UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Fundamentos de los Sistemas Operativos
Examen parcial - 11 de abril de 2019

1	2	3	test	extra	NOTA

Nombre y apellidos | **DNI/NIE**

SOLUCIONES

DURACIÓN: *Dispone de dos horas para realizar el examen.*

Lea las instrucciones para el test en la hoja correspondiente.

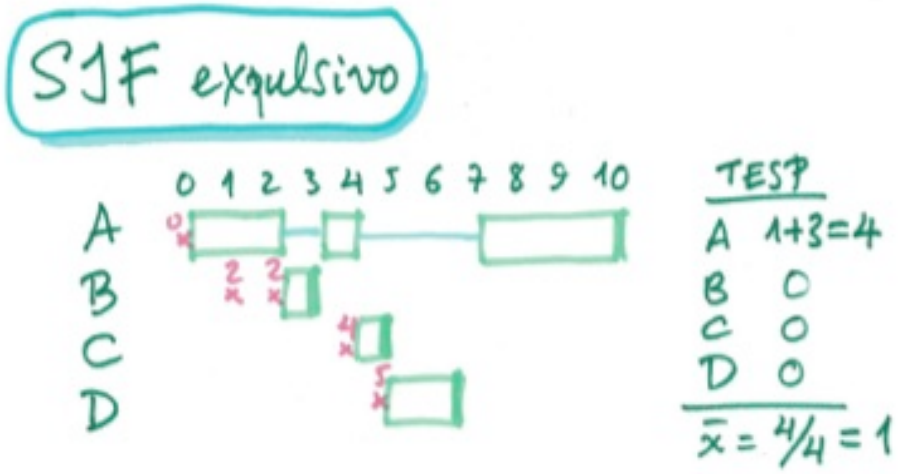
1 (1 punto) A un planificador de CPU llegan cuatro procesos, según el cuadro de la derecha. Para estas dos políticas: SJF expulsivo y Round Robin (Q=2), obtenga lo siguiente:

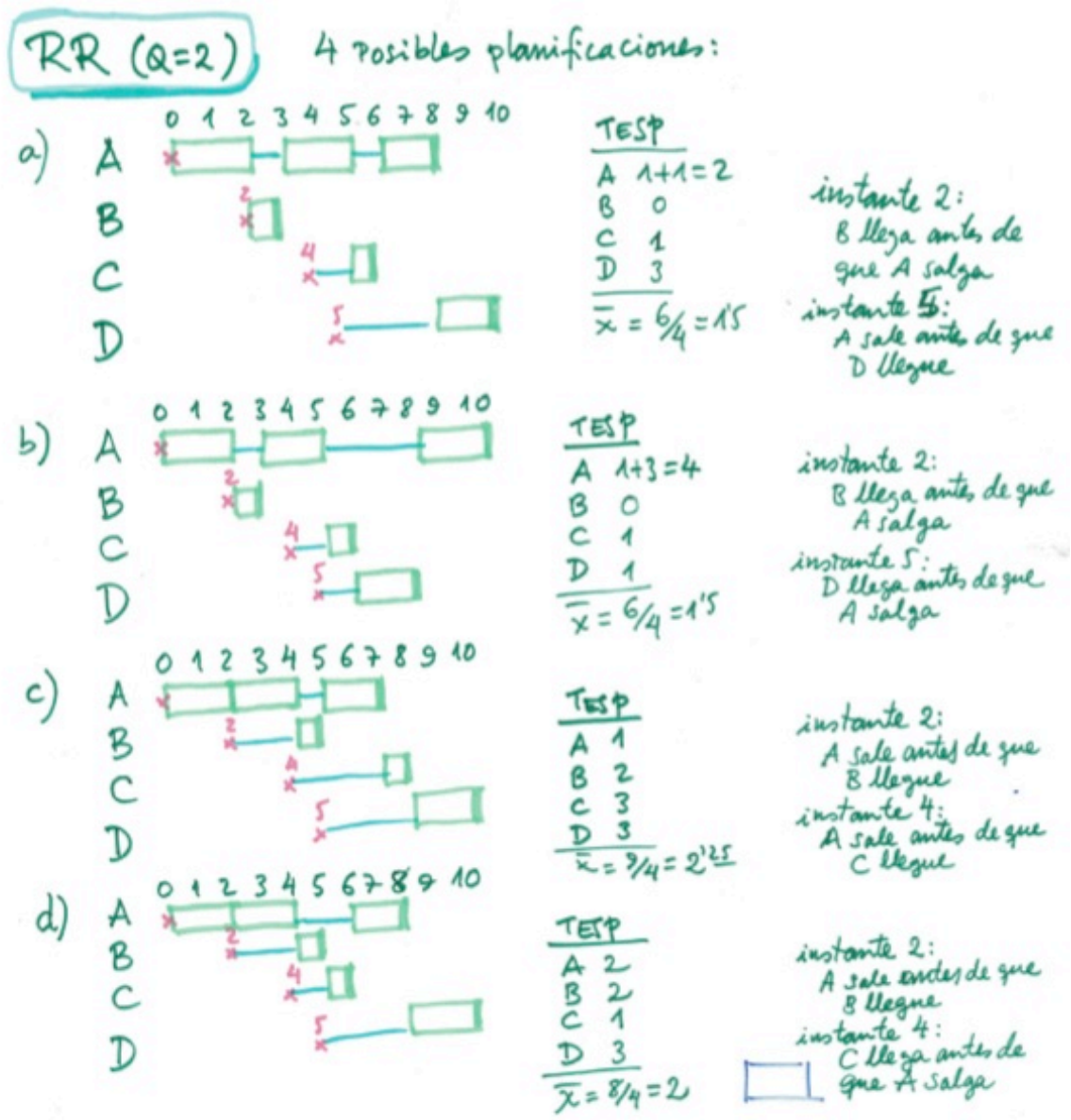
- Diagrama de Gantt con la planificación.
- Tiempo de espera de cada uno de los procesos y valor medio del conjunto.
- Número de cambios de contexto realizados durante la planificación.

proceso	llegada	duración
A	0	6
B	2	1
C	4	1
D	5	2

En el Round Robin se pueden dar entradas y salidas simultáneas de procesos, en los instantes 2, 4 y 5. Esto da lugar a cuatro planificaciones distintas, todas ellas válidas.

En las planificaciones, el número de cambios de contexto oscila entre 4 y 7, dependiendo de si se cuentan como cambios de contexto la entrada de A en el instante 0 y la finalización del último proceso.





2 (1'25 puntos) Para cada una de estas operaciones, señale cuáles deben ser privilegiadas y por qué. Utilice un máximo de 80 palabras para cada apartado.

Aquí va un esbozo de cuál podría ser la respuesta de cada apartado.

1. Modificar la cola de preparados.

Debe ser privilegiada, porque de lo contrario un proceso de usuario podría adelantar su posición en la cola, eliminar otros procesos, etc.

2. Volcar en disco el contenido de la memoria RAM.

Debe ser privilegiada, porque si se permite a un proceso copiar la memoria RAM, podría examinar su contenido y obtener información confidencial de otros procesos.

3. Reprogramar el temporizador del sistema.

Debe ser privilegiada, porque de lo contrario un proceso podría saltarse los controles de estancia máxima en el procesador (ej. en un algoritmo Round Robin).

4. Leer el contenido del vector de interrupciones.

No es necesario que sea privilegiada. El vector de interrupciones contiene las direcciones de las rutinas del servicio de interrupción, cuyo mero conocimiento no supone una amenaza a la seguridad.

5. Modificar el valor del registro base de memoria.

Nombre y apellidos	DNI/NIE

Debe ser privilegiada. El registro base de memoria señala dónde empieza la zona de memoria del proceso actual, y por tanto delimita qué zona es accesible por el proceso y qué zona está prohibida. Si el proceso pudiera modificar el registro base, podría ganar acceso a zonas arbitrarias de la memoria, comprometiendo con ello la integridad y la confidencialidad del sistema.

```
1  const int N = ...;
2  class Zapato { ... }
3  class Cesto {
4    Cesto (int capacidad);
5    void meterZapatoDerecho(Zapato);
6    void meterZapatoIzquierdo(Zapato);
7    Zapato sacarZapatoDerecho();
8    Zapato sacarZapatoIzquierdo();
9  }

10 Cesto cesto = new Cesto(N);

11 // hilos productores
12 void ObreroDerechos() {
13   while (true) {
14     ... fabricar un zapato derecho Z
15     cesto.meterZapatoDerecho(Z);
16   }
17 }

18 void ObreroIzquierdos() {
19   while (true) {
20     ... fabricar un zapato izquierdo Z
21     cesto.meterZapatoIzquierdo(Z);
22   }
23 }

24 // hilo empaquetador
25 void Empaquetador() {
26   while (true) {
27     Zapato izq,dch;
28     izq = cesto.sacarZapatoIzquierdo();
29     dch = cesto.sacarZapatoDerecho();
30     ... empaquetar zapatos (izq,dch)
31   }
32 }
```

3 (1'75 puntos) Tenemos un sistema concurrente que simula una fábrica de zapatos. En esta fábrica tenemos unos obreros que están constantemente fabricando zapatos de un pie (izquierdo o derecho). Cuando un obrero finaliza un zapato, lo pone en un cesto compartido. Aparte, un obrero empaquetador va sacando del cesto pares de zapatos (uno izquierdo y otro derecho) y los empaqueta.

El cesto tiene una capacidad limitada de N zapatos. Si un fabricante se encuentra el cesto lleno, se tiene que esperar. Si el empaquetador ve el cesto vacío o con zapatos de un solo pie, se tiene que esperar hasta que haya al menos un par completo.

El esquema del código de los procesos se muestra en el cuadro de la izquierda. Tal y como está el código, no hay ninguna protección sobre el uso concurrente del objeto `cesto` y tampoco se comprueban las condiciones de espera. Añada el código que hace falta para resolver esos defectos:

- Utilice unas variables de estado para conocer el estado del sistema.
- Coloque las condiciones de bloqueo/espera donde sea necesario.
- Delimite las zonas del código que deben ejecutarse en exclusión mutua.
- Use las primitivas **ENTRARSC**, **SALIRSC**, **DORMIR** y **DESPERTAR** para ejecutar las acciones de sincronización. Si se siente más cómoda/o con mutex y variables condición, hágalo con ellas (el tipo de herramienta de sincronización utilizada no influirá en la calificación).

```
   // variables de estado:
   // cantidad de zapatos de cada pie en la cesta
   int izdos=0, dchos=0;

   // para empaquetar el código de espera dentro
   // de los while
   void ESPERA() {
     SALIRSC(); DORMIR(); ENTRARSC();
   }

11 // hilos productores
12 void ObreroDerechos() {
13   while (true) {
14     ... fabricar un zapato derecho Z
     ENTRARSC();
       while (izdos+dchos==N) { ESPERA(); }
15     cesto.meterZapatoDerecho(Z);
       DESPERTAR();
     SALIRSC();
16   }
17 }

18 void ObreroIzquierdos() {
19   while (true) {
20     ... fabricar un zapato izquierdo Z
     ENTRARSC();
       while (izdos+dchos==N) { ESPERA(); }
21     cesto.meterZapatoIzquierdo(Z);
       DESPERTAR();
     SALIRSC();
22   }
23 }

24 // hilo empaquetador
25 void Empaquetador() {
26   while (true) {
27     Zapato izq,dch;
       ENTRARSC();
         while (izdos==0) { ESPERA(); }
28       izq = cesto.sacarZapatoIzquierdo();
         izdos--;
         DESPERTAR();
       SALIRSC();
       ENTRARSC();
         while (dchos==0) { ESPERA(); }
29       dch = cesto.sacarZapatoDerecho();
         dchos--;
         DESPERTAR();
       SALIRSC();
30     ... empaquetar zapatos (izq,dch)
31   }
32 }
```

Para solucionar este problema, necesitamos resolver varias cosas:

- El objeto cesto no está protegido, así que las operaciones que hacen los obreros debemos envolverlas en secciones críticas (líneas 15, 21 y 28-29).

- Debemos crear unas variables para conocer cuántos zapatos de cada pie se han colocado en la cesta. Con esas variables podemos colocar condiciones de espera justo antes de las líneas 15, 21 y 28.

NOTA 1. El código del empaquetador puede también implementarse comprobando una sola vez si (izdos==0 || dchos==0) y envolviendo las dos extracciones del cesto en una misma sección crítica. Esto hace que hasta que no haya un par completo disponible, el cesto no retira nada del cesto.

NOTA 2. Hay que tener en cuenta que este código no está preparado para gestionar una situación que se puede dar en el sistema: que el cesto se llene con zapatos de un solo pie. Si se llega a ese estado, el empaquetador no puede extraer un par completo y los obreros no pueden seguir colocando zapatos en el cesto: el sistema entra en interbloqueo. Es una situación que no se pide gestionar en el ejercicio propuesto. Dejamos como ejercicio para el estudiante el mejorar la solución de manera que evite el interbloqueo por «cesta llena de zapatos de un solo pie».