APELLIDOS	SOLUCIÓN	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 8 cuestiones, cuya valoración se indica en cada una de ellas.
- Recuerde que debe justificar sus cálculos o respuestas para obtener buena calificación.

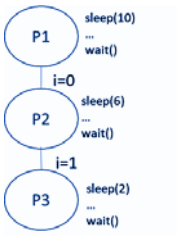
1. Suponga la ejecución del siguiente código sin errores y responda de forma justificada:

(1,2 puntos = 0,6 + 0,4 + 0,2)

```
1  #include <all_needed.h>
2  int main() {
3      int i;
4      pid_t pid;
5
6      for (i=0; i<2; i++) {
7          pid = fork();
8          if (pid != 0) {
9              sleep(10 - 4 * i));
10             break;
11         }
12     }
13     if (i == 2) sleep(2);
14     while (wait(NULL) =! -1);
15     exit(0);
16 }
```

1 **a)** Indique justificadamente el número de proceso que se crean al ejecutarlo y dibuje el esquema de parentesco entre dichos procesos.

Se crean 3 procesos. El proceso inicial (P1) ejecuta el bucle *for()* para i=0, creando un hijo con *fork()* (P2) y seguidamente pasa a suspensión durante 10 segundos, tras lo cual sale del bucle *for()* con break. El hijo (P2) crea un hijo (P3) pero está suspendido 6 segundos. El proceso P3 no ejecuta el bucle for() ya que i = 2 y pasa a suspensión durante 2 segundos. Todos los procesos creados esperan a sus posibles hijos. El esquema de procesos es una cadena



b) Indique de forma justificada, para cada uno de los procesos que se crean, si pueden permanecer zombies o no y por cuánto tiempo podrían estar en dicho estado.

Los posibles estados zombie sólo se dan en los procesos hijo. P2 está en suspensión 6 segundos y su padre 10 segundos, por lo tanto P2 está en estado zombie 4 segundos. P3 está en suspensión 2 segundos y su padre 6 segundos, por lo tanto también está en estado zombie 4 segundos.

c) Considerando un sistema multiprogramado y que el tiempo de ejecución de las instrucciones es despreciable en comparación con el tiempo de sleep ¿cuánto tiempo tarda en ejecutarse el programa?

Dado que los procesos se ejecutan concurrentemente el tiempo de ejecución es el mayor tiempo de suspensión, o sea 10 segundos

2. Sea un sistema de tiempo compartido con un planificador a corto plazo con dos colas de procesos preparados: Cola0 gestionada con un algoritmo Round Robin con quantum q=1ut y Cola1 gestionada con FCFS. La planificación entre colas es por prioridades expulsivas, siendo la Cola0 la más prioritaria. Los procesos nuevos y los que provienen de E/S acceden al sistema por la Cola0, y se degradan a la Cola1 cuando consumen un quantum de tiempo y no acaban su ráfaga de CPU. Si se producen varios eventos en un mismo instante el orden de inserción en cola de los procesos es: nuevo, procedente de E/S y fin de quantum. Existe un único dispositivo de E/S que utiliza una planificación FCFS. A dicho sistema llegan 3 procesos A, B y C, cuyos instantes de llegada y esquema de solicitud de ráfagas de CPU y E/S es el siguiente:

Proceso	Instante de llegada	Ráfagas de CPU y E/S
A	0	2CPU + 3E/S + 4CPU
B	1	1CPU + 1E/S + 2CPU
C	2	4CPU + 2E/S + 1CPU

(1,6 puntos=1,2+0,4)

a) Represente mediante diagrama temporal la ocupación de CPU, del periférico de E/S y de las colas de preparado

T	Cola1	Cola0	CPU	Cola E/S-->	E/S	Evento
0		(A)	A(1)			Llega A
1	A	(B)	B(0)			Llega B
2	A	(C)	C(3)		B(0)	Llega C
3	C, A	(B)	B(1)			
4	B, C		A(0)			
5	B		C(2)		A(2)	
6	B		C(1)		A(1)	
7	B		C(0)		A(0)	
8	B	(A)	A(3)		C(1)	
9	A		B(0)		C(0)	
10	A	(C)	C(0)			Acaba B
11			A(2)			Acaba C
12			A(1)			
13			A(0)			
14						Acaba A
15						
16						

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

	A	B	C
Tiempo Retorno	14 - 0 = 14	10 - 1 = 9	11- 2 = 9
Tiempo Espera	5	5	2

3. En el siguiente trozo de código falta completar las líneas 7,13 y 25, cada una con una instrucción.

(1,2 puntos = 0,4 + 0,4 + 0,4)

```
 1 #include <all_needed.h>
 2
 3 int V = 0;
 4 pthread_t thr1, thr2;
 5
 6 void *thread1( void *ptr ){
 7    /**complete**/
 8    V = 1;
 9    printf("T1 V=%d\n",V);
10 }
11
12 void *thread2( void *ptr ){
13    /**complete**/
14    V = V + 10;
15    printf("T2 V=%d\n",V);
16 }
```

```
18 int main(int argc, char* argv[]){
19    pthread_attr_t atrib;
20    pthread_attr_init(&atrib);
21
22    pthread_create(&thr1,&atrib, thread1,
23 NULL);
24    pthread_create(&thr2,&atrib, thread2,
25 NULL);
26
27    /**complete**/
28    V = V + 100;
...   printf("Main V=%d\n",V);
   }
```

A continuación, se proponen varias posibilidades para completar dichas líneas. Indique para cada una de estas opciones los mensajes que se mostrarán por pantalla tras su ejecución y justifique su respuesta.

a) línea 7 - sleep(1);
línea 13 - sleep(3);
línea 25 - sleep(2);

T1 V=1
Main V=101
El hilo main() termina antes que el hilo thread2 → el proceso finaliza sin que se pueda ejecutar el thread2.

b) línea 7 - sleep(5);
línea 13 - sleep(2);
línea 25 - pthread_exit(0);

T2 V=10
T1 V=1
La llamada pthread_exit() hace que termine el hilo main() en la línea 25 (sin ejecutar el printf () de la línea 27. Pero pthread_exit() hace que finalice el hilo main pero el proceso continua y permite que los otros dos hilos puedan finalizar, y por lo tanto puedan mostrar los valores de V al ejecutar sus respectivos printf(), primero el thread2 y después el thread1.

c) línea 7 - sleep(5);
línea 13 - pthread_join(thr1,NULL);
línea 25 - pthread_join(thr2,NULL);

T1 V=1
T2 V=11
Main V=111
En este caso, thread2 espera a que termine el thread1 y el hilo main espera a que termine el thread2.

4. Sea el siguiente código que hace uso de hilos y semáforos POSIX:

```
#include <semaphore.h>

sem_t sem_A, sem_B;

void *compute(void *param) {

   ...
}
```

```
int main(void) {

   phtread_t th[10];
   phtread_attr_t attr;
   int n;
   sem_t sem_C;

   sem_init(&sem_A, 0, 1);
   sem_init(&sem_B, 0, 0);
   sem_init(&sem_C, 0, 4);

   pthread_attr_init(&attr);
   for (n=0; n<10; n++) {
      pthread_create(&th[n], &attr, compute, NULL);
   }

   ...
}
```

El programador desea emplear los semáforos para resolver diferentes cuestiones que surgirán al ejecutarse concurrentemente los hilos que se crean.

(1,0 puntos = 0.4 + 0.6)

4

a) Indique tres situaciones o tres fines para los que son útiles los semáforos en los entornos de programación concurrente

- Para resolver el problema de la exclusión mutua al acceder a recursos compartidos.
- Para sincronizar procesos/hilos, esto es, establecer un orden o precedencia en la ejecución de los mismo o de parte del código.
- Para controlar el número de procesos/hilos que pueden acceder a un recurso.

b) Atendiendo tanto al ámbito en el que se han declarado los semáforos *semA*, *semB* y *semC*, así como a su inicialización en el código propuesto, y teniendo en cuenta que los semáforos serán empleados por los hilos *th[n]* en la función ***compute*()**, indique qué semáforo *semA*, *semB* o *semC* sería el indicado para cada una de las situaciones descritas en el apartado anterior (a).

- sem_init(&sem_A, 0, 1) ⇒ Control de acceso a zonas de exclusión mutua
- sem_init(&sem_B, 0, 0) ⇒ Sincronización de hilos
- sem_init(&sem_C, 0, 4) ⇒ En principio podría servir para controlar que el número de hilos que acceden a un recurso no sea superior a 4, no obstante la variable *sem_C* es local a la función ***main*()** por lo tanto los hilos ***compute*()** no podrán acceder a ella.

5. Teniendo en cuenta el mecanismo de herencia de procesos en Unix y las llamadas POSIX, responda a los siguientes apartados: **(1.2 puntos =0.6+0.6)**

5

a) Suponga que no se producen errores en las llamadas al sistema y complete el siguiente programa en C con las instrucciones y llamadas al sistema necesarias (una por línea con número subrayado) para que se llegue a ejecutar la siguiente línea de órdenes: **$ cat < f1 2> ferr | grep ".c" > f2**

```
 1  #include <unistd.h>
 2. #include <fcntl.h>
 3. #define readfile O_RDONLY
 4. #define newfile (O_RDWR | O_CREAT | O_TRUNC)
 5. #define mode644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
 6. int main() {
 7.     int pipeA[2];
 8.     int fd1,fd2;
 9.      pipe(pipeA);
10.     if (fork()) {
11.         fd1 = open("ferr", newfile, mode644);
12.         fd2 = open("f1",readfile);
13.           dup2(pipeA[1], STDOUT_FILENO);
14.           dup2(fd1, STDERR_FILENO);
15.           dup2(fd2, STDIN_FILENO);
16.         close(pipeA[0]); close(pipeA[1]); close(fd1); close(fd2);
17.         execlp("cat", "cat", NULL);
18.      } else {
19.         fd1 = open("f2",newfile, mode644);
20.          dup2(pipeA[0], STDIN_FILENO);
21.          dup2(fd1, STDOUT_FILENO);
22.         close(pipeA[0]); close(pipeA[1]);close(fd1);
23.         execlp("grep", "grep", ".c", NULL);
24.         }
25.      }
26.      return 0;
27. }
```

b) Rellene la tabla de descriptores de archivos del proceso padre tras ejecutar la línea 15 y del proceso hijo tras la línea 21. Las tablas deben ser correctas con los requisitos y la implementación de la sección a)

Tabla del Proceso padre en **15**	
0	f1
1	pipeA[1]
2	ferr
3	pipeA[0]
4	pipeA[1]
5	f err
6	f1
7	

Tabla del Proceso hijo en **21**	
0	pipeA[0]
1	f2
2	STDERR
3	pipeA[0]
4	pipeA[1]
5	f2
6	
7	

6. Suponga un sistema de archivos Minix de 1 GBytes con las siguiente 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 = 1 Zona = 2 KBytes**

(1,2 puntos = 0.8 +0.4)

6

a) Indique de forma justificada los tamaños de cada elemento de la cabecera al formatearlo para almacenar 32768 nodos-i (32K nodos-i).

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, 2 Bloque Mapa Nodos-i, 32 Bloque Mapa zonas y 512 Bloques para los Nodos-i.

Calculos:
MapaNodos-i → 32768 / 2K*8 = $2^5\ 2^{10}$ / $2^1\ 2^{10}\ 2^3$ = 2 → 2 Bloques
Nº máximo de zonas = 1 GB/2KB = 2^{30} / $2^1\ 2^{10}$ = 512 K zonas
Mapa Zonas → Nº de Zonas/ Tamaño en bits de la zona = 512 K/ 2K*8=$2^9\ 2^{10}$ / $2^1\ 2^{10}\ 2^3$ =2^5 =32 Bloques
Bloques Nodos-i → 32768 * 32 Bytes /2KBytes = $2^5\ 2^{10}\ 2^5$ / $2^1\ 2^{10}$ = 512 Bloques
1+1+2+32+512= 548 → El primer bloque de la Zona de datos

b) Dicho sistema contiene el directorio raíz y dentro de él 30 directorios vacíos y 30 archivos. La mitad de los archivos son archivos regulares y la otra mitad son enlaces simbólicos a esos archivos regulares.

Indique de forma justificada el número de i-nodos ocupados
Cada fichero o directorio creado ocupa un i-nodo.
Tenemos en cuenta que los enlaces simbólicos utilizan un i-nodo propio.
Por tanto, 1 i-nodo del directorio raiz + 30 i-nodos de los directorios vacios + 15 i-nodos de los ficheros regulares + 15 i-nodos de los archivo de tipo enlace, nos da un total de 61 i-nodos ocupados.

Indique de forma justificada el tamaño en Bytes del directorio raíz.

El directorio raíz tendrá: 2 entradas del . y .. + 30 entradas de los directorios + 15 entradas de los archivos regulares +15 entradas de los archivos de tipo enlace simbólico. Eso nos da (62 entradas x 16 Bytes) = 992 Bytes

7. Un sistema con paginación por demanda con dos niveles de paginación tiene un tamaño máximo de proceso de 4GBytes, un tamaño de página de 4KBytes y un total de 4096 entradas en el primer nivel de paginación. La siguiente tabla muestra información relativa al instante t=25 para el proceso P y el proceso S del sistema operativo, el cual se haya ubicado en memoria en los marcos con las direcciones más altas disponibles en este sistema.

Proceso	Marco	Página	T. último acceso	BitV (validez)
P	0x4A000	0xC71FF	5	1
P	0x4A001	0xC7200	10	1
P	-	0xA70C0	-	0
P	0x4A003	0xA73DC	15	1
S	0xFFFFE	0xB7001	20	1
S	0xFFFFF	0xB7002	25	1

Partiendo del enunciado y la información de la tabla, conteste cada apartado de forma justificada:

(1,1 punto = 0,5 + 0,2 + 0,4)

7

a) Formato de la dirección lógica y física con nombre y número de bits de cada campo o elemento:

Dirección lógica (nombre y número de bits de cada elemento)

Con el tamaño máximo de proceso de 4 Gbytes tendríamos 32 bits repartidos en 12 bits para el desplazamiento (tamaño página 4 Kbytes), 12 bits para representar el descriptor de página de primer nivel (4096 entradas) y el resto, 8 bits para el descriptor de página de segundo nivel

Dirección física(nombre y número de bits de cada elemento)

Dado que los marcos que delimitan la parte final de la memoria física están asignados a la componente S del Sistema Operativo (el marco 0xFFFFF sería el último y necesitamos 5 dígitos hex x 4 = 20 bits para el nº de marco y 12 bits para el desplazamiento (tamaño página= tamaño marco).

b) Determine el número máximo de descriptores de páginas de 2º nivel que el proceso P puede utilizar.

A partir del apartado a) tendríamos 8 bits para representar los descriptores de página de segundo nivel (2^8 = 512 descriptores) que multiplicado por las 4096 entradas de primer nivel nos daría 2^{12} x 2^8 = 2^{20}=1M (1048576) descriptores máximo.

c) Indique las Direcciones Físicas correspondiente, así como si se produce fallo de página, al solicitar acceso a las Direcciones lógicas propuestas a continuación:

Proceso → Dir. Lógica	***Dir. Física (o FP si hay fallo de página)***
P→ 0xA70C0102	Fallo de página (la página 0x A70C0 tiene bit de validez = 0 en la tabla).
P→ 0xA73DC102	0x4A003102 (del marco asignado la página 0xA73DC con bit de validez a 1).
S→ 0xB7000102	Fallo de página (la página 0x B7000 no se incluye en la tabla).
S→ 0x B7001003	0xFFFFE003 (del marco asignado la página 0x B7001 con bit de validez a 1 y el desplazamiento 003)

8. Considere el sistema descrito en la pregunta 7 y la asignación inicial de marcos que se detalla en su tabla para el instante t=25. Suponga que se utiliza paginación por demanda con un algoritmo **LRU** con reemplazo **LOCAL,** gestionado mediante contadores y que el sistema asigna 4 marcos al proceso P y 2 al proceso S.

(1,5 puntos = 0,2 + 1,0 + 0,3)

8

a) A partir del instante t=26 los procesos emiten la siguiente secuencia de direcciones lógicas (en hexadecimal): P:A70C0102, P:A74D0F02, P:A73DC102, P:C7200C10, P:C7200C11, P:A70C0102, S:B7003A00, S:B7001000

Obtenga la serie de referencias correspondiente a la secuencia anterior:
P:A70C0, P:A74D0, P:A73DC, P:C7200, P:A70C0, S:B7003, S:B7001

b) Rellene la siguiente tabla con la evolución del contenido de Memoria Principal para la serie de referencias obtenida en el apartado anterior. En cada recuadro anote en su parte superior la página que corresponda y en la inferior el valor del contador. Puede rellenar sólo las casillas donde haya algún cambio.

Página→	(asignación inicial)	P:A70C0	P:A74D0	P:A73DC	P:C7200	P:A70C0	S:B7003	S:B7001
Marco ↓	t= 25	t= 26	t= 27	t= 28	t= 29	t= 31	t= 32	t= 33
4A000	P:C71FF	P:C71FF	**P:A74D0**	P:A74D0	P:A74D0	P:A74D0	P:A74D0	P:A74D0
	5	5	**27**	27	27	27	27	27
4A001	P:C7200	P:C7200	P:C7200	P:C7200	P:C7200	P:C7200	P:C7200	P:C7200
	10	10	10	10	**29**	29	29	29
4A002	-	**P:A70C0**	P:A70C0	P:A70C0	P:A70C0	P:A70C0	P:A70C0	P:A70C0
	-	**26**	26	26	26	**31**	31	31
4A003	P:A73DC	P:A73DC	P:A73DC	P:A73DC	P:A73DC	P:A73DC	P:A73DC	P:A73DC
	15	15	15	**28**	28	28	28	28
FFFFE	S:B7001	S:B7001	S:B7001	S:B7001	S:B7001	S:B7001	**S:B7003**	S:B7003
	20	20	20	20	20	20	**32**	32
FFFFF	S:B7002	S:B7002	S:B7002	S:B7002	S:B7002	S:B7002	S:B7002	**S:B7001**
	25	25	25	25	25	25	25	**33**
Marque fallos de página y reemplazos:		FP	FP (R)				FP (R)	FP (R)

Indique el número de FALLOS DE PÁGINA TOTALES:
4 (3 de ellos con reemplazo)

c) Indique de forma justificada si el algoritmo LRU ha conseguido una gestión óptima de los marcos, es decir, si el número de fallos de página ha sido el mismo que habría obtenido el algoritmo de reemplazo óptimo.

No. Hay que reconsiderar los 3 reemplazos. En t=27 no habría cambios, pero en t=32 el algoritmo óptimo habría escogido como víctima la página S:B7002, con lo que en t=33 no habría ocurrido fallo de página.