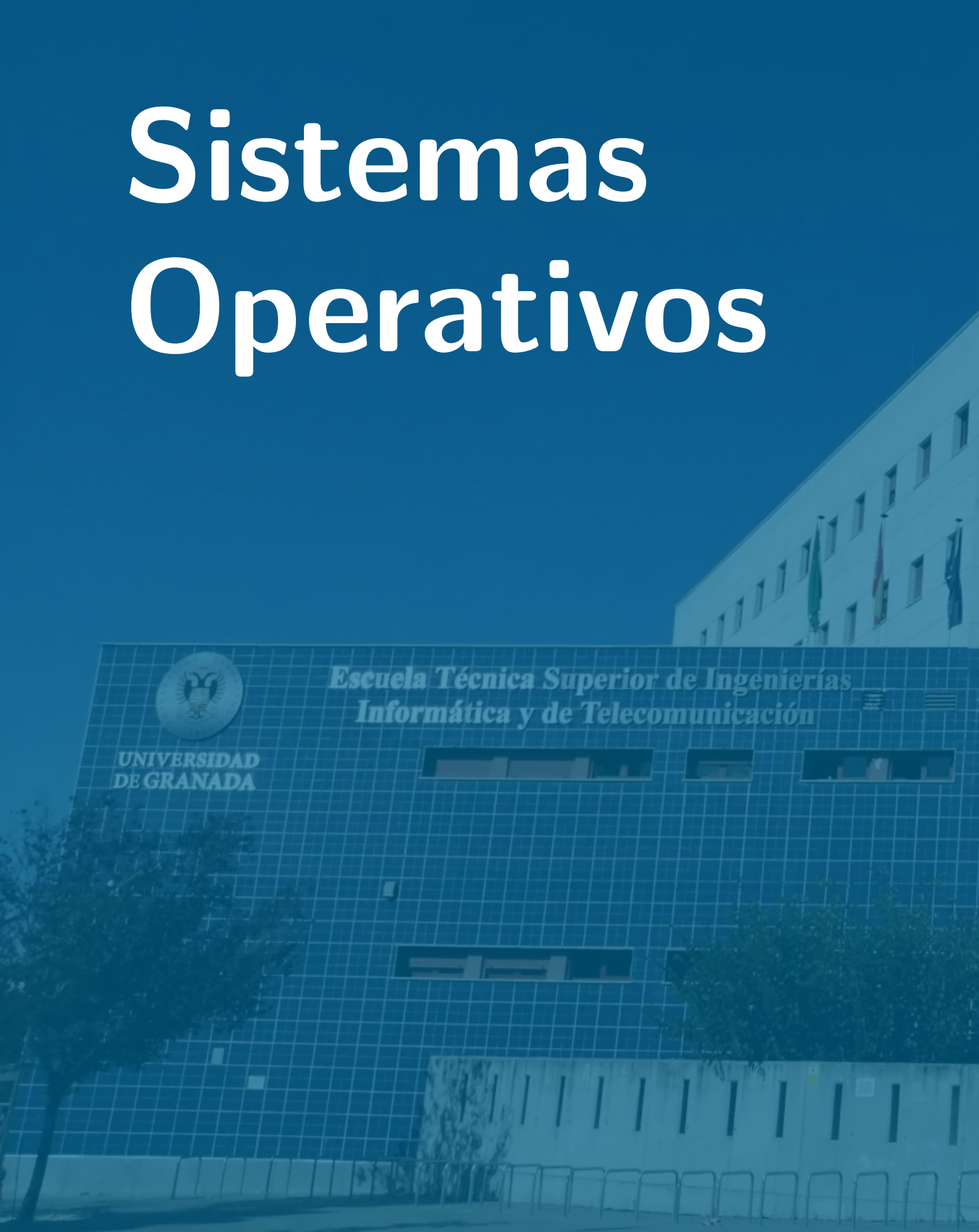
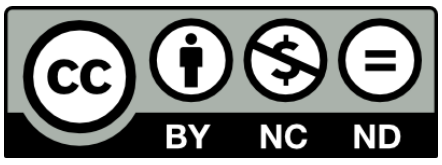
Sistemas Operativos



Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada



Sistemas Operativos

Los Del DGIIM, losdeldgiim.github.io

Arturo Olivares Martos

Granada, 2023-2024

1. Relaciones de Problemas

1.1. Estructuras de SO's

Ejercicio 1.1.1. Cuestiones generales relacionadas con un SO:

1. ¿Qué es el núcleo (kernel) de un SO?

   El kernel de un SO es un programa que reside en RAM que contiene las llamadas al sistema (funciones relacionadas directamente con el hardware del ordenador).

   Como son funciones a tan bajo nivel, no se puede acceder a ellas en modo usuario. Cuando se ejecuta una llamada al sistema, se cambia el bit de modo de la PSW a 0, y se entra en modo kernel.

2. ¿Qué es un modelo de memoria (interpretación del espacio de memoria) para un programa? Explique los diferentes modelos de memoria para la arquitectura IA-32.

   El modelo de memoria es la forma que tiene la CPU de interpretar los accesos a memoria. El espacio de direcciones son todas las direcciones disponibles en un ordenador, y va desde 0 hasta $2^n - 1$, siendo n el número de bits del bus de direcciones. Los diferentes modelos de memoria del IA-32, cuyo espacio de direcciones es $\{0, \ldots, 2^{32} - 1\}$, son:

   a) Flat memory model:
   Es un espacio lineal que direcciona todo el espacio de direcciones, desde 0 hasta $2^{32} - 1$. Puede direccionar cada byte, por lo que se dice que lo hace con granularidad de un byte.

   b) Segmented memory model:
   Usa segmentación. Cada dirección lógica consta de un selector de segmento y de un desplazamiento dentro de dicho segmento.

   Consta también de una tabla de segmentos en la que, entre otros aspectos, se encuentra dónde comienza cada segmento.

   Se puede identificar una gran cantidad de segmentos, cada uno con un tamaño límite de 2^{32} bytes (tamaño de la memoria).

   c) Real-adress mode memory model:
   Muy similar al modelo de memoria segmentada, aunque este se mantiene por compatibilidad hacia atrás con otras arquitecturas. En este caso hay limitaciones de tamaño tanto para los segmentos como para la memoria total.

3. ¿Cómo funciona el mecanismo de tratamiento de interrupciones mediante interrupciones vectorizadas? Explique que parte es realizada por el hardware y que parte por el software.

4. Describa detalladamente los pasos que lleva a cabo el SO cuando un programa solicita una llamada al sistema.

Ejercicio 1.1.2. Explique tres responsabilidades asignadas al gestor de memoria de un SO y tres asignadas al gestor de procesos.

Ejercicio 1.1.3. ¿Cómo gestionaría el sistema operativo la posibilidad de anidamiento de interrupciones?

Ejercicio 1.1.4. Contraste las ventajas e inconvenientes de una arquitectura de SO monolítica frente a una arquitectura microkernel.

Ejercicio 1.1.5. Cuestiones relacionas con virtualización:

1. ¿Qué se entiende actualmente por virtualización mediante hipervisor?

2. ¿Qué clases de hipervisores existen de manera general y que ventajas e inconvenientes plantea una clase con respecto a la otra?

Ejercicio 1.1.6. Cuestiones relacionadas con RTOS:

1. ¿Qué característica distingue esencialmente a un proceso de tiempo real de otro que no lo es?

2. ¿Cuáles son los factores determinantes del tiempo de respuesta en un RTOS, e.d. define determinismo y reactividad?

1.2. Procesos y Hebras

Ejercicio 1.2.1. Cuestiones generales sobre procesos y asignación de CPU:

1. ¿Cuáles son los motivos que pueden llevar a la creación de un proceso?

   EL fork

2. ¿Es necesario que lo último que haga todo proceso antes de finalizar sea una llamada al sistema para finalizar de forma explícita, por ejemplo `exit()`?

   No.

3. Cuando un proceso pasa a estado "BLOQUEADO", ¿Quién se encarga de cambiar el valor de su estado en el descriptor de proceso o PCB?

   El `context_switch()`, y dentro de él el dispatcher.

4. ¿Qué debería hacer cualquier planificador a corto plazo cuando es invocado pero no hay ningún proceso en la cola de ejecutables?

   Siempre hay un proceso kernel que se introduce.

5. ¿Qué algoritmos de planificación quedan descartados para ser utilizados en sistemas de tiempo compartido?

   First Come First Served: No, no contempla timeout y pueden entrar R dentro de r.

   Los apropiativos, en media, podrán funcionar.

Ejercicio 1.2.2. Cuestiones sobre el modelo de procesos extendido:

1. ¿Qué pasos debe llevar a cabo un SO para poder pasar un proceso de reciente creación de estado "NUEVO" a estado "LISTO"?

   Crear PCB y rellenarlo. Cargarlo en RAM.

   ¿Puede entrar en suspendido y listo? Sí, si no hay suficientes marcos de página al crear el proceso para pasarlo a listo.

2. ¿Qué pasos debe llevar a cabo un SO para poder pasar un proceso ejecutándose en CPU a estado "FINALIZADO"? Libera recursos, pone estado de PCB en finalizado, avisa al padre de que ha terminado y si tiene hijos, los cuelga del líder (init) de la sesión para mantener jerarquía padre-hijo.

   Esto lo realiza `sys_exit`, algoritmo de kernel.

3. Hemos explicado en clase que la función `context_switch()` realiza siempre dos funcionalidades y que además es necesario que el kernel la llame siempre cuando el proceso en ejecución pasa a estado "FINALIZADO" o "BLOQUEADO". ¿Qué funcionalidades debe realizar y en qué funciones del SO se llama a esta función?

   Planning y dispatch, arreglar los estados: ejecutándose $\longrightarrow$ x y z $\longrightarrow$ ejecutándose.

   Al context switch se le llama: Rutina de E/S.

   Funciones en las que se le llama:

- RSI de reloj.
- `sys_exit`.
- Al final de wait:
  - Si se ha terminado el hijo: Se sigue ejecutando.
  - Si no ha terminado el hijo: El proceso entra a bloqueados: Context switch: Planif. Dispatcher.

4. Indique el motivo de la aparición de los estados "SUSPENDIDO-BLOQUEADO" y "SUSPENDIDO-LISTO" en el modelo de procesos extendido.

   Puede darse el caso de que todos los procesos no puedan estar cargados a la vez en memoria principal. Uno podría pensar que esto se podría evitar no cargando en memoria principal más procesos cuando esta estuviese prácticamente llena, pero podría darse el caso de que todos los procesos que estuviesen en memoria principal fuesen de tipo E/S (ráfagas cortas, interrumpidos por frecuentes procesos largos de E/S) y, en cierto momento, todos estuviesen bloqueados. Entonces, el procesador estaría ocioso, pero tampoco podrían ejecutarse más programas ya que estos no cabrían en memoria. (Análogamente, el problema podría ser que todos los procesos fuesen de una ráfaga larga, por lo que el módulo de E/S no estaría haciendo nada, y los procesos de E/S estarían esperando cuando podrían estar bloqueados esperando al evento correspondiente).

   Llegados a este punto, aparece la memoria *swap* y el intercambio o *swapping* de procesos entre memoria principal y memoria secundaria. Cuando un proceso no se está ejecutando, parte de su imagen (como su programa o sus datos) pueden ser expulsados de memoria y almacenados en memoria secundaria. El `PCB` nunca podrá ser expulsado (ya que se perdería el control del registro), pero al liberar parte de la memoria principal ya podrían entrar nuevos procesos en memoria. El planificador a largo plazo se tendría que encargar de decidir qué procesos de los "nuevos" cargar en memoria (traer a "listos"), pero sería relevante que fuesen de ráfagas largas y pocas esperas de E/S, para equilibrar la mezcla.

   Es por esto que aparecen los dos nuevos estados. Cuando un proceso está bloqueado y el planificador a medio plazo (que es el encargado del *swapping*) considera que la espera puede ser larga, puede decidir expulsarlo a memoria secundaria. De igual forma, si considera que la llegada del evento puede ser inminente, puede traerlo a memoria principal. De igual forma, si un proceso en "listos" va a tardar mucho en ser planificado por el planificador a corto plazo, el planificador a medio plazo puede decidir expulsarlo a memoria secundaria.

Ejercicio 1.2.3. ¿Tiene sentido mantener ordenada por prioridades la cola de procesos bloqueados? Si lo tuviera, ¿en qué casos sería útil hacerlo? Piense en la cola de un planificador de E/S, por ejemplo el de HDD, y en la cola de bloqueados en espera del evento "Fin E/S HDD".

Al hablar de prioridades, es lógico pensar que el planificador de CPU funciona por prioridades, ...

Ejercicio 1.2.4. Explique las diferentes formas que tiene el kernel de ejecutarse en relación al contexto de un proceso y al modo de ejecución del procesador.

Usuario y kernel, ...

Ejercicio 1.2.5. Responda a las siguientes cuestiones relacionadas con el concepto de hebra:

1. ¿Qué elementos de información es imprescindible que contenga una estructura de datos que permita gestionar hebras en un kernel de SO? Describa las estructuras `task_t` y la `thread_t`.

   `task_t` :

   - pid
   - Lista de hebras.
   - Memoria de la tarea.
   - Estado del proceso.
   - Controlador de recursos del sistema.

   `thread_t` :

   - tid
   - Estado.
   - Contexto de registros

2. En una implementación de hebras con una biblioteca de usuario en la cual cada hebra de usuario tiene una correspondencia N:1 con una hebra kernel, ¿Qué ocurre con la tarea si se realiza una llamada al sistema bloqueante, por ejemplo `read()`?

   Bloquea a todas. Correspondencia 1 a 1 para evitarlo.

3. ¿Qué ocurriría con la llamada al sistema `read()` con respecto a la tarea de la pregunta anterior si la correspondencia entre hebras usuario y hebras kernel fuese 1:1?

   No pasaría nada.

Ejercicio 1.2.6. ¿Puede el procesador manejar una interrupción mientras está ejecutando un proceso sin hacer `context_switch()` si la política de planificación que utilizamos es no apropiativa? ¿Y si es apropiativa?

Ejercicio 1.2.7. Suponga que es responsable de diseñar e implementar un SO que va a utilizar una política de planificación apropiativa (*preemptive*). Suponiendo que el sistema ya funciona perfectamente con multiprogramación pura y que tenemos implementada la función `Planif_CPU()`, ¿qué otras partes del SO habría que modificar para implementar tal sistema? Escriba el código que habría que incorporar a dichas partes para implementar apropiación *(preemption)*.

Ejercicio 1.2.8. Para cada una de las siguientes llamadas al sistema explique si su procesamiento por parte del SO requiere la invocación del planificador a corto plazo (`Planif_CPU()`):

1. Crear un proceso, `fork()`.

   Si usamos una planificación apropiativa, sí sería necesario; ya que puede ser que el nuevo proceso creado sea el elegido por el planificador, pasando el que se estaba ejecutando a "listos", y el nuevo proceso a "ejecutándose".

   En el caso de que la planificación sea no apropiativa, no sería necesario llamar al planificador a corto plazo; ya que la creación de un proceso nuevo no provoca que el proceso que estaba ejecutándose deje de hacerlo.

2. Abortar un proceso, es decir, terminarlo forzosamente, `abort()`.

   En este caso, no habría ninguno proceso ejecutándose. Por tanto, para que el procesador no esté ocioso, se llama al planificador a corto plazo para decidir qué proceso de los listos pasará a ejecutarse.

3. Bloquear (suspender) un proceso, `read()` o `wait()`.

   En todos esos casos el proceso que estaba ejecutándose pasa a "bloqueado". Por tanto, no habría ninguno proceso ejecutándose y; análogamente al caso anterior, se llamaría al planificador a corto plazo.

4. Desbloquear (reanudar) un proceso, `RSI` o `exit()` (complementarias a las del caso anterior).

5. Modificar la prioridad de un proceso.

Ejercicio 1.2.9. En el algoritmo de planificación FCFS, el índice de penalización, $P = \frac{M+r}{r}$, ¿es creciente, decreciente o constante respecto a r (ráfaga de CPU: tiempo de servicio de CPU requerido por un proceso)? Justifique su respuesta.

Al ser FCFS, tenemos que el tiempo de espera M es constante e independiente respecto a r. Por tanto, cuanto menor es la ráfaga, mayor es la penalización. Es decir, es decreciente respecto a r.

Matemáticamente, como M es constante, podemos argumentarlo mediante derivación:

$$\frac{\partial P}{\partial r}(M, r) = \frac{r - (M + r)}{r^2} = -\frac{M}{r^2} < 0$$

Como la primera derivada es negativa y la función es diferenciable, tenemos que P es decreciente respecto a r.

Ejercicio 1.2.10. Sea un sistema multiprogramado que utiliza el algoritmo Por Turnos (Round-Robin, RR). 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 %?

Necesitamos $Q = 4 \cdot S$, ya que así por cada cuatro unidades de tiempo que se está ejecutando el proceso, se ejecuta una vez el kernel. Es decir, suponiendo que el proceso se ejecuta todo el tiempo en modo usuario, es necesario que $4/5$ de su tiempo de retorno se ejectuen seguidos sin consumir el "time slice"; por lo que tendría que usarse $Q = 4 \cdot S$.

Otra forma de verlo es, sabiendo que $T = Q + S$ y $Q = 0{,}8T$, tenemos que:

$$T = 0{,}8T + S \Longrightarrow 0{,}2T = S \Longrightarrow T = 5S \Longrightarrow Q = 4S$$

Ejercicio 1.2.11. Para la siguiente tabla que especifica una determinada configuración de procesos, tiempos de llegada a cola de listos y ráfagas de CPU; responda a las siguientes preguntas y analice los resultados:

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

Tabla 1.1: Configuración de procesos del Ejericio 1.2.11.

Observación. En los diagramas de ocupación de la CPU, las marcas de tiempo deberían ir justo en las líneas verticales, no en las casillas. Por ello, se opta por señalar justo antes de la marca de tiempo i y justo después de la i con i^- e i^+ respectivamente.

1. FCFS. Tiempo medio de respuesta, tiempo medio de espera y penalización.

																		T	**M**	**P**
A						L	L	L	L	L	E							6	5	6
B	E	E	E	E	E													5	0	1
C		L	L	L	L	E	E	E	E									8	4	2
D										L	L	E	E	E				5	2	5/3
E														L	E	E		3	1	3/2
	0^+					5^-	5^+				10^-	10^+				15^-	15^+			

Tabla 1.2: Diagrama de ocupación de memoria para FCFS.

Las medias aritméticas son:

$$\overline{T} = 5{,}4 \qquad \overline{M} = 2{,}4$$

2. SJF (ráfaga estimada coincide con ráfaga real). Tiempo medio de respuesta, tiempo medio de espera y penalización.

																		T	**M**	**P**
A						L	E											2	1	2
B	E	E	E	E	E													5	0	1
C		L	L	L	L	L	E	E	E	E								9	5	9/4
D										L	L	E	E	E				5	2	5/3
E														L	E	E		3	1	3/2
	0^+					5^-	5^+				10^-	10^+				15^-	15^+			

Tabla 1.3: Diagrama de ocupación de memoria para SJB.

Las medias aritméticas son:

$$\overline{T} = 4{,}8 \qquad \overline{M} = 1{,}8$$

3. SRTF (ráfaga estimada coincide con ráfaga real). Tiempo medio de respuesta, tiempo medio de espera y penalización.

																		T	**M**	**P**
A						L	E											2	1	2
B	E	E	E	E	E													5	0	1
C		L	L	L	L	L	E	E	E	E								9	5	9/4
D										L	L	E	E	E				5	2	5/3
E														L	E	E		3	1	3/2
	0^+				5^-	5^+				10^-	10^+				15^-	15^+				

Tabla 1.4: Diagrama de ocupación de memoria para SRTF.

Las medias aritméticas son:

$$\overline{T} = 4{,}8 \qquad \overline{M} = 1{,}8$$

4. RR ($q = 1$). Tiempo medio de respuesta, tiempo medio de espera y penalización.

																		T	**M**	**P**
A						L	E											2	1	2
B	E	L	E	L	E	L	L	E	L	L	E							11	6	11/5
C		E	L	E	L	L	E	L	E									8	4	2
D										L	E	L	E	L	E			6	3	2
E														E	L	E		3	1	3/2
	0^+				5^-	5^+				10^-	10^+				15^-	15^+				

Tabla 1.5: Diagrama de ocupación de memoria para RR($q = 1$).

Las medias aritméticas son:

$$\overline{T} = 6 \qquad \overline{M} = 3$$

5. RR ($q = 4$). Tiempo medio de respuesta, tiempo medio de espera y penalización.

																		T	**M**	**P**
A						L	L	L	L	E								5	4	5
B	E	E	E	E	L	L	L	L	L	E								10	5	2
C		L	L	L	E	E	E	E										7	3	7/4
D										L	L	E	E	E				5	2	5/3
E														L	E	E		3	1	3/2
	0^+				5^-	5^+				10^-	10^+				15^-	15^+				

Tabla 1.6: Diagrama de ocupación de memoria para RR($q = 4$).

Las medias aritméticas son:

$$\overline{T} = 6 \qquad \overline{M} = 3$$

Ejercicio 1.2.12. Utilizando los datos de la tabla 1.1, dibuje 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

Tenga en cuenta las siguientes suposiciones:

1. Todos los procesos inicialmente entran en la cola de mayor prioridad (menor valor numérico).
2. Cada cola se gestiona mediante la política RR y la política de planificación entre colas es por prioridades no apropiativa.
3. Un proceso en la cola i pasa a la cola $i+1$ si consume un quantum completo sin bloquearse.
4. Cuando un proceso llega a la cola de menor prioridad, permanece en ella hasta que finaliza.

																		T	**M**	**P**
A						E												1	0	1
B	E	L	E	E	L	L	L	E	L	L	L	L	L	L	E			15	10	5
C		E	L	L	L	E	E	L	L	L	L	E						11	7	$^{11}/_{4}$
D									E	E	E							3	0	1
E													E	E				2	0	1
	0^+				5^-	5^+				10^-	10^+				15^-	15^+				

Tabla 1.7: Diagrama de ocupación de memoria para el Ejercicio 1.2.12.

Notemos que, en naranja, se señala cuando un proceso pasa de la cola 1 a la cola 2; mientras que en rojo se señala cuando pasa de la 2 a la 3.