UNIVERSIDAD
DE GRANADA

Relación de problemas

Temas 1 y 2

Estructuras de sistemas operativos

Procesos y hebras

MAURICIO LUQUE JIMÉNEZ

17 de noviembre de 2025

Sistemas Operativos

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?

No, no es estrictamente necesario. Cuando un proceso finaliza de forma autónoma, sí debe realizar una llamada al sistema, pero también se puede dar el caso de que el proceso no se finalice de forma autónoma, sino que sea finalizado forzosamente por un proceso padre o por el propio sistema operativo (por ejemplo, en un caso de error fatal), en cuyo caso no se realiza una llamada al sistema. Lo mismo ocurre para sistemas monoprogramados, donde, si bien los procesos no pueden ser finalizados por procesos padre, pueden seguir siendo abortados por el propio sistema operativo.

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, esa tarea la realiza el sistema operativo. Un proceso puede bloquearse durante un cambio de contexto, en el que es el propio sistema operativo el que guarda la información relativa a ese proceso para volver a cargarlo cuando sea necesario. El PCB pertenece al sistema operativo y sólo el kernel lo puede manipular, de manera que ningún proceso puede modificarlo directamente.

c) ¿Qué debería hacer el planificador a corto plazo cuando es invocado pero no hay ningún proceso en la cola de ejecutables?

Si no hay procesos en la cola de preparados en el momento de invocar al planificador a corto plazo, el planificador no puede crear trabajos ni acceder a la cola de trabajos, por lo que la CPU permanece en estado inactivo (*idle)* hasta que un proceso se desbloquee o un proceso pase de la cola de trabajos a la cola de preparados.

d) ¿Qué algoritmos de planificación quedan descartados para ser utilizados en sistemas de tiempo compartido?

Quedan descartados los algoritmos no apropiativos, ya que no permiten expulsar procesos de la CPU hasta que estos mismos se terminen o se bloqueen, lo que impide repartir la CPU entre diferentes usuarios para que puedan ejecutar procesos en un mismo intervalo de tiempo.

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?

Al tratarse de ráfagas muy cortas, cada proceso se ejecuta en CPU durante intervalos muy breves de tiempo, lo que hace que por cada vez que entra a la CPU, al cabo de muy poco tiempo tiene que salir para volver a entrar en la cola de ejecutables. De esta manera, el cociente se dispara porque los tiempos de CPU (denominador) son muy bajos, mientras que el tiempo total de respuesta (numerador) es muy alto, especialmente en comparación con la ráfaga de CPU, lo que dispara el valor del cociente.

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.

No. El planificador a corto plazo se dedica a examinar los procesos existentes en la cola de preparados, por lo que lo que pase antes, más aún si es de la creación de un proceso, no requiere de este planificador.

b) Abortar un proceso, es decir, terminarlo forzosamente.

Sí, si asumimos que el proceso a abortar está actualmente ejecutándose. En este caso, el planificador a corto plazo, al ser el encargado de asignar recursos de CPU, debe encargarse de seleccionar el siguiente proceso para pasar a ejecutarse en la CPU.

c) Suspender o bloquear un proceso.

Sí. En el momento en el que un proceso queda bloqueado y la CPU queda libre, es necesario invocar al planificador a corto plazo para elegir a otro proceso para ocupar la CPU.

d) Reanudar un proceso (inversa al caso anterior).

Sí, si el proceso a reanudar debe entrar en la CPU. Para reanudar un proceso, éste debe pasar por la cola de preparados, donde el planificador puede asignarle tiempo y recursos de CPU. Por ejemplo, si se usa una política apropiativa por prioridades y un proceso se reanuda con mayor prioridad que el que está actualmente ejecutándose, el planificador debe ser invocado para liberar CPU y asignarle recursos a dicho proceso.

e) Modificar la prioridad de un proceso.

Sí, si la nueva prioridad altera el orden de ejecución. A la hora de asignar recursos y tiempo de CPU, el planificador debe escoger a qué procesos les da prioridad en dicha asignación, lo cual puede ocurrir en políticas con varias colas en las que se puedan traspasar procesos.

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%?

Siendo Q el valor útil para el usuario, puesto que es el tiempo en el que realmente se están ejecutando procesos, la proporción de tiempo útil respecto al tiempo total es el cociente entre el tiempo útil y el tiempo en el que se ejecutan procesos y se cambia de contexto (mediante el despachador), es decir, Q+S. De esta manera, tenemos:

$$Tiempo\,útil\,CPU = \frac{Q}{Q+S} = 0'8 \rightarrow Q = 0'8(Q+S) \rightarrow 0'2Q = 0'8S \rightarrow Q = 0'8\frac{S}{0'2} \rightarrow Q = 4S$$

Así pues, el valor del quantum debe ser cuatro veces el valor del tiempo que tarda el despachador en cada cambio de contexto.

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 asegure que cada proceso "ve" la CPU al menos cada T segundos?

Del ejercicio anterior teníamos que el tiempo de cada ciclo del algoritmo por turnos era $Q+S$. De esta manera, si tenemos varios procesos, el tiempo total de cada ciclo será $N \times (Q+S)$. Para que un proceso "vea" la CPU cada T segundos, el tiempo total de cada ciclo debe ser menor que T, es decir, $N \times (Q+S) \leq T$. Despejando Q, tenemos:

$(Q+S) \leq \frac{T}{N} \rightarrow Q \leq \frac{T}{N} - S$ Es decir, el valor del quantum Q debe ser menor o igual que el el cociente entre el intervalo T y el número de procesos, menos el tiempo que tarde el despachador en cada cambio de contexto.

6. ¿Tiene sentido mantener ordenada por prioridades la cola de procesos bloqueados? Si lo tuviera, ¿en qué casos sería útil hacerlo?

Sí, tiene sentido ordenarla por prioridades ya que, al igual que en la cola de preparados puede ser necesario ejecutar antes procesos más importantes que otros, puede serlo a la hora de continuar la ejecución de procesos que habían salido de la CPU. Por ejemplo, en sistemas de tiempo real es necesario que las tareas más prioritarias se finalicen lo antes posible, y si necesitan algún recurso externo que pueda requerir tiempos de espera demasiado largo, interesa priorizar aquellos procesos que necesiten ser terminados urgentemente.

7. ¿Puede el procesador manejar una interrupción mientras está ejecutando un proceso si la política de planificación que utilizamos es no apropiativa?

Sí. Las interrupciones hardware se atienden siempre, independientemente de la política de planificación. Lo que sí ocurre con una política no apropiativa es que, después de atender la interrupción, el planificador va a seguir ejecutando el proceso que estaba en la CPU hasta el momento de la interrupción y sólo cambiará de proceso si éste termina de ejecutarse o se bloquea, pero no cambiará a otro proceso que haya podido llegar a la cola de preparados durante o después de la interrupción.

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? y ¿cuáles serían tales modificaciones?

Las partes del sistema operativo que habría que modificar serían:

- El temporizador del sistema, para provocar interrupciones de reloj en las que invocar al planificador a corto plazo si el proceso que se está ejecutando debe abandonar la CPU.

- El despachador, para realizar cambios de contexto en el momento en el que el planificador decida expulsar o ejecutar un proceso en la CPU.

- Las rutinas que cambian el estado de los procesos (nuevo, preparado, bloqueado, terminado, etc.) para decidir si hay que replanificar la ejecución con procesos que pasan a estar preparados para ejecutar.

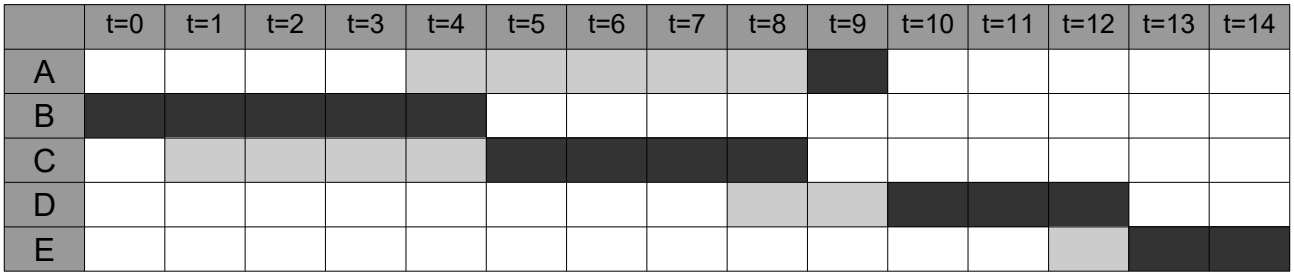
- PCB y colas de planificación para almacenar información relevante para el planificador a la hora de mantener un orden adecuado entre los procesos (prioridad, quantum, etc.).

9. En el algoritmo de planificación FCFS, la penalización ((t + tº de espera) / t), ¿es creciente, decreciente o constante respecto a t (tiempo de servicio de CPU requerido por un proceso)? Justifique su respuesta.

Suponiendo un tiempo de espera constante, el numerador (tiempo de servicio más tiempo de espera) no crecería a la misma velocidad que el denominador (tiempo de servicio), ya que el el tiempo de espera debería crecer en la misma proporción que el tiempo de servicio para que el cociente por éste diera el mismo resultado. De este modo, como crece a un ritmo inferior, la penalización es decreciente respecto a t. Explicado de otra manera, $P = \frac{t + t_{espera}}{t} = 1 + \frac{t_{espera}}{t}$, de manera que, si el tiempo de espera es constante, $\frac{t_{espera}}{t}$ cada vez es menor, lo que hace que la penalización sea decreciente respecto al tiempo de servicio.

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

$T_A=(9-4)+1=6$ $\quad$ $T_B=(0-0)+5=5$

$T_C=(5-1)+4=8$ $\quad$ $T_D=(10-8)+3=5$

$T_E=(13-12)+2=3$ $\quad$ $T_{MEDIO}=\frac{6+5+8+5+3}{5}=5'4\,segundos$

b) Tiempo medio de espera

$T_A=6-1=5$ $\quad$ $T_B=5-5=0$

$T_C=8-4=4$ $\quad$ $T_D=5-3=2$

$T_E=3-2=1$ $\quad$ $T_{MEDIO}=\frac{5+0+4+2+1}{5}=2'4\,segundos$

c) La penalización, es decir, el cociente entre el tiempo de respuesta y el tiempo de CPU.

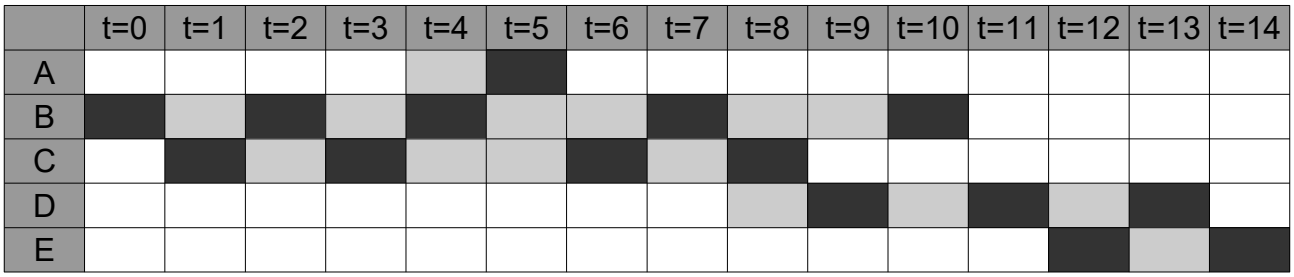
$T_A=\frac{6}{1}=6$ $\quad$ $T_B=\frac{5}{5}=1$

$T_C=\frac{8}{4}=2$ $\quad$ $T_D=\frac{5}{3}=1'66$

$T_E=\frac{3}{2}=1'5$ $\quad$ $T_{MEDIO}=\frac{6+1+2+1'66+1'5}{5}=2'43\,segundos$

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



- Tiempo medio de respuesta

$T_A = 6-4 = 2$ $T_B = 11-0 = 11$

$T_C = 9-1 = 8$ $T_D = 14-8 = 6$

$T_E = 15-12 = 3$ $T_{MEDIO} = \frac{2+11+8+6+3}{5} = 6\ segundos$

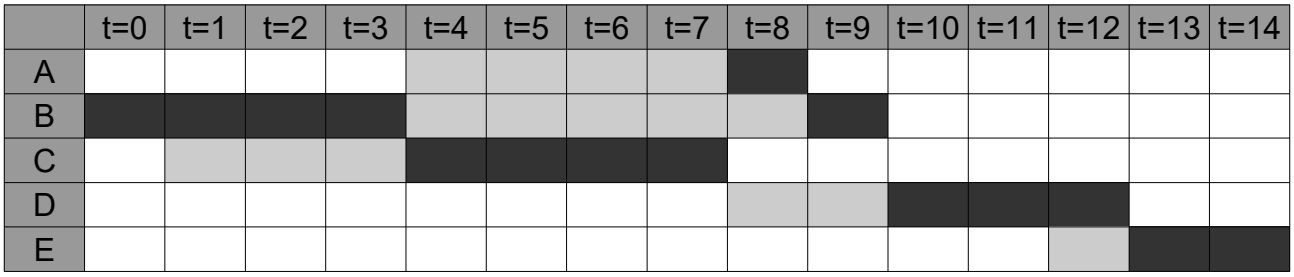
- Tiempo medio de espera

$T_A = 2-1 = 1$ $T_B = 11-5 = 6$

$T_C = 8-4 = 4$ $T_D = 6-3 = 3$

$T_E = 3-2 = 1$ $T_{MEDIO} = \frac{1+6+4+3+1}{5} = 3\ segundos$

b) Por Turnos con quantum q=4



- Tiempo medio de respuesta

$T_A = 9-4 = 5$ $T_B = 10-0 = 10$

$T_C = 8-1 = 7$ $T_D = 13-8 = 5$

$T_E = 15-12 = 3$ $T_{MEDIO} = \frac{5+10+7+5+3}{5} = 6\ segundos$

- Tiempo medio de espera

$T_A = 5-1 = 4$ $T_B = 10-5 = 5$

$T_C = 7-4 = 3$ $T_D = 5-3 = 2$

$T_E = 3-2 = 1$ $T_{MEDIO} = \frac{4+5+3+2+1}{5} = 3\ segundos$

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

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14
A															
B															
C															
D															
E															

- Tiempo medio de respuesta

$T_A = 6 - 4 = 2$ $T_B = 5 - 0 = 5$

$T_C = 10 - 1 = 9$ $T_D = 13 - 8 = 5$

$T_E = 15 - 12 = 3$ $T_{MEDIO} = \frac{2+5+9+5+3}{5} = 4'8\, segundos$

- Tiempo medio de espera

$T_A = 2 - 1 = 1$ $T_B = 5 - 5 = 0$

$T_C = 9 - 4 = 5$ $T_D = 5 - 3 = 2$

$T_E = 3 - 2 = 1$ $T_{MEDIO} = \frac{1+0+5+2+1}{5} = 1'8\, segundos$

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.

Sí, CFS favorece a procesos limitados por E/S. CFS mantiene un tiempo virtual de ejecución (*vruntime)* para cada proceso, y siempre busca ejecutar el proceso con menor *vruntime.* Cuando un proceso se bloquea por E/S su *vruntime* se queda congelado, sin aumentar, mientras que otros procesos, mismamente los limitados por CPU, siguen ejecutando y aumentando su *vruntime*. De esta manera, cuando el proceso de E/S vuelve a estar listo, tiene un *vruntime* menor, lo que hace que CFS lo elija para ejecutar por delante del proceso de CPU.

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?

El problema de que un proceso no llame a *wait* es que sus hijos quedan como procesos zombi. La función *wait* sirve para que un proceso padre pueda obtener información de aquellos procesos hijos que finalizan su ejecución, de manera que el kernel pueda liberar sus correspondientes entradas en la tabla de procesos. Si el padre nunca hace esa llamada, la entrada no se libera, acumulándose procesos zombi hasta que el sistema empiece a fallar al crear procesos nuevos.

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.

Los argumentos de la función clone son: la función que va a ejecutar el nuevo proceso, el espacio de direcciones, los indicadores o señales con los que se va a crear (que indican qué comparte la tarea actual con la nueva a crear) y los argumentos de la primera función. Para crear un proceso independiente, como no queremos que comparta información, basta especificar el indicador *SIGCHILD*, que sirve para indicar la señal que se enviará al padre cuando el hijo termine. En cambio, para crear una hebra, hay que añadir más indicaciones para que se comparta casi toda la información propia del proceso que crea la hebra: espacio de direcciones (*CLONE_VM),* tabla de descriptores de fichero (*CLONE_FILES*)*,* o una señal que indica que el nuevo proceso pertenece al mismo gruop de hebras que su creador (*CLONE_THREAD)*, entre otros.

b) Dibuje las principales estructuras de datos del kernel que reflejan las diferencias entre ambas