RIEPILOGO

open

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

int open(const char *pathname, int oflag, ... /* ,
    mode_t mode */);
```

Restituisce un fd se OK, altrimenti -1

flag di stato: O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_CREAT, O_EXCL, O_TRUNC, O_NOCTTY, O_NONBLOCK, O_SYNC (SVR4)

creat

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat( const char *pathname, mode_t mode );
```

Restituisce un fd aperto come write-only se OK, altrimenti -1

close

```
#include <unistd.h>

int close( int filedescriptor );
```

Restituisce 0 se OK, altrimenti -1

lseek

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek( int filedes, off_t offset, int whence );
```

Restituisce il nuovo offset se OK, altrimenti -1

whence: SEEK_SET, SEEK_CUR, SEEK_END

read

```
#include <unistd.h>

ssize_t read(int filedes, void *buff, size_t nbytes );
```

Restituisce il numero di bytes letti,
0 se alla fine del file
altrimenti -1

write

```
#include <unistd.h>

ssize_t write( int filedes, const void *buff, size_t
   nbytes );
```

Restituisce il numero di bytes scritti se OK, altrimenti -1

dup & dup2

```
#include <unistd.h>

int dup( int filedes );

int dup2( int filedes, int filedes2 );
```

Restituiscono entrambe il nuovo fd se OK, altrimenti -1

fcntl

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

int fcntl (int filedes, int cmd, ... /* int arg */ );
```

Il valore restituito dipende da *cmd* se O.K., altrimenti –1 su errore

cmd: F_SETFD, F_GETFD, F_SETFD, F_GETFL, F_SETFL, F_GETLK, F_SETLK, F_GETOWN, F_SETOWN

Funzioni *stat, fstat e lstat*

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

int stat (const char *pathname, struct stat *buf);

int fstat (int fd, struct stat *buf);

int lstat (const char *pathname, struct stat *buf);
```

Restituiscono 0 se OK, -1 in caso di errore

struct *stat*

```
struct stat {
  mode_t  st_mode;    /* file type & mode (permissions) */
  ino_t   st_ino;     /* i-node number (serial number) */
  dev_t   st_dev;     /* device number (filesystem) */
  dev_t   st_rdev;    /* device number for special files */
  nlink_t st_nlink;   /* number of links */
  uid_t   st_uid;     /* user ID of owner */
  gid_t   st_gid;     /* group ID of owner */
  off_t   st_size;    /* size in bytes, for regular files */
  time_t  st_atime;   /* time of last access */
  time_t  st_mtime;   /* time of last modification */
  time_t  st_ctime;   /* time of last file status change */
  long    st_blksize; /* best I/O block size */
  long    st_blocks;  /* number of 512-byte blocks allocated */
};
```


Macro per tipi di file

Macro	Type of file
S_ISREG()	regular file
S_ISDIR()	directory file
S_ISCHR()	character special file
S_ISBLK()	block special file
S_ISFIFO()	pipe or FIFO
S_ISLNK()	symbolic link (not in POSIX.1 or SVR4)
S_ISSOCK()	socket (not in POSIX.1 or SVR4)

permessi di accesso ai file

st_mode mask	Meaning
S_IRUSR	user-read
S_IWUSR	user-write
S_IXUSR	user-execute
S_IRGRP	group-read
S_IWGRP	group-write
S_IXGRP	group-execute
S_IROTH	other-read
S_IWOTH	other-write
S_IXOTH	other-execute

Funzione *access*

#include <unistd.h>

int access (const char **pathname*, int *mode*);

Restituisce 0 se OK, -1 in caso di errore

mode	Description
R_OK	test for read permission
W_OK	test for write permission
X_OK	test for execute permission
F_OK	test for existence of file

Funzione *umask*

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

mode_t umask (mode_t *cmask*);

Restituisce la maschera di creazione precedente (nota che non restituisce valori di errori)

Funzioni *chmod e fchmod*

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

int chmod (const char ***pathname*, mode_t *mode*);

int fchmod (int *fd*, mode_t *mode*);

Restituiscono 0 se OK, -1 in caso di errore

Costanti per *chmod*

mode	Description	
S_ISUID	set-user-ID on execution	4000
S_ISGID	set-group-ID on execution	2000
S_ISVTX	saved-text (sticky bit)	1000
S_IRWXU	read, write, and execute by user (owner)	700
S_IRUSR	read by user (owner)	400
S_IWUSR	write by user (owner)	200
S_IXUSR	execute by user (owner)	100
S_IRWXG	read, write, and execute by group	070
S_IRGRP	read by group	040
S_IWGRP	write by group	020
S_IXGRP	execute by group	010
S_IRWXO	read, write, and execute by other (world)	007
S_IROTH	read by other (world)	004
S_IWOTH	write by other (world)	002
S_IXOTH	execute by other (world)	001

Funzioni *chown*

#include <sys/types.h>
#include <unistd.h>

int chown (const char ***pathname*, uid_t *owner*, gid_t *group*);

int fchown (int *fd*, uid_t *owner*, gid_t *group*);

int lchown (const char ***pathname*, uid_t *owner*, gid_t *group*);

Restituiscono 0 se OK, -1 in caso di errore

funzioni *link* e *unlink*

#include <unistd.h>

int link (const char **path*, const char **newpath*);

int unlink (const char **pathname*);

Restituiscono 0 se OK, -1 in caso di errore

funzioni *remove* e *rename*

#include <stdio.h>

int remove (const char **pathname*);

int rename (const char **oldname*, const char **newname*);

Restituiscono 0 se OK, -1 in caso di errore

funzioni *symlink* e *readlink*

#include <unistd.h>

int symlink (const char **path*, const char **sympath*);

int readlink (const char **pathname*, char **buf*, int **bufsize*);

Se OK la prima restituisce 0 e la seconda il numero di byte letti. Entrambe restituiscono -1 in caso di errore

funzione *utime*

#include <sys/types.h>

#include <utime.h>

int utime (const char **pathname*, const struct utimbuf **times*);

Restituisce 0 se OK, -1 in caso di errore

```
struct utimbuf {
        time_t  actime;  /*access time*/
        time_t  modtime;  /*modification
time*/
}
```


funzioni *mkdir* e *rmdir*

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

int mkdir (const char **pathname*, mode_t *mode*);

int rmdir (const char **pathname*);

Restituiscono 0 se OK, -1 in caso di errore

chdir, fchdir e *getcwd*

#include <unistd.h>

int chdir (const char **pathname*);

int fchdir (int *fd*);

char *getcwd (char **buf*, size_t *size*);

Le prime 2 restituiscono 0 se OK e -1 in caso di errore

la terza restituisce *buf* se OK e NULL in caso di errore

funzioni *sync* e *fsync*

```
#include <unistd.h>

void sync(void);
int fsync(int filedes);
```

restituiscono 0 se O.K., altrimenti -1

modifica del *buffering*

```
#include <stdio.h>

void setbuf(FILE *fp, char *buf );
int setvbuf(FILE *fp, char *buf, int mode, size_t
  size);
```

Restituiscono 0 se OK, !=0 in caso di errore

parametri

Function	*mode*	*buf*	Buffer & length	Type of buffering
setbuf		nonnull	user *buf* of length BUFSIZ	fully buffered or line buffered
		NULL	(no buffer)	unbuffered
setvbuf	_IOFBF	nonnull	user *buf* of length *size*	fully buffered
		NULL	system buffer of appropriate length	
	_IOLBF	nonnull	user *buf* of length *size*	line buffered
		NULL	system buffer of appropriate length	
	_IONBF	(ignored)	(no buffer)	unbuffered

funzione *fflush*

```
#include <stdio.h>

int fflush(FILE *fp);
```

Restituisce 0 se OK, EOF in caso di errore

aprire uno *stream*

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

FILE *fopen(const char *pathname, cons char *type);

FILE *freopen(const char *pathname, cons char *type, FILE *fp);

FILE *fdopen(int fd, cons char *type);

Restituiscono un puntatore a file se OK,
NULL in caso di errore

tipi

type	Description
r or rb	open for reading
w or wb	truncate to 0 length or create for writing
a or ab	append; open for writing at end of file, or create for writing
r+ or r+b or rb+	open for reading and writing
w+ or w+b or wb+	truncate to 0 length or create for reading and writing
a+ or a+b or ab+	open or create for reading and writing at end of file

funzione *fclose*

```
#include <stdio.h>

int fclose(FILE *fp);
```

Restituisce 0 se OK, EOF in caso di errore

input di un carattere

```
#include <stdio.h>

int getc(FILE *fp);
int fgetc(FILE *fp);
int getchar(void);
```

Restituiscono il prossimo carattere se OK, EOF se alla fine del file o in caso di errore

funzioni *ferror* e *feof*

```
#include <stdio.h>

int ferror(FILE *fp);
int feof(FILE *fp);
```

Restituiscono true se la condizione è vera, false altrimenti

per resettare emtrambi i flag c'è la funzione:

```
void clearerr(FILE *fp);
```


funzione *ungetc*

```
#include <stdio.h>

int ungetc(int c, FILE *fp);
```

Restituisce *c* se OK, EOF in caso di errore

ouput di un carattere

```
#include <stdio.h>

int putc(int c, FILE *fp);
int fputc(int c, FILE *fp);
int putchar(int c);
```

Restituiscono *c* se OK, EOF in caso di errore

input di una linea

```
#include <stdio.h>

char *fgets(char *buf, int size, FILE *fp);
                                /* null byte terminated*/
char *gets(char *buf);  /* da non utilizzare */
```

Restituiscono *buf* se OK, NULL se alla fine del file o in caso di errore

ouput di una linea

```
#include <stdio.h>

int fputs(const char *str, FILE *fp); /* null byte non
                                         scritto */
int puts(const char *str); /* appende un newline... meglio
                              non utilizzarla */
```

Restituiscono un valore non negativo se OK, EOF in caso di errore

I/O diretto (binario)

```
#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nobj, FILE
   *fp);
size_t fwrite(const void *ptr, size_t size, size_t
   nobj, FILE *fp);
```

Restituiscono il numero di oggetti letti o scritti

input formattato

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *fp, const char *format, ...);
int sscanf(const char *buf, const char *format, ...);
```

Restituiscono il numero di oggetti assegnati se OK, EOF se alla fine del file o in caso di errore

ouput formattato

```
#include <stdio.h>

int printf(const char *format, ...);

int fprintf(FILE *fp, const char *format, ...);

int sprintf(char *buf, const char *format, ...);
```


posizionamento in uno stream

```
#include <stdio.h>
long ftell(FILE *fp);
```

restituisce l'indicatore della posizione corrente (misurato in byte) se O.K., -1L su errore.

```
long fseek(FILE *fp, long offset, int whence);
                                  /* simile a lseek */
```

restituisce 0 se O.K., diverso da 0 su errore.

```
void rewind(FILE *fp);
```


ancora posizionamento

```
int fgetpos(FILE *fp, fpos_t *pos);
int fsetpos(FILE *fp, const fpos_t *pos);
```

restituiscono 0 se O.K., diverso da 0 su errore

file descriptor

```
#include <stdio.h>

int fileno(FILE *fp);
```

restituisce il file descriptor associato allo stream

Funzioni di exit

```
#include <stdlib.h>
   void exit (int status);

#include <unistd.h>
   void _exit (int status);

#include <stdlib.h>
int atexit (void (*funzione) (void));
```


Variabili di Ambiente

```
#include <stdlib.h>

char * getenv (const char *name);

int putenv (const char *str);

int setenv (const char *name, const char *value, int
   rewrite);
```


Identificatori di processi

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid (void);
pid_t getppid (void);


uid_t getuid (void);
uid_t geteuid (void);

gid_t getgid (void);
gid_t getegid (void);
```


Funzione *fork*

```
#include <sys/types>
#include <unistd.h>

pid_t fork(void);
```

Restituisce 0 nel figlio, PID del figlio nel padre e -1 in caso di errore

Funzioni *wait e waitpid*

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait (int *statloc);

pid_t waitpid (pid_t pid, int *statloc, int
   options);
```

Restituiscono PID se OK, oppure -1 in caso di errore

Funzioni *exec*

#include < unistd.h>

int exec**l** (const char ***path*, const char ***arg0*, ../* (char *) 0 */);
int exec**v** (const char ***path*, char *const *argv*[ ]);

int exec**lee** (const char ***path*, const char ***arg0*, ../*(char *) 0, char *const *envp*[ ] */);
int exec**vee** (const char ***path*, char *const *argv*[ ], char *const *envp*[ ]);

int exec**lpp** (const char ***file*, const char ***arg0*, ../*(char *)0 */);
int exec**vpp** (const char ***file*, char *const *argv*[ ]);

Restituiscono -1 in caso di errore e non ritornano se OK.

Funzione *system*

#include <stdlib.h>

int system (const char ***cmdstring*);

Segnali in un sistema Unix

Name	Description	ANSI C	POSIX.1	SVR4	4.3+BSD	Default action
SIGABRT	abnormal termination (abort)	•	•	•	•	terminate w/ core
SIGALRM	time out (alarm)		•	•	•	terminate
SIGBUS	hardware fault			•	•	terminate w/ core
SIGCHLD	change in status of child		job	•	•	ignore
SIGCONT	continue stopped process		job	•	•	continue/ignore
SIGEMT	hardware fault			•	•	terminate w/ core
SIGFPE	arithmetic exception	•	•	•	•	terminate w/ core
SIGHUP	hangup		•	•	•	terminate
SIGILL	illegal hardware instruction	•	•	•	•	terminate w/ core
SIGINFO	status request from keyboard				•	ignore
SIGINT	terminal interrupt character	•	•	•	•	terminate
SIGIO	asynchronous I/O			•	•	terminate/ignore
SIGIOT	hardware fault			•	•	terminate w/ core
SIGKILL	termination		•	•	•	terminate
SIGPIPE	write to pipe with no readers		•	•	•	terminate
SIGPOLL	pollable event (poll)			•		terminate
SIGPROF	profiling time alarm (setitimer)			•	•	terminate
SIGPWR	power fail/restart					ignore
SIGQUIT	terminal quit character		•	•	•	terminate w/ core
SIGSEGV	invalid memory reference	•	•	•	•	terminate w/ core
SIGSTOP	stop		job	•	•	stop process
SIGSYS	invalid system call			•	•	terminate w/ core
SIGTERM	termination	•	•	•	•	terminate
SIGTRAP	hardware fault			•	•	terminate w/ core
SIGTSTP	terminal stop character		job	•	•	stop process
SIGTTIN	background read from control tty		job	•	•	stop process
SIGTTOU	background write to control tty		job	•	•	stop process
SIGURG	urgent condition			•	•	ignore
SIGUSR1	user-defined signal		•	•	•	terminate
SIGUSR2	user-defined signal		•	•	•	terminate
SIGVTALRM	virtual time alarm (setitimer)			•	•	terminate
SIGWINCH	terminal window size change			•	•	ignore
SIGXCPU	CPU limit exceeded (setrlimit)			•	•	terminate w/ core
SIGXFSZ	file size limit exceeded (setrlimit)			•	•	terminate w/ core

Funzione *signal*

#include <signal.h>

void (*signal(int *signo*, void (***func*)(int)))(int);

Restituisce SIG_ERR in caso di errore e il puntatore al precedente gestore del segnale se OK

Funzioni *kill* e *raise* (1)

#include <sys/types.h>
#include <signal.h>

int kill (pid_t *pid*, int *signo*);
int raise (int *signo*);

Restituiscono 0 se OK, -1 in caso di errore

Funzioni *pause e sleep*

#include <unistd.h>

int pause(void);

unsigned int sleep (unsigned int *secs*);

Funzione *pipe*

```
#include <unistd.h>

int pipe(int filedes[2] );
```

Restituisce 0 se OK, -1 in caso di errore

Attraverso l'argomento *filedes* ristituisce 2 file descriptor

popen e pclose

```
#include <stdio.h>

FILE *popen(const char *cmdstring, const char *type)
```

restituisce il file pointer se O.K., NULL su errore

```
int pclose(FILE *fp)
```

restituisce lo stato di terminazione di *cmdstring*, -1 su errore

FIFO (named pipes)

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

int mkfifo(const char *pathname, mode_t mode);
```

Restituisce 0 se OK, -1 in caso di errore

struttura **msqid_ds**

- ogni coda di messaggi ha la struttura

  msqid_ds

  associata ad essa che ne definisce lo stato corrente

```
struct msqid_ds {
  struct ipc_perm  msg_perm; /* see Section 14.6.2 */
  struct msg  *msg_first;  /* ptr to first message on queue */
  struct msg  *msg_last;   /* ptr to last message on queue */
  ulong        msg_cbytes; /* current # bytes on queue */
  ulong        msg_qnum;   /* # of messages on queue */
  ulong        msg_qbytes; /* max # of bytes on queue */
  pid_t        msg_lspid;  /* pid of last msgsnd() */
  pid_t        msg_lrpid;  /* pid of last msgrcv() */
  time_t       msg_stime;  /* last-msgsnd() time */
  time_t       msg_rtime;  /* last-msgrcv() time */
  time_t       msg_ctime;  /* last-change time */
};
```


Code di Messaggi

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key_t key, int flag);
```

- Crea o apre una coda di messaggi (in relazione a *key*)
- Restituisce ID della coda se OK, -1 in caso di errore

Funzione *msgctl*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf );
```

Restituisce 0 se OK, -1 in caso di errore

cmd = IPC_STAT, IPC_SET, IPC_RMID

Funzione *msgsnd*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

int msgsnd(int *msqid*, const void **ptr*, size_t *nbytes*, int *flag*);

Restituisce 0 se OK, -1 in caso di errore

flag può essere IPC_NOWAIT

Funzione *msgrcv*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

int msgrcv(int *msqid*, void **ptr*, size_t *nbytes*, long *type*, int *flag*);

Restituisce la dimensione del campo mtext se OK, -1 in caso di errore

flag = IPC_NOWAIT, MSG_NOERROR

Strutture associate ai semafori

```
struct semid_ds {
  struct ipc_perm  sem_perm;  /* see Section 14.6.2 */
  struct sem  *sem_base;  /* ptr to first semaphore in set */
  ushort       sem_nsems; /* # of semaphores in set */
  time_t       sem_otime; /* last-semop() time */
  time_t       sem_ctime; /* last-change time */
};
```

```
struct sem {
  ushort  semval;   /* semaphore value, always >= 0 */
  pid_t   sempid;   /* pid for last operation */
  ushort  semncnt;  /* # processes awaiting semval > currval */
  ushort  semzcnt;  /* # processes awaiting semval = 0 */
};
```


Funzione semget

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

int semget(key_t *key*, int *nsems*, int *flag*);

Restituisce ID del semaforo se OK, -1 in caso di errore

Funzione *semctl*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

int *semctl*(int *semid*, int *semnum*, int *cmd*, union semun *arg*);

cmd = IPC_STAT, IPC_SET, ipc_RMID, GETVAL, SETVAL, GETPID, GETNCNT, GETZNCT, GETALL, SETALL

```
union semun{
int         val;
struct semid_ds     *buf;
ushort      *array;}
```


Funzione *semop*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

int semop(int *semid*, struct sembuf *semoparray[ ]*, size_t *nops*);

Restituisce 0 se OK, -1 in caso di errore

```
struct sembuf {
  ushort  sem_num;  /* member # in set (0, 1, ..., nsems-1) */
  short   sem_op;   /* operation (negative, 0, or positive) */
  short   sem_flg;  /* IPC_NOWAIT, SEM_UNDO */
};
```

Struttura associata alla memoria condivisa

```
struct shmid_ds {
  struct ipc_perm  shm_perm; /* see Section 14.6.2 */
  struct anon_map *shm_amp;  /* pointer in kernel */
  int      shm_segsz;   /* size of segment in bytes */
  ushort   shm_lkcnt;   /* number of times segment is being locked */
  pid_t    shm_lpid;    /* pid of last shmop() */
  pid_t    shm_cpid;    /* pid of creator */
  ulong    shm_nattch;  /* number of current attaches */
  ulong    shm_cnattch; /* used only for shminfo */
  time_t   shm_atime;   /* last-attach time */
  time_t   shm_dtime;   /* last-detach time */
  time_t   shm_ctime;   /* last-change time */
};
```


shmget

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

int shmget(key_t *key*, int *size*, int *flag*);

Restituisce ID del segmento di memoria condivisa se OK, -1 in caso di errore

Funzione *shmctl*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

int shmctl(int *shmid*, int *cmd*, struct shmid_ds **buf*);

Restituisce 0 se OK, -1 in caso di errore

cmd= IPC_STAT, IPC_SET, IPC_RMID, SHM_LOCK, SHM_UNLOCK

Funzione *shmat*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

void *shmat(int *shmid*, void **addr*, int *flag*);

flag può essere SHM_RDONLY

Restituisce il puntatore al segmento se OK, -1 in caso di errore

Funzione *shmdt*

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

int shmdt(void **addr*);

Restituisce 0 se OK, -1 in caso di errore