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 9 cuestiones, cuya valoración se indica en cada una de ellas.**
- **Recuerde que debe justificar sus cálculos o respuestas para obtener buena calificación**

1. Utilizando llamadas al sistema para la creación de procesos y ejecución de programas, escriba el código necesario para que el resultado de la ejecución de dicho código sea la escritura en la salida estándar siguiente:

1. El listado del directorio actual (orden del shell "ls -la")
2. La frase "***Listado de directorio completado***"

(1,0 ptos)

```
1  /***listado.c ***/
2  #include "todas_las_cabeceras_necesarias.h"
3
4  main() {
5    pid_t pid;
     int status;
     pid=fork();
     if (pid==0) {
         if (execl("/bin/ls", "ls" , "-la" , NULL)<0){
            fprintf(stderr,"error en el ls\n");
            exit(-1);
          }
      } else {
          wait(&status);
          printf("Listado de directorio completado\n");
      }
}
```

2. Sea un sistema de tiempo compartido con planificador a corto plazo y una única cola de procesos preparados. Considere los procesos A, B y C, que llegan en los instantes 0, 1 y 2 respectivamente:

A(t=0)	3CPU+2E/S+1CPU

B(t=1)	1CPU+1E/S+1CPU

C(t=2)	2CPU+1E/S+1CPU

Rellene los siguientes diagramas para cada uno de los planificadores indicados y calcule los valores que se solicitan. La cola de E/S es FCFS. Para los casos en que en un mismo instante se produzcan varios eventos y el algoritmo de planificación no pueda decidir, el orden es: proceso nuevo, proceso que viene de E/S y fin de quantum.

(1,2 ptos = 0,6+0,6)

a) SRTF (Shortest-Remaining-Time-First)

T	Preparados	CPU	Cola E/S	E/S	Evento
0		A(2)			Llega A
1	A	B(0)			Llega B
2	C	A(1)		B	Llega C
3	B,C	A(0)			
4	C	B(0)		A	
5		C(1)		A	Fin B
6	A	C(0)			
7		A(0)		C	
8		C(0)			Fin A
9					Fin C
10					

	A	B	C	*Tiempo Medio*
Tiempo Retorno	8	4	7	19/3 = 6,33 u.t.
Tiempo Espera	2	1	3	6/3 = 2 u.t.
Utilización de CPU	9/9 = 100%			

b) RR (Round-Robin) q=1 u.t.

T	Preparados	CPU	Cola E/S	E/S	Evento
0		A(2)			Llega A
1	A	B(0)			Llega B
2	C	A(1)		B	Llega C
3	A,B	C(1)			
4	C,A	B(0)			
5	C	A(0)			Fin B
6		C(0)		A	
7			C	A	
8		A(0)		C	
9		C(0)			Fin A
10					Fin C

	A	B	C	*Tiempo Medio*
Tiempo Retorno	9	4	8	21/3 = 7 u.t.
Tiempo Espera	3	1	3	7/3 = 2,33 u.t.
Utilización de CPU	9/10 = 90%			

3. Un sistema maneja direcciones lógicas de 16 bits y soporta paginación pura (sin memoria virtual). El tamaño de página es de 4096 Bytes y las direcciones físicas de 20 bits.
Un programa *X* para este sistema contiene 13004 Bytes de código y 6900 Bytes de variables. Durante la compilación y enlazado de X se añade la biblioteca B con 5004 Bytes de código, 1648 Bytes de variables globales y 796 Bytes de constantes. Cuando se ejecuta el programa X, el sistema asigna al proceso el número máximo de páginas que permite la dirección lógica, ubica código, constantes y variables en páginas distintas y separa también los elementos propios del programa (X) y los de las bibliotecas (B). El compilador dedica a *stack* los 8 KBytes de las direcciones más altas del espacio lógico y el resto a *heap*.

(1,0 ptos = 0,3+ 0,3 + 0,2 + 0,2)

3

a) ¿En total, cuántas páginas tendrán permiso de ejecución (bit x) en el mapa de un proceso *P* que ejecute *X*?

6 páginas: cuatro de código de *X* y dos páginas de código de *B*
Con código de X : 13004Bytes/4KBytes = 13004/4096= 3,174 páginas → 4 páginas
Con código de B : 5004Bytes/4KBytes = 5004/4096= 1,22 páginas → 2 páginas

b) ¿Cuántas páginas de *P* tendrán permiso de escritura?

El número máximo de páginas del proceso es 16 páginas con Direc Lógicas de 16 bits y páginas de 4KBytes → 16- 12 = 4 → 2^4= 16 páginas

16 páginas – 6 páginas de código – 1 (constantes de *B*) = 9 páginas

Si no contamos las páginas no asignadas, tendríamos las páginas de variables (2+1) y las de pila (2) es decir 5 páginas.

c) ¿Qué rango de direcciones lógicas de *P* ocupará la pila?

de 0xE000 a 0xFFFF
con 8KBytes tenemos un rango de 0 a 0x1FFF (13 bits) → como está en las direcciones más altas, la última dirección es 0xFFFF, por lo que la primera será 0xFFFF-0x1FFF=0xE000

d) ¿Cuántas páginas estarán disponibles para *heap?*

16 -(6 código + 3 variable + 1 constante +2 stack) = 4 páginas

4. El siguiente programa (incompleto) se basa en el conocido "juego de la vida". Hay una matriz **`m`** de "células" (caracteres), que pueden estar "vivas" (carácter '*') o "muertas" (carácter en blanco). A cada iteración, cada célula evoluciona (nace o muere), en función del número de células vecinas vivas. Existe un hilo `Celula` por cada elemento de la matriz, que lo hace evolucionar. Recibe como parámetro un puntero al elemento de matriz m que le corresponde. Igual que para el programa "matrix" de la práctica de hilos, un hilo `Refrescador` visualiza periódicamente, de forma indefinida, el contenido de la matriz `m` en pantalla.

Conteste, justificando sus respuestas, a las siguientes cuestiones: **(1,1 ptos = 0,25 + 0,25 + 0,2 + 0,2 + 0,2)**

```
1  #include "Los_necesarios.h"
2  #define FILS 10
3  #define COLS 10
4  #define ITERACIONES 20
5
6  char m[FILS][COLS];
7  pthread_attr_t atrib;
8  pthread_t  hilo_celula[FILS][COLS];
9  pthread_t  hilo_refrescador;
10
11 int NumVecinos(char *ptr){
12  /*Lee 8 células vecinas a la dada, en
13 la matriz m y cuenta cuántas son '*'.
14 Se omite por simplicidad. */
15 }
16
17 void *Celula(void *ptr){
18   char* pchar= (char*) ptr;
19   int i, vec;
20   for(i=0; i<ITERACIONES; i++){
21     vec= NumVecinos(pchar);
22     if(*pchar=='*'){ //Célula viva
23        if ((vec!=2)&&(vec!=3))
24          *pchar=' '; //Morir
25        }else //Célula muerta
26          if (vec==3)
27            *pchar='*'; //Nacer
28     usleep(1000000); //Esperar 1seg
29   }
30 }
31 void *Refrescador(void *ptr){
32   //(se omite por simplicidad)
33 }
34
35 int main()
36 {int fil, col;
37
38  //Inicio aleatorio de células
39  for(fil=0; fil<FILS; fil++)
40   for(col=0; col<COLS; col++)
41    m[fil][col]=(rand()%10>4)?'*':' ';
42
43  pthread_attr_init(&atrib);
44
45  //Crear un hilo célula por elemento
46
47  //Crear hilo refrescador de pantalla
48  //(se omite por simplicidad)
49
50  //Esperar fin de hilos célula
51
52 }
```

a) Complete el código que en la línea 45 crearía un hilo célula para cada uno de los elementos de la matriz m, de forma que todos ellos funcionen concurrentemente. Haga uso exclusivamente de las variables y funciones que ya han sido definidas en el programa.

```
for(fil=0; fil<FILS; fil++)
 for(col=0; col<COLS; col++)
  pthread_create(&hilo_celula[fil][col],
                 &atrib,Celula,&m[fil][col]);
```

b) Complete el código que en la línea 50 realizaría la espera de todos los hilos célula.

```
for(fil=0; fil<FILS; fil++)
  for(col=0; col<COLS; col++)
    pthread_join(hilo_celula[fil][col],NULL);
```

c) ¿Cuál es el número máximo de hilos que pueden llegar a ejecutarse concurrentemente? Justifique la respuesta.

102 hilos: 100 hilos Celula (1 por cada elemento de la matriz) + Refrescador + hilo principal main

d) El programa no realiza la espera del hilo Refrescador. Explique qué ocurre con este hilo cuando el programa termina al alcanzar la línea 52.

Cuando el hilo principal (main) termina sin hacer una llamada pthread_exit, el sistema finaliza también la ejecución del resto de hilos del proceso, con lo que el hilo Refrescador finaliza.

e) Defina el concepto "condición de carrera". ¿Se puede dar una condición de carrera en este programa?

Una condición de carrera ocurre cuando varios hilos o procesos manipulan y acceden concurrentemente a los mismos datos y el resultado de la ejecución depende del orden concreto en que se produzcan los accesos. En este programa sí que se puede dar porque los hilos Celula manipulan y acceden concurrentemente a los mismos elementos de la matriz `m` sin ninguna sincronización, por lo que el resultado final puede ser distinto en cada ejecución.

NOTA: Definición de funciones:

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
int pthread_join(pthread_t thread, void **retval);
```

5. Sean tres procesos que se ejecutan concurrentemente y que contienen tres funciones f1(), f2() y f3() como se muestra en la tabla. Antes y después de la ejecución de estas funciones se debe definir un protocolo de entrada (PEx) y salida (PSx), utilizando **EXCLUSIVAMENTE** operaciones sobre dos semáforos A y B, para cumplir con las restricciones requeridas para cada los siguientes casos:

Caso a) f1(),f2() y f3() se tienen que ejecutar en exclusión mutua, pero sin ningún orden definido.

Caso b) La secuencia de ejecución de dichas funciones debe ser la siguiente f3(), f2() y f1().

Caso c) Se pueden ejecutar como máximo dos funciones concurrentemente (sin exclusión mútua).

Caso d) f1() y f2() pueden ejecutarse concurrentemente (sin exclusión mutua), pero f3() tiene que esperar la terminación de f1() y f2().

Proceso 1	Proceso 2	Proceso 3
sleep(5) . PE1 . f1() . PS1	sleep(5) . PE2 . f2() . PS2	sleep(10) . PE3 . f3() . PS3

(1,0 ptos = 0,25 + 0,25 + 0,25 + 0,25)

5 **Complete la tabla.** Indique el valor inicial de los semáforos para cada caso.
La columna "EJEMPLO" muestra cómo completar la tabla y no corresponde a ninguna secuencia lógica. Si no es necesario código en alguno caso de los protocolos de entrada y de salida indíquelo con un guión (como en EJEMPLO para PE3).

	EJEMPLO	Caso a)	Caso b)	Caso c)	Caso d)
Valor Inicial A	0	1	0	2	0
Valor Inicial B	0	---	0	---	0
PE1	P(A)	P(A)	P(A)	P(A)	---
PS1	V(B)	V(A)	---	V(A)	V(A)
PE2	P(B)	P(A)	P(B)	P(A)	---
PS2	P(A)	V(A)	V(A)	V(A)	V(B)
PE3	---	P(A)	---	P(A)	P(A);P(B)
PS3	V(B)	V(A)	V(B)	V(A)	----

6. Sea un sistema de memoria virtual con paginación por demanda y que utiliza un algoritmo **LRU** con reemplazo **LOCAL** de páginas. En este sistema, las direcciones lógicas son de 20 bits y las físicas de 16 bits, con un tamaño de página de 4KB. La memoria principal es de 20 KB (5 marcos) e inicialmente la memoria está vacía. En el sistema existen dos procesos A y B, a los cuales el sistema operativo asigna los marcos 1,2 y 3 al proceso A y los marcos 4,5 al proceso B.

(1,3 ptos = 0,4 + 0,2 + 0,4 + 0,3)

6

a) Indique el formato de las direcciones lógicas y físicas, con el nombre de los campos, sus tamaños y los bits de delimitación de cada campo.

Dirección lógica

Página (8 bits)	*Desplazamiento (12 bits)*

Dirección física

Marco (4 bits)	*Desplazamiento (12 bits)*

b) Obtenga a partir de la siguiente secuencia de direcciones lógicas, la serie de referencias:
(A,2D4B8), (A,2D4B9), (A,2D4EB), (B,2D4EB), (B,2D86F), (B,0B621), (B,0B815), (A,2C4B8), (A,2CA23), (B,946C3), (B,2D1A7), (B,2D1A1), (A,2D4C7), (B,0BA31), (B,0BA32), (A,2CA24), (A,2CA25)

A,2D B,2D B,0B A,2C B,94 B,2D A,2D B,0B A,2C

c) Complete la tabla de evolución del contenido de la memoria principal con la serie de referencias obtenida en el apartado anterior. Indique también los fallos de página producidos.

Marco	A,2D	B,2D	B,0B	A,2C	B,94	B,2D	A,2D	B,0B	A,2C
1	**2D**	2D	2D	2D	2D	2D	2D	2D	2D
2				**2C**	2C	2C	2C	2C	2C
3									
4		**2D**	2D	2D	**94**	94	94	**0B**	0B
5			**0B**	0B	0B	**2D**	2D	2D	2D

FALLOS DE PÁGINA TOTALES : 7

d) Calcule las áreas activas para los dos procesos, con un tamaño de ventana de 6, en el instante final de la secuencia del apartado **b**. ¿Cambiaría el número de marcos asignado a cada proceso, en caso afirmativo cuál sería su propuesta?

El tamaño de las áreas activa para el proceso A es 2 (área activa = {2D,2C}) y para el B es de 3 (área activa {94,0B,2D}). Por lo tanto al proceso A se le ha asignado un marco más de los que necesita. En cambio el proceso B necesita tres marcos diferentes, al acceder a tres páginas diferentes. Por lo que la asignación más óptima sería proceso A: dos marcos y B: tres marcos.

7. El programa Descrip.c genera dos procesos (padre e hijo) y antes de ser ejecutado los archivos A.txt, B.txt contienen 10 y 16 caracteres, respectivamente, como se muestra a continuación:

Descrip.c

```
int main() {
   int fd1, fd2, fd3, p[2];
   char buffer[50];

   fd1 = open( "A.txt", O_RDONLY );
   read(fd1, buffer, 5);
   fd2 = open( "B.txt", O_WRONLY );
   write(fd2, buffer, 5);
   dup2(fd1,5);
   read(5, buffer, 3);
   fork();
   write(fd2, buffer, 3);
   read(fd1, buffer, 1);
   /*write(fd1, buffer, 1);*/ /*comentario*/
   exit(1);
}
```

A.txt

```
1234567890
```

B.txt

```
AAAABBBBCCCCDDDD
```

(1.1 puntos =0.5+0.4+0.2)

7 **a)** Para cada uno de los procesos que se generan indique, el contenido de tabla de descriptores de archivos después de ejecutar su última instrucción y antes de hacer exit()

	Tabla Proceso Padre		Tabla Proceso Hijo
0	*STDIN*	0	*STDIN*
1	*STDOUT*	1	*STDOUT*
2	*STDERR*	2	*STDERR*
3	*A.txt*	3	*A.txt*
4	*B.txt*	4	*B.txt*
5	*A.txt*	5	*A.txt*
6		6	
7		7	

b) Indique los contenidos de A.txt y B.txt, tras finalizar la ejecución de dichos procesos

A.txt	B.txt
1234567890	*12345678678CDDDD*

c) Indique el resultado de ejecutar el código anterior si se descomenta la línea /*comentario*/

Dado que en este proceso el descriptor fd1 se obtiene sólo con permisos de lectura --> Esta llamada al sistema para realizar una escritura no tendría efecto, no podría llevarse a cabo

8. Dado el siguiente listado del contenido de un directorio en un sistema POSIX:

```
drwxr-xr-x   2 pep     alumne         4096 ene  8     2013  .
drwxr-xr-x  11 pep     alumne         4096 ene 10     14:39 ..
-rwsr--r-x   1 pep     alumne      1139706 ene  9     2013  intercanvi
-rw-------   1 pep     alumne       634310 ene  9     2013  f1
-rw-r--r--   1 ana     profes       104157 ene  9     2013  f2
-rw-rw-r--   1 pep     alumne       634310 ene  9     2013  f3
lrwxrwxrwx   1 pep     alumne       634310 ene  9     2013  f4 ->f1
```

Donde el programa `intercanvi` requiere el nombre de dos archivos como argumentos. El programa `intercanvi` intercambia el contenido de dos archivos, accediendo exclusivamente a los archivos y con buffers en memoria, es decir, la orden `intercanvi fich1 fich2`, tiene como resultado que el contenido inicial de `fich1` se copia en `fich2` y el contenido inicial de `fich2` se copia en `fich1`. Rellene la tabla e indique en caso de éxito cuales son los permisos que se comprueban y, en caso de error, cuál es el permiso que falla y porqué.

(1,0 puntos = 0,25 + 0,25 + 0,25 + 0,25)

8

Usuario	Grupo	Orden	¿Funciona?
ana	**profes**	**./intercanvi f1 f2**	**NO**

Justifique

Para ejecutar la orden hace falta permisos de ejecución sobre intercanvi, de lectura y escritura sobre f1 y lectura y escritura sobre f2.
La orden falla, ya que, a pesar de poder ejecutar intercanvi siendo (ana, profes) y al ejecutarla y pasar a ser (pep, profes), al estar el bit de SETUID activado, cuando se comprueban los permisos del fichero f2, el cual es propiedad de (ana, profes), vemos que sólo tenemos permisos de lectura sobre el fichero, ya que entraríamos a comprobar la segunda tripleta de permisos, la de profes.

ana	**profes**	**./intercanvi f3 f4**	**SI**

Justifique

Para ejecutar la orden hace falta permisos de ejecución sobre intercanvi, de lectura y escritura sobre f3 y lectura y escritura sobre f4, que como es un enlace simbólico a f1, realmente los permisos se deben tener sobre f1.
(ana, profes) puede ejecutar intercanvi y pasa a ser (pep, alumne), al cambiar el usuario efectivo al del propietario de los ficheros (pep) puede tanto leer y escribir sobre f3 como leer y escribir sobre f1

pau	**alumne**	**./intercanvi f1 f3**	**NO**

Justifique

Para ejecutar la orden hace falta permisos de ejecución sobre intercanvi, de lectura y escritura sobre f1 y lectura y escritura sobre f3.
Falla al intentar ejecutar intercanvi, ya que los del grupo alumno no tienen permisos de ejecución sobre el fichero

pep	**alumne**	**./intercanvi f2 f1**	**NO**

Justifique

Para ejecutar la orden hace falta permisos de ejecución sobre intercanvi, de lectura y escritura sobre f2 y lectura y escritura sobre f1.
Falla al intentar escribir sobre f2, ya que los del grupo de otros no tienen permisos de escritura

9. Un sistema de archivos minix contiene las siguientes entradas de directorio:

1	.
1	..
3	app
4	tmp
6	media

3	.
1	..
17	doom3

4	.
1	..
13	trash
41	emacs

6	.
1	..
15	home

15	.
6	..
20	luis
23	marta

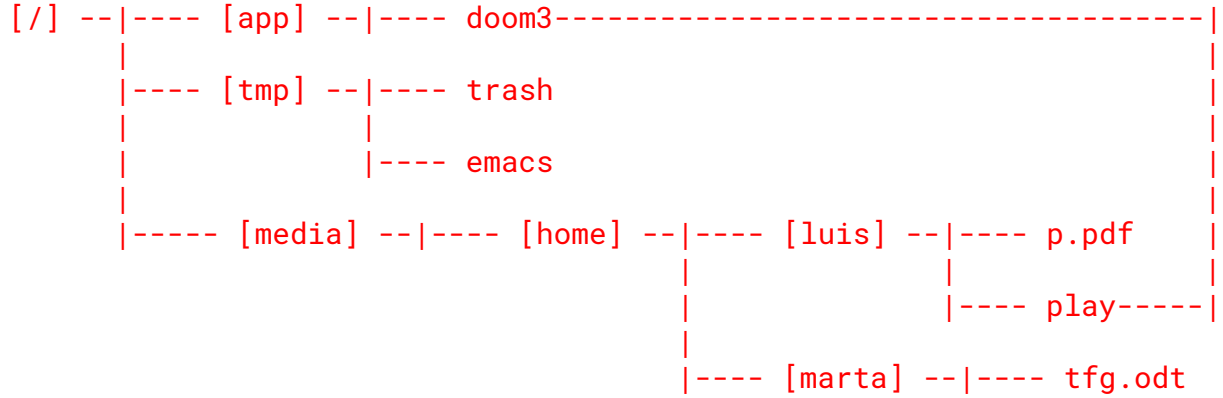
20	.
15	..
50	p.pdf
17	play

23	.
15	..
64	tfg.odt

(1,3 puntos = 0.4+0.2+0.2+0.5)

9

a) Dibuje el árbol de archivos del sistema, indicando para cada archivo su tipo.



[*xxx*] directorio, *xxx* archivo regular

b) Indique de forma justificada si hay algún enlace físico entre archivos regulares

Sí, los archivos */app/doom3* y */media/home/luis/play* son en realidad enlaces físicos a un mismo archivo regular, ambos apuntan al mismo nodo-i, el 17.

c) Indique de forma justificada si existen enlaces físicos entre archivos directorios, si es así exponga en qué casos

Sí, existen enlaces físicos a directorios. Todos los archivos de directorio tienen al menos dos referencias, una desde su propio directorio [.] y otra desde el directorio padre (exceptuando el directorio /).
Así pues ,el directorio [/media/home], que está en el nodo-i 15, tiene una referencia desde su padre [/media] y desde sí mismo [.].

d) Si el sistema de archivos está implementado sobre una partición formateada con 1 zona = 1 bloque = 1KB, con nodos-i de 32Bytes, 16K nodos-i y dispone de 64K bloques de tamaño total. Indique (en KB) el tamaño de la cabecera de dicha partición, exponiendo claramente el tamaño de cada elemento.

La cabecera ocupa: [1 (boot)+1 (superbloque)+n (mapa nodos-i) + m (mapa zonas) + N nodos-i] bloques
En 1 bloque = 1KB hay 1Kx8 bits = 8K bits
n = 16K/8K = 2 bloques
m = 64K/8K = 8 bloques

Un nodo-i ocupa 32B, en 1 bloque = 1KB cabe: 1KB/32B = 32 nodos-i
N = 16K/32 = 512 bloques
Sustituyendo:
Cabecera: 1 + 1 + 2 + 8 + 512 = 524 bloques = 524KB