APELLIDOS		NOMBRE		Grupo
DNI		Firma		

- No desgrape las hojas. Conteste exclusivamente en el espacio reservado para ello.
- Utilice letra clara y legible. Responda de forma breve y precisa.
- El examen consta de **6 cuestiones**, cuya valoración se indica en cada una de ellas.
- Recuerde que debe justificar sus cálculos o respuestas para obtener una buena calificación.

1. Considere los programas A y B cuyos archivos de código ejecutable se encuentran en el directorio actual y obtenidos al compilar los códigos fuentes siguientes:

/***Código Programa A.c ***/

```
#include <.........
int main(int argc, char **argv) {
    pid_t pid;

    pid=fork();
    if (pid>0) {
         sleep(10);
    }
    execl("./B",”B”, "1",NULL);
}
```

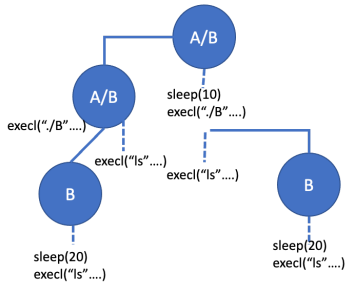
/***Código Programa B.c ***/

```
#include <.........
int main(int argc, char *argv[]) {
    int option;
    pid_t pid;

    if (argc< 2) {
        exit(-1);
     }
    option=atoi(argv[1]);
    pid=fork();
    if (pid==0) {
        sleep(20*option);
    }
    execl("/bin/ls","ls","-la",NULL);
}
```

Indique de forma razonada: **(1,5 puntos = 0,6 + 0,5+0,4)**

1 **a)** Indique el número de procesos que se crean al ejecutar la orden “**./A**” y dibuje el esquema de parentesco que existe entre dichos procesos



Al ejecutar ./A se crean un total de **4 procesos, los 4 procesos creados ejecutan la orden ls.** La secuencia de creación es: 2 procesos son creados al ejecutar el código de A (padre e hijo) cada uno de ellos cambia su imagen de memoria por el código de B, creando a su vez un proceso hijo cada uno al ejecutar el código B

b) Indique el número de procesos que se crean al ejecutar la orden “**./B 1**” y cuantas veces se muestra el listado del directorio actual en el terminal

Se crean **dos procesos, un padre y un hijo**, la función execl() sustituye el código de ambos proceso, pero el hijo queda suspendido durante 20 segundos antes de hacer el execl(). Ambos procesos, padre e hijo, muestran por pantalla el listado del directorio actual al ejecutar la orden “ls -la”

c) Justifique la existencia o no de procesos zombies y huérfanos al ejecutar la orden “./B 1”

El proceso hijo permanece suspendido durante 20 segundos, mientras el padre finaliza su ejecución. Es bastante probable que el proceso hijo se quede huérfano. No es probable que se quede zombie

2. Sea un sistema de tiempo compartido con un planificador a corto plazo que consta de dos colas, una gestionada con FCFS (**ColaF)** y otra gestionada con Round Robin con q=10ut (**ColaR**). Los procesos nuevos y los procedentes de E/S siempre van a la ColaR. La ColaR es la más prioritaria y un proceso **es degradado** a la cola ColaF tras consumir un quantum q en CPU. Los procesos siempre son añadidos/incorporados al final de las colas. La gestión intercolas es por prioridades expulsivas. Todas las operaciones de E/S se realizan en un único dispositivo con política de servicio FCFS. En este sistema se solicita ejecutar el siguiente grupo de trabajos:

Proceso	Instante de llegada	Ráfagas de CPU y E/S
A	0	20 CPU + 30 E/S + 30 CPU+10 E/S+10 CPU
B	5	20 CPU + 10 E/S + 20 CPU
C	10	40 CPU + 20 E/S + 10 CPU

(1,8 puntos=1,2+0,6)

a) Represente mediante diagrama temporal la ocupación de CPU, del periférico de E/S y de las colas de preparado. Por coherencia la tabla representa intervalos de 10ut

T	ColaR	ColaF	CPU	Cola E/S	E/S	Evento
0	~~A20~~		A20			
10	C40 ~~B20~~	A10	B20			
20	~~C40~~	B10,A10	C40			
30		C30,B10,~~A10~~	A10			
40		C30,~~B10~~	B10		A30	
50		~~C30~~	C30	B10	A20	
60			C20	B10	A10	
70	~~A30~~	C10	A30		B10	
80	~~B20~~	A20,C10	B20			
90		B10,A20,~~C10~~	C10			
100		B10,A20	A20		C20	
110		B10	A10		C10	
120	~~C10~~	B10	C10		A10	
130	~~A10~~	B10	A10			FIN C
140		~~B10~~	B10			FIN A
150						FIN B
160						

b) Indique tiempo de Retorno y Tiempo de Espera para cada proceso

	A	B	C
Tiempo Retorno	140 - 0 = 140	150 - 5 = 145	130- 10 = 120
Tiempo Espera	40	75	50

El tiempo de retorno corresponde al intervalo de tiempo que el proceso está en el sistema→ Se estima mediante la diferencia entre el Instante de llegada- Instante de finalización de un proceso.
Tiempo de Espera corresponde al tiempo que los procesos pasan en la cola de preparados esperando que se les asigne la CPU

3. El código mostrado intenta modelar el acceso a las aulas para la realización de exámenes de forma segura durante el COVID-19. Se trata de controlar tanto la entrada como la salida de las aulas, utilizando una solución de sincronización **basada en semáforos.** Cada alumno convocado a examen está representado por un hilo que ejecuta la función `hilo_alumno()`, y realiza las siguientes acciones, que han sido implementadas en las funciones indicadas:

1. Esperar fuera del edificio hasta la hora en la que se ha convocado, `ESPERE_FUERA_HORA_CONVOCADO()`
2. Ir al aula asignada usando el acceso correspondiente, `IR_AL_AULA_ASIGNADA()`
3. Entrar en el aula y sentarse, `ENTRAR_AULA();`
4. Hacer el examen. es decir, lo que estás haciendo ahora, `HACER_EXAMEN()`
5. Dejar el examen en la mesa, salir del aula y dirigirse a la salida del edificio asignada, `SALIR_AULA()`

```
1  #include <semaphore.h>
2  #define AFORO 15
3  #define ALUMNOS 20
4
5  sem_t sem_E, sem_S, sem_A;
6  void *hilo_alumno(void *param) {
7
8    ESPERE_FUERA_HORA_CONVOCADO();
10   sem_wait(&sem_A);
11   IR_AL_AULA_ASIGNADA();
12
13   sem_wait(&sem_E);
14   ENTRAR_AULA();
15   sem_post(&sem_E);
16
17   HACER_EXAMEN();
18
19   sem_wait(&sem_S);
20   SALIR_AULA();
21   sem_post(&sem_S);
22
23   sem_post(&sem_A);
24 }
```

```
25 int main(void) {
26
27    pthread_t th[ALUMNOS];
28    pthread_attr_t attr;
29    int n;
30
31    sem_init(&sem_E, 0, 2);
32    sem_init(&sem_S, 0, 1);
33    sem_init(&sem_A, 0, AFORO);
34
35    pthread_attr_init(&attr);
36    for (n=0; n<ALUMNOS; n++) {
37      pthread_create(&th[n], &attr, hilo_alumno, NULL);
38   }
39
40   for (n=0; n<ALUMNOS; n++) {
41      pthread_join(th[n], NULL);
42   }
43
44   PROFESOR_RECOGE_EXAMENES();
45 }
```

Utilice los semáforos (`sem_E, sem_S, sem_A`) e hilos ya propuesto en el código y resuelva las cuestiones.

(1,5 puntos =0,4+ 0,6 + 0,5)

3

a) Escriba el código necesario en la línea 40 (*main()*) para asegurar la adecuada finalización de todos los hilos creados y que el profesor recoja los exámenes (función `PROFESOR_RECOGE_EXAMENES()`) cuando todos los alumnos han salido de clase. Justifique brevemente la necesidad del código que ha añadido.

```
for (n=0; n<ALUMNOS; n++) {
   pthread_join(th[n], NULL);
```

No es adecuado utilizar pthread_exit() ya que se debe ejecutar PROFESOR_RECOGE_EXAMENES.

b) Inicialice *sem_E* y *sem_S* en *main()* y escriba en función `hilo_alumno()` las operaciones necesarias sobre dichos semáforos de manera que se cumpla un acceso al aula (función `ENTRAR_AULA()`) limitado a 2 alumnos al mismo tiempo (concurrentemente) y la salida (función `SALIR_AULA()`) a un alumno cada vez.

Inicializado a 2 el semáforo sem_E y a 1 sem_S,y las operaciones wait(P) y post (V) sobre dichos semáforos cómo protocolo de entrada y salida a *ENTRAR_AULA()* y *SALIR_AULA()* se cumple el acceso.

```
13   sem_wait(&sem_E);
14   ENTRAR_AULA();
15   sem_post(&sem_E);
```

```
19 sem_wait(&sem_S);
20   SALIR_AULA();
21   sem_post(&sem_S);
```

c) Inicialice *sem_A* y escriba en función `hilo_alumno()` las operaciones necesarias sobre dicho semáforo de manera que se cumpla el aforo máximo del aula (definido por la constante `AFORO`). En caso de que se complete el aforo, el alumno debe esperar fuera del edificio a que salga algún alumno.

Hay que proteger todo el bloque de acciones, desde que entra hasta que se sale del aula, tal como se muestra en el código (en Rojo) con operaciones sobre sem_A incializado a AFORO. No es correcto limitar solo *HACER_EXAMEN()* ya que provocaría que los alumnos puedan entrar en el aula con el aforo ya completo, y que esperen dentro del aula

4. Sea el siguiente código que emplea llamadas al sistema relacionadas con el sistema de archivos:

(1.4 puntos =0.6+0.6+0.2)

4

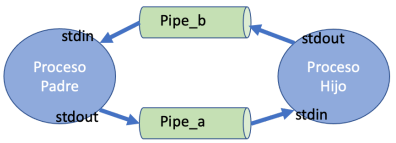
```
1.  #include <......
2.
3.  int main(void) {
4.    int pipe_a[2], pipe_b[2];
5.    unsigned char c;
6.    pipe(pipe_a);
7.    pipe(pipe_b);
8.    if (fork() == 0) {
9.      dup2(pipe_a[0], STDIN_FILENO);
10.     dup2(pipe_b[1], STDOUT_FILENO);
11.     close(pipe_a[0]); close(pipe_a[1]);
12.     close(pipe_b[0]); close(pipe_b[1]);
13.     while (read(STDIN_FILENO, &c, 1)) {
14.       write(STDOUT_FILENO, &c, 1);
15.     }
16.   } else {
17.     dup2(pipe_b[0], STDIN_FILENO);
18.     dup2(pipe_a[1], STDOUT_FILENO);
19.     close(pipe_a[0]); close(pipe_a[1]);
20.     close(pipe_b[0]); close(pipe_b[1]);
21.     fprintf(stdout, "Galleta ");
22.     while (read(STDIN_FILENO, &c, 1)) {
23.       write(STDERR_FILENO, &c, 1);
24.       write(STDOUT_FILENO, &c, 1);
25.     }
26.   }
27. }
```

a) Suponga que al ejecutar el código todas las llamadas se realizan con éxito. Indique el contenido de las tablas de descriptores de archivos para los procesos creados (padre e hijo) **justo tras la ejecución** de la línea indicada:

Tabla del Proceso **padre** línea **20**	
0	pipe_b[0]
1	pipe_a[1]
2	STDERR
3	
4	pipe_b[0]
5	
6	

Tabla del Proceso **hijo** línea **11**	
0	pipe_a[0]
1	pipe_b[1]
2	STDERR
3	
4	
5	pipe_b[0]
6	pipe_b[1]

b) En el código se crean unas tuberías con las que interactúan los procesos creados estableciendo una comunicación entre ellos. Realice un diagrama que muestre dicha comunicación



c) Describa qué hacen los procesos que se crean al ejecutar dicho programa desde un terminal, especificando la información que comparten o se transmiten y si se observa algún resultado en el terminal.

Ambos procesos padre e hijo, leen (read) la información que les llega por stdin carácter a carácter y la escriben (write) en stdout configurando un bucle de comunicación. Se puede decir que redirigen su entrada a su salida. Como las entradas y salida de ambos procesos están redirigidas a tuberías, reenvían todo lo que les llega de un tubo al otro.
El padre, antes de entrar en el bucle (While) escribe en stdout (pipe_a) el texto "Galleta", este texto circulará indefinidamente de un proceso a otro caracter a caracter. Observando el código del padre, vemos que dentro del bucle escribe todos los caracteres también en stderr, que no está redirigida, por lo como resultado se mostrará la palabra "Galleta " en el terminal escrita indefinidamente.

5. Una partición con capacidad de 128MBytes se formatea con una versión de MINIX y las siguientes características:

- Nodos-i con un tamaño de 32 Bytes con 7 punteros directos a zonas, 1 indirecto y 1 doblemente indirecto
- Punteros a zona de 16 bits (2Bytes)
- Entradas de directorio de 16 Bytes (14 Bytes para el nombre y 2 Bytes para el nodo-i)
- **1 Bloque = 2 KBytes; 1 Zona = 1 Bloque**

(1,8 puntos = 1,2 +0.6)

5

a) Indique de forma justificada los tamaños de cada elemento de la cabecera al formatearlo para almacenar 8192 nodos-i (8K nodos-i), así como el número de la primera zona de datos y el total de zonas de datos

Arranque	Super bloque	Mapa de bits de Nodos-i	Mapa de bits de Zonas	Nodos- i	Zonas de datos

1 Bloque de Arranque, 1 Super Bloque, 1 Bloque Mapa Nodos-i, 4 Bloque Mapa zonas y 128 Bloques para los Nodos-i.

Justificación:

Para Mapa de nodos-i es necesario un bit por cada Nodo-i, ajustando el tamaño a un número exacto de bloques

MapaNodos-i → 8Knodos-i/ 2K*8 = $2^3\ 2^{10} / 2^1\ 2^{10}\ 2^3 < 1$ → **1 Bloque**

Para Mapa de Zonas es necesario un bit por cada zona de la partición, ajustando a un número exacto de bloques

Nº máximo de zonas = 128MBytes /2KB = $2^{27} / 2^1\ 2^{10}$ = **64 K zonas**

Mapa Zonas → Nº de Zonas/ Tamaño en bits de la zona = 64 K/ 2K*8=$2^6\ 2^{10} / 2^1\ 2^{10}\ 2^3 = 2^2$ =**4 Bloques**

Bloques Nodos-i → 8K nodos-i * 32 Bytes /2KBytes = $2^3\ 2^{10}\ 2^5 / 2^1\ 2^{10}$ = $=2^7$ = **128 Bloques**

Cabecera 1+1+1+4+128= 135 Bloques

Como un bloque= a un zona → El primer bloque de la Zona de datos será el 135

Zonas de Datos =64Kzonas - 135zonas de la cabecera =65536-135=65401 zona de datos

b)Suponga que este sistema de ficheros tiene un directorio raíz "/" en el que se han creado 2 subdirectorios (*/dir1, /dir2*) y 5 ficheros regulares de 20KBytes (*/fich1, /fich2, /fich3, /fich4, /fich5*). Indique de forma justificada los bloques de la cabecera y zonas de datos a los que se requiere acceder para leer los 20KBytes del fichero regular */fich1*.

1. Lectura bloque de la cabecera en el que está el inodo 1 del raíz (se trata del primer bloque que contiene nodos-i) =>se identifica el número de zonade datos (puntero directo del nodo-i=1) que contiene las entradas de dicho directorio. El directorio "/" contiene un total de 9 entradas (/., /.., /dir1, /dir2, /fich1,/fich2, /fich3, /fich4,/fich5) → 9 entradas * 16Bytes <2KBytes → el directorio"/" ocupa una zona
2. Lectura de la zona de datos del raíz con el contenido del directorio / y por tantos de todas sus entradas con Nombre y Nodo-i de cada uno de los archivos. Se consulta la entrada /fich1 y su número de nodo-i =>se identifica el inodo del fichero que queremos leer
3. Lectura del bloque de la cabecera que contiene el inodo de /fich1→ esta en el bloque 1º que hemos leido => identificamos los 7 punteros directos a zona a leer => identificamos el puntero indirecto a zona y los punteros de 2 nivel

Bloque y zonas que se requiere acceder para leer los 20K de /fich1:

-1 bloque de la cabecera con el nodo-i del / + 1 zona de datos del raíz + 1 bloque de la cabecera con el nodo-i de /fich1(este bloque es el mismo que el del raíz) + 7 zonas de datos (14KBytes) apuntadas por los punteros directos del inodo de /fich + 1zona de datos (con metadatos) apuntada por el puntero indirecto del inodo de /fich1 y que contiene el puntero de 3 zonas de datos +3 zonas de datos del archivo /fich1=

1Bloque con nodos-i (nodo-i del / y del /fich1) + 1 zona del /+ 1 Zona de /fich con metadatos + 10 zonas de datos de /fich1

6. Sea un sistema con Memoria Virtual, paginación por demanda con reemplazo LOCAL, tamaño de página de 64KBytes y un total de 8192 marcos (2^{13}marcos). En dicho sistema se están ejecutando los procesos A y B con espacios lógicos que ocupan 5 páginas (en concreto las páginas de la 0 a la 4). El sistema ha reservado un total de tres marcos al proceso A (0x1,0x3,0x4) y dos marcos al proceso B (0x0, 0x2). Las tablas de páginas (los tiempos están en decimal) de los procesos A y B en el instante t=10 contienen:

Tabla Páginas Proceso A (en t=10)

	marco	Bit validez	Bit referencia	Instante última referencia	Instante carga en memoria
0	0x3	v	1	10	1
1	-	i	0	3	2
2	0x1	v	1	8	6
3	-	i	0	2	8
4	0x4	v	0	5	4

Tabla Páginas Proceso B(en t=10)

	marco	Bit validez	Bit referencia	Instante último referencia	Instante carga en memoria
0	-	i	0	4	7
1	0x2	v	1	9	9
2		i	0	1	3
3	0x0	v	0	6	5
4	-	i	1	0	1

(2,0 puntos = 0,4 + 0,6 + 0,6 + 0,4)

6

a) Describa el formato de la dirección física con número de bits para cada elemento:

Tamaño de página = 64KBytes = 2^{16} → son necesarios 16 bits tanto para desplazamiento de página como de desplazamiento de marco.

Número de marcos = 8192 marcos = 2^{13}marcos→ son necesarios 13 bits para el número de marco

Formato de la dirección Física de 13+16 =29bits

b28 ----------b16,b15----------b0

Número de marco ----13 bits----	Desplazamiento ---16 bits-----

Utilizando la información de las tablas de páginas, indique en t=10 la dirección física que corresponde a la dirección lógica **A:0x00047A2**

DL= A: 0x00047A2 → Número de página 0x000 con desplazamiento 47A2

La página 0(0x000) está ubicada en el marco 0x3 y su bit de validez está a válido → DF = 0x347A2,= 0x000347A2

b) Suponga que en los instantes t=11 y t=12, la CPU emite las direcciones lógicas **A: 0x0031AB2** y **B:0x0031AB2**. Indique cómo quedarían los contenidos de las tablas de páginas de los procesos A y B después de emitirse dichas direcciones lógicas si se utiliza un algoritmo **LRU con reemplazo LOCAL** (escriba sólo lo que cambie respecto a t=10)

Tabla de Páginas del Proceso A

marco	Bit validez	Bit referencia	Instante última referencia	Instante carga en memoria
0x4	v	1	11	11
-	i	-	-	-

Tabla de Páginas del Proceso B

marco	Bit validez	Bit referencia	Instante última referencia	Instante carga en memoria
0x0	v	1	12	5
-	i	1	0	1

Justifique brevemente su respuesta

DL A:0x0031AB2 → Página 0x003; desplazamiento 1AB2 → La página 3 del proceso A no está ubicada en memoria → Fallo de página al aplicar un algoritmo LRU Local se escoge como víctima la página del proceso A que hace más tiempo que no ha sido referenciada → Pagina víctima 0x4 --> La nueva página se ubica en el marco 0x4 y se actualizan tanto su bit de validez cómo su bit de referencia e instantes de carga.

DL B:0x0031AB2 → Página 0x003; desplazamiento 1AB2 → La página 3 está ubicada en memoria en el marco 0x0→ Por tanto no hay Fallo de página ni hay que aplicar un algoritmo de búsqueda, pero sí que hay que actualizar los datos de la tabla de página del proceso B→ actualizar el bit de referencia y el instante de la última referencia

c) Suponga que en t=10 se está utilizando un algoritmo de **2ª Oportunidad con reemplazo LOCAL** e indique (en hexadecimal) un rango de al menos 1024 direcciones lógicas consecutivas del proceso B que podrían ser emitidas por la CPU a partir del instante t=10 sin causar fallos de página.

Dado que el proceso B tiene ubicada en memoria tanto la página 1 como la 3 cualquier conjunto de DL de la página 1 ó 3 no producirían fallos de página.
Por lo tanto las DL que podrían emitir la CPU son:

- desde la 0x0030000 hasta 0x003FFFF (este rango es de 64K direcciones diferentes)
- desde la 0x0010000 hasta 0x001FFFF (este rango es de 64K direcciones diferentes)

d) Indique el tamaño de la fragmentación interna y externa que podría llegar a producirse al ejecutar un proceso P cuyo tamaño de espacio lógico es de 4KBytes en este sistema.
Dado que el proceso P tiene un tamaño de 4KBytes → Ocupa una único marco en Memoria 64KBytes → 64KBytes-4KBytes =60KBytes de fragmentación interna
No existe Fragmentación externa en Paginación