

# Aide mémoire Unix et librairie standard C v2.0

Michel Meynard

15 septembre 2014

## 1 Erreurs

Les appels systèmes (man 2) retournent souvent un entier positif ou nul en cas de succès et -1 en cas d'échec. Dans ce cas, `errno` est une variable statique contenant le code d'erreur `#include <errno.h> extern int errno;`

**afficher l'erreur** `void perror (const char *s)` affiche `s` puis l'erreur suivie d'un retour ligne. Entêtes : `stdio.h`.

**description de l'erreur** `char *strerror (int errnum)` retourne une chaîne décrivant l'erreur. Entêtes : `string.h`.

## 2 Gestion des Entrées/Sorties

### 2.1 Appels systèmes (2)

`pathname` est un nom de fichier.

**ouverture** `int open(const char *pathname, int mode)` ouverture selon un mode d'accès parmi `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_APPEND`; retourne un descripteur de fichier  $\geq 0$  ou -1 si erreur.

**création** `open(const char *pathname, int m, mode_t droits)` création avec `m=O_CREAT | O_WRONLY | O_TRUNC` avec des droits constitués par `droits & ~umask`. Par exemple, `droits=0644`. Entêtes : `sys/types.h`, `sys/stat.h`, `fcntl.h`

**lecture** `ssize_t read(int fd, void *buf, size_t count)` tente de lire `count` octets depuis `fd` dans le `buf`; retourne le nb d'octets lus, -1 si erreur, 0 si fin de fichier. Entêtes : `unistd.h`.

**écriture** `ssize_t write(int fd, const void *buf, size_t count)` écrit dans `fd` `count` octets depuis `buf`; retourne le nb d'octets écrits, -1 si erreur. Entêtes : `unistd.h`.

**déplacement** `off_t lseek(int fd, off_t offset, int whence)` déplace la tête de lect/écrit. selon `whence` parmi `SEEK_SET`, `SEEK_CUR`, `SEEK_END`. Retourne la position mesurée en octets depuis le début du fichier. Entêtes : `sys/types.h`, `unistd.h`.

**fermeture** `int close(int fd)` ferme le descripteur

**vidage tampon** `int fsync (int fd)`. Entêtes : `unistd.h`.

**duplication** `int dup(int oldfd)` retourne un nouveau descripteur sur la même entrée dans la table des fichiers ouverts. Entêtes : `unistd.h`.

**opérations sur fichier ouvert** `int fcntl(int fd, int cmd)` permet de manipuler le fichier ouvert (verrouiller, dupliquer, gérer signaux E/S, gérer bail). Entêtes : `unistd.h`, `fcntl.h`.

**tronquer ou étendre** `int truncate(const char *path, off_t length); int ftruncate(int fd, off_t length)` réduit ou allonge la taille d'un fichier ouvert ou non. Entêtes : `sys/types.h`, `unistd.h`.

**verrouiller** `int flock(int fd, int operation)` verrouillage coopératif avec opération dans `LOCK_SH` `shared`,

`LOCK_EX` exclusif, `LOCK_UN` déverrouiller. Entêtes : `sys/file.h`.

### 2.2 Fonctions de bibliothèque (3)

Les flux (`FILE *`) `stdin`, `stdout`, `stderr` préexistent. Par défaut, l'entête `stdio.h` doit être inclus. `EOF` est une macro valant -1.

**ouverture** `FILE *fopen(const char *path, const char *mode); FILE *fdopen(int fd, const char *mode); FILE *freopen(const char *path, const char *mode, FILE *stream)` avec `mode` dans `r`, `r+`, `w`, `w+`, `a`, `a+`. Ouverture d'un flux, conversion d'un descripteur en flux, ouverture d'un nouveau fichier dans un flux existant.

**fermeture** `int fclose(FILE *fp)`. Entêtes : `stdio.h`

**lecture, écriture** `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream); size_t fwrite (const void *ptr, size_t size, size_t nmemb, FILE *stream)` lit/écrit dans `stream` `nmemb` champs de `size` octets dans la zone pointée par `ptr`. Retourne le nb d'octets lus.

**lectures** `int fgetc(FILE *stream); char *fgets(char *s, int size, FILE *stream); int getc(FILE *stream); int get-char(void); char *gets(char *s)` retourne un char ou une ligne dans un flux ou `stdin`.

**rejet** `int ungetc(int c, FILE *stream)` rejette le char et pointe sur celui-ci.

**écritures** `int fputc(int c, FILE *stream); int fputs(const char *s, FILE *stream); int putc(int c, FILE *stream); int putchar(int c); int puts(const char *s)` écrit un char ou une chaîne dans un flux ou `stdout`.

**déplacement et position** `int fseek(FILE *stream, long offset, int whence); long ftell(FILE *stream); void rewind(FILE *stream); int fgetpos(FILE *stream, fpos_t *pos); int fsetpos(FILE *stream, fpos_t *pos)` `fseek` déplace selon `whence` parmi `SEEK_SET`, `SEEK_CUR`, `SEEK_END`; `ftell` indique la position; `rewind` remet au début; `fsetpos` utilise une structure de position complexe.

**vidage tampon utilisateur** `int fflush(FILE *flux)`. Entêtes : `stdio.h`

**test Fin de Fichier** `int feof(FILE *stream)`. Entêtes : `stdio.h`

**descripteur** `int fileno(FILE *stream)` ret le descripteur de fichier. Entêtes : `stdio.h`

**erreur** `int ferror(FILE *stream)` retourne vrai si une erreur est survenue. Entêtes : `stdio.h`.

## 2.3 E/S formatées (3)

Par exemple, `printf("f = %6.2f et ch = %s\n",f,s)`

**printf** *int printf(const char \*format, ...);*  
*int fprintf(FILE \*stream, const char \*format, ...);*  
*int sprintf(char \*str, const char \*format, ...);*  
*int snprintf(char \*str, size\_t size, const char \*format, ...)*  
écriture formatée dans stdout ou un flux ou une chaîne. Les arguments doivent correspondre aux spécifications de conversion respectant la syntaxe suivante : `[%flags][width][.precision][length]type`.

### 2.3.1 spécifications de conversion de printf

**type** (obligatoire)

- i ou d** l'int est représenté en décimal (la précision indique le nb mini de chiffres (1));
- u ou o ou x ou X** l'int non signé est représenté en décimal ou octal ou hexa; la précision indique le nb mini de chiffres (1);
- f ou F** le double est représenté en décimal non normalisé (sans exposant); la précision indique le nb de chiffres après la virgule (6);
- e ou E** le double est représenté en décimal normalisé (avec exposant); la précision indique le nb de chiffres après la virgule (6);
- g ou G** le double est représenté en f ou e selon la valeur de l'exposant;
- a ou A** le double est représenté en hexadécimal normalisé;
- c** l'int non signé est représenté comme caractère;
- s** la chaîne (char \*) est affichée; la précision indique le nb maxi de car;
- p** le pointeur est représenté en hexa;

**flags**

un ou plusieurs attributs suivants :

- 0** remplissage avec des 0 plutôt que des espaces pour les valeurs numériques;
- alignement à gauche;
- espace ou +** pour les nombres positifs, préfixer par un espace ou un +;
- #** change la forme : préfixe 0 (octal) ou 0x (hexadécimal), point décimal affiché même si pas utile.

**length**

largeur minimale de champ; littéral entier ou \* si définie dans l'argument suivant.

**précision**

- pour les types d, i, o, u, x, X : nombre minimum de chiffres;
  - pour les types a, A, e, E, f et F : le nombre de décimales;
  - pour les types g, G : le nombre maximum de chiffres significatifs;
- la précision peut être variable et définie dans l'argument suivant (.\*)

**length** (modificateur de longueur)

valable pour les types d, i, o, u, x, X;

**hh** argument ptr sur char; (half half)

**h** argument ptr sur short int; (half)

**l** argument ptr sur long int;

**ll** argument ptr sur long long int;

**L** argument ptr sur long double (a, A, e, E, f, F, g, ou G);

**z** argument ptr sur size\_t (entier long non signé);

**scanf** *int scanf(const char \*format, ...);*  
*int fscanf(FILE \*stream, const char \*format, ...);*  
*int sscanf(const char \*str, const char \*format, ...)*  
Exemple : `scanf("%d %f",&i,&f)` : lit une séquence formatée de caractères en entrée et convertit certains facteurs en représentation interne dans des arguments pointeurs. Retourne le nb de correspondances effectuées ou EOF si erreur.

La chaîne **format** est une séquence de directives contenant :

**blanc(s)** correspond à 0 ou des blancs (espace, \t, \n);

**car ordinaire** correspond à ce car;

**%...** correspond à un format;

### 2.3.2 spécifications de format de scanf

`[%*][a][lngmaxi][length]type`

**type** (obligatoire)

**i,d,u,o,x,X** entier correspondant aux spécif. de printf;

**f,e,g,a** flottant correspondant aux spécif. de printf;

**s** chaîne sans blanc (ajout auto de \0);

**c** lngmaxi (1) caractères sont lus (pas d'ajout de \0);

**classe [a-z]** séquence de minuscules, **[~0-9]** séquence de tout sauf chiffre;

**p** pointeur : l'argument doit être un **void\*\***;

**n** le nb de car précédemment lus est affecté dans l'argument **int\***;

**\*** (oubli de conversion)

un facteur correspondant est lu en entrée mais ne nécessite aucun argument;

**a** (allocation automatique)

l'argument **char\*\*** est affecté d'une zone dynamique de taille correspondant au facteur d'entrée

**lngmaxi**

La lecture des caractères s'arrête lorsque ce maximum est atteint

**length** (modificateur de longueur)

idem printf : hh, h, l, ll, L, z

## 3 Gestion du Système de Fichier

**vérifier droits** *int access(const char \*pathname, int mode)* ret 0 si accès dans le mode est autorisé : existence, lisibilité, inscript., exéc. du fichier (F\_OK R\_OK, W\_OK, X\_OK). Entêtes : `unistd.h`

**changer droits** *int chmod(const char \*path, mode\_t mode)* 4\*3=12 bits définissant les droits d'un fichier du plus signif au moins : S\_ISUID (Set User Id), S\_ISGID, S\_ISVTX (sticky), S\_IRUSR, S\_IWUSR, S\_IXUSR, S\_I(R|W|X)GRP, S\_I(R|W|X)OTH. Entêtes : `sys/stat.h`

**changer proprio** *int chown(const char \*path, uid\_t owner, gid\_t group)* change le propriétaire du fichier. Entêtes : `unistd.h`

**déplacer** *int rename(const char \*oldpath, const char \*newpath)* renomme ou déplace le fichier. Entêtes : `stdio.h`

**créer lien** *int link(const char \*oldpath, const char \*newpath)* crée un lien supplémentaire. Entêtes : `unistd.h`

**supprimer** *int unlink(const char \*pathname)* supprime un lien dur et éventuellement le fichier. Entêtes : `unistd.h`

**créer lien symb** *int symlink(const char \*cible, const char \*nom)* crée un lien symbolique. Entêtes : `unistd.h`

**lire lien symb** *int readlink(const char \*path, char \*buf, size\_t bufsiz)*; écrit dans le buf, le contenu du lien symb. Entêtes : `unistd.h`

**état d'un fichier** *int stat(const char \*path, struct stat \*buf)*; *int fstat(int fd, struct stat \*buf)*; *int lstat(const char \*path, struct stat \*buf)* lit l'état d'un fichier par son nom, son descripteur ou celui du lien symb (i-node, proprio., taille, nb liens, heures). Entêtes : `sys/types.h`, `sys/stat.h`, `unistd.h`

```
struct stat struct stat {
    dev_t st_dev; /* ID du périph. contenant le fic */
    ino_t st_ino; /* Numéro inoead */
    mode_t st_mode; /* Protection */
    nlink_t st_nlink; /* Nb liens matériels */
    uid_t st_uid; /* UID propriétaire */
    gid_t st_gid; /* GID propriétaire */
    dev_t st_rdev; /* ID périphérique (si fic spécial) */
    off_t st_size; /* Taille totale en octets */
    blksize_t st_blksize; /* Taille de bloc pour E/S */
    blkcnt_t st_blocks; /* Nombre de blocs alloués */
    time_t st_atime; /* Heure dernier accès */
    time_t st_mtime; /* Heure dernière modification */
    time_t st_ctime; /* Heure dernier changement état */
};
```

**macros fonctions sur stat.st\_mode** S\_ISREG(m), S\_ISDIR(m), S\_ISCHR(m), S\_ISBLK(m), S\_ISFIFO(m), S\_ISLNK(m), S\_ISSOCK(m) avec m=etat.st\_mode;

**macros masques sur stat.st\_mode** S\_IRWXU, S\_IRUSR, S\_IWUSR, S\_IXUSR, S\_IRWXG, S\_IRGRP, ..., S\_IRWXXO, ..., S\_IXOTH à tester avec `etat.st_mode&masque`

**changer le masque des droits** *mode\_t umask(mode\_t mask)* retourne le masque précédent (par défaut `mask=S_IWGRP/S_IWOTH=022`)

**modifier date** *int utimes(const char \*filename, const struct timeval times[2])* met à jour la date de dernier accès et modification (`touch(1)`). Entêtes : `sys/time.h`

## 3.1 Répertoires

**création/suppr** *int mkdir(const char \*pathname, mode\_t mode)*; *int rmdir(const char \*pathname)* crée, supp un répertoire. Entêtes : `unistd.h`

**ouvrir/fermer DIR** *\*opendir(const char \*name)*; *DIR \*fdopendir(int fd)*; *int closedir(DIR \*dir)* Ouvre un rép avant de le parcourir. Entêtes : `sys/types.h`, `dirent.h`

**lire entrée suiv.** *struct dirent \*readdir(DIR \*dir)* la structure `dirent` contient un champ `char d_name[]`. Entêtes : `dirent.h`

```
struct dirent struct dirent {
    ino_t d_ino; /* numéro d'inoead */
    off_t d_off; /* décalage dirent suivante */
    unsigned short d_reclen; /* longueur de cet enr. */
    unsigned char d_type; /* type du fichier */
    char d_name[256]; /* nom du fichier */
};
```

**position** *off\_t telldir(DIR \*dir)* ret la position actuelle. Entêtes : `dirent.h`

**se déplacer** *void seekdir(DIR \*dir, off\_t offset)* se déplace à la position `offset`.

**revenir au début** *void rewinddir(DIR \*dir)*. Entêtes : `dirent.h`

**filtrer et trier** *int scandir(const char \*dir, struct dirent \*\*\*namelist, int (\*filter)(const struct dirent \*), int (\*compar)(const struct dirent \*\*, const struct dirent \*\*))* Par exemple, permet de ne sélectionner que les fichiers C dans l'ordre alphabétique inverse!

## 4 Gestion des processus

**répertoire de travail** *char \*getcwd(char \*buf, size\_t size)*; écrit le chemin d'accès absolu dans le buffer puis le retourne. Entêtes : `unistd.h`

**identifiants** *pid\_t getpid(void)*; *pid\_t getppid(void)*; *uid\_t getuid(void)*; *uid\_t geteuid(void)* ret le pid, ppid, user id, effective user id du pus courant. Entêtes : `unistd.h`, `sys/types.h`

**user id** *uid\_t getuid(void)*; *uid\_t geteuid(void)* retourne l'id de l'utilisateur réel ou effectif

**infos user** *struct passwd \*getpwuid(uid\_t uid)* retourne un ptr sur une struct :

```
struct passwd {
    char *pw_name; /* Nom d'utilisateur */
    char *pw_passwd; /* Mot de passe de l'u. */
    uid_t pw_uid; /* Identifiant de l'u. */
    gid_t pw_gid; /* Identifiant du groupe */
    char *pw_gecos; /* Nom réel */
    char *pw_dir; /* Répertoire personnel */
    char *pw_shell; /* Interpréteur de com. */
};
```

**création** *pid\_t fork(void)* crée un pus enfant, copie du parent; ret -1, 0 (enfant), pid du parent. Entêtes : `unistd.h`

**envoi signal** *int kill(pid\_t pid, int sig)* envoie un signal parmi SIGKILL, SIGINT, ... Entêtes : `signal.h`

**exec** *int execve(const char \*fichier, char \*const argv[], char \*const envp[])* (2)  
*extern char \*\*environ; int execl(const char \*path,*

*const char \*arg, ...)*  
*int execlp(const char \*file, const char \*arg, ...)*  
*int execl(const char \*path, const char \*arg,..., char \**  
*const env[])*  
*int execv(const char \*path, char \*const argv[])*  
*int execvp(const char \*file, char \*const argv[])*  
recouvrement du pus par un fichier exécutable : l pour  
liste des arg, p pour utiliser PATH, e pour env. En-  
têtes : **unistd.h**

**wait** *pid\_t wait(int \*status); pid\_t waitpid(pid\_t pid, int*  
*\*status, int options);* attend le chgt d'état d'un pus  
enfant. Entêtes : **sys/types.h, sys/wait.h**

**gestion de signal** *int sigaction(int signum, const struct*  
*sigaction \*act, struct sigaction \*oldact)* récupère et re-  
définit le gestionnaire de signal (fon), les drapeaux ...  
Entêtes : **signal.h**

**exit** *void exit(int status)* fin du pus et renvoie status &  
255 au pus parent (par wait). Entêtes : **stdlib.h**. Fin  
normale (0) sinon (>0).

**attendre un signal** *int pause(void);*

## 5 Chaines de caractères

Une chaîne est une séquence de char terminée par un  
'\0'. Le fichier **string.h** doit être inclus.

**longueur** *size\_t strlen(const char \*s)*

**copier** *char \*strcpy(char \*dest, const char \*src); char*  
*\*strncpy(char \*dest, const char \*src, size\_t n)* la dest  
doit être assez grande.

**copier dans le tas** *char \*strdup(const char \*s); char*  
*\*strndup(const char \*s, size\_t n)* ret une chaîne al-  
louée dynamiquement (malloc) qui est une copie de s;  
doit être libérée (free)!

**concaténer** *char \*strcat(char \*dest, const char \*src);*  
*char \*strncat(char \*dest, const char \*src, size\_t n)*

**comparer** *int strcmp(const char \*s1, const char \*s2); int*  
*strncmp(const char \*s1, const char \*s2, size\_t n)* ret  
nul si égal, négatif si s1<s2, positif sinon; pour accents  
et casse voir strcoll et strcasecmp.

**chercher un char** *char \*strchr(const char \*s, int c); char*  
*\*strrchr(const char \*s, int c)* ret un ptr sur la première  
(resp dernière) occurrence de c dans s, NULL si pas  
trouvé.

**chercher sous-chaîne** *char \*strstr(const char \*meule,*  
*const char \*aiguille)* ret un ptr sur la première occu-  
rence de aiguille dans meule, NULL si pas trouvé.

**chercher un des car** *char \*strpbrk(const char \*s, const*  
*char \*ensemble)* ret un ptr sur la première occurrence  
d'un char d'ensemble dans s, NULL si pas trouvé.

**découper en jeton** *char \*strtok(char \*s, const char \*se-*  
*par)* ret itérativement une sous-chaîne de s sans sépara-  
teur ou bien NULL. Une suite de séparateurs équivaut  
à un, les sép. de début et de fin sont ignorés. Modifie  
s!

## 6 Allocation et libération dyna- mique de mémoire

Entêtes : **stdlib.h**

**allocation** *void \*malloc(size\_t size)* ret un ptr sur une  
zone du tas de taille size octets; pas d'init.

**allocation init** *void \*calloc(size\_t nmemb, size\_t size)*  
ret un ptr sur une zone du tas de taille size\*nmemb  
octets; init. à 0!

**élargissement d'alloc** *void \*realloc(void \*ptr, size\_t*  
*size)* ret un ptr sur une (nouvelle) zone de taille size  
et dont le début contient ce qu'il y avait dans \*ptr; la  
suite n'est pas init.; l'ancienne zone est libérée auto.

**libération** *void free(void \*ptr)* libère la zone allouée par  
xxxalloc

## 7 Gestion des erreurs

Les appels systèmes et les fonctions de bibliothèque ne fonc-  
tionnant pas correctement retournent -1 ou NULL et posi-  
tionnent **errno**;

**errno** var globale entière indiquant le numéro d'erreur  
lorsque un appel système plante;

**message d'erreur** *char \*strerror (int errnum); void per-*  
*ror (const char \*s);* retourne ou affiche le msg d'erreur  
correspondant à errno.

## 8 Gestion des signaux

NSIG types de signaux émis lors d'un évènement syst ou  
explicitement;

**envoyer un signal** *int kill(pid\_t pid, int sig);*

**struct sigaction** *struct sigaction {*  
*void (\* sa\_handler)(int sig); // le gestionnaire*  
*sigset\_t sa\_mask; // signaux à bloquer pendant l'exécution*  
*int sa\_flags; // diverses options dont SA\_RESTART*  
*}*

**association signal gst** *int sigaction(int signum, const struct*  
*sigaction \*act, struct sigaction \*oldact);*

**Quelques signaux** **SIGSEGV, SIGINT, SIGTERM, SIG-**  
**KILL, SIGUSR1, SIGPIPE, SIGCHLD, SIGALRM, ...**

**alarme programmée** *unsigned int alarm(unsigned int*  
*nb\_sec);*

## 9 Tubes

**tube** *int pipe(int tube[2]);* retourne -1 si erreur

**tube nommé** *int mkfifo(const char \*pathname,*  
*mode\_t mode);* crée un tube nommé; rdv sur  
l'ouverture en lecture et en écriture;