Usos Open

- Abrir fichero que ya existe
    - Para leer

```
int fd,ret;
fd=open("datos",O_RDONLY);
if (fd<0){
       perror("error al abrir el fichero datos\n");
       exit(1);
}
ret=read(fd,….);
```

    - Para escribir

```
int fd,ret;
fd=open("datos",O_WRONLY);
if (fd<0){
       perror("error al abrir el fichero datos\n");
       exit(1);
}
ret=write(fd,….);
```

    - Para leer/escribir

```
int fd,ret;
fd=open("datos",O_RDWR);
if (fd<0){
       perror("error al abrir el fichero datos\n");
       exit(1);
}
Ret=read(fd,…);
ret=write(fd,….);
```

- Crear y Abrir fichero de datos (se supone que no existe). Si existe no pasa nada.
    - Ej: escritura

```
int fd,ret;
fd=open("datos",O_WRONLY|O_CREAT, S_IRUSR| S_IWUSR);
if (fd<0){
       perror("error al abrir/crear el fichero datos\n");
       exit(1);
}
ret=write(fd,….);
```

- Crear, Abrir y, si existe, borrar contenido de fichero de datos. Si existe se borrará el contenido, si no existe se creará.
    - Ej: escritura

```
int fd,ret;
fd=open(“datos”,O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR| S_IWUSR);
if (fd<0){
       perror(“error al abrir/crear el fichero datos\n”);
       exit(1);
}
ret=write(fd,….);
```

Lecturas

- A tener en cuenta: Para usar la entrada/salida/salida error std no hace falta hacer open. El resto de dispositivos si.
- Datos con tamaño conocido: Ej. Int

```
#define MAX_INT 10
int fd,ret,v_ent[MAX_INT];
char buffer[128];
fd=open(“datos”,O_RDONLY);
ret=read(fd,v_ent,MAX_INT*sizeof(int));
if (ret<0) {
       perror(“Error de lectura\n”);exit(1);
}
if (ret==0){
       sprintf(buffer,”No he leído ningún numero\n”);
else{
       sprintf(buffer,”He leído %d enteros\n”,ret/sizeof(int));
}
write(1,buffer,strlen(buffer));
```

- N bytes

```
#define MAX_SIZE 256
int fd,ret;
char v_char[MAX_SIZE],buffer[128];
fd=open(“datos”,O_RDONLY);
ret=read(fd,v_char,MAX_SIZE);
if (ret<0) {
       perror(“Error de lectura\n”);exit(1);
}
if (ret==0){
       sprintf(buffer,”No he leído ningún byte\n”);
else{
       sprintf(buffer,”He leído %d bytes\n”,ret);
}
write(1,buffer,strlen(buffer));
```

Escrituras

- Datos con tamaño conocido: Ej int

```
#define MAX_INT 10
int fd,ret,v_ent[MAX_INT];
char buffer[128];
fd=open(“datos”,O_WRONLY);
inicializa(v_ent,MAX_INT);
ret=write(fd,v_ent,MAX_INT*sizeof(int));
if (ret<0) {
       perror(“Error de escritura\n”);exit(1);
}
if (ret==0){
       sprintf(buffer,”No he escrito ningún numero\n”);
else{
       sprintf(buffer,”He escrito %d enteros\n”,ret/sizeof(int));
}
write(1,buffer,strlen(buffer));
```

- N bytes

```
#define MAX_SIZE 256
int fd,ret;
char v_char[MAX_SIZE],buffer[128];
fd=open(“datos”,O_WRONLY);
inicializa(v_char,MAX_SIZE);
ret=write(fd,v_char,MAX_SIZE);
if (ret<0) {
       perror(“Error de escritura\n”);exit(1);
}
if (ret==0){
       sprintf(buffer,”No he escrito ningún byte\n”);
else{
       sprintf(buffer,”He escrito %d bytes\n”,ret);
}
write(1,buffer,strlen(buffer));
```

Lecturas y Después escrituras

- Datos de tamaño fijo: Ej int (suma fichero_entrada fichero_salida)

```
int fd,fd2,ret,num,suma=0;
if ((fd=open(argv[1],O_RDONLY)<0) error_cs(“Open datos entrada”,1);
if ((fd2=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR| S_IWUSR)<0)
     error_cs(“Open datos salida”,1);
ret=read(fd,&num,sizeof(num));
while(ret>0){
       suma=suma+num;
       ret=read(fd,&num,sizeof(int));
       // if ((ret<0) || ((ret>0) && (ret<sizeof(int))) ERROR
}
write(fd2,&suma,sizeof(int));
}
```

- N bytes: Leemos, hacemos algo con la entrada , y escribimos

```
int fd,fd2,ret;
char entrada[128];
if ((fd=open(argv[1],O_RDONLY)<0) error_cs("Open datos entrada",1);
if ((fd2=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR| S_IWUSR)<0)
     error_cs("Open datos salida",1);
ret=read(fd,entrada,sizeof(entrada));
while(ret>0){
       FiltraEntrada(entrada,ret);
       write(fd2,entrada,ret);
       ret=read(fd,entrada,sizeof(entrada));
}
```

- Conversión de tipos: int a string : Leo números de la entrada std, en ascii, separados por espacios, los sumo y los escribo en salida std (en ascii)

```
#define fd_in 0
#define fd_out 1
int lee_num_entrada(char *num,int max_size)
{
int ret,pos=0;
ret=read(fd_in,&num[pos],1);
while((pos<max_size) && (ret>0) && (num[pos]!=' ')){
       pos++;
       ret=read(fd_in,&num[pos],1);
}
if ((pos<max_size) && (num[pos]==' ')){
       num[pos]='\0';
       return 1;
}else return 0;
}
main(…){
int ret,num,suma=0;
char num_entrada[64],suma_salida[128];
while( lee_num_entrada(num_entrada,64)>0){
       num=atoi(num_entrada);
       suma=suma+num;
}
sprintf(suma_salida,"%d",suma);
write(fd_out,suma_salida,strlen(suma_salida));
}
```

Comunicar procesos con pipes

- A tener en cuenta: Si el proceso P1 y P2 quieren comunicarse con una pipe, tengo que asegurar que los dos pueden acceder a los canales. Si la pipe es sin nombre, P1 y P2 tienen que estar emparentados
    - P1 es padre de P2. Hay que crear la pipe antes de crear P2

```
int p[2]; // Para los dos canals
int pid;
pipe(p); // Ya esta creada. P[0] es de lectura y p[1] de escritura
pid=fork(); // creamos P2
if (pid==0){
       // P2 : Podemos usar la pipe
}else if (pid>0){
       // P1 : Podemos usar la pipe
}else if (pid<0) ERROR
```

- P1 y P2 son hijos del mismo proceso. Hay que crear la pipe antes de crear el primero de los dos.

```
int p[2]; // Para los dos canals
int pid_p1,pid_p2;
pipe(p); // Ya esta creada. P[0] es de lectura y p[1] de escritura
pid_p1=fork(); // creamos P2
if (pid_p1==0){
       // P1: Podemos usar la pipe
}else if (pid_p1>0){
       pid_p2=fork();
       if (pid_p2==0){
              //P2: Podemos usar la pipe
       }else if (pid_p2>0)while(waitpid(-1,NULL,0)>0);
       Else if (pid_p2<0) ERROR
}else if (pid_p1<0) ERROR
```