1. **Cuestiones sobre procesos, y asignación de CPU:**
   a. **¿Es necesario que lo último que haga todo proceso antes de finalizar sea una llamada al sistema para finalizar? ¿Sigue siendo esto cierto en sistemas monoprogramados?**
   Sí, es necesario en todos los casos, incluso en sistemas multiprogramados. En general, cuando un proceso ejecuta la última instrucción, solicita al sistema operativo su finalización con la llamada *exit*. El padre de un proceso puede finalizar la ejecución de sus hijos con las llamadas *abort* o *kill*.

   b. **Cuando un proceso se bloquea, ¿deberá encargarse él directamente de cambiar el valor de su estado en el descriptor de proceso o PCB?**
   No, es el planificador a corto plazo *(dispatcher)* el que se encarga de guardar el contexto del proceso en su PCB y por tanto también se encarga de cambiar el valor de su estado.

   c. **¿Qué debería hacer el planificador a corto plazo cuando es invocado pero no hay ningún proceso en la cola de ejecutables?**
   El planificador a corto plazo invoca al que se conoce como *"proceso nulo"* que está siempre listo y tiene la prioridad más baja.

   d. **¿Qué algoritmos de planificación quedan descartados para ser utilizados en sistemas de tiempo compartido?**
   Se suelen descartar aquellos que no se basen en quantums de tiempo, es decir, aquellos que no reparten el tiempo de CPU en porciones pequeñas. La esencia de los sistemas de tiempo compartido es repartir de forma equitativa el tiempo de CPU, por tanto, es necesario repartirlo en porciones pequeñas.

2. **La representación gráfica del cociente *[(tiempo_en_cola_ejecutables + tiempo_de_CPU)/tiempo_de_CPU]* frente a *tiempo_de_CPU* suele mostrar valores muy altos para ráfagas muy cortas en casi todos los algoritmos de asignación de CPU. ¿Por qué?**
   Si se obtienen valores muy altos de ese cociente, es porque el numerador es más grande que el numerador, lo que se traduce que el tiempo que pasa cierto proceso el la cola de ejecutables es mayor que el tiempo que se le ha asignado para la CPU. Esto quiere decir que las ráfagas son demasiado cortas y no favorecen a que los procesos largos trabajen con la CPU.

3. **Para cada una de las llamadas al sistema siguientes, especificar y explicar si su procesamiento por el sistema operativo requiere la invocación del planificador a corto plazo:**
    a. **Crear un proceso**
    Sí, si la política de planificación es apropiativa, excepto Round-Robin, ya que no se le puede quitar un proceso hasta que no acabe el quantum.

    b. **Abortar un proceso, es decir, terminarlo forzosamente**
    Sí hay que invocar al planificador a corto plazo, ya que habrá que expulsar a dicho proceso de ejecución para dar paso a otro.

    c. **Suspender o bloquear un proceso**
    Igual que el apartado anterior.

    d. **Reanudar un proceso (inversa al caso anterior)**
    Depende de la política de planificación. Estás pasando un proceso de estado suspendido a estado preparado, esencialmente es lo mismo que en el apartado a.

    e. **Modificar la prioridad de un proceso**
    Misma respuesta que apartados a y d.

4. **Sea un sistema multiprogramado que utiliza el algoritmo Por Turnos *(Round-Robin)*. Sea S el tiempo que tarda el despachador en cada cambio de contexto. ¿Cuál debe ser el valor de quantum Q para que el porcentaje de uso de la CPU por los procesos de usuario sea del 80%?**
    El tiempo que un proceso está en la CPU es: T = Q+S
    También sabemos que 0.8T = Q, por tanto, T = Q/0.8

    Sustituimos en la expresión de arriba:
    Q/0.8 = Q+S
    Q = 0.8(Q+S)
    Q = 0.8Q + 0.8S
    Q - 0.8Q = 0.8S
    0.2Q = 0.8S
    Q = 4S

5. **Sea un sistema multiprogramado que utiliza el algoritmo Por Turnos *(Round-Robin)*. Sea S el tiempo que tarda el despachador en cada cambio de contexto, y N el número de procesos existente. ¿Cuál debe ser el valor de quantum Q para que se asegura que cada proceso "ve" la CPU al menos cada T segundos?**
    N*(Q+S) <= T

6. **¿Tiene sentido mantener ordenada por prioridades la cola de procesos bloqueados? Si lo tuviera, ¿en qué casos sería útil hacerlo?**
   Sí tiene sentido, pero solo si el suceso por el cual los procesos están bloqueados es el mismo. Por ejemplo una impresora, tiene sentido que dentro de la cola de bloqueados de la impresora los procesos tengan prioridad, pero el resto de procesos no tiene nada que ver.

   El mantener ordenados los procesos por prioridades en la cola de procesos bloqueados es útil para casos en los que un proceso en la cola de bloqueados acabe de desbloquearse y éste tenga más prioridad que cualquiera de los de la cola de listos. Otro caso sería por ejemplo traer un proceso bloqueado con mayor prioridad a memoria principal si el sistema operativo tiene motivos para creer que el evento al cual está esperando va a ocurrir pronto (este último cambio de estado no se suele admitir por ser muy costoso).

7. **¿Puede el procesador manejar una interrupción mientras está ejecutando un proceso si la política de planificación que utilizamos es no apropiativa?**
   Claro, ya que un proceso no puede estar acaparando tiempo de CPU indefinidamente, y por tanto se sacará al proceso de ejecución para dar paso a otro.

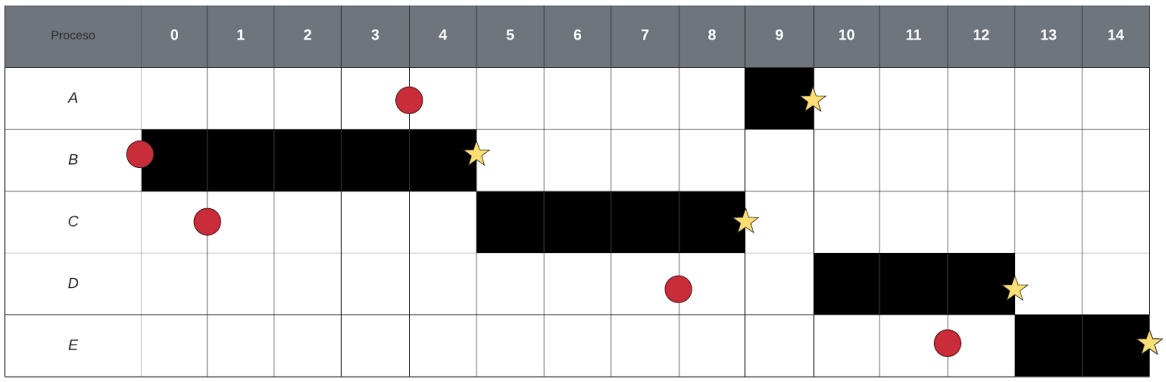
8. **Suponga que es responsable de diseñar e implementar un sistema operativo que va a utilizar una política de planificación apropiativa. Suponiendo que tenemos desarrollado el algoritmo de planificación a tal efecto, ¿qué otras partes del sistema operativo habría que modificar para implementar tal sistema? ¿Cuáles serían tales modificaciones?**
   En el caso de políticas Round-Robin, habría que modificar concretamente las rutinas de tratamiento de interrupción del reloj, ya que es quien se encarga de cambiar de proceso cuando un proceso agota su quantum.

   Para otras políticas apropiativas, habría que modificar lo último que hace cada proceso, y en concreto, habría que indicarle que hiciese una llamada al planificador. Además, habría que modificar también la rutina de tratamiento de interrupciones en general (por ejemplo, en procesos que cambian de estado cuando esperan a E/S).

9. **En el algoritmo de planificación FCFS, la penalización *((t+t_espera)/t),* ¿es creciente, decreciente o constante con respecto a t *(tiempo de servicio de CPU requerido por un proceso)*? Justifique su respuesta.**
   Es decreciente, ya que a medida que aumenta t y suponiendo que el tiempo de espera es siempre el mismo, el cociente resulta menor, es decir, la penalización resultante es menor a medida que aumenta el tiempo de servicio de CPU requerido por un proceso.

10. En la tabla siguiente se describen cinco procesos:

Proceso	Tiempo de creación	Tiempo de CPU
A	4	1
B	0	5
C	1	4
D	8	3
E	12	2



Si suponemos que tenemos un algoritmo de planificación que utiliza una política FIFO (primero en llegar, primero en ser servido), calcula:

a) **Tiempo medio de respuesta**
Proceso A: 10-4 = 6
Proceso B: 5
Proceso C: 9-1 = 8
Proceso D: 13-8 = 5
Proceso E: 15-12 = 3

Tiempo medio de respuesta: 27/5 = 5,4

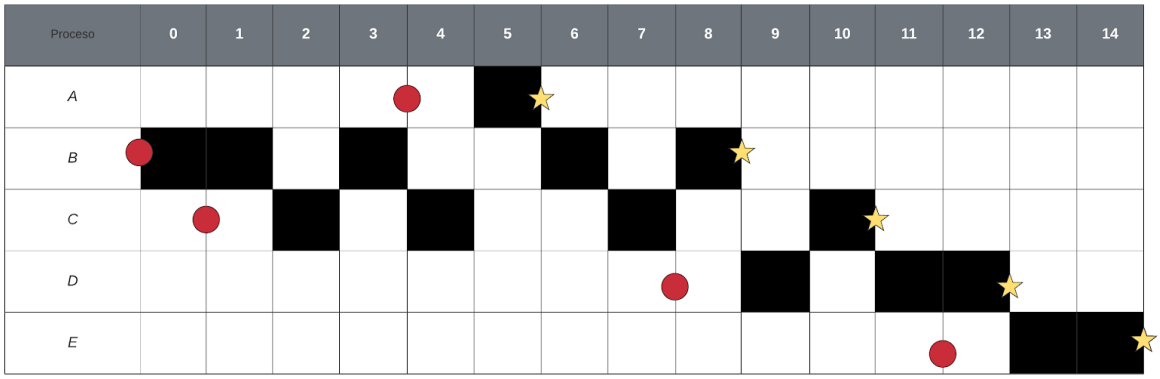
b) **Tiempo medio de espera**
Proceso A: 9-4 = 5
Proceso B: 0
Proceso C: 5-1 = 4
Proceso D: 10-8 = 2
Proceso E: 13-12 = 1

Tiempo medio de espera: 14/5 = 2,8

c) **La penalización, es decir, el cociente entre el tiempo de respuesta y el tiempo de CPU**
Proceso A: 6/1 = 6
Proceso B: 5/5 = 1 (sin penalización)
Proceso C: 8/4 = 2
Proceso D: 5/3 = 1,67
Proceso E: 3/2 = 1,5

11. Utilizando los valores de la tabla del problema anterior, calcula los tiempos medios de espera y respuesta para los siguientes algoritmos:

a) **Por Turnos con quantum q=1**



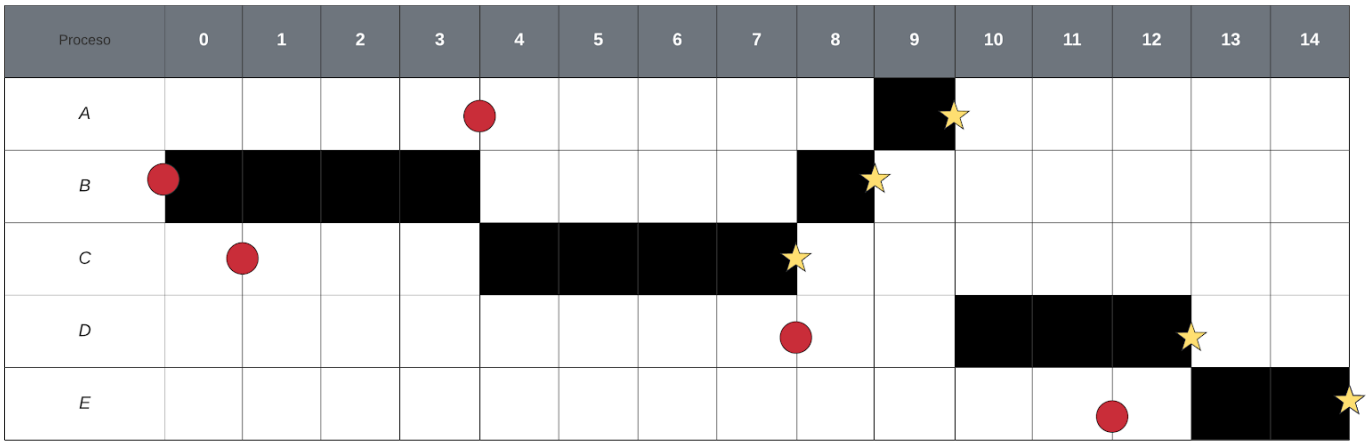
Tiempos de espera:
Proceso A: 1
Proceso B: 0
Proceso C: 1
Proceso D: 1
Proceso E: 1

Tiempo medio de espera: 4/5 = 0,8

Tiempos de respuesta:
Proceso A: 6-4 = 2
Proceso B: 2-0 = 2
Proceso C: 3-1 = 2
Proceso D: 10-8 = 2
Proceso E: 15-12 = 3

Tiempo medio de respuesta: 11/5 = 2,2

b) Por Turnos con quantum q=4



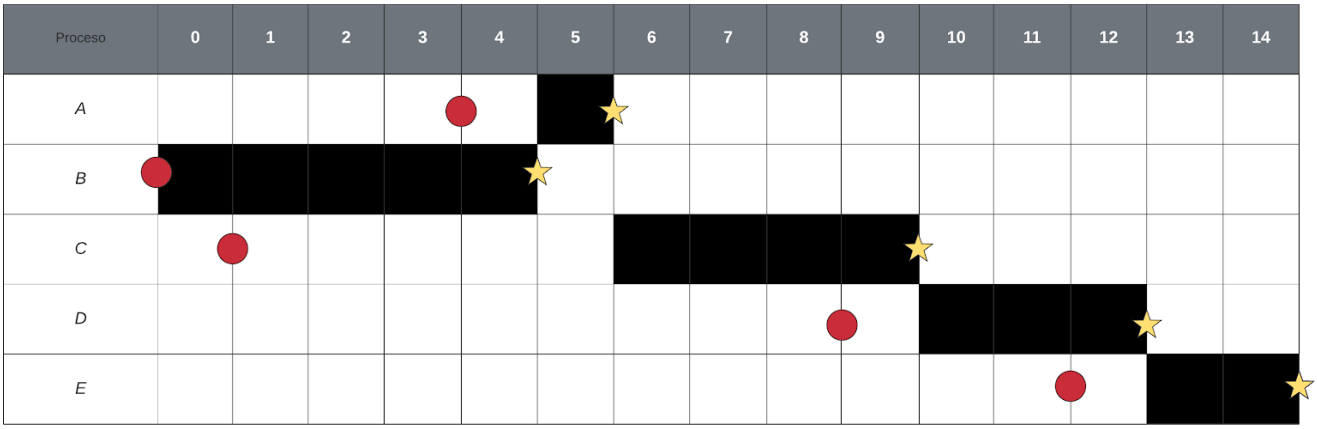
Tiempos de espera:
Proceso A: 9-4 = 5
Proceso B: 0
Proceso C: 4-1 = 3
Proceso D: 10-8 = 2
Proceso E: 13-12 = 1

Tiempo medio de espera: (5+0+3+2+1)/5 = 11/5 = 2,2

Tiempos de respuesta:
Proceso A: 10-4 = 6
Proceso B: 4-0 = 4
Proceso C: 8-1 = 7
Proceso D: 13-8 = 5
Proceso E: 15-12 = 3

Tiempo medio de respuesta: 25/5 = 5

c) El más corto primero (SJF). Suponga que se estima una ráfaga igual a la real.



Tiempos de espera:
Proceso A: 5-4 = 1
Proceso B: 0
Proceso C: 6-1 = 5
Proceso D: 9-8 = 1
Proceso E: 1

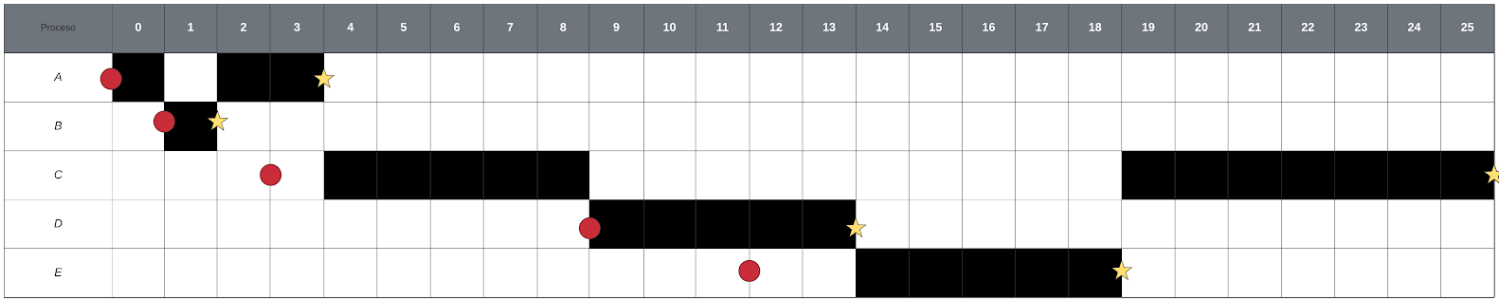
Tiempo medio de espera: 8/5 = 1,6

Tiempos de respuesta:
Proceso A: 6-4 = 2
Proceso B: 5
Proceso C: 10-1 = 9
Proceso D: 13-8 = 5
Proceso E: 15-12 = 3

Tiempo medio de respuesta: 24/5 = 4,8

12. Calcula el tiempo de espera medio para los procesos de la tabla utilizando el algoritmo: el primero más corto apropiativo (o primero el de tiempo restante menor, SRTF).

Proceso	Tiempo de creación	Tiempo de CPU
A	0	3
B	1	1
C	3	12
D	9	5
E	12	5



Tiempos de espera:
Proceso A: 0
Proceso B: 0
Proceso C: 4-3 = 1
Proceso D: 0
Proceso E: 14-12 = 2

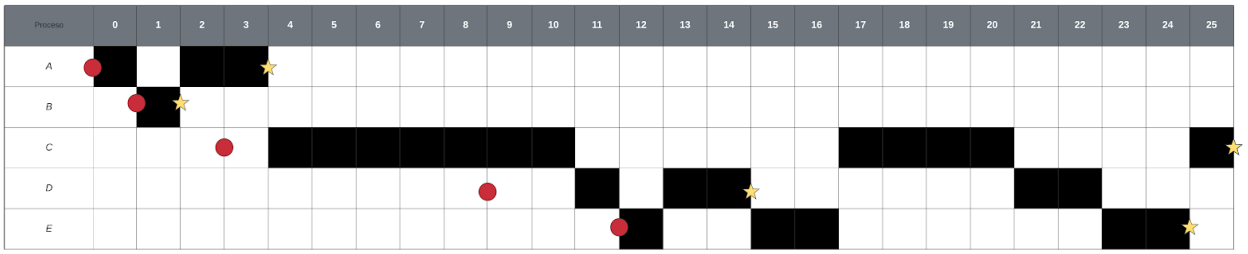
Tiempo medio de espera: 3/5 = 0,6

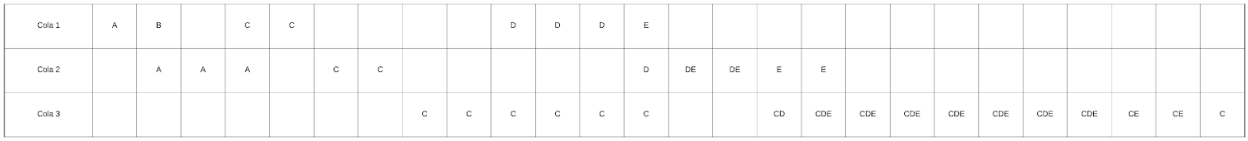
13. Utilizando la tabla del ejercicio anterior, dibuja el diagrama de ocupación de CPU para el caso de un sistema que utiliza un algoritmo de colas múltiples con realimentación con las siguientes colas:

Cola	Prioridad	Quantum
1	1	1
2	2	2
3	3	4

Y suponiendo que:

a) **Todos los procesos inicialmente entran en la cola de mayor prioridad (menor valor numérico). Cada cola se gestiona mediante la política Por Turnos.**
b) **La política de planificación entre colas es por prioridades no apropiativo.**
c) **Un proceso en la cola *i* para a la cola *i+1* si consume un quantum completo sin bloquearse**
d) **Cuando un proceso llega a la cola de menor prioridad, permanece en ella hasta que finalice.**





14. **Suponga que debe maximizar la eficiencia de un sistema multiusuario y que está recibiendo quejas de muchos usuarios sobre los pobres tiempos de respuesta (o tiempos de vuelta) de sus procesos. Los resultados obtenidos con una herramienta de monitorización del sistema nos muestran que la CPU se utiliza al 99'9% de su tiempo y que los procesadores de E/S están activos solo un 10% de su tiempo. ¿Cuáles pueden ser las razones de estos tiempos de respuesta pobres y por qué?**
Si se quejan los usuarios es porque antes funcionaba de otra manera.

    La razón más probable es que el quantum sea demasiado grande, lo que inevitablemente se traduce en tiempos de respuesta muy elevados.

    En este tipo de ejercicios Patricia prefiere que elijas cuáles son las razones y expliques por qué, pero voy a comentar una a una.

    a) **El quantum en la planificación Round-Robin es muy pequeño**
    Patricia la daría por buena si eliges esta también, ya que, aunque con este motivo los procesos se moverían muy rápido por todo el sistema, se generaría mucha sobrecarga por muchos cambios de contexto. No es el motivo principal pero se acepta.

    b) **La memoria principal es insuficiente**
    Si fuese insuficiente, significaría que se está utilizando memoria secundaria, y esto no tiene sentido ya que los procesadores de E/S están activos sun 10% del tiempo. Si este número fuese más grande, sí tendría sentido.

    c) **El sistema operativo tiene que manejar mucha memoria principal por lo que las rutinas de gestión de memoria están consumiendo todos los ciclos de CPU**
    Misma razón que en el apartado anterior.

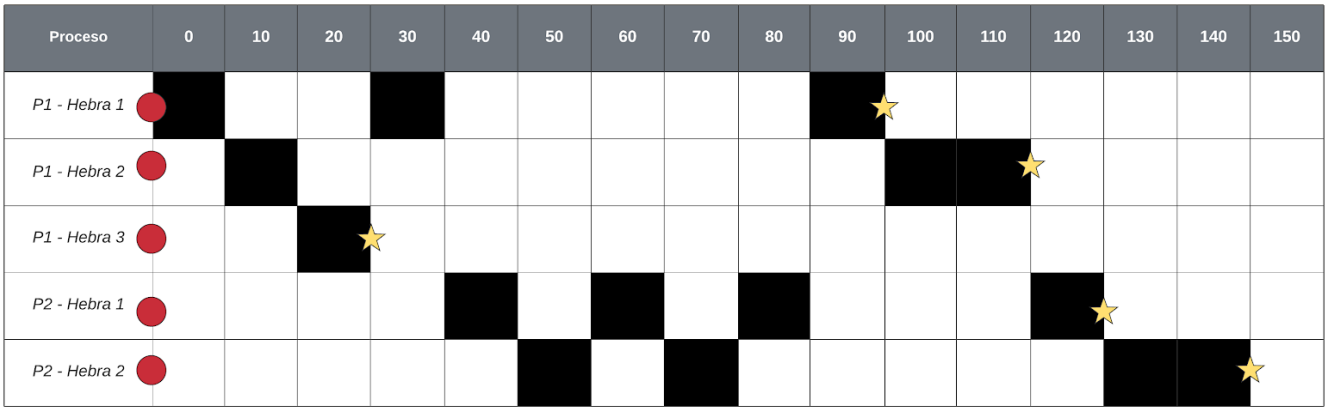
    d) **La CPU es muy lenta**
    Si la CPU es muy lenta, lo sería también al principio. Es decir, si los usuarios no se quejaron en un principio, significa que antes el sistema funcionaba bien y ahora no, por lo que la CPU no tiene sentido que sea lenta, si no, lo hubiese sido siempre.

15. **Compare el rendimiento ofrecido al planificar el conjunto de tareas multi-hebras descrito en la tabla y bajo las siguientes configuraciones:**
    a) **Sistema operativo multiprogramado con hebras de usuario. En este sistema se dispone de una biblioteca para la programación con hebras en el espacio de usuario. El algoritmo de planificación de CPU utilizado por el SO es *Round-Robin* con un *quantum* de 50 u.t (unidades de tiempo). El planificador de la biblioteca de hebras reparte el *quantum* del proceso (tarea) entre las hebras utilizando *Round-Robin* con un *quantum* para cada hebra de 10 u.t. Suponga que no existe coste en el cambio de contexto entre hebras ni entre procesos.**
    b) **Sistema operativo multiprogramado con hebras kernel. El SO planifica las hebras usando *Round-Robin* con un quantum de 10 u.t. Como en el apartado anterior, suponga que no existe coste en la operación de cambio de contexto. Considere además que las operaciones de E/S de un proceso únicamente bloquean a la hebra que las solicita.**

    Suponga en ambos casos que los dos procesos están disponibles y que el planificador entrega la CPU al proceso P1. Para realizar la comparación represente en cada caso el diagrama de ocupación de CPU y calcule el grado de ocupación de CPU (tiempo CPU ocupada/tiempo total).

Proceso	Hebras	Ráfaga de CPU	Tiempo de E/S	Ráfaga de CPU
P1	Hebra 1	20	30	10
	Hebra 2	30	-	-
	Hebra 3	10	-	-
P2	Hebra 1	30	30	10
	Hebra 2	40	-	-

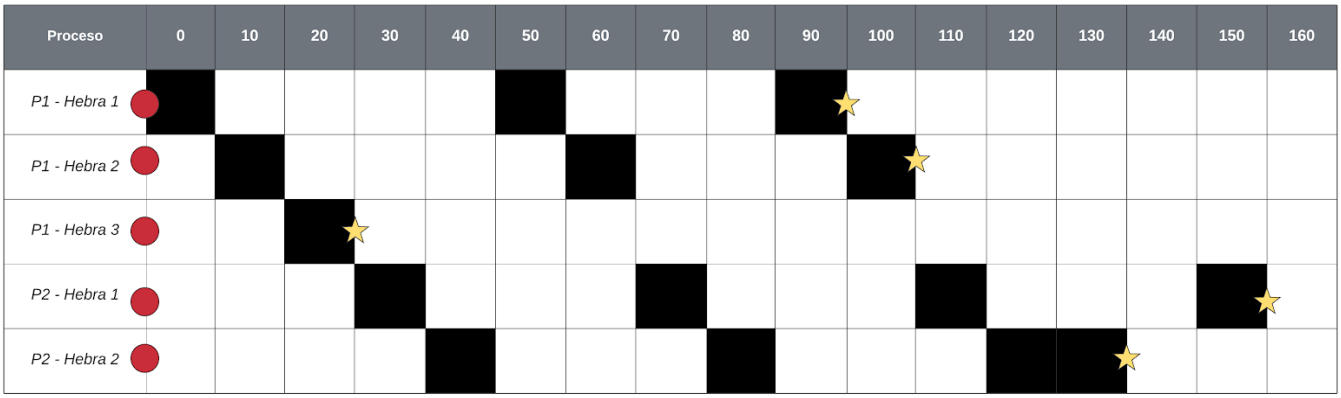
Apartado a)



Se reparte de la siguiente forma: a cada proceso le corresponden 50 unidades de tiempo, y dentro de cada proceso, se le reparte a cada hebra el tiempo de CPU 10 unidades de tiempo cada vez. La hebra 1 del proceso 1, cuando consume la primera ráfaga de 20 unidades de tiempo se bloquea esperando E/S, por lo tanto, el resto de hebras de ese proceso también se bloquea, y se le pasa el tiempo de CPU al proceso 2.

El grado de ocupación de CPU es:
150/150 = 1 = 100%

Apartado b)



El grado de ocupación de CPU es:
150/160 = 93,75%

16. **¿El planificador CFS de Linux favorece a los procesos limitados por E/S (cortos) frente a los procesos limitados por CPU (largos)? Explique cómo lo hace.**
    El planificador CFS guarda la cantidad de tiempo de CPU que una determinada tarea ha consumido. Este valor se denomina *tiempo de ejecución virtual* o *virtual runtime* y se calcula sumando el tiempo que ha consumido cierta tarea de CPU, su prioridad y su peso. Por tanto, un proceso que esté limitado por E/S *(que sea corto)*, normalmente se pasa bloqueado mucho tiempo, y por tanto el valor de la suma mencionada anteriormente es muy bajo porque apenas ha consumido tiempo de CPU.

    A menor tiempo de virtual runtime, más necesidad tiene esa tarea de usar el procesador.

    vruntime = tiempo de CPU + prioridad + peso

17. **¿Cuál es el problema que se plantea en Linux cuando un proceso no realiza la llamada al sistema *wait* para cada uno de sus procesos hijos que han terminado su ejecución? ¿Qué efecto puede producir esto en el sistema?**
    Cuando un proceso hijo termina pero no *"se le espera"*, se convierte en un zombie. El kernel mantiene algo de información sobre estos procesos para que si el padre decide *"esperarlo"* en un momento futuro, éste se pueda identificar. Si no se le espera nunca, el zombie estará ocupando espacio en la tabla de procesos y si esta tabla se llena, no se podrán crear más procesos.

18. **La orden clone sirve tanto para crear un proceso en Linux como una hebra.**
    a. **Escriba los argumentos que debería tener clone para crear un proceso y una hebra**
    Desde el punto de vista kernel no hay diferencia entre una hebra y un proceso en Linux, por tanto, los argumentos de la llamada al sistema clone serán los mismos para un proceso y una hebra y en concreto son:
        - Un puntero a una función, que el proceso hijo empezará ejecutando
        - Puntero a void que especifica la localización de la pila usada por el proceso hijo
        - Los distintos argumentos y flags a usar pasados como punteros a void

    b. **Dibuje las principales estructuras de datos del kernel que reflejan las diferencias entre ambas**
    Aquí habría que especificar sobre todo las diferencias entre task_struct y demás estructuras de **Linux**.