Programación UNIX

1) Nociones básicas:

int main (int argc, char ** argv) { ... }

Para saber si una llamada al sistema ha producido algún error, se ha de comprobar si su valor devuelto es -1. Para saber cual error es, tenemos la variable externa **errno**, que nos dará el número de error. Con este en **error.h**, sabremos la descripción del error usando:

char sys_errlist[sys_nerr];
sys_nerr = Nº total de errores del sistema posible.

Tambíen se puede usar, para mostrarlo directamente por pantalla:

stdio.h → void perror (char * msg);
perror("mensaje"); → "mensaje: descripción del error"

__FILE__ → Nombre del fichero.
__LINE__ → Nº de linea.

2) El sistema de ficheros:

a) Arquitectura:

Estructura → sector de arranque / superbloque / lista de i-nodos / bloques de datos //

+ Superbloque → sys/filsys.h
  (Tamaño del Sistema de Ficheros, Lista de bloques libres, Indice del siguiente bloque libre de datos en la lista de bloques libres de datos, Tamaño de la lista de i-nodos, Total de i-nodos libres, Lista de i-nodos libres, Indice del siguiente i-nodo libre en la lista, Campo de bloqueo, Flag de modificación.)
+ I-nodo → sys/ino.h
  (UID, GID, Tipo (ordinario, directorio, especial de dispositivo (de caracter / de bloque), tubería), Permisos, Fechas (creación, modificación, acceso), Tamaño, Bloques de dirección.)

b) Ficheros ordinarios:

+ **open** → Abre (e incluso puede crear) un fichero para su manejo.
  sys/types.h, sys/stat.h, fcntl.h
  int open (char * ruta, int flag [, mode_t modo]);
  - ruta = ruta absoluta o relativa del fichero, de longitud máxima PATH_MAX.
  - flag = O_RDONLY (solo lectura), O_WRONLY (solo escritura), O_RDWR (L/E), O_NDELAY (indicador especial, ver pagina 54 del libro), O_APPEND (añadir), O_CREAT (crear), O_EXCL (junto a O_CREAT, da un error si existe), O_TRUNC (borra el contenido).
  - modo = permisos en octal (0600 = rw-------, 0666 = rw-rw-rw-).
  Devuelve un entero que nos permitirá acceder al descriptor del fichero.
+ **read** → Lee bytes de un fichero.
  unistd.h
  int read (int manejador, char * buf, unsigned nbyte);
  - manejador = Nº de descriptor del fichero.
  - buf = buffer donde se almacenarán los bytes leidos.

- nbytes = Nº de bytes a leer.
Devuelve el Nº de bytes leidos realmente.

+ **write** → Escribe bytes en un fichero
unistd.h
int write (int manejador, char * buf, unsigned nbyte);
- manejador = Nº de descriptor del fichero.
- buf = buffer donde están almacenados los bytes a escribir.
- nbytes = Nº de bytes a escribir.
Devuelve el Nº de bytes escritos realmente.

+ **close** → Cierra un fichero.
unistd.h
int close (int manejador);
Devuelve 0 si todo va bien.

+ **creat** → Crea un fichero, para escribir y vacío.
fcntl.h
int creat (char * path, mode_t mode);
Devuelve el Nº de descriptor del fichero.

+ **dup** → Duplica un descriptor.
unistd.h
int dup (int manejador);
Devuelve el nuevo Nº de descriptor del fichero.

+ **lseek** → Modifica la posición del puntero de L/E.
sys/types.h, unistd.h
off_t lseek (int manejador, off_t offset, int whence);
- offset = desplazamiento.
- whence = SEEK_SET (desplazamiento desde el inicio del fichero), SEEK_CUR (desplazamiento desde la posición actual), SEEK_END (desplazamiento desde el final del fichero).
Devuelve la nueva posición del puntero de L/E.

+ **fsync** → Actualiza el contenido del fichero con el buffer actual.
unistd.h
int fsync (int manejador);

+ **<u>Control de ficheros abiertos</u>** → fcntl
sys/types.h, unistd.h, fcntl.h
int fcntl (int manejador, int cmd, union { intval; struct flock * lockdes; } arg);
- cmd = comando
  + F_DUPFD = Devuelve un descriptor libre >= arg.val, copiando el contenido del actual.
  + F_GETFD = Se realiza la comparación: EAX & 1
    0 = El fichero se cerrará tras ejecutar la llamada exec.
    1 = El fichero no se cerrará tras ejecutar la llamada exec.
    (close-on-exec)
  + F_SETFD = Fija el indicador close-on-exec con arg.val = {0, 1}.
  + F_GETFL = Devuelve los flags de apertura del fichero..
  + F_SETFL = Cambia los flags con arg.val.
  + F_GETLK = Devuelve el primer cerrojo en arg.lockdes. Si no hay ninguno arg.lockdes.l_type = F_UNLCK.
  + F_SETLK = Mete un cerrojo al fichero con arg.lockdes.
  + F_SETLKW = Como F_SETLK, pero si no lo puede poner duerme el proceso hasta lograrlo.

```
struct flock {
        short l_type;       // Tipo de cerrojo.
                            // F_RDLCK = lectura.
                            // F_WRLCK =  escritura.
                            // F_UNLCK = eliminar el cerrojo.
        short l_whence;     // Inicio de la región a bloquear.
                            // SEEK_SET = principio.
                            // SEEK_CUR = actual.
                            // SEEK_END = final.
        off_t l_stat;       // Offset de inicio de la región a bloquear, con
                            // base a l_whence.
        off_t l_len;        // Tamaño de la región. 0 = Hasta el final.
        off_t l_pid;        // PID del proceso que fijo el cerrojo.
        long l_sysid;       // ID del sistema que fijó el cerrojo.
}
```

+ **<u>Adminstración de ficheros</u>**
  + **stat, lstat, fstat** → Devuelven la información del i-nodo de un fichero.
    sys/types.h, sys/stat.h
    int stat (char * path, struct stat * buf);
    int lstat (char * path, struct stat * buf);
    int fstat (int manejador, struct stat * buf);
    <u>struct stat</u>:
    + dev_t st_dev = Nº de dispositivo que contiene el i-nodo.
    + ino_t st_ino = Nº de inodo.
    + ushort st_mode = Modo del fichero:
        + Bits 0-2: Permiso para los otros usuarios.
        + Bits 3-5: Permiso para los usuarios del mismo grupo.
        + Bits 6-8: Permiso para el usuario creador del fichero.
        + Bit 9: Stiky bit.
        + Bit 10: Cambiar el GID al ejecutar.
        + Bit 11: Cambiar el UID al ejecutar.
        + Bit 12-15: Tipo de fichero.
            + 1000 = Ordinario
            + 0100 = Directorio
            + 0010 = Dispositivo de caracteres
            + 0110 = Dispositivo de bloque
            + 0001 = Tubería
            + 1010 = Enlace simbólico
            + 1100 = Conector
    + ushort st_nlink = Nº de enlaces al fichero.
    + uid_t st_uid = UID del propietario.
    + gid_t st_gid = GID del propietario.
    + dev_t st_rdev = ID de dispositivo (solo para ficheros especiales).
    + off_t st_size = Tamaño del fichero.
    + time_t st_atime = Fecha del último acceso.
    + time_t st_mtime = Fecha de la última modificación.
    + time_t st_ctime = Fecha de creación.
    <u>Modos</u>: if((mode & constante) == constante) { ... }
    S_IFMT 0170000 Tipo de fichero

S_IFREG	0100000	Ordinario
S_IFDIR	0040000	Directorio
S_IFCHR	0020000	Dispositivo de caracteres
S_IFBLK	0060000	Dispositivo de bloques
S_IFIFO	0010000	Tuberías
S_IFSOCK	0140000	Conector
S_ISUID	0004000	Activar UID al ejecutar
S_ISGID	0002000	Activar GID al ejecutar
S_ISVTX	0001000	Stiky bit (varios procesos compartirán el mismo segmento de código y este permanecerá siempre en memoria principal).

+ **chmod, fchmod** → Cambia los permisos de un fichero.
  sys/types.h, sys/stat.h
  int chmod (char * path, mode_t modo);
  int fchmod (int manejador, mode_t modo);
+ **access** → Determina la accesibilidad de un fichero por parte de un proceso.
  unistd.h
  int access (char * path, int amode);
  - amode = tipo de acceso → R_OK, W_OK, X_OK.
+ **umask** → Establece una máscara de permisos que no se podrán poner al crear un fichero.
  sys/types.h, sys/stat.h
  mode_t umask (mode_t cmask);
  - cmask = Nueva máscara. Ejemplo 0066 → Si queremos crear un fichero con el permiso 0666, se nos quedará al final en 0600.
  Devuelve la máscara anterior.
+ **rename** → Cambia el nombre del fichero.
  stdio.h
  int rename (const char * fichero, const char * nuevonombre);
  Devuelve 0 si todo fue bien.
+ **chown, fchown** → Cambian de propietario y de grupo el fichero.
  sys/types.h, sys/stat.h
  int chown (char * path, uid_t owner, gid_t group);
  int fchown (int handler, uid_t owner, gid_t group);
  Para no alterar alguno de los dos campos se ha de usar UID_NO_CHANGE o GID_NO_CHANGE. Devuelve 0 si todo va bien.
+ **utime** → Cambia la fecha de acceso y modificación.
  sys/types.h, utime.h
  int utime (char * path, struct utimebuf * times);
  struct utimebuf
  + time_t actime; // Fecha de acceso.
  + time_t modtime; // Fecha de modificación.
+ **truncate, ftruncate** → Reduce el tamaño del fichero.
  int truncate (char * path, unsigned long length);
  int ftruncate (int manejador, unsigned long length);
+ **Información del usuario y el grupo**
  pwd.h
  struct passwd * getpwuid (uid_t uid);
  struct passwd:
  - pw_name = Nombre del usuario.

grp.h
struct group * getgrgid (gid_t gid);
struct group:
- gr_name = Nombre del grupo.

Otras funciones:
getpwent, setpwent, endpwent, getpwnam. /etc/passwd
getgrent, setgrent, endgrent, getgrnam. /etc/group

+ **lockf** → Comparte y bloquea ficheros de forma obligatoria.
unistd.h
int lockf (int handler, int function, long size);
- function = Tipo de acción.
F_ULOCK = Desbloquear región bloqueada.
F_LOCK = Bloquear región, sino dormir hasta conseguirlo.
F_TLOCK = Bloquear región, sino devuelve -1, en vez de dormir.
F_TEST = Comprueba si está bloqueada una región (0 = no / -1 = sí).

c) <u>Directorios y ficheros especiales</u>:

+ **<u>Acceso a directorios</u>**

+ **mknod, mkdir** → Crean un directorio.
sys/types.h, sys/stat.h
int mknod (char * path, mode_t mode, int dev);
int mkdir (char * path, mode_t mode);
- mode = Tipo de fichero y permisos.
- dev = Nº de dispositivo, solo para archivos especiales.

+ **rmdir** → Borra un directorio.
unistd.h
int rmdir (char * path);

+ **link** → Crea un nuevo enlace a un i-nodo.
unistd.h
int symlink (char * viejo, char * nuevoenlacesimbolico);
int link (char * viejo, char * nuevo enlace);

+ **unlink** → Borra un enlace a un i-nodo. Cuando se borran todos los enlaces, se borra el fichero, quedando libre el i-nodo.
unistd.h
int unlink (char * path);

+ **chdir** → Cambia el directorio actual de trabajo.
unistd.h
int chdir (char * path);
int fchdir (int handler);

+ **getcwd** → Devuelve el directorio actual.
unistd.h
char * getcwd (char * buf, int size);

+ **chroot** → Cambia el directorio raíz del proceso (solo la puede llamar root).
unistd.h
int chroot (char * path);

+ **opendir** → Abre un directorio.
sys/types.h, dirent.h
DIR * opendir (char * dirpath);
DIR:
- int dd_fd;

- long dd_loc;
- long dd_size;
- long dd_bbase;
- long dd_entno;
- long dd_bsize;
- char * dd_buf;

Devuelve NULL en caso de error.

+ **readdir** → Lee la siguiente entrada del directorio.

sys/types.h, dirent.h
struct dirent * readdir (DIR * dirp);
struct dirent:
- ino_t d_ino; // i-nodo asociado.
- short d_reclen; // Longitud de la entrada.
- short d_namlen; // Longitud de la cadena d_name.
- char d_name[_MAXNAMLEN + 1]; // Nombre del fichero.

Cuando llega al final devuelve NULL.
(Nota: Cuando no funciones _MAXNAMLEN, probar con MAXNAMLEN.)

+ **closedir** → Cierra un directorio.

sys/types.h, dirent.h
int closedir (DIR * dirp);

+ **seekdir, telldir, rewinddir**

sys/types.h, dirent.h
void seekdir (DIR * dirp, long posnueva); // Manda a pos al puntero de lectura.
long telldir (DIR * dirp); // Da la posición actual del puntero de lectura.
void rewinddir (DIR * dirp); // Situa al inicio el puntero de lectura.

+ **<u>Acceso a ficheros especiales</u>**

+ **E/S sobre terminales**

close(0); open("/dev/tty", O_RDONLY); // Manejador 0 = Entrada.
close(1); open("/dev/tty", O_WRONLY); // Manejador 1 = Salida.
close(2); open("/dev/tty", O_WRONLY); // Manejador 2 = Salida error.

sys/types.h, sys/utmp.h
struct utmp * getutent (); // Devuelve la siguiente entrada de /etc/utmp, o
// NULL si llega al final.
struct utmp:
- char ut_user[8]; // Nombre del usuario.
- char ut_id[4]; // ID de /etc/inittab.
- char ut_line[12]; // Nombre del fichero dispositivo asociado (console,
// tty##, ln##, etc...)
- pid_t ut_pid; // ID del proceso.
- short ut_type; // Tipo entrada: EMPTY, RUN_LVL, BOOT_TIME,
// OLD_TIME, NEW_TIME, INIT_PROCESS,
// LOGIN_PROCESS, USER_PROCESS,
// DEAD_PROCESS, ACCOUNTING.
- struct exit_status:
  - short e_terminatio; // Estado de terminación del proceso.
  - short e_exit; // Estado de salida del proceso.
- unsigned short ut_reserved1;
- char ut_host[16]; // Nombre del ordenador.
- unsigned long ut_addr; // Dirección ip del ordenador.

+ **Control de terminales**

sys/ioctl.h

int ioctl (int handler, int request, arg);

(Ver sección 7 del manual de UNIX.)

Ejemplo para manejar la entrada (termios.h):

```
char mem[256], c;
int i = 0;
struct termios old, param;
ioctl(0, TIOCGETA, &old); // Leer la cfg de la entrada.
param = old;
param.c_lflag &= ~(ICANON | ECHO); // Quitar entrada canónica y con eco.
param.c_cc[4] = 1; // Devolver el control tras leer un caracter.
ioctl(0, TIOCSETA, &param);
do {
    read(0, &c, 1);
    if(c == 128) { // Backspace
        if(i > 0) {
            i--;
            putchar('\b');
            putchar(' ');
            putchar('\b');
        }
    } else {
        mem[i++] = c;
        putchar(c);
    }
    fflush(stdout);
} while(i < 255 && c != '\n');
mem[i] = 0;
ioctl(0, TIOCSETA, &old);
// Ver termio (7) para terminarles asíncronas.
```

+ **Otras funciones**

sys/types.h, sys/stat.h

int mknod (char * path, mode_t mode, int dev);

Para crear ficheros especiales es necesario emplear mknod.

- mode: (Ver función stat.) Tipo de fichero y permisos. Ej:
  mknod("pepe", S_IFDIR | 0600, 0);
- dev: Para los ficheros especiales hay que indicarlo. Existen tres funciones para manejar variables de tipo dev_t:
  + dev_t makedev (major_t Bmayor, minor_t Bmenor);
  + major_t major (dev_t dev);
  + minor_t minor (dev_t dev);

+ <u>Administración del sistema de ficheros</u>

+ **mount** → Monta un sistema de ficheros.

  int mount (char * spec, char * dir, int rwflag);

  - spec: Ruta del dispositivo que vamos a montar ("/dev/cdrom").
  - dir: Ruta del directorio sobre el que montar el dispositivo ("/mnt/cdrom").
  - rwflag: (1 = escritura prohibida).

  Devuelve 0 si todo va bien.

+ **umount** → Desmonta un sistema de ficheros.

int umount (char * dir);

+ **sync** → Sincroniza la memoria y el disco duro, para dar consistencia.
  unistd.h
  void sync (void);
+ **ustat** → Estado del sistema de ficheros.
  sys/types.h, ustat.h
  int ustat (dev_t dev, struct ustat * buf);
  - buf:
    - daddr_t f_tfree; // Nº de bloques libres.
    - ino_t f_tinode; // Nº de inodos libres.
    - char f_fname[6]; // Nombre del sistema de ficheros.
    - char f_fpack[6]; // Nombre del paquete del sistema de ficheros.
+ **Otras** → setmntent, getmntent, endmntent, statfs, fstatfs.

d) <u>Funciones de manejo de la memoria</u>: (string.h)

+ **memmove** → Copia una región de memoria.
  void * memmove (void * dest, const void * orig, size_t tam);
+ **memcpy** → Copia una región de memoria.
  void * memcpy (void * dest, const void * orig, size_t tam);
+ **memccpy** → Copia una región de memoria, hasta encontrarse con el caracter c.
  void * memccpy (void * dest, const void * orig, int c, size_t tam);
+ **memset** → Rellena una región con un valor dado.
  void * memset (void * region, int valor, size_t n);

e) <u>Manejo de cadenas</u>:

+ **Salida con formato:**
  int printf (const char * formato[, argumentos]);
  int fprintf (FILE * f, const char * formato[, argumentos]);
  int sprintf (char * cad, const char * formato[, argumentos]);
  - formato: %[flags][ancho][.precisión][{h | l | L }] tipo
  - flags:

-	Alineación a la izquierda (por defecto a la derecha)
+	Pone el signo + para los positivos
0	Rellena con 0 el ancho
#	Muestra 0, 0x, con o, x, X. Con los decimales fuerza a poner el punto.
h	short
l	long, double
L	long, double

  - tipo:

d	int	decimal	i	int	decimal
u	uint	decimal	o	uint	octal
x	uint	hexadecimal	X	uint	hexadecimal (en mayúsculas)
f	float		e	double	
E	double		c	char/int	
s	char *, '\0'		g, G	el más compacto entre f o e y E	

  %p = void * %n = int * %% = Escribe '%'

+ **Entrada con formato**:
  int scanf (const char * formato[, argumentos]);
  int fscanf (FILE * f, const char * formato[, argumentos]);
  int sscanf (char * cad, const char * formato[, argumentos]);

- formato: %[*][ancho][{h | l }] tipo
  * = Suprime la entrada del dato.

+ **Funciones de manipulación:**

```
char * strcat (char * cad, char * cad2);
        cad += cad2;

char * strcpy (char * cad, char * cad2);
        cad = cad2;

char * strchr (const char * cad, int c);
char * strrchr (const char * cad, int c);
        Devuelve un puntero al 1er char igual que c (strrchr también admite '\0').

size_t strlen (char * cad);
        Tamaño de cad (uint).

int strcmp (const char * cad, const char * cad2);
        < 0             cad < cad2
        = 0             cad = cad2
        > 0             cad > cad2

size_t strcspn (const char * cad, const char * patrón);
        "patrón" = conjunto de chars a buscar en cad.

char * strncat (char * cad, const char * cad2, size_t n);
char * strncpy (char * cad, const char * cad2, size_t n);
char * strncmp (char * cad, const char * cad2, size_t n);

size_t strspn (const char * cad, const char * patron);
        Busca en cad el patrón y da su posición.

char * strstr (const char * cad, const char * cad2);
        Busca cad2 en cad y da un puntero a su posición.

char * strtok (char * cad, const char * cad2);

char * strlwr (char * cad);
        Pasa cad a minúsculas.

char * strupr (char * cad);
        Pasa cad a mayúsculas.
```

+ **Conversión entre datos:** (stdlib.h)

```
double      atof        (const char * cad);
int         atoi        (const char * cad);
long        atol        (const char * cad);
int         toascii     (int c);                // = c & 0x00FF (ctype.h)
int         tolower     (int c);
int         toupper     (int c);
```

3) Procesos e hilos:

a) Gestión de procesos e hilos:

+ **Ejecución de programas:** (unistd.h)

int execl (char * ruta, char * arg0, ..., char * argN, NULL);
int execv (char * ruta, char * argv[]);
int execle (char * ruta, char * arg0, ..., char * argN, NULL, char * envp[]);
int execve (char * ruta, char * argv[], char * envp[]);
int execlp (char * comando, char * arg0, ..., char * argN, NULL);
int execvp (char * comando, char * argv[]);
- comando = nombre del ejecutable, que será buscado en los directorios de PATH.
  int main (int argc, char * argv[, char * envp]) { ... }

exec carga el programa, encima del que le llamara, destruyendo al proceso llamante. Por lo que al terminar el programa, no se vuelve al proceso llamante.

+ **Creación de procesos:** (unistd.h, sys/types.h)

pid_t fork (void);
Devuelve -1 en caso de error. Pero si todo va bien, el hijo recibe 0, el padre recibe el PID del hijo y ambos siguen su ejecución tras el fork().

+ **Terminación de procesos:**

stdlib.h → void exit (int estado);
Termina la ejecución de un proceso, retornando "estado" al proceso que lo invocó.

sys/types.h, sys/wait.h → pid_t wait (int * estado);
Suspende la ejecución del proceso, hasta que muere alguno de sus hijos. Hay una serie de macros para comprobar el valor de estado. Devuelven cierto cuando:
- WIFEXITED → Termina con la llamada exit.
- WEXITSTATUS → Si la anterior es cierta, esta da los 8 bits menos significativos.
- WIFSIGNALED → Terminado con laguna señal.
- WTERMSIG → Valor de la señal.
- WCOREDUMP → Terminado con un core dump.
- WIFSTOPPED → El proceso está parado.
- WSTOPSIG → Nº de señal que paró el proceso.

+ **Información sobre procesos:** (sys/types.h)

```
pid_t getpid (void);        // Da el PID del proceso actual.
pid_t getppid (void);       // Da el PID del padre del proceso actual.
pid_t getpgrp (void);       // Da el PID del grupo.
pid_t setpgrp (void);       // Cambia el PID de grupo del proceso actual,
                            // por el propio PID.
uid_t getuid (void);        // Da el UID del usuario que está utilizando la aplicación.
uid_t geteuid (void);       // Da el UID efectivo.
gid_t getgid (void);        // Da el GID del usuario.
gid_t getegid (void);       // Da el GID efectivo.

int setuid (uid_t uid);     // Cambia el ID dependiendo de los IDs efectivos
int setgid (gid_t gid);     // actuales.
```

+ **Variables de entorno:** (stdlib.h)

extern char ** environ;
char * getenv (char * name);
int putenv (char * cadena);

ulimit.h → long ulimit (int cmd, ...);
- cmd:
  - UL_GETFSIZE: Nº máximo de bloques que se pueden escribir.

- UL_SETFSIZE
- UL_GETMAXBRK: Tamaño máximo que se puede ocupar en memoria.

+ **Control de la memoria:**

```
extern _end, end;
extern _etext, etext;       // Dirección de la zona de texto.
extern _edata, edata;       // Dirección de la zona de datos.

int brk (char * endds);     // end = endds;
char * sbrk (int n);        // Incrementa n Bytes el segmento de datos.

void * malloc (size_t tamaño);   // Pide memoria.
void free (void * ptr);          // Libera memoria.

sys/lock.h   // Bloqueo del programa en memoria.
int plock (int op);
```

- op:
  - PRLOCK → Deja en memoria los segmentos de código.
  - TXTLOCK → Deja en memoria los segmentos de texto.
  - DATLOCK → Deja en memoria los segmentos de datos.
  - UNLOCK → Desbloquea la memoria.

+ **Hilos:** (Páginas 212-233)

+ **Errores:**

```
#define ErrorTotal (mensaje) \
        fprintf(stderr, "%s: %d: ERROR: %s - %s\n", __FILE__, \
                __LINE__, mensaje, strerror(errno)); \
        exit(-1);
```

b) <u>Señales</u>:

+ **Tipos de señales:**

```
SIGHUP    = Desconexión
SIGINT    = Interrupción
SIGQUIT   = Salir
SIGILL    = Instrucción ilegal
SIGTRAP   = Trace trap
SIGIOT    = I/O trap instruction
SIGEMT    = Emulador trap instruction
SIGFPE    = Error en coma flotante
SIGKILL   = Terminación abrupta
SIGBUS    = Error de bus
SIGSEGV   = Violación de segmento
SIGSYS    = Argumento erróneo en una llamada al sistema
SIGPIPE   = Intento de escritura en una tubería de la que no hay nadie leyendo
SIGALRM   = Despertador
SIGTERM   = Finalización controlada
SIGUSR1   = Señal Nº 1 de usuario
SIGUSR2   = Señal Nº 2 de usuario
SIGCLD    = Terminación del proceso hijo
SIGPWR    = Fallo de alimentación
```

+ **Señales en UNIX S5:** (signal.h)

```
int kill (pid_t pid, int sig); // Envía una señal a un proceso.
```

- pid: > 0 PID al que mandamos la señal.

= 0 Se envía la señal a todos los procesos del mismo grupo de proceso.
= -1 Se envía la señal a todos los procesos del mismo usuario.
< -1 Se envía la señal a todos los procesos del mismo gid.

- sig: Nº de señal a enviar.

int raise (int sig);
Envía una señal al propio proceso que llama a la función.

void (* signal (int sig, void (* función) ())) ();
Asigna una función para el tratamiento de una señal enviada al proceso.
- sig: Nº de señal.
- función: SIG_DFL = Usar manejador por defecto.
SIG_IGN = Ignorar la señal.
función con el formato:
void funcion (int sig);
void funcion (int sig, int code, struct sigcontext * scp);

Devuelve SIG_ERR en caso de error, ejemplo:
if(signal(SIGINT, funcionSenyal) == SIG_ERR) { ... error ...}

unistd.h → int pause (void);
Suspende la ejecución hasta que recibe una señal.

+ **Señales del sistema 4.3 BSD:**

SIGVTALRM	= Alarma de un temporizador en tiempo virtual.
SIGPROF	= Alarma de un temporizador.
SIGIO	= Señal de entrada/salida asíncrona.
SIGWINCH	= Cambio del tamaño de una ventana.
SIGSTOP	= Señal de parada de un proceso.
SIGTSTP	= Señal de parada procedente de un terminal.
SIGCONT	= Continuar.
SIGTTIN	= La reciben procesos en 2º plano, que intentan leer datos en un terminal de control.
SIGTTOU	= La reciben procesos en 2º plano, que intentan escribir datos en un terminal de control.
SIGURG	= Indica que ha llegado un dato urgente a través de un canal de E/S.
SIGXCPU	= Se ha superado el tiempo de CPU del proceso actual.
SIGXFSZ	= Se ha superado el tamaño máximo del fichero que puede manejarse.

+ **Funciones de tiempo:** (sys/time.h, time.h)
int stime (long * tp); // Fija la fecha del sistema.
time_t time (time_t * tloc); // Da los segundos pasados desde el 1-1-1970.
int gettimeofday (struct timeval * tp, struct timezone * tzp);
int settimeofday (struct timeval * tp, struct timezone * tzp); // (Páginas 283-284)
clock_t times (struct tms * buffer);
Tiempo de CPU en modo:
+ clock_t tms_utime; // Usuario
+ clock_t tms_stime; // Supervisor
+ clock_t tms_cutime; // Usuario, de los procesos hijo.
+ clock_t tms_cstime; // Supervisor, de los procesos hijo.

+ **Temporizadores:** (unistd.h)

unsigned alarm (unsigned sec);
Activa un temporizador, que cuando hayan pasado sec segundos, se enviará una señal SIGALRM al proceso que llamó a la función. Si se llama varias veces a alarm se restaura el valor de sec. Y con sec a valor 0, se quita el temporizador. Sec puede valer como mucho MAX_ALARM (unos 31 días).

struct tm * localtime (const time_t * tiempo); // time.h
struct tm:
+ int tm_sec; // 0-59
+ int tm_min; // 0-59
+ int tm_hour; // 0-23
+ int tm_mday; // 1-31
+ int tm_mon; // 1-12
+ int tm_year; // 1900-...
+ int tm_wday; // 0 (Domingo) - 6 (Sábado)
+ int tm_yday; // 0-365
+ int tm_isdst; // Indicador de corrección de la fecha

4) Comunicación entre procesos:

a) Comunicación mediante tuberías:

+ **Sin nombre:** (unistd.h)
  int pipe (int handler[2]);
  Devuelve 0 si todo fue bien. Handler contiene dos manejadores:
  - handler[0] = Fichero de solo lectura, del que leeremos datos.
  - handler[1] = Fichero de solo escritura, en el que escribiremos datos.
+ **Con nombre:** (sys/types.h, sys/stat.h)
  int mkfifo (char * ruta, mode_t modo);
  Deuvelve 0 o -1 en caso de fallo. Modo son los permisos de la tubería.
  Si existe falla y errno = EEXIST.
+ **Multiplexación:**
  sys/types.h, unistd.h, time.h, sys/time.h
  int select (int nfds, int readfds, int writefds, int exceptfds, struct timeval * timeout);
  - nfds: Bits que se examinarán (0..nfds-1).
  - readfds: Descriptores de los ficheros de lectura.
  - writefds: Descriptores de los ficheros de escritura.
  - exceptfds: Descriptores de los ficheros especiales.
  - timeout: Tiempo máximo para la función select.

  FD_ZERO (fd_set * fdset); // Pone a 0 los bits de fdset.
  FD_SET (int fd, fd_set * fdset); // Activa el bit fd de fdset.
  FD_CLEAR (int fd, fd_set * fdset); // Desactiva el bit fd de fdset.
  FD_ISSET (int fd, fd_set * fdset); // Comprueba el bit fd de fdset.

b) Comunicación entre procesos: (sys/types.h, sys/ipc.h)

+ **Mecanismos IPC:** Semáforos, memoria compartida y mensajes.
  types.h, sys/ipc.h
  key_t ftok (char * path, char id);
  Genera claves que usar de nombre para los mecanismos que creamos.
+ **Semáforos:** (sys/sem.h) (cerrado (varl ≤ 0 < val) abierto)
  int semget (key_t key, int nsems, int semflg);

Devuelve un manejador para el semáforo, que heredan los hijos.
- key: Nombre clave, si vale IPC_PRIVATE, se crea uno nuevo con un nombre clave libre en el sistema.
- nsems: Nº de semáforos.
- semflag: Permisos del semáforo, con IPC_CREAT, crea el semáforo si no existe. IPC_EXCL (exclusividad).

int semctl (int semid, int semnum, int cmd, union semun arg);
- arg → { int val; struct semid_ds * buf; ushort * array; }
- cmd: Comando.

GETVAL	Devuelve el valor de un semáforo.
SETVAL	Establece el valor de un semáforo (arg.val).
GETPID	Da el PID del último proceso en usar el semáforo.
GETNCNT	Da el Nº de procesos que esperan que incremente el semáforo.
GETZCNT	Da el Nº de procesos que esperan que sea 0 el semáforo.
GETALL	Lee el valor de todos los semáforos (arg.array).
SETALL	Cambia el valor de todos los semáforos (arg.array).
IPC_STAT	Lee la información administrativa asociada (arg.buf).
IPC_SET	Modifica la información administrativa asociada (arg.buf).
IPC_RMID	Indica los semáforos a borrar dentro del semid.

int semop (int semid, struct sembuf * sops, int nsops);
- sops: Array de operaciones.
  - ushort sem_num; // Nº del semáforo.
  - short sem_op; // Operación (incrementar o decrementar) (1 o -1).
  - short sem_flg; // Máscara de bits.
    - IPC_NOWAIT: No espera a que el semáforo esté abierto (IPC_WAIT por defecto).
    - SEM_UNDO: Deshacer la operación al finalizar el proceso.
- nsops: Nº de elementos en sops.

+ **Memoria compartida:** (sys/shm.h)

int shmget (key_t key, int size, int shmflg); // Devuelve un manejador.
- size: Tamaño de la zona.
- shmflg: (IPC_CREAT | permiso).

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

IPC_STAT	Lee el estado.
IPC_SET	Modifica el estado.
IPC_RMID	Borra la zona de memoria compartida.
SHM_LOCK	Bloquea el segmento compartido.
SHM_UNLOCK	Desbloquea el segmento compartido.

- shmid_ds:
  - struct ipc_per shm_per; // Permisos.
  - int sh_segsz; // Tamaño del segmento.
  - int pad1; // Usado por el sistema.
  - ushort shm_lpid; // PID del último proceso en operar con el // segmento.
  - ushort shm_cpid; // PID del proceso creador.
  - ushort shm_nattach; // Nº de procesos unidos al segmento.
  - short pad2; // Usado por el sistema.
  - time_t shm_atime; // Fecha de la última modificación.

- time_t shm_dtime; // Fecha de la última separación.
- time_t shm_ctime; // Fecha del último cambio.

char * shmat (int shmid, char * shmaddr, int shmflag); // Atar.
int shmdt (char * shmaddr); // Desatar.
- shmaddr: Dirección virtual donde queremos que empiece la zona de memoria compartida. Normalmente es 0.
- shmflag: SHM_RDONLY = Segmento de solo lectura.
La función shmat devuelve la dirección virtual real del segmento.

+ **Colas de mensajes:** (sys/msg.h)

int msgget (key_t key, int msgflg); // Devuelve un manejador.
- msgflg: (IPC_CREAT | permiso).

int msgctl (int msqid, int cmd, struct msqid_ds * buf);
- cmd: IPC_STAT Lee el estado.
IPC_SET Modifica el estado.
IPC_RMID Borra el buzón de mensajes.

int msgsnd (int msqid, void * msgp, int msgsz, int msgflg);
int msgrcv (int msqid, void * msgp, int msgsz, long msgtyp, int msgflg);
- msgp: Mensaje.
- msgsg: Tamaño del mensaje.
- msgflg:
  - IPC_NOWAIT = No espera a que se vacíe o haya datos en el buzón.
  - IPC_WAIT = Espera a que se vacíe o haya datos en el buzón.
- msgtyp:
  - = 0 → Leer el 1er mensaje de la cola.
  - > 0 → Leer el 1er mensaje de tipo msgtyp.
  - < 0 → Leer el 1er mensaje de tipo menor o igual que (-msgtyp).

c) <u>Comunicaciones en red</u>:

+ **Direcciones de red:**

```
struct sockaddr {
        ushort sa_family;    // Familia de conectores (AF_xxx)
        char sa_data[14];    // Dirección
}

struct in_addr { // netinet/in.h
        u_long s_addr; // 32b con la id de la red y del nodo
}

struct sockaddr_in {
        short sin_family;              // AF_INET
        u_short sin_port;              // 16b con el Nº de puerto
        struct in_addr sin_addr;       // 32b con la dirección IP
        char sin_zero[8];              // 8b no usados
}

struct sockaddr_un { // sys/un.h
        short sun_family;              // AF_UNIX
        char sun_path[108];            // Ruta
```

}

+ **Llamadas para el manejo de conectores:** (sys/types.h)
  + **Abrir un punto terminal en un canal:**
    int socket (int af, int type, int protocol);
    - af: Familia de conectores.
      AF_UNIX Protocolo interno, para la comunicación entre procesos de la misma máquina.
      AF_INET Protocolo de internet (TCP, UDP).
    - type: Semántica de la comunicación. Conector con un protocolo:
      SOCK_STREAM Orientado a conexión (circuito virtual).
      SOCK_DGRAM No orientado a conexión (datagrama).
      SOCK_RAW Interno de la red, para detalles internos de la red (solo root).
      SOCK_SEQPACKET No orientado a conexión, con envío fiable y secuencial para datagramas.
      SOCK_RDM No orientado a conexión, con envío fiable y secuencial de paquetes.
    - protocol: Nº de protocolo (0 = que elija el sistema).
    Devuelve un manejador.
  + **Nombre de un conector:** (sys/vn.h, sys/netinet.h) (AF_UNIX, AF_INET)
    int bind (int sfd, const void * addr, int addrlen);
    - sfd: Manejador.
    - addr: { struct sockaddr_in | struct sockaddr_un }
    - addrlen: sizeof(...)
    Asocia una dirección con un conector.
  + **Disponibilidad para recibir peticiones de servicio:**
    int listen (int sfd, int backlog);
    - sfd: Manejador.
    - backlog: Longitud de la cola.
    Habilita una cola asociada al conector, que recibe peticiones de los clientes.
    Tiene que ser un conector SOCK_STREAM.
  + **Petición de conexión:**
    int connect (int sfd; const void * addr, int addrlen);
    Realiza una conexión con la dirección indicada.
  + **Aceptar una conexión:**
    int accept (int sfd, void * addr, int * addrlen);
    Extrae la primera petición de la cola creada por listen. Devuelve un descriptor.
  + **Manejo de mensajes:**
    int recv (int sfd, void * buf, int len, int flags);
    int recvfrom (int sfd, void * buf, int len, int flags, void * from, int fromlen);
    int recvmsg (int sfd, struct msghdr msg[], int flags);
    - from: Dirección del conector que ha enviado los datos.
    - msg: Mensaje.
      caddr_t msg_name; // Dirección.
      int msg_namelen; // Tamaño de msg_name.
      struct iovec * msg_iov; // Array de bloques de memoria donde escribir lo // leido.
      caddr_t msg_accrights; // Derechos de acceso.
      int msg_accrightslen; // Tamaño de msg_accrights.
    - flags:
      MSG_PEEK = Se lee sin borrar del buffer.
      MSG_OOB = Se leen los mensajes con más prioridad.

int send (int sfd, void * buf, int len, int flags);
int sendto (int sfd, void * buf, int len, int flags, void * to, int tolen);
int sendmsg (int sfd, struct msghdr msg[], int flags);
- flags: 0, MSG_OOB (mensaje urgente).

+ **Cierre del canal:**
int close (int sfd);

+ **Funciones varias:**
sys/utsname.h → int uname (struct utsname * name);
Da información sobre el sistema actual. (Página 440)

netinet/in.h, arpa/inet.h
unsigned long inet_addr (const char * dirip);
char * inet_ntoa (struct in_addr in);
Convierte de cadena a número y al revés, direcciones ip.

int rresvport (int * port); // Reserva el uso de un puerto.
1-1023 → Reservados por el sistema.
1024-5000 → Gestionados por el sistema.

sys/types.h, netinet/in.h
unsigned long htonl (unsigned long hostlong);
unsigned short htons (unsigned short hostshort);
unsigned long ntonl (unsigned long netlong); // BE → Formato de la máquina.
unsigned short ntons (unsigned short netshort); // BE → Formato de la máquina.
Transforma números de Big Endian, a Little Endian y al revés.

netinet/in.h, netdb.h // Gestión de /etc/hosts
struct hostent * gethostent (void);
struct hostent * gethostentbyname (char * name);
struct hostent * gethostentbyaddr (const char * addr, int len, int type);
int sethostname (int stayopen);
int endhostent (void);

```
struct hostent {
        char * h_name;       // Nombre
        char ** h_aliases;   // Lista de alias
        int h_adrtype;       // AF_INET
        int h_length;        // Nº de direcciones
        char ** h_addr_list; // Lista de direcciones ip
}
```

+ **Otras llamadas de red:**
  + **Nombres de un conector:**
    int getsockname (int sfd, void * addr, int addrlen);
    int getpeername (int sfd, void * addr, int addrlen);
  + **Nombre de nodo actual:** (unistd.h)
    int gethostname (char * hostname, size_t size);
    - size: Tamaño de hostname.
  + **Tuberías con conectores:**
    int socketpair (int family, int type, int protocol, int sockvec[2]);
    - family: AF_UNIX.

- type: SOCK_STREAM, SOCK_DGRAM.
- protocol: 0.

+ **Cierre de un conector:**
  int shutdown (int sfd, int how);
  - how:
    SHUT_RD = Deshabilita la recepción.
    SHUT_WD = Deshabilita el envio.
    SHUT_RDWD = Deshabilita todo (igual que close).

+ **Ejemplo de transferencia de ficheros:**
  (Páginas 467-482)

+ **sleep:**
  unistd.h
  unsigned int sleep (unsigned int segundos);

5) Manipulación de la pantalla:

a) Compilar con ncurses:

```
#include <curses.h>

gcc prog.c -o prog -lcurses
```

b) Inicialización y finalización:

+ **Inicialización:**

```
WINDOW * initscr (void); // Devuelve un puntero a stdscr (NULL == error).
SCREEN * newterm (const char * type, FILE * outfd, FILE * infd);
SCREEN * set_term (SCREEN * new);
```

+ **Finalización:**

```
int endwin (void);
void delscreen (SCREEN * sp);
```

+ **Ejemplo:**

```
if(initscr() != NULL) {
        printw("Hola mundo.");
        refresh();
        getch();
        endwin();
}

SCREEN * scr = newterm(NULL, stdout, stdin);
if(scr != NULL) {
        if(set_term(scr) != NULL) {
                printw("Hola mundo.");
                refresh();
                getch();
        }
        endwin();
        delscreen(scr);
}
```

c) Input/Output:

+ **Salida (Output):**
  - Caracteres:

```
int addch (chtype ch);
int mvaddch (int y, int x, chtype ch);
int waddch (WINDOW * scr, chtype ch);
int mvwaddch (WINDOW * scr, int y, intx, chtype ch);

int echochar (chtype ch);
int wechochar (WINDOW * scr, chtype ch);

int insch (chtype ch);
int winsch (WINDOW * scr, chtype ch);
int mvinsch (int y, intx, chtype ch);
int mvwinsch (WINDOW * scr, int y, intx, chtype ch);
```

Atributos: ('A' | ...)

A_NORMAL	normal
A_STANDOUT	brillante
A_UNDERLINE	subrayado
A_REVERSE	revés
A_BLINK	parpadear
A_DIM	fina
A_BOLD	negrita
A_INVIS	invisible
A_CHARTEXT	con máscara

Caracteres ASCII: (ver curs_addch (3))

ACS_ULCORNER	┌
ACS_LLCORNER	└
ACS_URCORNER	┐
ACS_LRCORNER	┘
ACS_HLINE	–
ACS_VLINE	|

  - Cadenas:

```
int addchstr (const chtype * chstr);
int addchnstr (const chtype * chstr, int n);
int waddchstr (WINDOW * scr, const chtype * chstr);
int waddchnstr (WINDOW * scr, const chtype * chstr, int n);
int mvaddchstr (int y, int x, const chtype * chstr);
int mvaddchnstr (int y, int x, const chtype * chstr, int n);
int mvwaddchstr (WINDOW * scr, int y, int x, const chtype * chstr);
int mvwaddchnstr (WINDOW * scr, int y, int x, const chtype * chstr, int n);

int addstr (const chtype * chstr); // waddstr, mvaddstr, mvwaddstr.
int addnstr (const chtype * chstr, int n); // waddnstr, mvaddnstr, mvwaddnstr.
```

  - Varias:

```
int bkgd (const chtype ch); // Caracter de relleno.
chtype getbkgd (WINDOW * win);

int box (WINDOW * win, chtype verch, chtype horch);
```

```
int border (WINDOW * win, chtype ls, chtype rs, chtype ts, chtype bs, chtype tl,
            chtype tr, chtype bl, chtype br);
int hline (chtype ch, int n); // mvhline, mvwhline, whline.
int vline (chtype ch, int n); // mvvline, mvwvline, wvline.

int erase (void); // werase.
int clear (void); // wclear.
int clrtobot (void); // wclrtobot → Borra el resto de la pantalla desde el cursor.
int clrtoeol (void); // wclrtoeol → Borra el resto de la linea.

int scrollock (void);
int refresh (void); // wrefresh.

void getmaxyx (WINDOW * win, int ymax, int xmax); // Es una macro.
```

+ **Entrada (Input):**

```
int getch (void); // wgetch. ^A = Control + A.
int echo (void); // wecho.
int noecho (void); // wnoecho.

int getstr (char * str); // wgetstr.
int getnstr (char * str, int n); // wgetnstr.
int scanw (char * fmt, args...); // wscanw.
```

d) Color:

```
bool has_colors (void); // 0 = No se pueden usar los colores.

Colores → COLOR_*
          * → BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE.

int start_color (void);
int init_pair (short pair, short f, short b); // COLOR_PAIR(n)
- f = texto.
- b = fondo.

int attron (int attrs); // Activa un color y atributo.
int attroff (int attrs); // Desactiva un color y atributo.
```

e) Manejo de ventanas:

```
WINDOW * newwin (int nlines, int ncols, int y, int x);
WINDOW * subwin (WINDOW * orig, int nlines, int ncols, int y, int x);
WINDOW * derwin (WINDOW * orig, int nlines, int ncols, int y, int x);
WINDOW * dupwin (WINDOW * win);
```

f) Varias funciones:

```
int putwin (WINDOW * win, FILE * filep); // Escribe en filep.
WINDOW * getwin (FILE * filep); // Lee de un filep.

int scr_dump (const char * filename);
int scr_restore (const char * filename);
```

```
int baudrate (void);
char erasechar (void);
char killchar (void);
char * termname (void); // = $TERM
char * longname (void); // = Descripción de la terminal

void getmaxyx (WINDOW * win, int y, int x);
void getyx (WINDOW * win, int y, int x);
void getbegyx (WINDOW * win, int y, int x);
void getparyx (WINDOW * win, int y, int x);
// Es una macro, por ello no es necesario usar & delante de x o de y.
```

a) <u>Solución al kbhit() del dos</u>:

```
nodelay(win, TRUE);
c = getch();
if(c == ERR)
   NadaPulsado();
else
   TeclaPulsada();
```