UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

1	2	3	4	test	extra	NOTA

Fundamentos de los Sistemas Operativos
Examen parcial - 3 de abril de 2017

Nombre y apellidos **DNI/NIE**

SOLUCIONES	

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

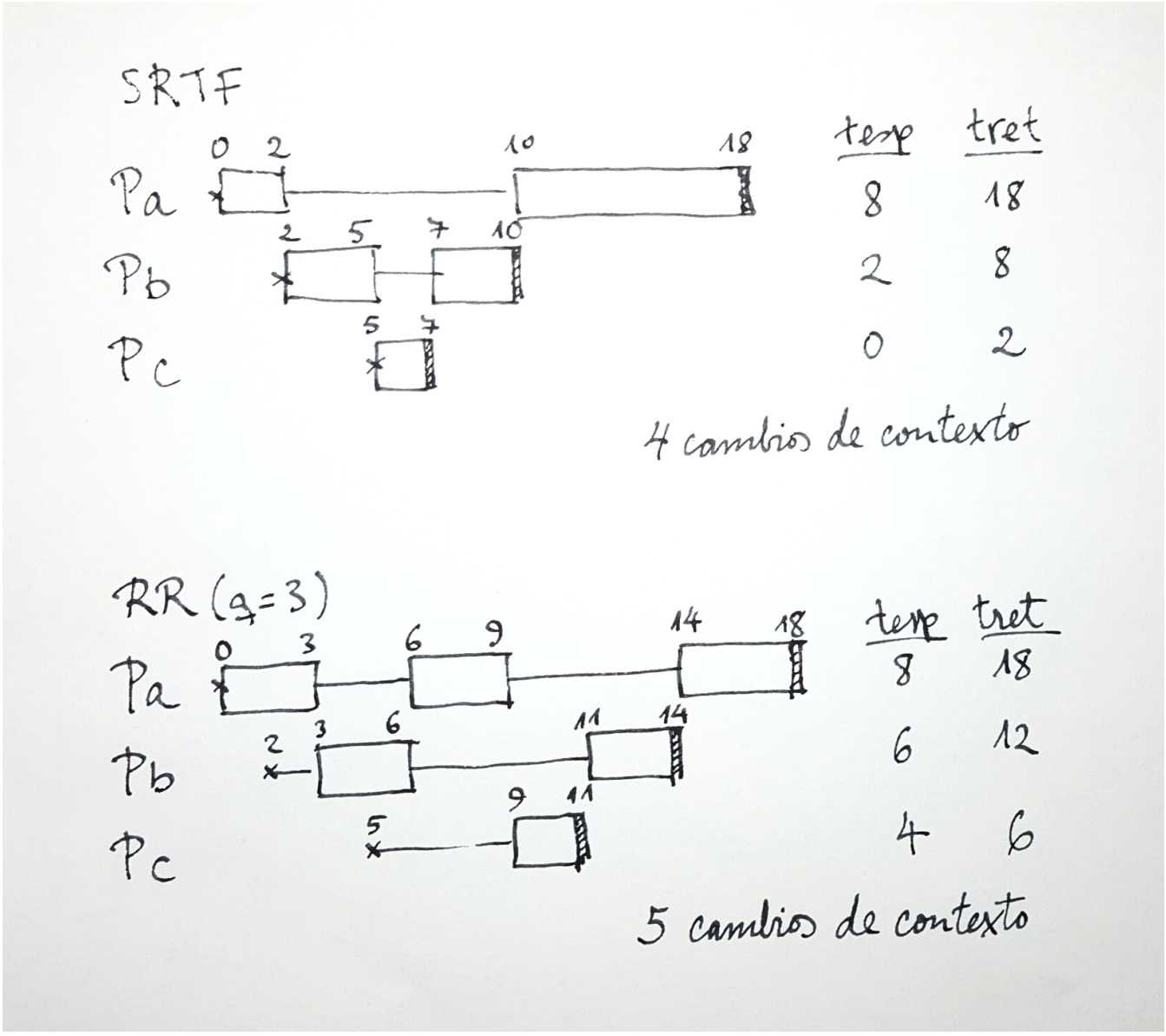
Lee las instrucciones para el test en la hoja correspondiente.

1 (1'25 puntos) A un planificador de CPU llegan tres procesos, según el cuadro adjunto. Aplica las dos políticas SRTF (SJF expulsivo) y RR (Q=3) y, para cada una de ellas, obtén lo siguiente:

- Diagrama de Gantt o similar con la planificación.
- Tiempo de espera y de retorno de cada uno de los procesos.
- Número de cambios de contexto realizados durante la planificación.

proceso	llegada	duración
Pa	0	10
Pb	2	6
Pc	5	2

Solución en la figura inferior:



2 (0'50 puntos) ¿Qué diferencias hay entre un micronúcleo y un núcleo monolítico?

Pregunta básica de conceptos del Tema 1. La diferencia radica en la arquitectura del código.

3 (1 punto) Muchos informáticos sostienen que uno de los beneficios del sistema operativo es que los desarrolladores de aplicaciones no tienen que estar modificando su *software* cada vez que hay un cambio en las tecnologías *hardware*, especialmente cuando aparece una nueva tecnología de almacenamiento (ej. discos SSD). Elabora unos argumentos que apoyen esa afirmación. No uses más de 150 palabras en tu exposición.

Pregunta de aplicación de conceptos trabajados en el Tema 1. Imprescindible mencionar que el SO actúa como intermediario entre las aplicaciones y el hardware de E/S a través de una interfaz de programación (API). Esta API se diseña de manera que sea *independiente del dispositivo*, abstrayendo detalles que puedan variar según las características del hardware. Si se puede desarrollar un manejador interno del SO que adapte el nuevo hardware a la misma API, las aplicaciones no necesitan ninguna modificación, ya que la API del SO permanece inalterada. Sólo hay que actualizar el SO.

4 (1'25 puntos) En una tienda de pájaros tenemos una jaula con canarios que revolotean en su interior. Regularmente cada canario quiere comer de un comedero de alpiste, en el que solamente puede haber tres pájaros al mismo tiempo. Si un pájaro quiere comer y el comedero está lleno, se debe esperar a que haya hueco. Por su parte, el encargado de la tienda de vez en cuando repone el alpiste del comedero. Mientras el encargado está reponiendo, ningún pájaro puede estar comiendo: el encargado debe esperar a que se quede vacío el comedero y, una vez vacío, ningún pájaro entra a comer hasta que el encargado termina de reponer.

El algoritmo general de un pajarito y del encargado se muestra en el siguiente cuadro.

```
void pajarito () {
  while ( pajaritoVivo() ) {
      … revolotear por la jaula
      … esperar a que haya hueco y que
      el encargado no esté reponiendo el
      alpiste
      COMER();
  }
}
```

```
void encargado() {
  while ( ! jubilado() ) {
      … hacer otras cosas
      … esperar a que no haya pájaros
      comiendo
      REPONER_ALPISTE();
  }
}
```

TAREA. Tienes que arreglar el algoritmo para que los pajaritos y el encargado se sincronicen entre todos ellos conforme al enunciado expuesto. Puedes utilizar variables de estado y operaciones básicas de sincronización (entradas y salidas en sección crítica, colas de espera, etc.). Si ya conoces el uso de semáforos, puedes resolverlo con ellos si lo prefieres (no habrá diferencia en la calificación de la pregunta).

NOTA. Escribe tu intento aunque no tengas la solución completa y perfecta para todos los escenarios. En ese caso, describe las limitaciones que sepas que tiene tu propuesta.

Nombre y apellidos	DNI/NIE

Solución: primera versión, con sincronización de bajo nivel

Este algoritmo en realidad es una variante simplificada del «problema de los lectores y escritores», con un máximo de tres lectores (los pájaros) y un solo escritor (el encargado de la tienda). Cualquier solución al problema de los lectores y escritores puede adaptarse con facilidad a este ejercicio.

A continuación se muestra una solución con herramientas de bajo nivel:

Variables compartidas:

```
int comiendo = 0;            // cantidad de pájaros que están comiendo
bool reponiendo = false;     // ¿quiere el encargado reponer?
```

```
void pajarito () {
  while ( pajaritoVivo() ) {
      … revolotear por la jaula
      ENTRARSC();
      while (comiendo==3 OR reponiendo){
        SALIRSC();
        DORMIR();
        ENTRARSC();
      }
      comiendo++;
      SALIRSC();

      // ojo, queda fuera de la sección
      // crítica,para que varios pájaros
      // puedan comer al mismo tiempo
      COMER();

      ENTRARSC();
      comiendo--;
      DESPERTAR();
      SALIRSC();
  }
}
```

```
void encargado() {
  while ( ! jubilado() ) {
      … hacer otras cosas
      ENTRARSC();
      reponiendo = true;
      while (comiendo>0) {
        SALIRSC();
        DORMIR();
        ENTRARSC();
      }
      REPONER_ALPISTE();
      reponiendo = false;
      DESPERTAR();
      SALIRSC();
  }
}
```

Es importante dejar la operación **COMER()** fuera de una sección crítica. Si la envolvemos dentro de una S.C., no se permitirá a más de un pájaro estar en el comedero y por tanto no se cumpliría eficazmente la característica de que puedan comer hasta tres juntos.

Ojo, en la anterior solución no hace falta usar la variable **reponiendo**, ya que el encargado mientras está reponiendo alpiste retiene el uso de la sección crítica e impide a los pájaros usar el comedero. Pero esta variable impide el bloqueo indefinido del encargado en caso de que continuamente lleguen pájaros: los pájaros que intenten comer después de que **reponiendo==true** se quedarán bloqueados en el bucle **while**.

Segunda versión, con semáforos

El código equivalente al anterior algoritmo, pero utilizando semáforos, podría ser así:

```
Semáforo mutex = 1;
Semáforo cola = 0;
int esperando = 0;

void bloquear() {
  esperando++;
  V(mutex);
  P(cola);
  P(mutex);
}

void despertar() {
  while (esperando>0) {
    esperando--;
    V(cola);
  }
}
```

```
void pajarito () {
  while ( pajaritoVivo() ) {
      … revolotear por la jaula
      P(mutex);
      while (comiendo==3 OR reponiendo){
        bloquear();
      }
      comiendo++;
      V(mutex);

      // ojo, queda fuera de la sección
      // crítica,para que varios pájaros
      // puedan comer al mismo tiempo
      COMER();

      P(mutex);
      comiendo--;
      despertar();
      V(mutex);
  }
}
```

```
void encargado() {
  while ( ! jubilado() ) {
      … hacer otras cosas
      P(mutex);
      reponiendo = true;
      while (comiendo>0) {
        bloquear();
      }
      REPONER_ALPISTE();
      reponiendo = false;
      despertar();
      V(mutex);
  }
}
```

Nombre y apellidos	DNI/NIE

Tercera versión, con semáforos

El segundo algoritmo con semáforos se puede optimizar para situar en colas distintas el bloqueo/desbloqueo del encargado y los pajaritos. La optimización lleva a un código más difícil de entender y verificar. El siguiente código se ofrece solamente como referencia para el estudio; no es un código que se pueda elaborar fácilmente durante la realización de un examen.

```
Semáforo mutex=1, colaEncargado=0, colaPajaritos=0;
int comiendo = 0, pajEsperando = 0;
bool reponiendo = false;
```

```
void pajarito () {
  while ( pajaritoVivo() ) {
       … revolotear por la jaula
       P(mutex);
       while (comiendo==3 OR reponiendo) {
         pajEsperando++;
         V(mutex);
         P(colaPajaritos);
         P(mutex);
       }
       comiendo++;
       V(mutex);

       COMER();

       P(mutex);
       comiendo--;
       if (comiendo==0 AND reponiendo) {
         V(colaEncargado);
       }
       elsif (pajEsperando>0) {
         pajEsperando--;
         V(colaPajaritos);
       }
       else V(mutex);
  }
}
```

```
void encargado() {
  while ( ! jubilado() ) {
       … hacer otras cosas
       P(mutex);
       reponiendo = true;
       if (comiendo>0) {
         V(mutex);
         P(colaEncargado);
       }

       REPONER_ALPISTE();

       reponiendo = false;
       for i in 1..min(3,pajEsperando) {
         pajEsperando--;
         V(colaPajaritos);
       }
       V(mutex);
  }
}
```

Cuarta solución con semáforos, compacta

Los semáforos permiten una solución mucho más compacta:

Variable compartida:

```
Semáforo comedero = 3;
```

```
void pajarito () {
  while ( pajaritoVivo() ) {
      … revolotear por la jaula
      P(comedero);
      COMER();
      V(comedero);
  }
}
```

```
void encargado() {
  while ( ! jubilado() ) {
      … hacer otras cosas
      P(comedero);
      P(comedero);
      P(comedero);
      REPONER_ALPISTE();
      V(comedero);
      V(comedero);
      V(comedero);
  }
}
```

En este algoritmo, el encargado acapara todos los *créditos* para poder usar el comedero, con lo cual impide que mientras repone alpiste pueda haber algún pájaro. Si el encargado se encuentra con uno o varios pájaros comiendo, se quedará esperando a que los pájaros liberen sus semáforos.

Eso sí, este algoritmo no resuelve el problema de que continuamente lleguen pajaritos al comedero y con ello impidan trabajar al encargado (los otros algoritmos sí lo resuelven, por medio de la variable **reponiendo**).