1 Système de fichier

`ssize_t read(int fd, void *buf, size_t count);`
Lit count bytes du fichier fd dans le buffer buf

`ssize_t write(int fd, const void *buf, size_t count);`
Ecrit count bytes dans le fichier fd depuis de buffer buf

`int open(const char *pathname, int flags, mode_t mode);`
Ouvre un fichier.
flags : O_RDONLY, O_WRONLY, O_RDWR, O_CLOEXEC, O_CREAT, O_DIRECTORY, O_EXCL, O_NOCTTY, O_NOFOLLOW, O_TMPFILE, O_TRUNC
mode : les droits

`int close(int fd);`
Ferme le file descriptor

`pid_t wait(int *status);`
Attends un fils

`pid_t waitpid(pid_t pid, int *status, int options);`
Attends un processus
options à 0

`int creat(const char *path, mode_t mode);`
Créer un fichier à l'addresse indiquée
mode : les droits

`int link(const char *oldpath, const char *newpath);`
Créer un raccourcis dur

`int unlink(const char *pathname);`
Supprime le lien sur un fichier

```
int chdir(const char *path);

int chmod(const char *path, mode_t mode);
int stat(const char *pathname, struct stat *statbuf);
struct stat {
dev_t st_dev; /* ID of device containing file */
ino_t st_ino; /* Inode number */
mode_t st_mode; /* File type and mode */
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; /* Block size for filesystem I/O */
blkcnt_t st_blocks; /* Number of 512B blocks allocated
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 */
};

S_ISREG(m)
S_ISDIR(m)
off_t lseek(int fd, off_t offset, int whence);
```

2 Réseau et communication

`int sockfd = socket(int socket_family, int socket_type, int protocol);`
socket_vamily : AF_INET (réseau) ou AF_UNIX (local)
socket_type : SOCK_STREAM (TCP) ou SOCK_DGRAM (UDP)
protocol : ?

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);;
```

```
int listen(int sockfd, int backlog);
```

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr,
socklen_t addrlen);
```

Structure de sockaddr :
`struct sockaddr`

```
u_short sa_family;
char sa_data[14];
;
```

```
int dup(int oldfd);
```

```
int dup2(int oldfd, int newfd);
```
newfd devient la copie de oldfd

```
int pipe(int pipefd[2]);
```
Créer deux fd, en mettant l'entrée du premier dnas la sortie du deuxième

3 Execution parallèle

3.1 Threads

`int pthread_create(pthread_t* pthread, const pthread_attr_t *attr,void* (*fonction)(void *), void *arg);` Créer un nouveau thread
pthread -> l'identité du nouveau thread
attr -> attributs du thread (mettre NULL)
fonction -> fonction à éxécuter en parallèle
arg -> argument de la fonction

```
pthread_t pthread_self(void)
```
Renvoie l'identité du thread

```
int pthread_equal(pthread_t t1, pthread_t t2);
```
Vérifie si les deux thread sont égaux.

```
int pthread_exit(void* value_ptr);
```
Termine le thread

```
int pthread_join(pthread_t tid, void **value);
```
appel bloquant jusqu'a ce que le thread tid termine.
Si value n'est pas Null, pointe sur la valeur de retour du thread

3.2 Mutexes

`pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;`
Création d'un mutex.

`int pthread_mutex_lock(pthread_mutex_t *mutex);`
Lock le mutex (bloquant)

`int pthread_mutex_trylock(pthread_mutex_t *mutex);`
Lock le mutex (non bloquant, renvoie 0 en cas de succès)

`int pthread_mutex_unlock(pthread_mutex_t *mutex);`
Dévérouillage d'un mutex

3.3 Semaphore

`int sem_init(sem_t* sem, int pshared, unsigned int value);`
Initialise la nouvelle sémaphore sem avec la valeur value.
pshared = 0 -> sémaphore partagée entre tous les processus légers
pshared != 0 -> sémaphore partagée entre tous les processus

`int sem_destroy(sem_t* sem);`
Détruit la sémaphore sem

`int sem_wait(sem_t* sem);`
Attends d'avoir un cookie

`int sem_post(sem_t* sem);` Rends le cookie

4 Contrôle

```
void exit(int stat_us);
int execve(const char *filename, char *const argv[], char *const envp[]);
pid_t fork(void);
time_t time(time_t *t);
pid_t getpid(void);
pid_t getppid(void);
int rename(const char *oldpath, const char *newpath);
int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);
int kill(pid_t pid, int sig); Envoie un signal
```
SIGINT -> interruption (Ctrl-C)

`int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`
Donne l'action à faire lors de la reception du signal
signum -> numéro du signal
act -> action à réaliser
oldact -> récupère l'ancien handler

Structure de sigaction :
struct sigaction
void (*sa_handler) (int) ; /* Fonction de handler*/
sigset_t sa_mask ; /* Les signaux masqués */
int sa_flags ; /* flags */