Funciones Módulo 2

Tratamiento de errores

```
void perror(const char *str)
```

Sirve para imprimir un mensaje de error.

Entrada y salida de archivos regulares

```
int open(const char *path, int oflags);
int open(const char *path, int oflags, mode_t mode);
```

Sirve para abrir un archivo y obtener su descriptor.

`path`: Dirección donde se encuentra el archivo.

`oflags`: Banderas para aplicar al archivo.
- `O_RDONLY` - Solo lectura
- `O_WRONLY` - Solo escritura
- `O_RDWR` - Lectura y escritura
- `O_APPEND` - Agregar información al final del archivo
- `O_TRUNC` - Elimina toda información del archivo
- `O_CREAT` - Crea un nuevo archivo (incluir modo)
- `O_EXCL` - Combinada con , hará que el programa falle si ya existe el archivo

`mode`: Establece los permisos sobre el archivo
- `S_IRUSR` - Lectura para el propietario
- `S_IWUSR` - Escritura para el propietario
- `S_IXUSR` - Ejecución para el propietario
- `S_IRGRP` - Lectura para el grupo
- `S_IWGRP` - Escritura para el grupo
- `S_IXGRP` - Ejecución para el grupo
- `S_IROTH` - Lectura para otros
- `S_IWOTH` - Escritura para otros
- `S_IXOTH` - Ejecución para el otros

```
ssize_t read(int fildes, void *buf, size_t nbytes);
```

Lee los datos de un archivo específico, dado por el descriptor `fildes`.

> `fildes`: Descriptor que indica el archivo donde leer.
>
> `buf`: Lugar donde se almacenan los datos leídos.
>
> `nbytes`: Límite de bytes que pueden llegarse a leer. Si se supera ese límite, los datos se truncan. (Se pierden datos)
>
> **`ssize_t`**: Se devuelve el número de bytes leídos. Si este número es negativo se ha producido algún error.

```
ssize_t write(int fildes, const void *buf, size_t nbytes);
```

Escribe los datos en el archivo especificado mediante el descriptor indicado.

> `fildes`: Descriptor que indica el archivo donde escribir
>
> `buf`: Lugar de donde se leen los datos para escribir.
>
> `nbytes`: Límite de bytes que pueden llegarse a escribir. Si no se llega a ese límite, los datos se truncan. (Se pierden datos)
>
> **`ssize_t`**: Se devuelve el número de bytes escritos. Si este número es negativo se ha producido algún error.

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

Cambia el `current_offset` de un archivo (posición de lectura/escritura actual).

> `fildes`: Descriptor que indica el archivo que estamos tratando.
>
> `offset`: Nuevo lugar donde apunta `current_offset` medido en bytes.
>
> `whence`: Forma de interpretar el nuevo offset. Tiene tres valores:
>
> `SEEK_SET`: Desde el inicio del archivo
> `SEEK_CUR`: Desde la localización actual de `current_offset`
> `SEEK_END`: Desde el final del archivo

```
int close(int fildes);
```

Cierra un archivo abierto. Devuelve 0 si se realiza con éxito y -1 si no.

DESCRIPTORES POR DEFECTO

`STDIN_FILENO`: Entrada estándar de un proceso
`STDOUT_FILENO`: Salida estándar de un proceso
`STDERR_FILENO`: Salida de error estándar

Comprobar el tipo de archivo

`S_ISLNK`: Enlace simbólico
`S_ISREG`: Archivo regular
`S_ISDIR`: Directorio
`S_ISCHR`: Dispositivo de caracteres
`S_ISBLK`: Dispositivo de bloques
`S_ISFIFO`: Cauce con nombre (FIFO)
`S_ISSOCK`: Socket

Máscaras de permisos

`S_IFMT` `0170000`: máscara de bits para los campos de bit del tipo de archivo (no POSIX)
`S_IFSOCK` `0140000`: socket (no POSIX)
`S_IFLNK` `0120000`: enlace simbólico (no POSIX)
`S_IFREG` `0100000`: archivo regular (no POSIX)
`S_IFBLK` `0060000`: dispositivo de bloques (no POSIX)
`S_IFDIR` `0040000`: directorio (no POSIX)
`S_IFCHR` `0020000`: dispositivo de caracteres (no POSIX)
`S_IFIFO` `0010000`: cauce con nombre (FIFO) (no POSIX)
`S_ISUID` `0004000`: bit SUID
`S_ISGID` `0002000`: bit SGID
`S_ISVTX` `0001000`: sticky bit (no POSIX)
`S_IRWXU` `0000700`: user (propietario del archivo) tiene permisos de lectura, escritura y ejecución
`S_IRUSR` `0000400`: user tiene permiso de lectura (igual que S_IREAD, no POSIX)
`S_IWUSR` `0000200`: user tiene permiso de escritura (igual que S_IWRITE, no POSIX)
`S_IXUSR` `0000100`: user tiene permiso de ejecución (igual que S_IEXEC, no POSIX)
`S_IRWXG` `0000070`: group tiene permisos de lectura, escritura y ejecución
`S_IRGRP` `0000040`: group tiene permiso de lectura
`S_IWGRP` `0000020`: group tiene permiso de escritura
`S_IXGRP` `0000010`: group tiene permiso de ejecución
`S_IRWXO` `0000007`: other tienen permisos de lectura, escritura y ejecución
`S_IROTH` `0000004`: other tienen permiso de lectura
`S_IWOTH` `0000002`: other tienen permiso de escritura
`S_IXOTH` `0000001`: other tienen permiso de ejecución

Cambio de permisos

```
mode_t umask(mode_t mask);
```

Fija la máscara de creación de permisos para el proceso, devolviendo el valor anteriormente establecido. Esta máscara es usada en **open**.

```
int chmod(const char *path, mode_t mode);
```

Cambiaremos los permisos del archivo `path`, según los especificados en `mode`.

```
int fchmod(int fildes, mode_t mode);
```

Igual que la anterior solo que este opera con el descriptor de un archivo previamente abierto con **open** (`fildes`).

Otras máscaras de permisos

`S_IRWXU 00700`: user (propietario del archivo) tiene permisos de lectura, escritura y ejecución
`S_IRUSR 00400`: lectura para el propietario (= S_IREAD no POSIX)
`S_IWUSR 00200`: escritura para el propietario (= S_IWRITE no POSIX)
`S_IXUSR 00100`: ejecución/búsqueda para el propietario (=S_IEXEC no POSIX)
`S_IRWXG 00070`: group tiene permisos de lectura, escritura y ejecución
`S_IRGRP 00040`: lectura para el grupo
`S_IWGRP 00020`: escritura para el grupo
`S_IXGRP 00010`: ejecución/búsqueda para el grupo
`S_IRWXO 00007`: other tienen permisos de lectura, escritura y ejecución
`S_IROTH 00004`: lectura para otros
`S_IWOTH 00002`: escritura para otros
`S_IXOTH 00001`: ejecución/búsqueda para otros

Manejo de directorios

```
DIR *opendir(const char *dirname);
```

Abre el directorio especificado en `dirname`, devolviendo su estructura `DIR`. Si hay algún fallo se devuelve `NULL`.

```
struct dirent *readdir(DIR *dirp);
```

Lee la entrada donde esté el puntero de lectura, devolviendo su `struct dirent`. Cuando finaliza su ejecución, el puntero de lectura pasará a la siguiente posición. Se le deberá pasar como parámetro la estructura `DIR` (`dirp`). Si se produce un error devuelve `NULL`.

```
int closedir(DIR *dirp);
```

Cierra un directorio. Devuelve 0 si se ejecuta con éxito. Si hay algún error devuelve -1.

```
void seekdir(DIR *dirp,  long loc);
```

Sitúa el puntero de la siguiente lectura de un directorio en `loc`. El valor de este último argumento se obtiene de la siguiente llamada.

```
long telldir(DIR *dirp);
```

Devuelve la posición actual del puntero de lectura de un directorio.

```
void rewinddir(DIR *dirp);
```

Coloca el puntero de lectura al principio del directorio.

Recorrer un directorio completo

```
int nftw (const char *dirpath, int (*func) (const char *pathname,
          const struct stat *statbuf, int typeflag, struct FTW *ftwbuf),
          int nopenfd, int flags);
```

Esta función recorre cada archivo/directorio de un directorio, y le aplica una función especificada como parámetro.

`dirpath`: Ruta del directorio a recorrer.

`func`: Función que se ejecutará para cada archivo/directorio. La función tendrá que tener los parámetros indicados en la llamada.*

`nopenfd`: Este parámetro indica el número de descriptores máximo que puede crear la función. Cuando estos descriptores se necesitan superar, se irán abriendo y cerrando los ya existentes.

`flags`: Modifican la operación de la función:

- `FTW_DIR`: Realiza un chdir(cambia de directorio) en cada directorio antes de procesar su contenido. Se utiliza cuando func debe realizar algún trabajo en el directorio en el que el archivo especificado por su argumento pathname reside.
- `FTW_DEPTH`: Realiza un recorrido postorden del árbol. Esto significa que nftw llama a func sobre todos los archivos (y subdirectorios) dentro del directorio antes de ejecutar func sobre el propio directorio.
- `FTW_MOUNT`: No cruza un punto de montaje
- `FTW_PHYS`: Indica a nftw que nos desreferencie los enlaces simbólicos. En su lugar, un enlace simbólico se pasa a func como un valor typeflag de FTW_SL

* Los parámetros de la función `func`, deben ser los siguientes:

`pathname`: Indica la ruta del archivo
: Puntero a una estructura stat con información del archivo
: Información adicional sobre el archivo.

- `FTW_D`: Es un directorio
- `FTW_DNR`: Es un directorio que no se puede leer
- `FTW_DP`: Estamos haciendo un recorrido posorden de un directorio, es decir, los archivos y subdirectorios de este ya se han leido.
- `FTW_F`: No es un directorio ni enlace simbólico.
- `FTW_NS`: stat ha fallado sobre este archivo
- `FTW_SL`: Enlace simbólico
- `FTW_SLN`: Enlace simbólico perdido.

`ftwbuf`: Puntero a una estructura FTW

Estructuras

```
struct stat {
    dev_t st_dev;              /* no de dispositivo (filesystem) */
    dev_t st_rdev;             /* no de dispositivo para archivos especiales */
    ino_t st_ino;              /* no de inodo */
    mode_t st_mode;            /* tipo de archivo y mode (permisos) */
    nlink_t st_nlink;          /* número de enlaces duros (hard) */
    uid_t st_uid;              /* UID del usuario propietario (owner) */
    gid_t st_gid;              /* GID del usuario propietario (owner) */
    off_t st_size;             /* tamaño total en bytes para archivos regulares */
    unsigned long st_blksize;  /* tamaño bloque E/S para el sistema de archivos*/
    unsigned long st_blocks;   /* número de bloques asignados */
    time_t st_atime;           /* hora último acceso */
    time_t st_mtime;           /* hora última modificación */
    time_t st_ctime;           /* hora último cambio */
};
```

Obtener las estructuras de un archivo

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

Sirve para obtener los metadatos de un archivo.

`path`: Ruta de directorios hacia el archivo

`buf`: Estructura donde se guardan los metadatos

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

Es igual que la función explicada anteriormente solo que cuando lo realizamos sobre un enlace blando, devolverá los metadatos del archivo al que el enlace hace referencia.

Directorios

```
typedef struct _dirdesc {
    int dd_fd;
    long dd_loc;
    long dd_size;
    long dd_bbase;
    long dd_entno;
    long dd_bsize;
    char *dd_buf;
} DIR;
```

```c
struct dirent {
    long d_ino;          /* número i-nodo */
    char d_name[256];    /* nombre del archivo */
};


struct FTW {
    int base;   /* Desplazamiento de la parte base del pathname */
    int level;  /*Profundidad del archivo dentro recorrido del arbol */
};
```

Control de señales

```c
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

- `sa_handler` especifica la acción que se va a asociar con la señal signum pudiendo ser:
  - `SIG_DFL` para la acción predeterminada.
  - `SIG_IGN` para ignorar la señal
  - Un puntero a una función manejadora para la señal

- `sa_mask` permite establecer una máscara de señales que deberían bloquearse durante la ejecución del manejador de la señal. Además, la señal que lance el manejador será bloqueada, a menos que se activen las opciones `SA_NODEFER` o `SA_NOMASK`.

- Para asignar valores a `sa_mask` , se usan las siguientes funciones:

  - `int sigemptyset(sigset_t *set);` - Inicializa a vacío un conjunto de señales (devuelve 0 si tiene éxito y –1 en caso contrario).
  - `int sigfillset(sigset_t *set);` - Inicializa un conjunto con todas las señales (devuelve 0 si tiene éxito y –1 en caso contrario).
  - `int sigismember(const sigset_t *set, int senyal);` - Determina si una señal `senyal` pertenece a un conjunto de señales `set` (devuelve 1 si la señal se encuentra dentro del conjunto, y 0 en caso contrario).
  - `int sigaddset(sigset_t *set, int signo);` - Añade una señal a un conjunto de señales `set` previamente inicializado (devuelve 0 si tiene éxito y –1 en caso contrario).
  - `int sigdelset(sigset_t *set, int signo);` - Elimina una señal signo de un conjunto de señales `set` (devuelve 0 si tiene éxito y –1 en caso contrario).

- `sa_flags` especifica un conjunto de opciones que modifican el comportamiento del proceso de manejo de señales. Se forma por la aplicación del operador de bits OR a cero o más de las siguientes constantes:

    - `SA_NOCLDSTOP`: Si signum es SIGCHLD, indica al núcleo que el proceso no desea recibir notificación cuando los procesos hijos se paren (esto es, cuando los procesos hijos reciban una de las señales: SIGTSTP, SIGTTIN o SIGTTOU).
    - `SA_ONESHOT` o `SA_RESETHAND` : Indica al núcleo que restaure la acción para la señal al estado predeterminado una vez que el manejador de señal haya sido llamado.
    - `SA_RESTART` : Proporciona un comportamiento compatible con la semántica de señales de BSD haciendo que ciertas llamadas al sistema reinicien su ejecución cuando son interrumpidas por la recepción de una señal.
    - `SA_SIGINFO`: El manejador de señal toma 3 argumentos, no uno. En este caso, se debe configurar sa_sigaction en lugar de sa_handler.

- El parámetro **`siginfo_t`** para sa_sigaction es una estructura con los siguientes elementos:

```
siginfo_t {
    int si_signo; /* Número de señal */
    int si_errno; /* Un valor errno */
    int si_code; /* Código de señal */
    pid_t si_pid; /* ID del proceso emisor */
    uid_t si_uid; /* ID del usuario real del proceso emisor */
    int si_status; /* Valor de salida o señal */
    clock_t si_utime; /* Tiempo de usuario consumido */
    clock_t si_stime; /* Tiempo de sistema consumido */
    sigval_t si_value; /* Valor de señal */
    int /* señal POSIX.1b */ si_int;
    void * si_ptr; /* señal POSIX.1b */
    void * si_addr; /* Dirección de memoria que ha producido el fallo */
    int si_band; /* Evento de conjunto */
    int si_fd; /* Descriptor de fichero */
}
```

Control de procesos

```
pid_t getpid(void);
```

Devuelve el PID del proceso que lo invoca

```
pid_t getppid(void);
```

Devuelve el PID del proceso padre que lo invoca

```
uid_t getuid(void);
```

Devuelve el identificador de usuario real del proceso que la invoca

```
uid_t geteuid(void);
```

Devuelve el identificador del usuario efectivo del proceso que la invoca

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

Devuelve el identificador de grupo real del proceso que la invoca

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

Devuelve el identificador del grupo efectivo del proceso que la invoca

```
pid_t fork();
```

Crea un nuevo proceso y devuelve su PID

```
pid_t wait(int *wstatus);
```

Suspende la ejecución del proceso hasta que uno de sus hijos termina. `wstatus` contiene el código de terminación del hijo.

```
pid_t waitpid(pid_t pid, int *wstatus);
```

Espera al hijo indicado en el parámetro `pid`. El parámetro `pid` puede tener diferentes valores, que realizarán diferentes funciones:

`<-1`: espera a cualquier hijo cuyo id de grupo de proceso sea igual al valor absoluto del PID.
`-1`: Espera a cualquier hijo
`0`: espera a cualquier hijo cuyo id de grupo de proceso es igual al id del proceso llamado
`>0`: espera al hijo cuyo ID es igual al del valor del pid.

Llamadas exec

```
int execl(const char *path, const char *arg, ...);
int execlp (const char *file, const char *arg, ...);
int execle(const char *path, const char *arg , ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

- `*arg0, *arg1...argn` son los punteros a los distintos argumentos de la ejecución. El último debe de ser `NULL`.
- **`execv`** y **`execvp`** proporcionan un vector de punteros a cadenas de caracteres, que representan la lista de argumentos disponible para el nuevo programa.
- **`execle`** especifica el entorno del proceso que ejecutará el programa mediante un parámetro adicional que va detrás del puntero N.

Llamada clone

```
int clone(int (*func) (void *), void *child_stack, int flags,
          void *func_arg);
```

`clone` crea un proceso hijo ejecutando la función pasada como parámetro. El argumento de la función se introducirá en el parámetro `func_arg`.

Cauces

Cauces con nombre

```
int mknod(const char *FILENAME, mode_t MODE, dev_t DEV);
```

Esta función creará un archivo llamado `FILENAME`. El parámetro `MODE`, especifica los valores almacenados en el campo `st_mode` del i-nodo:

> `S_IFCHR`: representa el valor del código de tipo de archivo para un archivo de dispositivo orientado a caracteres
> `S_IFBLK`: representa el valor del código de tipo de archivo para un archivo de dispositivo orientado a bloques
> `S_IFSOCK`: representa el valor del código de tipo de archivo para un socket.
> `S_IFIFO`: representa el valor del código de tipo de archivo para un FIFO.

```
int mkfifo(const char *FILENAME, mode_t MODE);
```

En esta llamada `FILENAME`, proporciona el nombre del archivo y `MODE` sus permisos. Los archivos FIFO se eliminan con la llamada al sistema unlink.

Cauces sin nombre

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

Si esta llamada tiene éxito, se devolverán en `fd`, dos nuevos descriptores. Normalmente será `fd[0]` para lectura, y `fd[1]` para escritura.

```
int close(int fd);
```

Cierra el descriptor.

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

Esta función crea una copia del descriptor pasado en el parámetro `oldfd`

```
int dup2(int oldfd, int newfd);
```

Copia es descriptor `oldfd` en `newfd`

Gestión de señales

```
int kill(pid_t pid, int sig);
```

La llamada **kill** se puede utilizar para enviar cualquier señal a un proceso o grupo de procesos.

- Si `pid` es positivo, entonces se envía la señal `sig` al proceso con identificador de proceso igual a pid. Se devuelve 0 si hay éxito , o un número negativo en caso contrario.
- Si el `pid` es 0, entonces `sig` se envía a cada proceso en el grupo de procesos del proceso actual.
- Si `pid` es igual a -1 entonces se envía la señal `sig` a cada proceso, excepto al primero desde los números más altos en la tabla de procesos hasta los más bajos.
- Si `pid` es menor que -1, entonces se envía `sig` a cada proceso en el grupo de procesos `-pid`.
- Si `sig` es 0, entonces no se envía ninguna señal, pero sí se realiza la comprobación de errores.

```
int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
```

La llamada al sistema **sigaction** se emplea para cambiar la acción tomada por un proceso cuando recibe una determinada señal. Se devolverá 0 en caso de éxito y -1 en caso de fracaso. El significado de los parámetros de la llamada es el siguiente:

- `signum` especifica la señal y puede ser cualquier señal válida salvo `SIGKILL` o `SIGSTOP`.
- Si `act` no es NULL , la nueva acción para la señal signum se instala como act.
- Si `oldact` no es NULL, la acción anterior se guarda en `oldact`.

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

Se emplea para examinar y cambiar la máscara de señales. (La máscara es el conjunto de señales bloqueadas). Retorna 0 en caso de éxito y -1 en caso de error. Los parámetros a introducir son los siguientes:

- El argumento `how` indica el tipo de cambio. Los valores que puede tomar son los siguientes:
    - `SIG_BLOCK`: El conjunto de señales bloqueadas es la unión del conjunto actual y el argumento `set`.

    - `SIG_UNBLOCK`: Las señales que hay en `set` se eliminan del conjunto actual de señales bloqueadas. Es posible intentar el desbloqueo de una señal que no está bloqueada.
    - `SIG_SETMASK`: El conjunto de señales bloqueadas se pone según el argumento `set`.

- `set` representa el puntero al nuevo conjunto de señales enmascaradas. Si set es diferente de `NULL`, apunta a un conjunto de señales, en caso contrario **`sigprocmask`** se utiliza para consulta.
- `oldset` representa el conjunto anterior de señales enmascaradas. Si `oldset` no es NULL, el valor anterior de la máscara de señal se guarda en `oldset`. En caso contrario no se retorna la máscara la anterior.

```
int sigpending(sigset_t *set);
```

La llamada sigpending permite examinar el conjunto de señales bloqueadas y/o pendientes de entrega. La máscara de señal de las señales pendientes se guarda en `set`. Retorna 0 en caso de éxito y -1 en caso de error.

```
int sigsuspend(const sigset_t *mask);
```

La llamada sigsuspend reemplaza temporalmente la máscara de señal para el proceso con la dada por el argumento mask y luego suspende el proceso hasta que se recibe una señal.