

TGR Entrada/salida (Sistemas Operativos)

1. Sea el siguiente código

```
#include "includes.h"
int realiza_trabajo(int fd, char *msg){
    write(fd,msg,strlen(msg));
}
int realiza_trabajo_1(int fd){
    realiza_trabajo(fd,"Realizando Trabajo 1");
}
int realiza_trabajo_2(int fd){
    realiza_trabajo(fd,"Realizando Trabajo 2");
}

int main(){
    int fd;
    fd= open("salida.txt",O_WRONLY|O_CREAT, 0664);
```

int copia = dup(fd);

close(fd);

dup(STDOUT_FILENO); → fd sigue siendo 3 pero
en 3 tenemos salida standard.

realiza_trabajo_1(fd);

close(fd);

dup(copia);

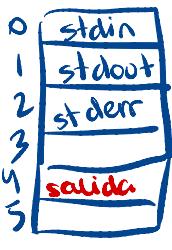
close(copia);

realiza_trabajo_2(fd);

close(fd);

}

Modifíquese dicho código para que todo lo que se escribiría en el fichero “salida.txt” durante la llamada a la función **realiza_trabajo_1**, se muestre por la salida estándar (STDOUT_FILENO), pero que la función **realiza_trabajo_2** siga almacenando sus datos en el fichero “salida.txt”.



El código tal y como está,
escribe en “salida.txt”
“Realizando Trabajo 1”
“Realizando Trabajo 2”



2. Asúmase que tenemos un disco duro con 250 cilindros numerados del 0 al 249, cuya cabeza está sobre el cilindro 100 (atendiendo una petición de acceso previa) y se está moviendo en sentido ascendente ($0 \rightarrow 249$). Si en el instante actual la cola de peticiones de acceso a cilindros contiene las peticiones: $\langle 99, 102, 125, 29, 247, 95, 110, 90, 198, 10 \rangle$, indíquese en qué orden se atenderán dichas peticiones si se utiliza un algoritmo de planificación:

FCFS o FIFO:	99	102	125	29	247	95	110	50	198	10
SSTF:	99	102	95	90	110	125	198	247	29	10
SCAN:	102	110	125	198	247	95	90	29	10	
C-SCAN:	102	110	125	198	247	10	29	90	95	99

3. En un momento dado, la cola de peticiones de E/S para acceder a cilindros de un disco duro contenía las peticiones: $\langle 88, 33, 18, 90 \rangle$. Desconocemos el cilindro que estaba siendo accedido antes de estas peticiones, pero sabemos que dichas peticiones fueron atendidas en el orden siguiente $\langle 88, 33, 18, 90 \rangle$. Indíquese si los siguientes algoritmos pudieron haber sido (o no) usados para la planificación del acceso al disco. Razónese brevemente la respuesta:

algor.	sí/no	breve justificación de la respuesta
FIFO:	sí	las peticiones se atienden por orden de llegada
SSTF:	no	si empieza en el 88 debería ir al 90, no al 33
SCAN:	sí	estando en el 88 y desplazándose ascendente.

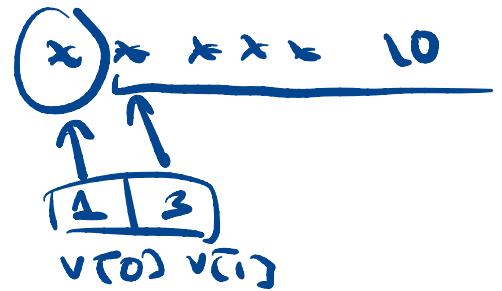
4. Suponiendo que el fichero "fichero.txt" contiene: "ABCDEFGHIJKLMNPQRSTUVWXYZ\n", y que el siguiente código no produce ningún error de ejecución:

```
#include "includes.h"

int lee1(int fd, char BUF[]) {
    read(fd,BUF,1);
    read(fd,BUF,3);
}
int lee2(int fd, char BUF[]) {
    pread(fd,BUF,1,0);
    pread(fd,BUF,3,0);
}
int lee3(int fd, char BUF[]) {
    struct iovec iov[2];
    iov[0].iov_base= BUF;    iov[0].iov_len= 1;
    iov[1].iov_base= BUF+1;  iov[1].iov_len= 3;
    readv(fd,iov,2);
}

int main(){
    int fd = open("fichero.txt",O_RDONLY);
    char BUF[6]= {'*', '*', '*', '*', '*', '\0'};

    #ifdef CASO_A
        lee1(BUF);
    #endif
    #ifdef CASO_B
        lee2(BUF);
    #endif
    #ifdef CASO_C
        lee3(BUF);
    #endif
}
```



#ifdef CASO_A lee1(BUF); #endif	#ifdef CASO_B lee2(BUF); #endif	#ifdef CASO_C lee3(BUF); #endif
---------------------------------------	---------------------------------------	---------------------------------------

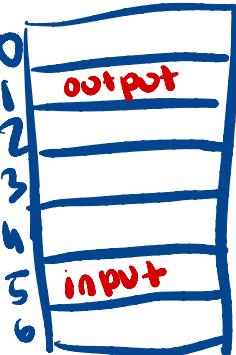
```
printf("%s",BUF);
close(fd);
}
```

Muéstrese qué se imprime por pantalla en el “printf” si se llama a lee1(), lee2(), o lee3() respectivamente.

lee1() → B C D * *
lee2() → A B C * *
lee3() → A B C D *

5. Dado el siguiente programa...

```
11: #include "includes.h"
12: int main(){
13:     int fd1= open("input.txt",O_RDONLY); //existe en el directorio actual
14:     int fd2= open("output.txt",O_WRONLY|O_CREAT,0664); //se crea el fichero
15:     dup (fd1);
16:     close(STDOUT_FILENO);           // equivale a close(1)
17:     close(STDERR_FILENO);          // equivale a close(2)
18:     dup (fd2);
19:     close(STDIN_FILENO);           // equivale a close(0)
110:    printf("prueba");
111:    close(fd1);close(fd2);
112:    exit(0);
113: }
```



De las siguientes afirmaciones, elija la correcta y justifique su elección.

- A El programa produce un error porque el *printf* quiere escribir "prueba" en el fichero "input.txt" que está abierto como de sólo lectura.
- B El programa imprime "prueba" por pantalla.

F. Ninguna de las anteriores, lo que se hace es imprimir "prueba" en el fichero "output.txt".

6. Indíquese qué sucede (¿Se escribe en pantalla?, ¿Se escribe en el fichero "input.txt"? cuando el siguiente programa ejecuta `printf("prueba")` (línea 7). Justifíquese brevemente su respuesta.

```
0  Stdin  
1  input  
2  
3  input  
4  
5  
6
```

1.1: #include "includes.h"
1.2: int main(){
1.3: int fd1= open("input.txt",O_RDONLY); //existe en el directorio actual
1.4: close(STDOUT_FILENO); //equivale a close(1)
1.5: close(STDERR_FILENO); //equivale a close(2)
1.6: dup (fd1);
1.7: printf("prueba");
1.8: exit(0);
1.9: }

OJO! No se escribe "prueba" en el fichero "input.txt" porque tiene permisos de solo lectura.

7. Para las siguientes sentencias indíquese con qué método(s) de entrada/salida (E/S) se relaciona (polling, E/S por interrupciones o E/S por DMA):

- Se genera una interrupción al finalizar una transferencia.

DMA y interrup.

- Es la CPU la que realiza la transferencia de datos cuando estos están listos.

Polling e interrup.

- La CPU debe suministrar al controlador el tipo de operación (lectura o escritura), la dirección de transferencia de datos y cantidad de bytes a transferir.

DMA (los transfiere al controlador DMA)

- Durante la transferencia de varios datos, la sincronización se consigue al preguntar la CPU al dispositivo si está listo.

Polling

8. Si vamos a transferir 1Mbyte a un dispositivo de E/S. ¿Quién generará más interrupciones durante dicha operación?. ¿Un controlador que use E/S programada (polling), otro que use E/S por interrupciones, u otro que use DMA?. Justifique brevemente la respuesta.

9. Tengo un disco de 2 caras, con 80 cilindros y un total de 18 sectores por pista. Cada sector contiene 512 bytes (0.5Kbytes). Si lo he formateado usando bloques de 2Kbytes, ¿Cuántos bloques son accesibles para el Sistema Operativo?

8. El controlador por interrupciones porque generará una cada vez que se transmite un dato. DMA generará una sola al terminar la transferencia completa y polling no genera interrupciones.



$$2 \cdot 80 \cdot 18 = \text{no sectores} = 2880$$

$$2880 \cdot 0.5 = 1440 \text{ KB}$$

$$1440 / 2 = 720 \text{ bloques accesibles.}$$

TGR Entrada/salida (Sistemas Operativos)

1. Sea el siguiente código

```
#include "includes.h"
int realiza_trabajo(int fd, char *msg){
    write(fd,msg,strlen(msg));
}
int realiza_trabajo_1(int fd){
    realiza_trabajo(fd,"Realizando Trabajo 1");
}
int realiza_trabajo_2(int fd){
    realiza_trabajo(fd,"Realizando Trabajo 2");
}

int main(){
    int fd;
    fd= open("salida.txt",O_WRONLY|O_CREAT, 0664);

    //código de redirección
    int copia = dup(fd);
    close(fd);
    dup(STDOUT_FILENO);

    realiza_trabajo_1(fd);

    //código para restaurar redirección
    close(fd);
    dup (copia);
    close(copia);

    realiza_trabajo_2(fd);
    close(fd);
}
```

Modifíquese dicho código para que todo lo que se escribiría en el fichero “**salida.txt**” durante la llamada a la función **realiza_trabajo_1**, se muestre por la salida estándar (STDOUT_FILENO), pero que la función **realiza_trabajo_2** siga almacenando sus datos en el fichero “**salida.txt**”.

2. Asúmase que tenemos un disco duro con 250 cilindros numerados del 0 al 249, cuya cabeza está sobre el cilindro 100 (atendiendo una petición de acceso previa) y se está moviendo en sentido ascendente (0 → 249). Si en el instante actual la cola de peticiones de acceso a cilindros contiene las peticiones: ⟨99, 102, 125, 29, 247, 95, 110, 90, 198, 10⟩, indíquese en qué orden se atenderán dichas peticiones si se utiliza un algoritmo de planificación:

FCFS o FIFO:	99	102	125	29	247	95	110	90	198	10
SSTF:	99	102	95	90	110	125	198	247	29	10
SCAN:	102	110	125	198	247	99	95	90	29	10
C-SCAN:	102	110	125	198	247	10	29	90	95	99

3. En un momento dado, la cola de peticiones de E/S para acceder a cilindros de un disco duro contenía las peticiones: ⟨88, 33, 18, 90⟩. Desconocemos el cilindro que estaba siendo accedido antes de estas peticiones, pero sabemos que dichas peticiones fueron atendidas en el orden siguiente ⟨88, 33, 18, 90⟩. Indíquese si los siguientes algoritmos pudieron haber sido (o no) usados para la planificación del acceso al disco. Razónese brevemente la respuesta:

algor.	sí/no	breve justificación de la respuesta
FIFO:	sí	Peticiones en estricto orden de llegada
SSTF:	no	El ⟨90⟩ debería haber sido la segunda petición (tras atender ⟨88⟩)
SCAN:	sí	El ascensor bajaba inicialmente ⟨88, 33, 18⟩, y al subir atiende ⟨90⟩

4. Suponiendo que el fichero “fichero.txt” contiene: "ABCDEFGHIJKLMNPQRSTUVWXYZ\n", y que el siguiente código no produce ningún error de ejecución:

```
#include "includes.h"

int lee1(int fd, char BUF[]) {
    read(fd,BUF,1);
    read(fd,BUF,3);
}
int lee2(int fd, char BUF[]) {
    pread(fd,BUF,1,0);
    pread(fd,BUF,3,0);
}
int lee3(int fd, char BUF[]) {
    struct iovec iov[2];
    iov[0].iov_base= BUF;    iov[0].iov_len= 1;
    iov[1].iov_base= BUF+1;  iov[1].iov_len= 3;
    readv(fd,iov,2);
}

int main(){
    int fd = open("fichero.txt",O_RDONLY);
    char BUF[6]= {'*', '*', '*', '*', '*', '\0'};

    #ifdef CASO_A
        lee1(BUF);
    #endif
    #ifdef CASO_B
        lee2(BUF);
    #endif
    #ifdef CASO_C
        lee3(BUF);
    #endif
}
```

<code>#ifdef CASO_A lee1(BUF); #endif</code>	<code>#ifdef CASO_B lee2(BUF); #endif</code>	<code>#ifdef CASO_C lee3(BUF); #endif</code>
--	--	--

```
    printf("%s",BUF);
    close(fd);
}
```

Muéstrese qué se imprime por pantalla en el “printf” si se llama a `lee1()`, `lee2()`, o `lee3()` respectivamente.

- `Lee1()` imprime por pantalla: `BCD**`, pues el primer `read` hace `BUF[0]=[A]`, pero a continuación, el segundo `read` lee `[BCD]`, y sobreescribe `BUF[0..2]=[BCD]`.
- `Lee2()` imprime por pantalla: `ABC**`, pues el primer `read` lee la primera 'A' (`BUF[0]=[A]`), pero a continuación, el segundo `pread` lee `[ABC]` (de nuevo desde la posición 0), y sobreescribe `BUF[0..2]=[ABC]`.
- `Lee3()` imprime por pantalla: `ABCD*`, pues `readv` hace dos lecturas. En la primera, se lee el primer byte del fichero y se guarda en `BUF` (`BUF[0]=[A]`). En la segunda lectura `readv` lee `[BCD]`, y coloca ese contenido en `BUFF+1`, esto es, hace `BUF[1..3]=[BCD]`.

5. Dado el siguiente programa...

```
11: #include "includes.h"
12: int main(){
13:     int fd1= open("input.txt",O_RDONLY); //existe en el directorio actual
14:     int fd2= open("output.txt",O_WRONLY|O_CREAT,0664); //se crea el fichero
15:     dup (fd1);
16:     close(STDOUT_FILENO);           // equivale a close(1)
17:     close(STDERR_FILENO);          // equivale a close(2)
18:     dup (fd2);
19:     close(STDIN_FILENO);           // equivale a close(0)
110:    printf("prueba");
111:    close(fd1);close(fd2);
112:    exit(0);
113: }
```

De las siguientes afirmaciones, elija la correcta y justifique su elección.

- A El programa produce un error porque el *printf* quiere escribir "prueba" en el fichero "input.txt" que está abierto como de sólo lectura.
- B El programa imprime "prueba" por pantalla.

C. Habiendo cerrado "1" (línea 6) y "2" (línea 7) el *dup(fd2)* de la línea 8 duplica la referencia del descriptor "fd2" que apunta al fichero "output.txt" , y coloca dicho duplicado en la entrada "1" de la Tabla de Descriptores de ficheros abiertos. Esto es una "redirección" del fichero "output.txt" a la salida estándar. Por ello el *printf* de la línea 10 imprime "prueba" en el fichero "output.txt".

6. Indíquese qué sucede (¿Se escribe en pantalla?, ¿Se escribe en el fichero "input.txt"? cuando el siguiente programa ejecuta `printf("prueba")` (línea 7). Justifíquese brevemente su respuesta.

```
1.1: #include "includes.h"
1.2: int main(){
1.3:     int fd1= open("input.txt",O_RDONLY); //existe en el directorio actual
1.4:     close(STDOUT_FILENO);           //equivale a close(1)
1.5:     close(STDERR_FILENO);          //equivale a close(2)
1.6:     dup (fd1);
1.7:     printf("prueba");
1.8:     exit(0);
1.9: }
```

El programa no escribe nada por pantalla ni en el fichero "input.txt". El motivo radica en que el "printf" quiere escribir "prueba" en el fichero "input.txt", que está abierto como de sólo lectura. Nótese que esto se debe a que en las líneas 4 y 6 se trata de redireccionar la salida estándar al fichero "input.txt" (pero no es posible escribir en él por estar abierto en modo de sólo lectura)

7. Para las siguientes sentencias indíquese con qué método(s) de entrada/salida (E/S) se relaciona (polling, E/S por interrupciones o E/S por DMA):

- Se genera una interrupción al finalizar una transferencia.

E/S por interrupciones y E/S por DMA.

- Es la CPU la que realiza la transferencia de datos cuando estos están listos.

polling y E/S por interrupciones.

- La CPU debe suministrar al controlador el tipo de operación (lectura o escritura), la dirección de transferencia de datos y cantidad de bytes a transferir.

E/S por DMA

- Durante la transferencia de varios datos, la sincronización se consigue al preguntar la CPU al dispositivo si está listo.

polling

8. Si vamos a transferir 1Mbyte a un dispositivo de E/S. ¿Quién generará más interrupciones durante dicha operación?. ¿Un controlador que use E/S programada (polling), otro que use E/S por interrupciones, u otro que use DMA?. Justifique brevemente la respuesta.

El controlador que use Interrupciones, pues genera una interrupción cada vez que un dato es transferido, mientras que el DMA potencialmente podrán originar sólo una interrupción al final de la transferencia. Por su parte polling no genera interrupciones.

9. Tengo un disco de 2 caras, con 80 cilindros y un total de 18 sectores por pista. Cada sector contiene 512 bytes (0.5Kbytes). Si lo he formateado usando bloques de 2Kbytes, ¿Cuántos bloques son accesibles para el Sistema Operativo?

2 caras x 80 cilindros ==>160 pistas

160 pistas x 18 sectores/pista x 0.5Kb/sector x 0.5 bloques/KB= 160x9x0.5 bloques = 720 bloques