Opérations sur le système de fichier :

`int creat(const char *name, mod_t mode)`	Crée un fichier avec le chemin d'accès `name` et les droits d'accès `mode`. `mode` est en base octale. Chaque groupe de 3 bits indique les droits d'accès du propriétaire, du groupe et des autres.
`int link(const char *oldpath, const char *newpath)`	Crée un lien direct `newpath` vers un fichier dont le chemin d'accès est `oldpath`
`int unlink(const char *pathname)`	Efface un lien donc le chemin d'accès est `pathname`
`int symlink(const char *oldpath, const char *newpath)`	Crée un lien indirect/symbolique (c.a.d un lien vers un lien) `newpath` vers un fichier donc le chemin d'accès est `oldpath`.
`int rename(const char *oldpath, const char *newpath)`	Renomme un fichier dont le chemin d'accès est `oldpath` vers le chemin d'accès `newpath`.
`int mkdir(const char *name, mod_t mode)`	Crée un répertoire dont le chemin d'accès est `name`
`int rmdir(const char *name)`	Efface un répertoire dont le chemin d'accès est `name`

Opérations sur les entrées/sorties et les fichiers

`ssize_t read(int fd, void *buffer, size_t longueur)`	Lit séquentiellement maximum `longueur` octets depuis le fichier dont le descripteur est `fd` et les copier à l'adresse pointée par `buffer`. Le résultat indique le nombre d'octet effectivement lus, il peut être inférieur à `longueur`. Si il est égal à zéro, il indique la fin de fichier.
`ssize_t write(int fd, void *buffer, size_t longueur)`	Écrit séquentiellement maximum `longueur` octets dans le fichier dont le descripteur est `fd` depuis le buffer dont l'adresse est pointée par `buffer`. Le résultat indique le nombre d'octets effectivement écrits, il peut-être inférieur à longueur.
`int open(const char *nom, int flags)`	Ouvre un fichier dont le chemin d'accès est `nom`, selon les `flags` indiqués (lecture seule, écriture seule, etc). Le résultat est un entier qui si il est positif est un descripteur de fichier utilisable dans `read` et `write`
`int close(int fd)`	Ferme le fichier dont le descripteur est `fd`.
`int dup(int oldfd)`	Duplique un descripteur existant vers un autre, en choisissant le descripteur le plus petit disponible pour le processus appelant. Le résultat, si positif ou nul, est le nouveau descripteur dupliqué et permet d'accéder au même fichier que celui dont le descripteur a été dupliqué.
`int dup2(int oldfd, int newfd)`	Même fonction que `dup`, mais ferme `newfd` avant de dupliquer `oldfd`.
`off_t lseek(int fd, off_t offset, int whence)`	Déplace la tête de lecture associée à un fichier dont le descripteur de fichier est `fd`. **Attention :** Certains descripteurs ne sont pas « seekables », car on n'en connaît pas le contenu à l'avance. Ex : l'entrée du clavier.
`int stat(const char *path, struct stat *buf);`	Retourne des informations sur le statut d'un fichier dont le chemin d'accès est donné.
`int fcntl(int fd, int cmd, ... /* arg */ );`	Effectue des opérations de contrôle de propriétés sur un descripteur de fichier.

Opérations sur les processus :

`void _exit(int status)`	Quitte le processus appelant et indique un code de sortie.
`pid_t fork()`	Duplique le processus appelant. Le processus fils reçoit la valeur de retour `0`. Le processus père reçoit le `pid` du fils
`int execve(const char *path, const char *argv[], char * const envp[])`	Remplace le processus appelant par le fichier indiqué en paramètre dans `path`, puis l'execute. Ne retourne que si il y a une erreur.
`pid_t wait(int *status)`	Bloque jusqu'à ce que le processus fils ait retourné son core de sortie, qu'on obtient dans `status`.
`pid_t getpid()`	Récupère le `pid` du processus appelant.
`pid_t getppid()`	Récupère le `pid` du père du processus appelant.
`int chdir(const char *path)`	Change le répertoire de travail courant du processus appelant.

Opérations de communication inter-processus :

`int pipe(int pipefd[2])`	Crée un tube de communication FIFO. Toute donnée écrite dans `pipefd[1]` apparait dans `pipefd[0]`.