--------FUNCIONES MÁS USADAS-------

- **getopt:** Sirve para recibir opciones y argumentos
  `getopt(int argc,char** argv,const char *optstring)`

  return: devuelve el siguiente carácter de opción que se ha encontrado.
  -1 ->si no quedan más

  Ej:

```
char* comando;
while((c=getopt(argc,argv,"c"))!=-1){
if(c=='c'){
comando=malloc(strlen(argv[optind]));
strcpy(comando,argv[optind]);
printf("%s\n",comando);
}
}
```

- **sscanf:** leer datos de una cadena y almacenarlos en variables usando un formato
  `int sscanf(const char *str, const char *format, ...);`
  return: nº de elementos correctamente leídos

  Ej:

```
char input[] = "Juan 25";
char name[20];
int age;
int count = sscanf(input, "%s %d", name, &age);
if (count == 2) BIEN
else ERROR
```

- **strsep:** se utiliza para dividir una cadena en tokens
  `char *strsep(char **stringp, const char *delim);`
  return: devuelve el token o NULL
  Ej:

```
char* str = strdup(input);
while ((token = strsep(&str, delimiter)) != NULL) {
printf("Token: %s\n", token);
}
```

- **getc:** leer un el carácter disponible del fichero\entrada estándar
  `int getc(FILE *stream);`
  return: devuelve el char o EOF
  Ej:

```
int c;
while ((c = getc(stdin)) != EOF) {
putchar(c); // Muestra el carácter en la salida estándar
}
return 0;
```

- **putc** escribe un caracter en la salida estándar/fichero
  `int fputc(int c, FILE *stream);`
  return: devuelve el char o EOF
  Ej:

```
c = 'H';
if (putc(c, archivo) != c) {
printf("Error al escribir el carácter.\n");
}
```

- **strlen:** Obtiene la longitud de la cadena.
  `size_t strlen(const char *s);`

- **strcat:** concatena dos cadenas.
  `char *strcat(char *dest, const char *src);`
  Ej:

```
char str1[20] = "Hola";
char str2[] = " mundo!";
strcat(str1, str2); // Concatena str2 al final de str1
printf("%s\n", str1); // Imprime "Hola mundo!"
```

- **strstr:** encuentra la primera aparición de una subcadena en una cadena
  `char *strstr(const char *haystack, const char *needle);`
  return: devuelve un puntero al primer carácter de la subcadena encontrada dentro de la cadena principal.
  Ej:

```
char str[] = "Hola mundo, este es un ejemplo";
char sub[] = "mundo";
char *resultado = strstr(str, sub); // Busca la subcadena "mundo"
dentro de str
if (resultado != NULL) ERROR
```

- **strchr:** encuentra la primera aparición del char especificado en una cadena
  `char *strchr(const char *s, int c);`
  return: Devuelve un puntero al carácter encontrado dentro de la cadena, o un puntero nulo (NULL)
  Ej:

```c
char str[] = "Hola mundo";
char ch = 'm';
char *resultado = strchr(str, ch); // Busca el carácter 'm'
dentro de str
if (resultado != NULL) {
printf("Carácter encontrado: %c\n", *resultado);
} else {
printf("Carácter no encontrado.\n");
}
```

- **sprintf:** Formatea una cadena de caracteres y almacena su resultado en un buffer.ç
  `int sprintf(char *str, const char *format, ...);`
  return: Devuelve el número de caracteres escritos en el búfer, sin incluir el carácter nulo de terminación ('\0').
  Ej:

```c
char buffer[20];
int numero = 42;
int caracteres_escritos = sprintf(buffer, "El número es: %d",
numero);
printf("Cadena generada: %s\n", buffer);
printf("Caracteres escritos: %d\n", caracteres_escritos);
return 0;
```

- **memcpy:** copia un bloque de memoria desde una ubicación de origen.
  `void *memcpy(void *dest, const void *src, size_t n);`
  Ej:

```c
char origen[] = "Hola, mundo!";
char destino[20];
memcpy(destino, origen, sizeof(origen));
```

- **fgets :** función en el lenguaje de programación para leer una línea de texto (\n)
  `char *fgets(char *cadena, int size, FILE *stream);`
  - cadena:puntero a un arreglo de caracteres.
  - longitud: num max de caracteres incluyendo un\0.
  - archivo: stdin,file..

  return: NULL/ puntero cadena leída.
  Ej:

```
char cadena[50];
```

```
printf("Introduce una cadena: ");
fgets(cadena, sizeof(cadena), stdin);

printf("La cadena que ingresaste es: %s", cadena);

return 0;
```

- **strtol:** convierte una cadena en un 'long int'.
  `long int strtol(const char *nptr, char **endptr, int base);`
  - nptr:puntero que se quiere convertir a numero
  - fin: puntero a puntero char almacena la posición del primer caracter no válido después de la conversión
  - base: base numérica (10 decimal)(0 base 16)...

  return: el número

```
char cadena[] = "12345";
char *resto;
long numero;

numero = strtol(cadena, &resto, 10);

printf("Número convertido: %ld\n", numero);
printf("Resto de la cadena: %s\n", resto);
```

- **strcmp:** compara dos cadenas de caracteres.
  `int strcmp(const char *s1, const char *s2);`

  return:
  if(s1>s2) return positivo
  if(s1<s2) return negativo
  if(s1==s2)return 0

---FORK---
pid_t fork
waitpid

—CADENAS DE CARACTERES—
execl
sigaction

-------API POSIX PARA FICHEROS--------

- **int creat (const char * pathname , mode_t mode);**
  hace lo mismo que open con las flags O_CREAT|O_WRONLY|O_TRUNC (mode->flags: S_IRUSR | S_IWUSR)

- **int close (int fd)**
  Cierra el fichero que tiene el fd que se le pase
  return: 0 si no da error, EOF si da error
  Ej:

```
fclose(alumnos);
```

- **off_t lseek (int fd , off_t offset , int whence);**
  Mover el puntero de lectura/escritura a una ubicación específica dentro del archivo.
  - fd: El descriptor de fichero
  - offset: El desplazamiento que se va a realizar
  - whence:
    - SEEK_SET: desde el inicio
    - SEEK_CUR: desde donde esté
    - SEEK_END: desde el final

  return: nueva posición actual dentro del archivo
  Ej: (En este caso pone el puntero al principio)

```
// Desplazo el puntero del fichero el offset necesario
lseek(fdo, 0, SEEK_SET);
```

- **ssize_t read/write (int fd , void * buf , size_t count);**
  Leer/escribir datos de un fichero abierto con descriptor fd
  - fd: Descriptor del fichero
  - buf: donde se almacenan los datos leidos
  - count: Numero maximo de bytes que se leen

  return: numero de bytes leidos , -1 en caso de error y 0 al acabar el fichero(read)
  Ej: (Practica 2 ej 2)

```c
int readBytes;
while (readBytes = read(fdo, buff, sizeof(buff)))
{
    // write funciona igual que read pero escribiendo
    if (write(fdd, buff, readBytes) == -1)
    {
        // Cierra los ficheros
        close(fdd);
        close(fdo);
        err(3, "putc() failed!!");
        return -1;
    }
}
```

- **int unlink (const char * pathname);**
  Elimina un archivo del sistema de ficheros
  return: 0 si sale bien y , -1 en caso de error
  Ej:

```c
const char *nombreArchivo = "ejemplo.txt";

int resultadoUnlink = unlink(nombreArchivo);
```

- **int stat (const char * pathname , struct stat * statbuf);**
- **int fstat (int fd , struct stat * statbuf);**
- **int lstat (const char * pathname , struct stat * statbuf) ;**
  Devuelve información sobre un archivo (pahtname) en el struct statbuf
  statbuff:
    - st_mode: tipo de archivo
    - st_size: tamaño de archivo

  lstat no sigue los enlaces simbolicos y te da información del enlace, pero stat te da información del archivo al que apunta
  return: 0 si sale bien y , -1 en caso de error
  Ej:(Práctica 2 ejercicio 3)

```c
struct stat statbuf;
// lstat devuelve información sobre un archivo concreto,
if (lstat(argv[1], &statbuf) == -1)
{
    perror("lstat");
    exit(EXIT_FAILURE);
}
```

------API POSIX PARA DIRECTORIOS-----

- **DIR * opendir (char * dirname) ;**
  Abre el directorio con nombre dirname

- **struct dirent * readdir (DIR * dirp) ;**
  Lee los archivos que hay dentro del directorio y los devuelve en un struct

```
struct dirent {
      ino_t d_ino; Inode number
      off_t d_off; Not an offset; see below
      unsigned short d_reclen; Length of this record
      unsigned char d_type; Type of file; not supported by all
filesystem types
      char d_name[256]; Null-terminated filename
};
```

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

mkdir: crea un directorio con un nombre y protecci´on
rmdir: borra el directorio vac´ıo con un nombre
rewinddir: sit´ua el puntero de posici´on en la primera entrada
chdir: cambia el directorio actual
getcwd: obtener el directorio actual
rename: cambiar el nombre de una entrada del directorio

------API STDLIB PARA FICHEROS-------

long ftell (FILE * stream);
void rewind (FILE * stream);
int fsetpos (FILE * stream , fpos_t * pos);
int fsetpos (FILE * stream , const fpos_t * pos);
int fflush (FILE * stream);
void setbuf (FILE * stream , char * buf);
void setbuffer (FILE * stream , char * buf , size_t size);
void setlinebuf (FILE * stream);
int setvbuf (FILE * stream , char * buf , int mode , size_t size);

MAKEFILE

```
SRC = $(wildcard *.c)
BIN = $(SRC:%.c=%)
OBJ = $(SRC:%.c=%.o)
```

```
CC = gcc
CFLAGS = -g -pthread -O0
LDFLAGS = -g -pthread
LIBS = -lrt

all: $(BIN)

%.o: %.c Makefile
$(CC) $(CFLAGS) -c -o $@ $<

$(BIN): %: %.o
$(CC) $(LDFLAGS) -o $@ $^ $(LIBS)

.PHONY: clean

clean:
-rm $(BIN) $(OBJ)
```

COMO HACER UN ZIP

```
zip -r directorio.zip directorio/
```

SI ESO NO VA:

```
tar -cvzf nombre_comprimido.tar.gz directorio_a_comprimir
```