortie standard. Un filtre affiche ses erreurs sur la sortie standard en erreur. Un filtre n'a aucune interaction avec l'utilisateur
- Exemples : cat, wc, grep

Il arrive qu'on veuille utiliser ces filtres dans un contexte différent du contexte par défaut : lecture dans un fichier ou écriture dans un fichier

La **duplication** du descripteur du fichier permet de rediriger

- l'entrée vers le fichier désiré (initialement prévue sur l'entrée standard)
- la sortie vers le fichier désiré (initialement prévue sur la sortie standard)

Remarque : La redirection n'est pas utile lorsqu'on écrit la totalité de l'application. Elle est nécessaire uniquement lorsqu'on utilise des binaires utilisant des fichiers de noms internes fixés

La primitive dup permet de dupliquer le descripteur spécifié en paramètre, dans la **première entrée libre** de la table des fichiers ouverts

La primitive dup2 permet

- de fermer l'entrée spécifiée en second paramètre
- puis de copier le descripteur spécifié dans l'entrée spécifiée

```
#include <unistd.h>

int dup (int origine);

int dup2(int origine, int destination);
```

Écrire une commande mon_cp, permettant de recopier le contenu d'un fichier source dans un fichier destination puis d'afficher ensuite le contenu du même fichier source à l'écran de l'utilisateur.

Cette commande utilisera, pour les deux affichages demandés, la fonction afficher écrite précédemment.

Elle sera utilisée ainsi : *mon_cp source destination*

```
#include <stdio.h>
#include <fcntl.h>
#define TAILLE 100

int main(){
    int fic;
    int nb;
    char tmp[TAILLE];

    printf("\n *** DEBUT DU PROGRAMME\n");
    /* Ouverture du fichier a afficher */
    fic = open("ex_dup.c", O_RDONLY);
    if (fic == -1) {
        perror("Probleme d'ouverture ");
        exit(2);
    }

    /* Duplication sur l'entree 0 */
    close(0);
    dup(fic);

    /* Affichage du fichier en utilisant l'entree 0 */
    while ((nb = read(0, tmp, TAILLE))
        write(1, tmp, nb);

    printf("\n\n *** FIN DU PROGRAMME\n");
}
```

Demande d'ouverture du fichier de nom « ex_dup.c »
L'ouverture est en lecture seule

Fermeture du fichier de nom interne 0 (probablement le clavier). L'entrée 0 est alors libre

Duplication du fichier de nom interne « fic »
dans l'entrée 0 (obligatoirement, la 1^{re} libre)
L'entrée 0 est alors redirigée vers le fichier fic

Lecture d'un bloc de TAILLE octets dans le fichier de nom interne 0. Du fait de la redirection, la lecture sera faite dans le fichier « ex_dup.c »