

Rellene sus datos personales en la hoja de respuestas múltiples;

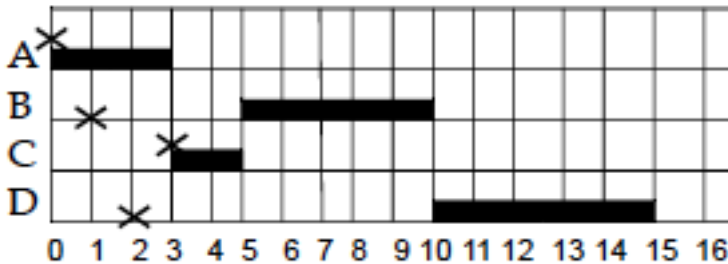
Puntuación sobre 10: la puntuación cada acierto es 10/nº items

Cada fallo penaliza con el valor de un acierto


Responda en la hoja de respuestas asignando **A=VERDADERO; B=FALSO**.

PREGUNTAS SOBRE GENERALIDADES SOBRE PROCESOS Y PLANIFICACION:

Sea el siguiente diagrama de asignación de CPU en que los procesos A, B, C y D tienen ráfagas de 3s, 5s, 2s y 5s respectivamente



En relación al diagrama anterior conteste a las preguntas 1,2,3,4

1 V	El tiempo de espera del proceso A es 0
2 F	El tiempo de espera del proceso C es 4
3 F	La penalización del proceso A es 0
4 V	La penalización del proceso D es 2,6
5 F	Cuando el S.O. detiene un proceso ejecutándose porque le va a asignar la CPU a otro, no es necesario salvar los valores actuales de los registros de la CPU porque ya se han ido salvando adecuadamente durante la ejecución de las instrucciones máquina por las que ha ido pasando la ejecución del proceso.
6 F	El algoritmo de asignación de CPU FIFO o FCFS (First Come First Served) es válido para entornos interactivos.
7 V	El planificador a corto plazo se ejecuta mas veces por unidad de tiempo que el planificador a largo plazo.
8 F	En el algoritmo de Barrido Ciclico con quantum Q: si el proceso actual se bloquea antes de finalizar el quantum entoces es imposible que en este momento se asigne la CPU a otro proceso en este momento, debiendo quedar ociosa hasta que acabe el quantum para que pueda pasar a ejecutarse otro proceso.
9 F	En el PCB o Bloque de Control de Proceso podemos ver en cualquier momento el valor actual que los registros de la CPU toman para cada uno de los procesos.
10 F	En el PCB o Bloque de Control de Proceso se salva el valor de las variables usadas en el código de cada proceso.
11 V	Es imposible que desde el estado “Bloqueado” un proceso pase directamente a estado “Ejecutandose” 
12 F	Solo puede haber 1 planificador a medio plazo a lo sumo.
13 F	En la política de planificación FCFS (First Come First Served) o FIFO, la penalización de una ráfaga es mayor para ráfagas largas que para ráfagas cortas.

14V	En un algoritmo de “Colas múltiples con realimentación” si se decide que cuando un proceso se desbloquea prosiga en la misma cola en que estaba antes de bloquearse, es porque suponemos que en nuestro sistema los procesos tienen sus ráfagas de duración parecida.
15F	La operación de cambiar la asignación de CPU de una hebra a otra (dentro de un mismo proceso) es más corta si se trata de “hebras Nucleo” que si se trata de “hebras de usuario”.
16V	La política de planificación “El más corto primero” trata mejor a las ráfagas cortas si hay apropiatividad (derecho preferente o desplazamiento). que si no la hay.
17V	Si hay una mezcla de trabajos (unos con muchas E/S, otros con mucho cálculo) entonces se podrá aprovechar mejor el tiempo de CPU
18V	Si las tareas de gestión de hebras (creación, reparto de tiempo entre hebras...) se realizan en funciones de librería incluidas por el compilador, entonces decimos que son hebras de usuario.

19F	En el algoritmo de planificación “El más corto primero” con derecho preferente el tiempo de espera es independiente del tiempo de cpu de la ráfaga.
20V	En el caso de “hebras de usuario”: La operación de cambiar la asignación de CPU de una hebra a otra es más corta que la de cambiar la asignación de CPU de un proceso a otro.
21F	En el caso de “hebras Núcleo”, si una hebra queda bloqueada ello implica que todas las demás del mismo proceso también queden bloqueadas.
22V	En la política de planificación de Barrido Cíclico con quantum Q, el tiempo de espera de una ráfaga es mayor para ráfagas largas que para ráfagas cortas.
23F	En la política de planificación de Barrido Cíclico con quantum Q, la penalización de una ráfaga es mayor para ráfagas largas que para ráfagas cortas.
24F	En la política de planificación FCFS (First Come First Served) o FIFO, el tiempo de espera de una ráfaga es mayor para ráfagas largas que para ráfagas cortas.
25F	Si tenemos una política de planificación no apropiativa (sin desplazamiento), no se puede manejar una interrupción mientras se está ejecutando un proceso.
26V	Siempre que el proceso actual no desea seguir ejecutándose ha de llamarse al planificador .a corto plazo.
27V	Una ventaja de las “hebras de usuario” es que se pueden ejecutar en cualquier S.O. mejorando la transportabilidad del software.

Suponga que tenemos política de planificación Round-Robin **no apropiativa** (sin desplazamiento o si derecho preferente) y tenemos los siguientes procesos:

- A: En estado **Ejecutándose** y le queda cierto de tiempo para finalizar el *quantum*;
- B: **Preparado** para ejecutarse;
- C: **Bloqueado** esperando una entrada/salida de disco.

Tenemos estas prioridades:

B es el menos prioritario, después está A, y C es el más prioritario.

Cuando se produce una interrupción señalando fin de quantum..... (Responda Verdadero/Falso cada uno de los ítems siguientes)

28- F El proceso C pasa a **Preparado**

29- V El proceso A sigue en estado **Ejecutándose**

30- F El proceso B pasa a **Ejecutándose**.

Si en lugar de producirse fin de quantum se produce el fin de la operación de E/S que esperaba C.....

(Responda Verdadero/Falso cada uno de los ítems siguientes)

31- F El proceso C pasa a **Ejecutándose**

32- F El proceso B pasa a **Ejecutándose**.

33- V El proceso A no cambia su estado (es el que está **Ejecutándose**)

Tenemos un algoritmo de asignación de CPU de “Colas múltiples con realimentación”. Queda definido así:

- Tres colas gestionadas mediante **Round Robin**:
 - Cola 1 con quantum = 2 ms
 - Cola 2 con quantum = 4 ms
 - Cola 3 con quantum = 8 ms
- Algoritmo entre colas: **prioridades no apropiativo**, cola 1 mayor prioridad, cola 3 menor prioridad.
- Los procesos entran inicialmente en la cola 1
- Cuando un proceso se bloquea, al regresar a la cola de ejecutables entra en la cola 1
- Un proceso se traspa a una cola de menor prioridad cuando agota un quantum de tiempo completo.

Tenemos los siguientes procesos:

Proceso A: Momento de creación t=0;

Comportamiento: 1 ms de CPU, 3ms de bloqueo y así cíclicamente hasta completar 4ms de tiempo total de CPU.

Proceso B: : Momento de creación t=2;

Comportamiento: 3 ms de CPU, 4ms de bloqueo y así cíclicamente hasta completar 6ms de tiempo total de CPU.

Proceso C: Momento de creación t=3;

Comportamiento: una única ráfaga de 23 ms de CPU sin ráfagas de bloqueo.

Realice el diagrama de asignación de CPU (que no ha de entregar) para poder responder lo siguiente:

34- V El proceso B satisface el criterio de pasar a la cola 2 en el momento t=4

35- F El proceso A satisface el criterio de pasar a la cola 2 en el momento t = 7.

36- V El proceso A nunca llega a la cola 2.

37- F En t=4, justo antes de elegir el siguiente proceso para asignarle la CPU, la cola de ejecutables nº1 tiene como primer proceso a A.

38- F El tiempo de espera total del proceso A es de 3 ms.

39- F El proceso C nunca llega a la cola 3.

40- V Entre t=1ms y t=2ms no hay ningún proceso ejecutable

PREGUNTAS SOBRE IMPLEMENTACION EN LINUX:

41 F	En un kernel apropiativo, cuando se produce la interrupción de fin de quantum la apropiación ocurre exactamente en el instante de fin de quantum, independientemente del código que se estuviera ejecutando por parte del proceso actual.
42 F	El kernel sabe quién es el proceso actual porque hay un determinado valor de estado en el elemento state de task_struct.
43 V	Todo proceso cuando termina pasa por el estado EXIT_ZOMBIE
44 V	El valor de retorno que un proceso hijo pasa al padre se almacena en la task_struct del hijo
45 V	<pre>struct task_struct *task; for (task = current; task != &init_task; task = task->parent) {printf("%s[%d]\n", task->pid)</pre> <p>El anterior fragmento de código dentro del kernel muestra en pantalla el pid del proceso actual, de su padre, de su abuelo, bisabuelo, tatarabuelo, etc... hasta llegar al proceso init.</p>
46 F	El anterior fragmento de código dentro del kernel muestra en pantalla el pid de todos los procesos existentes.
47 F	La principal razón de que task_struct y thread_info se combinen en una union es la de ahorrar espacio.
48 F	Cuando un proceso lanza varias hebras, no se crea una task_struct para cada una sino que todas ellas comparten la task_struct del padre.
49 V	Durante la actuación de copy_process se establece el estado del hijo a TASK_UNINTERRUPTIBLE porque se desea que la totalidad del proceso hijo se realice sin procesar señales.

50 V	Sea el proceso A que hace un fork creando al proceso B y situémonos en el caso de que se dispone de la técnica copy-on-write; cuando el proceso B haga un exec entonces se asignará un nuevo espacio de memoria para almacenar el código del programa aludido en exec
51 V	Sea el proceso A que hace un fork creando al proceso B y situémonos en el caso de que se dispone de la técnica copy-on-write; la zona de datos de ambos procesos no se duplica sino que es físicamente compartida por ambos procesos, JUSTO TRAS EL FORK.
52 V	El compilador añade automáticamente una llamada a exit cuando main termina en el caso de que el programador no la haya incluido
53 V	Para cada recurso que esté utilizando un proceso, cuando éste termina se decrementa el contador correspondiente que indica el número de procesos que lo están utilizando.
54 F	Cuando un proceso termina sin hacer wait, se produce una terminación en cascada en que los hijos que tenga son terminados obligatoriamente
55 F	En la clase de planificación CFS el tiempo virtual vruntime de un proceso es menor al aumentar el valor numérico de prioridad del proceso
56 F	En la clase de planificación CFS la magnitud “peso” de un proceso aumenta a mayor valor numérico de prioridad de un proceso
57 V	El planificador principal va recorriendo las distintas cases de planificacion en orden de mayor a menor importancia encontrando la primera que no esté vacía, y el método de planificación asociado determina el siguiente proceso a ejecutar.
58 V	Una entidad de planificación se representa mediante una instancia de sched_entity
59 V	Un proceso aislado puede ser una entidad de planificación
60 F	Cuando un proceso tiene su campo policy igual a SCHED_IDLE entonces dicho proceso es la tarea vacia consistente en un bucle infinito que se ejecuta cuando no existe ningún proceso a ejecutar.
61 F	Cuando el flag TIF_NEED_RESCHED cambia a estado establecido se produce un cambio de asignación de CPU inmediatamente.
62 V	En la estructura sched_class se recogen diversos punteros a funciones que implementan las operaciones de cada método de planificación.
63 F	La clase de planificación a la que pertenece un proceso determinado se almacena en el campo policy de la task_struct del proceso.
64 V	Para dar el control a un nuevo proceso se debe ejecutar la función pick_next_task cuyo puntero está en la clase de planificación de la task_struct del proceso actual.
65 V	Aunque haya varias CPUs no es posible ejecutar un proceso en varias CPUs simultáneamente.
66 V	El método de planificación periódico de la clase de planificación a que corresponde el proceso actual es activado por el planificador periódico scheduler_tick.
67 F	Cuando se vuelve desde modo kernel al espacio de usuario se chequea el flag TIF_NEED_RESCHED
68 F	Dentro de las funciones definidas en un programa, cada vez que se retorna de una función (en el espacio de usuario) se chequea el flag TIF_NEED_RESCHED
69 V	En CFS, se llama latencia al menor periodo de tiempo en que se asegura que todos los procesos han sido elegidos para ejecutarse.
70 F	En CFS, a mayor valor de latencia hay un mayor coste de tiempo de CPU en realizar cambios de contexto.
71 V	En CFS se elige el proceso con un menor valor de vruntime
72	El valor de vruntime de un proceso se almacena en su task_struct. (VER NOTA1)
73 V	El valor de vruntime se actualiza siempre que haya que reelegir un nuevo proceso.
74 V	El rbtree usado en CFS para almacenar los tiempos vruntime almacena ese dato solo de los procesos ejecutables.
75 F	En un kernel apropiativo está prohibido retirar la CPU a un proceso que esté ejecutando código kernel
76	El flag preempt_count se almacena en la task_struct del proceso. (VER NOTA1)
77 V	Si un proceso tiene el flag preempt_count a un valor >0 entonces no se le puede retirar el control de la CPU.
78 F	Mientras un proceso está ejecutando código de usuario podría tener el flag preempt_count a un valor >0.
79 F	Si el flag TIF_NEED_RESCHED está establecido y hay otro proceso más prioritario que el actual, entonces se realiza el cambio de contexto independientemente del valor que tenga preempt_count
80 F	El flag preempt_count es un único flag para todo el sistema.
81 V	Cuando el kernel libere todos los lock del proceso actual llegando preempt_count a valer 0, entonces se chequea TIF_NEED_RESCHED y si está activo se invoca al planificador.

NOTA1: PREGUNTAS ANULADAS. Realmente vruntime está en la sched_entity y no en task_struct, y preempt_count está en thread_info y no en task_struct, pero en ambos casos “es como si lo estuviera”, ya que muchas veces se alude a task_struct como el conjunto de todos los datos que se guardan sobre el proceso. Por lo tanto veo mejor anular estos item.