

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 11 cuestiones, cuya valoración se indica en cada una de ellas.

1. Indique cuales de las siguientes afirmaciones son verdaderas (V) y cuales son falsas (F).

Nota: Dos respuestas incorrectas restan una correcta.

(0,8 puntos)

1	Los sistemas Operativos presentan dos modos de ejecución que dan soporte al procesador: modo usuario y modo núcleo	V/F <b>F</b>
	Los procesadores actuales disponen de al menos dos modos de ejecución, el bit de modo indica el modo actual	V/F <b>V</b>
	Los modos de ejecución permiten la protección en el acceso al hardware de la maquina, como la memoria y los registros de los controladores de dispositivo	V/F <b>V</b>
	En un sistema cuya carga son procesos orientados principalmente a CPU la multiprogramación no supone una mejora significativamente ni en la productividad ni el tiempo de espera medio.	V/F <b>V</b>
	La solicitud de interrupciones provocan el cambio de modo núcleo a modo usuario	V/F <b>F</b>
	Cuando finaliza la ejecución de código del sistema operativo, se cambia a modo usuario para ejecutar aplicaciones de usuario	V/F <b>V</b>
	Los traps o interrupciones por llamadas al sistema ocurren en puntos del código previstos por el programador de la aplicación.	V/F <b>V</b>
	La librería de C "stdio" no hace ninguna llamada al sistema	V/F <b>F</b>

2. Justifique, para cada situación, qué ha podido ocurrir en el sistema para que se produzca cada una de las siguientes transición de estado

- De Preparado a en Ejecución
- De Suspendido a Preparado
- De en Ejecución a Suspendido

(0,6 puntos)

2	a) El proceso que está en ejecución abandona el estado de ejecución porque finaliza, o porque se suspende o porque es expulsado por el planificador. El planificador escoge a un proceso de la cola de preparados para asignarle el procesador
	b) Finaliza una espera (fin de operación de E/S asíncrona, fin de un sleep, muerte de un hijo estando el proceso padre en un wait, etc) por lo que el proceso implicado pasa de suspendido a preparado
	c) El proceso que está en ejecución realiza una llamada al sistema bloqueante, como por ejemplo, una lectura de un tubo vacío, una operación de E/S, una llamada wait habiendo procesos hijos activos en el sistema, una llamada a sleep, una llamada pthread_mutex_lock sobre un mutex ocupado, una operación P sobre un semáforo a cero, etc.

3. Dado el siguiente código cuyo fichero ejecutable ha sido generado con el nombre "Ejemplo1".

```

1  /** Ejemplo1***/
2  #include "todas_las_cabeceras_necesarias.h"
3
4  main()
5  { int i=0, retraso1=1, retraso2=1;
6    pid_t pid, pid_x;
7
8    for (i=0; i<2; i++)
9    {
10     pid=fork()
11     if (pid==0)
12     { sleep(retraso1);
13       exit(0);
14     }
15   }
16   sleep(retraso2);
17   while (wait(NULL) != -1);
18   exit(0);
19 }

```

Indique de forma justificada:

- El número de procesos que se generan al ejecutarlo y dibuje el esquema de parentesco entre procesos.
- Indique de forma justificada la posibilidad de que se generen procesos **zombies** o **huérfanos** para cada uno de los siguientes casos:  $retraso1=retraso2$ ,  $retraso1 \gg retraso2$  y  $retraso2 \gg retraso1$ .

(1,0 puntos)

3	<p>a) El número de procesos creados al ejecutar ejemplo1 son tres, un padre y dos hijos. El esquema sería el siguiente</p> <div data-bbox="651 1120 847 1312" data-label="Diagram"> <pre> graph TD     P1((P1)) --&gt; F1((F1))     P1 --&gt; F2((F2)) </pre> </div> <p>b) En función de las variables retraso1 y retraso2</p> <ul style="list-style-type: none"> <li>Retraso1 &gt; retraso2 → Funcionamiento normal, cuando los hijos salen de la espera el padre ya está esperándolos en el bucle del wait</li> <li>Retraso1=retraso2 → en función del orden de ejecución puede que en un transitorio alguno de los hijos se quede zombie, pero es poco probable</li> <li>Retraso1 &lt; retraso2 → cuanto mayor sea la diferencia más tiempo podrán permanecer los hijos en estado zombie, ya que mas tiempo pasará antes de que el proceso padre realice las llamadas a wait.</li> </ul> <p>Tal y como está estructurado el código, sólo en el caso de algún error o la llegada de una señal que afecte al proceso padre (antes de que acaben los procesos hijos) se podría dar la circunstancia de que los procesos hijos se queden huérfanos.</p>
---	--

- Un sistema dispone de un planificador a corto plazo compuesto por 3 colas (C0, C1, C2), gestionadas con prioridades expulsivas siendo C0 la cola más prioritaria y C2 la menos prioritaria. Todos los procesos nuevos llegan a la cola C0 (mas prioritaria) y el sistema les asocia un contador que se inicializa a 0. El contador se incrementa en 1 cada vez que el proceso consume un ciclo de CPU. Cuando un contador toma el **valor 2** el proceso asociado se degrada a la cola con prioridad inmediatamente inferior e inicializa su contador a 0. Cuando un proceso alcanza la cola menos prioritaria permanece en ella hasta finalizar su ejecución.

El sistema está dotado de un único dispositivo de E/S gestionado con FCFS. Los algoritmos de planificación que rigen cada una de las colas son los siguientes:

**C0: Round Robin con q=1**

**C1: SRTF**

**C2: FCFS**

A dicho sistema llegan 4 procesos cuyo perfil de ejecución e instante de llegada son:

Proceso	Perfil de ejecución	Instante de llegada
A	4 CPU + 4 E/S + 2 CPU	0 (1°)
B	3 CPU + 2 E/S + 1 CPU	0 (2°)
C	1 CPU + 2 E/S + 5 CPU	0 (3°)
D	1 CPU + 1 E/S + 3 CPU	0 (4°)

(1.4 puntos)

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

4 a)	FCFS Cola 2	SRTF Cola 1	RR Cola 0	CPU	Cola E/S	E/S	Evento
0			D,C,B,A	A			Llegan A,B, C, D
1			A,D,C,B	B			
2			B,A,D,C	C			
3			B,A,D	D		C	
4			B,A	A	D	C	
5		A	C,B	B		D	
6		B A	D,C	C			
7		C B A	D	D			
8		D C B A		B			
9		D C A		A		B	
10		D C		A		B	
11		B D C		B		A	
12		D C		D		A	Fin de B
13		C		D		A	
14				C		A	Fin D
15	A			C			
16	CA			A			
17	C			A			
18				C			Fin A
19				C			
20							Fin C
21							
22							

b) Indique cuál será el Tiempo medio de espera, el Tiempo medio de retorno y la Utilización de la CPU.

4	<p>b)</p> <p>Tiempo medio de Espera= <math>(8+6+12+8)/4 = 34/4=8.5</math></p> <p>Tiempo medio de Retorno= <math>(18+12+20+14)/4 = 64/4=16</math></p> <p>Utilización CPU= <math>20/20= 1</math> (100%)</p>
---	---



5. Dado el siguiente código cuyo archivo ejecutable ha sido generado con el nombre "Hilos1".

```

1  /*** Hilos1 ***/
2  #include <stdio.h>
3  #include <pthread.h>
4  pthread_t hilo1, hilo2, hilo3;
5
6  void *fun_hilo( void *ptr )
7  {int sec;
8      sec=(int)ptr;
9      usleep(sec*1000000);
10     printf("Yo he esperado %d segundos\n",sec);
11 }
12
13 void *fun_hilo3( void *ptr )
14 {int sec;
15     sec=(int)ptr;
16     usleep(sec*1000000);
17     pthread_join( hilo1, NULL);
18     printf("Yo he esperado %d segundos\n",sec);
19 }
20
21 int main()
22 {pthread_attr_t atrib;
23     pthread_attr_init( &atrib );
24     pthread_create( &hilo1, &atrib, fun_hilo, (void *)5);
25     pthread_create( &hilo2, &atrib, fun_hilo, (void *)1);
26     pthread_create( &hilo3, &atrib, fun_hilo3, (void *)3);
27     pthread_join( hilo3, NULL);
28 }
```

Indique de forma ordenada las cadenas que imprime el programa en la Terminal tras su ejecución, así como el tiempo aproximado que tardará en terminar el programa. Justifique su respuesta.

(0,8 puntos)

**5**  
**Yo he esperado 1 segundos**  
**Yo he esperado 5 segundos**  
**Yo he esperado 3 segundos**

El hilo principal crea 3 hilos hilo1, hilo2 e hilo3 y espera con pthread\_join a que el hilo3 termine. Los hilos 1 y 2 ejecutan la función func\_hilo, mientras que hilo3 ejecuta func\_hilo3. Todos los hilos, después de ser creados, se suspenden con usleep() un intervalo que depende del valor de su parámetro. El hilo1 se suspende durante 5 segundos, el hilo 2 durante 1 segundo, mientras que el hilo3 lo hace durante 3 segundos. El hilo 3 después de 3 segundos suspendido, queda a la espera de que termine el hilo1. Lo cual nos garantiza que hilo 1 finaliza y que han transcurrido 5 segundos. Mientras tanto el hilo 2 ha estado suspendido 1 segundo, ha escrito su mensaje y ha terminado. Como el hilo principal espera a que el hilo 3 termine, el **programa tardará 5 segundos en terminar**.

6. Sea un computador dotado de una memoria física es de 2GB, cuyos procesos disponen de 16 GB de espacio de direccionamiento lógico. Calcule:

- El tamaño de página empleado en un esquema de gestión de memoria mediante paginación, con un máximo de 512 Kpáginas por proceso.
- Número de marcos del sistema, para una gestión de memoria mediante segmentación paginada con un número máximo de 128 segmentos por proceso, con 128 K (131.072) páginas por segmento.

(0,7 puntos)

**6**

a) Dado que  $16\text{Gbytes} = 2^4 * 2^{30} = 2^{34}$  se necesitan 34 bits para la dirección lógica  
Dado que  $512\text{Kpáginas} = 2^{19}$  se necesitan 19 bits indicar el número de página lógica.  
Como  $34 - 19 = 15$ , entonces, el tamaño de la página es de  $2^{15} = 32\text{ KB}$   
El formato de la dirección lógica será:

----- 34 bits -----

Número de página 19 bits	Desplazamiento de página 15 bits
-----------------------------	-------------------------------------

c) Dado que  $16\text{Gbytes} = 2^4 * 2^{30} = 2^{34}$  se necesitan 34 bits para la dirección lógica  
Con  $128 = 2^7$  segmentos por proceso con  $128\text{kpáginas} = 2^7 * 2^{10} = 2^{17} \rightarrow 7$  bits para los segmentos y 17 bits para la página lógica  $\rightarrow$  por tanto tenemos  $34 - 17 - 7 = 10$  bits para el desplazamiento de página  $\rightarrow$  el tamaño de la página es de  $2^{10} = 1024\text{ Bytes}$ . El formato de la dirección lógica es:

----- 34 bits -----

Número de Segmento 7 bits	Número de Páginas 17	Desplazamiento de página 10
------------------------------	-------------------------	--------------------------------

Dado que la memoria física es de  $2\text{GB} = 2^{31}\text{Bytes} \rightarrow$  la dirección física tiene 31 bits, de los cuales 10 bits son para el desplazamiento de página y  $31 - 10 = 21$  para el número de marco  $\rightarrow$  el número total de marcos de dicha memoria será  $2^{21} = 2\text{ M marcos}$  (2.097.152). El formato de la dirección física será:

----- 31 bits -----

Número de marco 21 bits	Desplazamiento de página 10
----------------------------	--------------------------------

7. El sistema operativo de cierto computador gestiona la memoria virtual mediante paginación con páginas de 1KB, asigna marcos libres en orden creciente de direcciones y aplica un algoritmo reemplazo LRU de ámbito local. En un momento dado, se desea ejecutar un proceso P al que se le asignan 3 marcos (del 0 al 2). Durante su ejecución el proceso P referenciará la siguiente secuencia de **direcciones lógicas**:

**0, 4, 8, 12, 1024, 0, 4, 8, 12, 2048, 0, 4, 8, 12, 3072, 0, 4, 8, 12, 4096**

- a) Suponga que P no tiene ningún página cargada en memoria e indique de forma justificada las direcciones lógicas que producirán los tres primeros fallos de página al ejecutar dicho proceso.
- b) Indique y justifique el número total fallos de página que se producirán al ejecutar el proceso P completo.
- c) Indique el contenido de todos los descriptores de la tabla de páginas del proceso P (con bit de validez y número de marco) justo después del último acceso.
- d) Indique de forma justificada, la última dirección física a la que accede este proceso.

(1.0 puntos)

**7**

a) La serie de referencias a que corresponde dichas direcciones lógicas es: 0,1,0,2,0,3,0,4

<b>0, 4, 8, 12</b>	<b>,1024</b>	<b>, 0, 4, 8, 12</b>	<b>, 2048</b>	<b>,0, 4, 8, 12</b>	<b>,3072</b>	<b>,0, 4, 8, 12</b>	<b>,4096</b>
Página 0	Página 1	Página 0	Página 2	Página 0	Página 3	Página 0	Página 4

Las direcciones lógicas que producen los tres primeros fallos de páginas, serán aquellas que correspondan a las tres páginas distintas invocadas por primera vez  $\rightarrow 0, 1024, 2048$

b) Dado que se hace referencia a 5 páginas distintas y se utiliza un algoritmo LRU de ámbito local con 3 marcos de memoria física, el número total de fallos de página es 5.

<b>Pag 0</b>	Pag 0	<b>Pag 0</b>	Pag 0	<b>Pag 0</b>	Pag 0	<b>Pag 0</b>	Pag 0
	<b>Pag 1</b>	Pag 1	Pag 1	Pag 1	<b>Pag 3</b>	Pag 3	Pag 3
			<b>Pag 2</b>	Pag 2	Pag 2	Pag 2	<b>Pag 4</b>
FP	FP	FP	FP	R	FP	R	FP

c) En la memoria física: Marco 0: Página 0; Marco 1: Página 3; Marco 2: Página 4  
Tabla de páginas del proceso:

Nº Página	Bit de validez	Marco
0	valido	0
1	invalido	1
2	invalido	2
3	valido	1
4	valido	2
5	invalido	-

d)

La última dirección física a la que se accede es la correspondiente a la dirección lógica 4094, que corresponde a la página 4 con desplazamiento 0 → La página 4 está en el marco 2 → La dirección física es  $2 \cdot 1024 + 0 = 2048$

8. Se ejecutan concurrentemente los procesos A, B y C que aparecen en la tabla adjunta.

a) Indique, justificando su respuesta, todos los posibles valores que puede alcanzar la variable x

// Variables compartidas int x=1;                      Semáforos S1=0, S2=0, S3=1;		
// Proceso A	// Proceso B	// Proceso C
P(S1) x = x + 3; // sección 1 V(S2) P(S3) x = x + 5; // sección 2	P(S2) x = 2*x; // sección 3 V(S1) P(S2) x = 4*x; // sección 4 V(S3)	P(S3) x = x + 2; // sección 5 V(S2)

b) Suponga que antes de crear los procesos A, B y C se ejecuta las órdenes V(S2) y P(S3) e indique, justificando su respuesta, todos los posibles valores que puede alcanzar la variable x

(0,7 puntos)

8	a) Dado que los Semáforos se encuentran inicializados a S1=0, S2=0, S3=1; Existe un único orden de ejecución posible del código marcado como “//sección x” que vendrá dado por : Proceso C → P(S3), sección 5 (x=X+2=3), Proceso B → P(S2), sección 3 (x=2*x=6), V(S1), Proceso A → P(S1) sección 1 (x=x+3=9), V(S2) Proceso B → P(S2) sección 4 (x=4*x=36), V(S3) Proceso A → P(S3) sección 2 (x=x+5= 41)
	b) Antes de comenzar la ejecución de los procesos A, B y C se ejecutan las instrucciones V(S2) ; P(S3) . Esto deja los contadores de los semáforos como S1=0, S2 =1 , S3=0; <u>Opción 1:</u> En este caso el proceso C quedaría suspendido en su primera instrucción P(S3) mientras que el proceso B es el único que podrá avanzar, siendo el orden de ejecución de las secciones: Proceso C → P(S3) → C suspendido en S3 Proceso B → P(S2), sección 3 (x=2*x=2), V(S1), P(S2) → B suspendido en S2 Proceso A → P(S1), sección 1 (x=x+3=5), V(S2), P(S3) → A suspendido en S3 Proceso B → sección 4 (x=4*x=20), V(S3) Proceso C → sección 5 (x= x+2= 22) , V(S2) En este caso x=22 y A se queda suspendido en el semáforo S3 <u>Opción 2:</u> Comienza la ejecución con el proceso B Proceso B → P(S2), sección 3 (x=2*x=2), V(S1), P(S2) → B suspendido en S2 Proceso A → P(S1), sección 1 (x=x+3=5), V(S2), P(S3) → A suspendido en S3 Proceso C → P(S3) → C suspendido en S3 Proceso B → sección 4 (x=4*x=20), V(S3) Proceso A → sección 2 (x=x+5 =25) En este caso X=25 y el proceso C se queda suspendido en S3

9. Suponga activados todos los permisos necesarios para ejecutar el código del programa cuyo archivo fuente es “Ene17a.c” y que lo ejecutamos en el directorio donde se encuentra “Ene17a.c”. Analice el código de programa “Ene17a.c” e indique:

- El contenido de las tablas de descriptores, durante la ejecución del código, exactamente en los puntos donde figuran los comentarios `/* Tablas punto X */` (con X 1,2,3,4) para los procesos involucrados.
- Indique el esquema de comunicación que se generará entre los procesos con tubo o tubos y archivos involucrados.
- Indique de forma justificada lo que se muestra por pantalla, suponga que no se producen errores en ninguna llamada.
- Indique de forma justificada cuál será el contenido del archivo `"f2"` después de la ejecución del programa

Nota: **cat**: muestra en la salida estándar lo que lee por la entrada estándar,

**grep "cadena"**: escribe en la salida estándar las líneas de la entrada estándar que contienen "cadena"

**wc -l**: escribe en la salida estándar el número de líneas que lee de la entrada estándar

```

1  /** código de Ene17a.c */
2  #include "todas_las_cabeceras_necesarias.h"
3
4  int main(int argc, char *argv[])
5  { //proceso p1
6      int i, fd[2] /*tubo1*/, fd2[2] /*tubo2*/, x;
7      pipe(fd);
8      if (fork() == 0)
9      { //proceso h1
10         x=open("ene17a.c", O_RDONLY);
11         dup2(x, 0);
12         dup2(fd[1], 1);
13         close(fd[0]); close(fd[1]);
14         /* Tabla punto 1 */
15         execlp("cat", "cat", NULL);
16         fprintf(stderr, "mensaje 1");
17         exit(1);
18     }
19     pipe(fd2);
20     if (fork() == 0)
21     { //proceso h2
22         close(fd[0]); close(fd[1]);
23         x=open("f2", O_CREAT|O_TRUNC|O_WRONLY, 0666);
24         dup2(fd2[0], 0); dup2(x, 1);
25         close(fd2[0]); close(fd2[1]);
26         /* Tabla punto 2 */
27         execlp("wc", "wc", "-l", NULL);
28         fprintf(stderr, "mensaje 2");
29         exit(1);
30     }
31     if (fork() == 0)
32     { //proceso h3
33         dup2(fd[0], 0);
34         close(fd[0]); close(fd[1]);
35         dup2(fd2[1], 1);
36         close(fd2[0]); close(fd2[1]);
37         /* Tabla punto 3 */
38         execlp("grep", "grep", "fork", NULL);
39         fprintf(stderr, "mensaje 3");
40         exit(1);
41     }
42     /* Tabla punto 4 */
43     close(fd[0]); close(fd[1]);
44     close(fd2[0]); close(fd2[1]);
45     wait(NULL);
46     wait(NULL);
47     wait(NULL);
48     return 0;
49 }
```

(1,2 puntos)







<b>10</b>	Analizando los permisos de cambia_claves vemos que todos los usuarios (useri, grpj) pueden ejecutarlo pero, a causa del bit SETGID, pasarán a tener el grupo del propietario (useri, grpa)				
	<b>Usuario</b>	<b>Grupo</b>	<b>Orden</b>	<b>¿Funciona?</b>	<b>Observaciones</b>
	user3	grpb	./cambia_clave claves_web	NO	El usuario efectivo en este caso es (user3,grpa). El archivo claves_web no tiene permisos de lectura y escritura para grpa
	user2	grpa	./cambia_clave claves_impr	NO	El usuario efectivo en este caso es (user2,grpa). El archivo claves_impr no tiene permisos de lectura y escritura para grpa
	user2	grpa	./cambia_clave claves_sala	SI	El usuario efectivo en este caso es (user2,grpa). El archivo claves_sala tiene permisos de lectura y escritura para grpa
	user3	grpb	./cambia_clave claves_sala	SI	El usuario efectivo en este caso es (user3,grpa). El archivo claves_sala tiene permisos de lectura y escritura para grpa

**11.** En un dispositivo de 512MBytes de capacidad, se crea un sistema de archivos MINIX con capacidad para 10500 nodos-i. Indique de forma justificada el espacio que ocupa cada uno de los elementos de la cabecera y el tamaño del espacio de datos después de dar formato al dispositivo.

**Nota:** Los tamaños estándar de MINIX son:

- Tamaño de bloque 1024 bytes con 1bloque=1zona
- Referencia a bloque 2 bytes
- Entrada de directorio 16 bytes
- Nodo-i de 32 bytes

(1.0 puntos)

<b>11</b>	<b>BA</b>	<b>SB</b>	<b>Mapa de nodos-i</b>	<b>Mapa de zonas</b>	<b>Nodos-I</b>	<b>Zonas de datos</b>
	1	1	2 bloques	64 bloques	329 bloques	523891 zonas

- Bloque de arranque : **1 Bloque**
- Superbloque : **1 Bloque**
- Mapa de Nodos-I  
Se necesitan 10500 bits para el mapa de nodos-i, un bit por nodo-i.  
Bloques para el mapa de nodos-i=  $10500/8192 = 1,2817... \rightarrow$  **2 Bloques**
- Mapa de zonas  
El número de zonas que hay en la partición es:  $512*1024*1024/1024 = 512*1024$  zonas  
Por tanto se necesitan  $512*1024$  bits para el mapa  
Bloques para el mapa de zonas=  $(512 * 1024 \text{ bits}) / (8 * 1024) =$  **64 Bloques**
- Nodos-i  
En este sistema hay 10500 nodos-i, cada nodo-i ocupa 32 bytes  $\rightarrow 10500 * 32 \text{ bytes}$   
Bloques para los nodos-i =  $10500 * 32 \text{ bytes} / 1024 \text{ bytes} = 328,... \rightarrow$  **329 Bloques**
- Zonas de Datos  
Si la partición le quitamos lo que ocupa la cabecera, sabremos cuántas zonas hay de datos:  
 $512*1024 - (1 + 1 + 2 + 64 + 329) = 523891$