APELLIDOS		NOMBRE	
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 tiene 12 cuestiones. Las cuestiones 5 y 6 valen 1 punto (cada una), todas las demás 0.8 puntos cada una.**

CUESTIONES

Explique por qué un sistema operativo necesita que el procesador disponga de al menos dos modos de ejecución.

1	Deben existir estos dos modos porque de esta forma se evita que un programa no autorizado pueda realizar cualquiera de las siguientes tareas: • Acceder a los dispositivos de E/S, tanto para programar sus controladoras como para atender sus interrupciones. • Acceder a memoria no asignada, ya que tampoco tendrán acceso a la MMU del procesador y sin ella no es posible acceder a cualquier posición de la memoria principal. • Gestionar los recursos comunes. Los únicos programas que podrán gestionar los recursos del sistema son aquellos que se ejecuten en el modo privilegiado (o modo "núcleo" o "del sistema"). Para poder utilizar estos programas deben solicitar una "llamada al sistema" y cambiar entonces el modo de ejecución. Aquellos programas que se ejecutan en el modo privilegiado forman el sistema operativo (normalmente es un solo programa y forma el NÚCLEO del sistema operativo), todos los demás programas se ejecutan en modo usuario y deberán ser ejecutados como PROCESOS por el núcleo que se acaba de mencionar. Si no hubiese dos modos de ejecución no se podría distinguir entre el núcleo y el resto de los programas. Por tanto, cualquier programa podría llevar a cabo las mismas tareas que normalmente realiza el núcleo y **no existiría ningún control** sobre el funcionamiento del sistema. Ejemplo: Versiones de MS-DOS desarrolladas para el i8086.

Razone por qué al crear un fichero en UNIX con la llamada creat("fichero", 0666); puede darse el caso de que dicho fichero tenga una palabra de protección "rw-r--r--", pero no una "rwx--xr--" (por ejemplo).

2	Porque todo proceso UNIX mantiene un atributo (llamado "máscara de creación de ficheros") que indica qué derechos **no deben otorgarse** a los ficheros de nueva creación (sean del tipo que sean: regulares, directorios, etc.). Este atributo puede consultarse con la orden "umask" sin argumentos y puede modificarse facilitando un argumento cuando se emplee esa misma orden. En el ejemplo planteado, la primera palabra de protección tiene menos derechos que los especificados como segundo argumento del creat() (el valor 0666 octal implica la palabra de protección rw-rw-rw), cosa válida al emplear una máscara 022, pero la segunda palabra tiene derechos de ejecución para el propietario y el grupo y esos derechos no habían sido solicitados al crear el fichero. Por tanto, ningún valor de la máscara podría haber dado como resultado tal palabra de protección.

Se tienen cuatro procesos A, B, C y D que únicamente necesitan una sola ráfaga de CPU de 6 unidades de tiempo, donde las cuatro unidades centrales se dedican a ejecutar un procedimiento de un monitor M compartido por los cuatro procesos (es decir, se utiliza una unidad inicial y otra final para ejecutar código externo al monitor, mientras que las unidades desde la segunda a la quinta se ejecutan dentro del monitor). El sistema utiliza un algoritmo de planificación Round-Robin con quantum de dos unidades de tiempo. Los cuatro procesos llegan respectivamente en los instantes 0, 1, 2 y 3. Indique el tiempo de retorno y el instante de finalización de cada uno de estos procesos si se ha empleado un algoritmo FCFS para gestionar las colas de entrada al monitor.
NOTA: *En caso de empate se considerará que entra antes en la cola de preparados un proceso NUEVO que otro que haya estado SUSPENDIDO y éste antes que otro que haya sido expulsado de la CPU. Además, si un proceso debe suspenderse, cuando vuelva a la CPU dispondrá del quantum completo (no sólo de la parte que le quedase por consumir).*

3

Retorno: A: 9-0=9, B: 16-1=15, C: 21-2=19, D: 24-3=21

Finalización: A: 9 B: 16 C: 21 D: 24

En un sistema multiprogramado con una política de planificación SRTF se tienen cinco procesos P1, P2, P3, P4 y P5. Cada uno de ellos llega al sistema en los instantes 0, 1, 2, 16 y 20 respectivamente. Además, cada uno necesita una sola ráfaga de CPU de duración 6, 6, 2, 8 y 3 unidades de tiempo respectivamente. Calcule la productividad, utilización de CPU y tiempo de espera medio en dicho sistema, para este caso concreto.

4

Productividad: 5/27
Utilización CPU: 2500/27 = 92.59%
Tiempo de espera medio: (2+7+0+3+0)/5=2.4

Implemente mediante un monitor una utilidad de sincronización a la que vamos a llamar "EVENTO" y que ofrecerá tres operaciones:

- *Esperar(n : integer)*: Suspende al proceso que la invoca hasta que el evento llegue al estado "ocurrido", pero sólo si el número de procesos que esperaba el evento es menor que el número facilitado como argumento. Si el evento ya se encontraba en estado "ocurrido", esta operación no tiene ningún efecto.
- *Ocurrir*: El proceso que la invoca hace pasar al evento al estado "ocurrido", liberando a todos los procesos suspendidos en él.
- *Reiniciar*: Devuelve el evento al estado "no ocurrido".

Inicialmente el EVENTO se encuentra en estado "no ocurrido".

5

```
type evento=monitor
 var
   estado : ( ocurrido, noOcurrido );
   espera : condition;
 procedure entry esperar( n : integer );
  begin
    if ( estado = noOcurrido ) and (espera.awaited < n )
    then begin
      espera.wait;
      espera.signal
    end
  end;
 procedure entry ocurrir;
  begin
    estado := ocurrido;
    espera.signal
  end;
 procedure entry reiniciar;
  begin
    estado := noOcurrido
  end;
begin
  estado := noOcurrido
end.
```

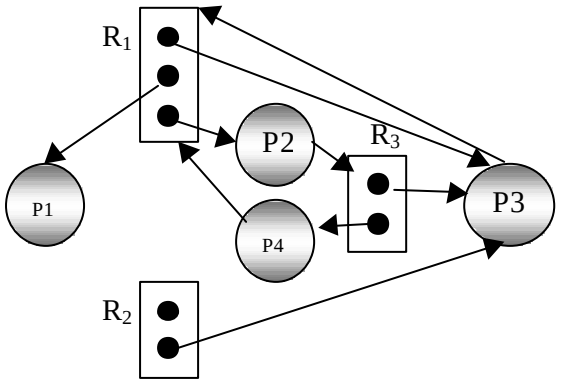
Se tienen cuatro procesos P1, P2, P3 y P4 que desean ejecutar los procedimientos que aparecen en los listados siguientes. En estos procedimientos, todos aquellos que se llaman "A" podrán ser ejecutados concurrentemente por dos procesos como máximo, mientras que los que empiecen con "B" deben ejecutarse en exclusión mutua (sólo entre los que empiecen con "B", no con los empezados con "A") y en el orden que sugiere el dígito que sigue a dicha letra (si dos procedimientos tienen el mismo dígito no debe imponerse un orden entre ellos, aunque deben seguir ejecutándose en exclusión mutua y terminar ambos antes de que pueda ejecutarse el procedimiento con el siguiente dígito). Utilice semáforos para proteger adecuadamente estos procedimientos e indique el valor inicial que deberá tener cada semáforo.

6

P1	***P2***	***P3***	***P4***
P(S1); A; *V(S1);* B1; *V(S2);* *P(S1);* A; *V(S1);* *P(S4);* B4; *V(S4);*	*P(S1);* A; *V(S1);* *P(S2);* B2; *V(S2);V(S3);* *P(S1);* A; *V(S1);*	*P(S2);* B2; *V(S2);V(S3);* *P(S1);* A; *V(S1);* *P(S4);* B4; *V(S4);*	*P(S1);* A; A; *V(S1);* *P(S3);P(S3);* B3; *V(S4);*

Semáforo	*S1*	*S2*	*S3*	*S4*					
Valor inicial	2	0	0	0					

Dado el siguiente estado de un sistema y suponiendo que los procesos P1 y P4 realizan seguidamente una petición cada uno de una instancia de R2 (se desconoce el orden en que efectúan tales peticiones) y que tras esto ningún proceso realiza ninguna petición más y no liberan los recursos hasta su terminación:



Indique si hay o no interbloqueo. Si lo hay, diga qué procesos están interbloqueados, si no lo hay, dé al menos una secuencia de terminación.

7

No están interbloqueados si P1 consigue completar primero su petición. Posibles secuencias de terminación:
<P1, P4, P2, P3>, <P1, P3, P2, P4>, <P1, P3, P4, P2>
Sí que estarán interbloqueados si la primera petición en ser atendida es la de P4. En ese caso, el interbloqueo afecta a todos los procesos.

Indique qué inconveniente plantea el algoritmo del banquero que lo hace difícilmente implementable en un sistema operativo actual.

8

Para poder utilizar este algoritmo cada proceso necesita informar al sistema sobre todos los recursos que podrá llegar a necesitar durante su ejecución. Para ello, o bien el proceso necesita realizar una llamada al sistema inicial que indique tal información (pero obviamente, existe la posibilidad de que la información facilitada sea incorrecta) o bien durante la compilación y enlace se ha averiguado tal información (cosa también imposible).

Sea un sistema de intercomunicación basado en mensajes con capacidad nula donde existen tres procesos P1, P2 y P3 que han llegado a la cola de preparados en ese orden en el instante cero. El sistema utiliza un algoritmo de

planificación FCFS. Cada llamada a procedimiento mostrada en el código (como An, Bn o Cn, siendo "n" un dígito) que aparece seguidamente necesita dos unidades de tiempo para ser completada. Las llamadas al sistema para enviar y recibir mensajes consumen un tiempo despreciable (puede asumirse que dicho tiempo es cero y que puede incluirse al final de la unidad de tiempo anterior a la aparición de la llamada). Indique para cada proceso su tiempo de retorno (en caso de que termine) o qué evento se ha quedado esperando en caso de que no llegue a terminar.

P1	***P2***	***P3***
A1;	receive(P1, msg1);	send(P1, msg1);
A2;	B1;	C1;
send(P2, msg1);	B2;	C2;
A3;	B3;	receive(P2, msg3);
receive(P3, msg2);	send(P3, msg2);	send(P1, msg2);
A4;	B4;	C3;

9

P1: 14.

P2: 20. Como alternativa, podría haber terminado en el instante 22, si el "send(P1,msg2)" realizado en el instante 18 ha devuelto un código de error y no ha suspendido a P3.

P3: Suspendido en el instante 18, al efectuar un send(P1, msg2) y P1 no esperar tal mensaje. Como alternativa, si el sistema devuelve un código de error en el instante 18 (ya que P1 ya no existe), este proceso terminaría en el instante 20 y su tiempo de retorno sería 20, obligando a P2 a terminar en el instante 22.

Dado el siguiente programa escrito en C con primitivas de sincronización POSIX, donde se asume que la función `printf` escribe directamente en memoria de vídeo y, por tanto, no llega a implicar la suspensión de los procesos (ya que no realiza una E/S que necesite espera):

```
#include <pthread.h>
#include <stdio.h>

pthread_mutex_t mutex =
PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond =
PTHREAD_COND_INITIALIZER;
pthread_cond_t seg =
PTHREAD_COND_INITIALIZER;
pthread_cond_t ter =
PTHREAD_COND_INITIALIZER;

void *hilo1() {
  pthread_mutex_lock(&mutex);
  printf( "Hilo 1 espera.\n" );
  pthread_cond_wait(&cond,&mutex);
  printf( "Hilo 1 avisa.\n" );
  pthread_cond_signal(&seg);
  pthread_mutex_unlock(&mutex);
  printf( "Termina hilo 1.\n" );
  pthread_exit(0);
}

void *hilo2() {
  pthread_mutex_lock(&mutex);
  printf( "Hilo 2 espera.\n" );
  pthread_cond_wait(&seg,&mutex);
  printf( "Hilo 2 avisa.\n" );
  pthread_cond_signal(&ter);
  pthread_mutex_unlock(&mutex);
  printf( "Termina hilo 2.\n" );
  pthread_exit(0);
}
```

```
void *hilo3() {
  pthread_mutex_lock(&mutex);
  printf( "Hilo 3 espera.\n" );
  pthread_cond_wait(&ter,&mutex);
  printf( "Hilo 3 liberado.\n" );
  pthread_mutex_unlock(&mutex);
  printf( "Termina hilo 3.\n" );
  pthread_exit(0);
}

void main() {
  pthread_t h1,h2,h3;


pthread_create(&h1,NULL,hilo1,NULL);

pthread_create(&h2,NULL,hilo2,NULL);

pthread_create(&h3,NULL,hilo3,NULL);

  printf( "Hilos creados.\n" );
  pthread_mutex_lock(&mutex);
  pthread_cond_signal(&cond);
  printf( "Esperando hilos.\n" );
  pthread_join(h1,NULL);
  printf( "Hilo 1 terminado.\n" );
  pthread_join(h2,NULL);
  printf( "Hilo 2 terminado.\n" );
  pthread_join(h3,NULL);
  printf( "Hilo 3 terminado.\n" );
  pthread_mutex_unlock(&mutex);
}
```

Indique qué se mostrará en pantalla al ejecutar el programa anterior, cuántos hilos terminarán la ejecución del código mostrado y en qué orden. Si algún hilo no consigue terminar indique por qué motivo no logra hacerlo. Asuma un algoritmo de planificación FCFS.

10

Salida en pantalla:
1) Hilos creados.
2) Esperando hilos.

No termina ningún hilo. El principal se queda suspendido en "pthread_join(h1,NULL);" pero sigue teniendo el mutex cerrado. Los hilos h1, h2 y h3 se quedan suspendidos al realizar su "pthread_mutex_lock(&mutex);" para tratar de adquirir el mutex.

Repita la cuestión anterior, pero ahora utilizando un algoritmo de planificación por prioridades estáticas expulsivas, donde las prioridades siguen este orden h1 > h2 > h3 > main. Es decir, "h1" es el más prioritario y "main" el menos.

11

Salida en pantalla:
1) Hilo 1 espera.
2) Hilo 2 espera.
3) Hilo 3 espera.
4) Hilos creados.
5) Esperando hilos.

No termina ningún hilo. El hilo h1 se ha suspendido al realizar el 'pthread_cond_wait(&cond, &mutex);" pues la condición todavía no ha ocurrido. Lo mismo sucede con los hilos h2 y h3 con sus primeras llamadas a esa misma operación sobre sus respectivas condiciones. No hay problemas con el mutex pues al suspenderse en la condición queda liberado. Finalmente, el hilo principal, tras haber creado a todos los anteriores, se suspende en el mismo lugar que en la cuestión anterior, por lo que todos los hilos han quedado suspendidos.

Dado el siguiente código en Ada95:

```
protected Objeto is
  function Lee_A return Integer;
  function Lee_B return Integer;
  entry Modifica_A(x : in Integer);
  procedure Modifica_B(y : in Integer);

private
  A,B : Integer:= 100; --valor inicial
end Objeto;

protected body Objeto is

  function Lee_A return Integer is
  begin
       return A;
  end Lee_A;

  function Lee_B return Integer is
  begin
       return B;
  end Lee_B;

  entry Modifica_A(x:in Integer) when B < 10 is
  begin
      A := A + x;
  end Modifica_A;

  procedure Modifica_B(y:in Integer) is
  begin
      B := B * y;
  end Modifica_B;
```

```
task T1;
task body T1 is
  j: integer;
begin
  loop
  j := Objeto.Lee_A;
  Objeto.Modifica_B(j);
  j := j + Objeto.Lee_B;
  Objeto.Modifica_A(j);
  end loop;
end T1;




task T2;
task body T2 is
  j: integer;
begin
  loop
  j:= Objeto.Lee_B;
  Objeto.Modifica_A(j);
  j := Objeto.Lee_A;
  Objeto.Modifica_B(j+1);
  end loop;
end T2;
```

```
end Objeto;
```

Razone si son posibles cada una de las siguientes situaciones:

a) Estando la tarea `T2` ejecutando el código de `Lee_B`, la tarea `T1` invoca a `Lee_A` y pasa a ejecutar su código.

b) Estando la tarea `T2` ejecutando el código de `Lee_B`, la tarea `T1` invoca a `Modifica_A` y pasa a ejecutar su código.

c) Estando la tarea `T1` ejecutando el código de `Modifica_B`, la tarea `T2` invoca a `Modifica_A` y pasa a ejecutar su código.

NOTA: En cada una de las situaciones, suponga que A = B = 100.

12	a)	Sí que es posible, ya que no existe exclusión mutua entre diferentes llamadas a funciones para un mismo objeto protegido. Múltiples funciones pueden ser ejecutadas concurrentemente.
	b)	No es posible. En Ada 95 no se permite que dentro de un mismo objeto haya una tarea ejecutando una función y otra ejecutando una entrada. La función trabaja como “lector” y la entrada como “escritor”. No puede haber lectores y escritores trabajando concurrentemente sobre el mismo objeto.
	c)	Tampoco es posible. En Ada 95 no se permite que dentro de un mismo objeto haya una tarea ejecutando una entrada y otra ejecutando un procedimiento. Tanto las entradas como los procedimientos tienen categoría de “escritores”. Cuando una tarea “escritora” modifica el objeto ninguna otra tarea puede acceder al mismo objeto.