



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, la 6 vale 2 (1.2+0.8) puntos y el resto 1 punto cada una.

Diga en qué consiste la técnica de multiprogramación e indique las ventajas de su uso. Analice si dicha técnica tiene o no sentido en el caso de que los procesos a ejecutar tuviesen una única ráfaga de CPU y ninguna de entrada salida, suponga que no hay interacción usuario máquina

1	<p>La multiprogramación consiste en la capacidad que tienen los sistemas de poder ejecutar varios procesos concurrentemente con una única CPU. Para ello se intercalan intervalos de CPU de un proceso con intervalos de E/S de otro proceso diferente. Esto implica que dichos procesos deben encontrarse simultáneamente en memoria.</p> <p>La principal ventaja es un mayor rendimiento del sistema y un aumento de utilización de CPU.</p> <p>Si los procesos tienen únicamente ráfagas de CPU, no puede haber concurrencia entre E/S y CPU, por tanto no aumentaría la utilización de CPU. Como debe haber cambios de contexto para que los procesos se ejecuten concurrentemente, con este tipo de procesos sólo produciría una disminución de la productividad.</p> <p>La multiprogramación no implica interacción usuario-máquina, por tanto, la ejecución de varios procesos concurrentemente no es percibida por el usuario y sólo tiene sentido si se aumenta la utilización de CPU y la productividad.</p>
---	---

Dado un computador dotado de sistema operativo

- Justifique la necesidad de que el procesador disponga de al menos dos modos de funcionamientos
- Indique para cada una de las acciones propuestas cuales se realizan en modo núcleo y cuales en modo usuario

modo usuario

2	a)	El objetivo principal de tener varios modos de funcionamiento es la protección del acceso al hardware de la máquina. Los programas de usuario necesitarán la intervención del sistema operativo, al que solicitan servicios mediante llamadas al sistema, para poder acceder al hardware. Cuando se hace una llamada al sistema el procesador cambia de modo usuario a modo núcleo. Se trata de impedir que los programas de usuarios accedan a memoria principal libremente, puedan monopolizar el uso de CPU, accedan a ciertos registros del sistema, accedan directamente a los dispositivos. Del conjunto de instrucciones del procesador, un subconjunto de ellas sólo se pueden ejecutar en modo privilegiado	
	b) Acciones Propuestas	Núcleo	Usuario
	Programar el controlador del disco	X	
	Seleccionar un proceso de la cola de preparados	X	
	Invocar una llamada al sistema		X
	Leer 256 bytes de un fichero	X	
	Ejecutar instrucciones que contiene expresión matemáticas complejas		X



Dada las siguientes transiciones entre procesos indique de forma justificada si son o no posibles exponiendo un ejemplo cuando sea posible:

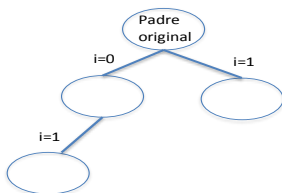
3	En ejecución a preparado Si es posible. Puede ocurrir en los casos en que el proceso en ejecución no haya finalizado su ráfaga de CPU y se le haya acabado el quantum que le asigno el sistema. También puede ocurrir en sistemas con planificadores expulsivos a procesos en ejecución, cuando llega un nuevo proceso a la cola de preparados con mayor prioridad que el que tiene asignada CPU.
	De preparado a suspendido No es posible. Si el proceso no solicita E/S o suspensión hasta que ocurra un evento, con la invocación de llamadas al sistema o ejecución de instrucciones, y para esto debe estar en ejecución, no puede pasar a suspendido.
	De suspendido a en ejecución No es posible. De suspendido debe pasar a preparado y ser seleccionado por el planificador. El único caso a considerar sería aquel en el que este es el único proceso del sistema y en ese caso sería el S.O.

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

```
1  /*** Ejemplo1***/  
2  #include "todas_las_cabeceras_necesarias.h"  
3  main()  
4  { int i=0;  
5    int pid;  
6  
7    while (i<2)  
8    {  
9      switch((pid=fork()))  
10     {  
11       case (-1): {printf("Error creando hijo\n");break;}  
12       case (0): {printf("Hijo %i creado\n",i);break;}  
13       default: {printf("Padre\n");}  
14     }  
15     i++;  
16   }  
17   exit(0);  
18 }
```

Indique de forma justificada:

- El número de procesos que se generan al ejecutarlo y el parentesco existente entre ellos.
- Indique que mensajes se mostrarán en pantalla, si la llamada fork() siempre tiene éxito.

4	<p>a) Se crean un total de 4 procesos al ejecutar este código un padre(P) que crea dos hijos Hijo 0 (creado con i=0) e Hijo 1 (creado con i=1). Además el Hijo 0 se convierte en padre y con i=1 crea un proceso Hijo.</p>  <pre>graph TD A([Padre original i=0]) --> i=0 B([Hijo 0 i=0]) A --> i=1 C([Hijo 1 i=1]) B --> i=1 D([Hijo 1 i=1])</pre>
---	--



b)

Si tiene éxito la llamada `fork()`, el orden de los mensajes depende del sistema, pero se imprimen los mensajes:

```
Hijo 0 creado
Hijo 1 creado
Padre
Padre
Hijo 1 creado
Padre
```

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  int main()
5  {
6      int status;
7      printf("Mensaje 1: antes de  exec()\n");
8      if (execl("/bin/ps", "ps", "-la", NULL) < 0)
9          { printf("Mensaje 2: después de  exec()\n");
10             exit(1); }
11
12     printf("Mensaje 3: antes del exit()\n");
13     exit(0);
14 }
```

Indique de forma justificada:

- Cuantos procesos se pueden llegar a crear durante su ejecución, en que número de líneas se lleva a cabo dicha creación y que mensaje imprime cada uno de ellos.
- Cuando aparecerán por la salida estándar "*Mensaje 3: antes del exit()*".

- 5** **a)** Se crea un único proceso al lanzar el ejecutable de este código. Este proceso imprimirá por pantalla el mensaje "Mensaje 1: antes de `execl()`". A continuación realizará la llamada `exec()` y se sustituirá el código propuesto arriba por el que contenga el fichero ejecutable que haya en `/bin/ps`. Si existe el fichero ejecutable `/bin/ps` se mostrará por pantalla el resultado de ejecutar `ps`. Si no existe el fichero ejecutable `/bin/ps` se mostrará por pantalla "Mensaje 2: después de `exec()`".
- b)** Este mensaje no aparecerá nunca en pantalla, ya que sólo hay dos posibilidades:
- Que `execl()` tenga éxito en ese caso, este código es sustituido por el del fichero `/bin/ps`
 - Que `execl()` no tenga éxito, se ejecutará el `exit(1)` de la línea 9 y finaliza el proceso.

Un Sistema Operativo de tiempo compartido acepta trabajos lanzados desde terminales locales (interactivos I), desde una línea de red (R) y trabajos en Batch (B). La gestión de trabajos es por colas multinivel sin realimentación. La política de planificación entre colas es por prioridades expulsivas. La cola con mayor prioridad es la de procesos interactivos (I) y la de menor la de Batch (B). Cada proceso va a la cola de su tipo y permanece en ella durante su vida. La política de cada cola es:

- Interactivos: R-R ($q=1$)
- Red: R-R ($q=2$)
- Batch: FCFS

El orden de llegada de los procesos a las colas es el siguiente: en primer lugar los nuevos, a continuación los procedentes de E/S y por último los que salen de CPU. Suponga que todas las

operaciones E/S se realizan en un único dispositivo con política de servicio FCFS y que se solicita ejecutar el siguiente grupo de trabajos:

Trabajo	Instante de llegada	Tipo	
T1	0	I	2CPU+2E/S+1CPU+2E/S+2CPU
T2	2	R	4CPU+1E/S+ 4CPU+1 E/S+1 CPU
T3	4	B	1CPU+1E/S+ 5CPU+1 E/S+1CPU
T4	5	I	2CPU+4E/S+1CPU
T5	21	B	2CPU

- a) Indique el diagrama de uso de CPU, relleno la tabla adjunta para cada instante de tiempo.
- b) Calcule el tiempo medio de espera en la cola de preparados, el tiempo medio de retorno y la utilización de CPU

6 a	T	Cola I	Cola R	Cola B	CPU	Cola E/S	E/S	Evento
	0	(T1)			T1			Llega T1
	1				T1			
	2		(T2)		T2		T1	Llega T2 T1 a E/S
	3				T2		T1	
	4	(T1)	T2	T3	T1			Llega T3
	5	(T4)	T2	T3	T4		T1	T1 a E/S Llega T4
	6	(T4)	T2	T3	T4		T1	
	7	(T1)	T2	T3	T1		T4	T4 a E/S
	8	(T1)	T2	T3	T1		T4	
	9		(T2)	T3	T2		T4	Fin de T1
	10		(T2)	T3	T2		T4	
	11	(T4)		T3	T4		T2	T2 a E/S
	12		(T2)	T3	T2			FIN de T4
	13				T2			
	14		(T2)	T3	T2			
	15			T3	T2			
	16			(T3)	T3		T2	T2 a E/S
	17		(T2)		T2		T3	T3 a E/S
	18			(T3)	T3			FIN de T2
	19		(T2)		T3			
	20				T3			Fin de T2
	21			T5	T3			Llega T5
	22			T5	T3			
	23			(T5)	T5		T3	T3 a E/S
	24			T3	T5			
	25			(T3)	T3			Fin T5
	26							Fin T3
27								
28								
6 b	Tiempo medio de espera=0+5+13+0+2/5 = 20/5=4							
	Tiempo medio de retorno=((9-0)+(18-2)+(26-4)+(12-5)+(25-21))/5= =9+16+22+7+4/5=11.6 u.t.							
	Utilización de CPU= 26/26= 100%							



Dado el problema de la sección crítica y sus posibles soluciones, diga si las siguientes sentencias son verdaderas o falsas:

7	V/F	
	V	Al conjunto instrucciones de un proceso, que acceden a datos compartidos con al menos otro proceso del sistema y que alguno de dichos procesos puede modificar, se le llama sección crítica.
	F	Una solución correcta al problema de la sección crítica debe garantizar la exclusión mutua de todo el código de los procesos que se ejecutan concurrentemente.
	V	Una solución propuesta al problema de la sección crítica utiliza la instrucción <code>test_and_set</code> que debe ejecutarse de forma atómica.
	F	Atómica significa que no puede ejecutarse con espera activa y es necesaria la intervención de un mutex
	F	La condición que debe cumplir cualquier solución válida al problema de la sección crítica y que establece que “ <i>si ningún proceso está ejecutando su sección crítica y hay otros que desean entrar a las suyas, entonces la decisión de qué proceso entrará a la sección crítica se ha de tomar en un tiempo finito y sólo depende de los procesos que desean entrar</i> ” se denomina espera limitada .
	V	La soluciones al problema de la sección crítica con espera activa presenta infrautilización del procesador
	V	Los mutex de POSIX son una solución al problema de la sección crítica sin espera activa

Dado el problema del productor consumidor estudiado en clase donde aparece tanto una sección crítica, como una relación de precedencia. Recuerde que los consumidores no deben consumir antes de que los productores hayan producido. Complete el siguiente código realizando operaciones sobre los semáforos *mutex*, *vacio* y *lleno*, que ya se encuentran declarados e inicializados. Justifique su solución indicando la utilidad de las operaciones propuestas.

<pre>#include <semaphore.h> #define N 20 int buffer[N]; int entrada, salida, contador sem_t mutex, lleno, vacio; void *func_prod(void *p) { int item; while(1) { item = producir() //Complete (1) buffer[entrada] = item; entrada = (entrada + 1) % N; contador = contador + 1; //Complete (2) } }</pre>	
<pre>void *func_cons(void *p) { int item; while(1) { //Complete (3) item = buffer[salida]; salida = (salida + 1) % N; contador = contador - 1; //Complete(4) consumir(item); } }</pre>	
<pre>void main () { sem_init(&mutex,0,1); sem_init(&vacio,0,N); sem_init(&lleno,0,0); }</pre>	



8	<p>a) Solución propuesta</p> <pre>(1) sem_wait(&vacio); sem_wait(&mutex) (2) sem_post(&mutex); sem_post(&lleno) (3) sem_wait(&lleno); sem_wait(&mutex) (4) sem_post(&mutex); sem_post(&vacio)</pre>
	<p>b) Utilidad o función de cada instrucción propuesta</p> <pre>(1) sem_wait(&vacio); sem_wait(&mutex) prot entrada productor (2) sem_post(&mutex); sem_post(&lleno) prot salida productor (3) sem_wait(&lleno); sem_wait(&mutex) prot entrada consumidor (4) sem_post(&mutex); sem_post(&vacio) prot salida consumidor</pre> <p>Los semáforos <i>lleno</i> y <i>vacio</i> tienen la función de evitar el acceso al <i>mutex</i> tanto de productores como de consumidores si no están preparados para ello.</p> <p>Los semáforos <i>lleno</i> y <i>vacio</i> controlan la secuenciación entre productores y consumidores de forma que los productores no accedan al <i>mutex</i> de la sección crítica si no hay huecos y los consumidores si no hay ítems. El semáforo <i>mutex</i> controla el acceso a la sección crítica en exclusión mutua.</p>

Indique las cadenas que imprime el programa en la Terminal tras su ejecución. Justifique su respuesta

<pre>void * funcion_hilo1(void * arg) { sleep(40); printf("Soy el hilo 1\n"); return null; }</pre>	<pre>void * funcion_hilo2(void * arg) { sleep(50); printf("Soy el hilo 0\n"); return null; }</pre>
<pre>int main (void) { pthread_t th1,th2; pthread_attr_t atrib; pthread_attr_init(&atrib); printf("Erase una vez dos hilos...\n"); pthread_create(&th1, &atrib, funcion_hilo1, null); pthread_create(&th2, &atrib, funcion_hilo2, null); exit(0); }</pre>	
9	<p>Solo se imprime " Erase una vez dos hilo ..."</p> <p>Como el hilo principal no espera a los otros hilos con <code>join</code>. Es muy probable, que el hilo principal finalice antes de que los otros hilos dejen de estar suspendidos</p>