UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

1	2	3	test	extra	NOTA

Fundamentos de los Sistemas Operativos
Convocatoria ordinaria - 23 de mayo de 2017
Segunda parte: temas 3, 4 y 5

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) Un sistema de archivos trabaja con bloques de datos de 1KiB y enlaces de 16 bits. Este sistema emplea una política de asignación de espacio indexada simple, con un único bloque de datos para los índices.

¿Cuál es el tamaño máximo que puede llegar a tener un archivo en este sistema?

El tamaño máximo está condicionado por el número de enlaces que caben en el bloque de índices. Como el bloque es de 1KiB y los enlaces son de 2 bytes (16 bits), en el bloque caben 1024/2 = 512 enlaces. Cada enlace apunta a un bloque de 1KiB, así que el tamaño total del archivo con 512 enlaces es de 512×1024 bytes = **512 KiB**.

Se desea que el sistema admita archivos más grandes y con este fin se proponen estas dos intervenciones: a) Duplicar el tamaño de los bloques de datos (pasar a 2KiB); b) permitir que el bloque de índices ocupe dos bloques de datos. ¿Cuál de las dos intervenciones elegirías tú y por qué?

Si duplicamos el tamaño del bloque de datos (2048 bytes), en el bloque de índices cabrán el doble de enlaces: 1024 enlaces de 2 bytes. Por tanto un archivo puede llegar a tener 1024 bloques de 2048 bytes, que son $1024 \times 2048 = 2^{10} \times 2^{11} = 2^{21} = 2$ MiB.

Por su parte, si permitimos que el bloque de índices ocupe dos bloques de datos, duplicaremos la cantidad de enlaces que puede almacenar, o sea que también pasamos a 1024 enlaces. Como los bloques siguen siendo de 1 KiB, el tamaño máximo del archivo será de 1024×1 KiB = 1 MiB.

Observando cómo resultan los tamaños máximos de ambas intervenciones, está claro que la primera opción (duplicar el tamaño del bloque de datos) aumenta mucho más el límite.

2 (1'5 puntos) En un sistema de memoria virtual paginada tenemos esta secuencia de referencias a páginas virtuales de un proceso: 1, 2, 3, 4, 2, 5, 1, 3, 4, 5, 3, 1, 5. El proceso tiene tres marcos asignados, inicialmente vacíos. Planifica esta secuencia con estos dos algoritmos: óptimo (OPT) y segunda oportunidad. Indica en cada caso cuándo se produce un fallo de página y qué página se reemplaza.

Esta es la planificación según el algoritmo óptimo.

acceso	1	2	3	4	2	5	1	3	4	5	3	1	5
marcos	1--	12-	123	124	124	154	154	354	354	354	354	154	154
víctima				3		2		1				3/4	
fallo	F	F	F	F		F		F				F	

Se produce un total de siete fallos de página.

En el penúltimo acceso, a la página 1, se pueden elegir como víctimas tanto la página 3 como la 4, ya que no se van a acceder más en el futuro.

Esta es la planificación según la «segunda oportunidad».

acceso	1	2	3	4	2	5	1	3	4	5	3	1	5
marcos	»1+	»1+ 2+	»1+ 2+ 3+	4+ »2- 3-	4+ »2+ 3-	»4+ 2- 5+	4- 1+ »5+	3+ »1+ 5-	»3+ 1- 4+	3- 5+ »4+	3+ 5+ »4+	»3- 5- 1+	»3- 5+ 1+
víctima				1		3	2	4	5	1		4	
fallo	F	F	F	F		F	F	F	F	F		F	

Se produce un total de diez fallos de página.

En la descripción de los marcos, se usan estas convenciones:
- **»** es el marco al cual apunta la *manecilla del reloj* que empezará a buscar posibles víctimas.
- **+** la página tiene el bit de referencia activo
- **-** la página tiene el bit de referencia desactivado

---

3 (1'25 puntos) Tenemos un baño público en el que cabe un número ilimitado de personas. Cada cierto tiempo un empleado realiza la limpieza del baño, actuando de la siguiente forma: pone un letrero para impedir que entren personas al baño, se espera a que quienes están dentro acaben y salgan, y una vez que el recinto está vacío, realiza la limpieza. Cuando termina de limpiar, retira el letrero y se marcha, quedando el baño libre para que entren nuevos usuarios.

Se quiere implementar un software que simule este baño, con hilos que implementan a las personas que interactúan con él. A continuación se muestran los bocetos de las funciones que ejecutarán los hilos usuarios y el hilo limpiador:

```
void usar_baño () {
  … esperar a que el baño esté libre
  … y llegados a este punto…
  … hacer lo habitual en un baño
  … y salir del baño
}
```

```
void limpiar_baño() {
  … poner letrero «baño cerrado»
  … esperar a que se quede vacío
  … limpiar el baño
  … quitar letrero
}
```

TAREA. Tienes que añadir al boceto de código las acciones de sincronización necesarias para cumplir con las especificaciones descritas arriba. Utiliza **semáforos** como herramienta de sincronización.

NOTA. Si te cuesta implementar el problema con semáforos, utiliza las herramientas de "esperar" y "despertar" que vimos en clase, o al menos expresa algorítmicamente cómo hay que sincronizar los procesos. Así podrás tener algo de puntuación en este ejercicio.

El sistema planteado no es más que una variante del **problema de los lectores y escritores**. Es una variante con prioridad a los escritores (el limpiador) y con la característica de que solamente hay un proceso escritor. Por tanto, puede servir como solución cualquier algoritmo que resuelva el *segundo problema de los lectores y escritores*, o en general que dé prioridad a los escritores.

Aquí se muestra una posible implementación, adaptada a las especificaciones del enunciado. La solución utiliza un *mutex* y dos colas de espera: una para el limpiador y otra para los usuarios que se encuentren con que se está limpiando el baño.

Nombre y apellidos	DNI/NIE

```
int usuarios_esperando = 0;
int usuarios_dentro = 0;
bool letrero_puesto = false;

Semáforo mutex = 1;
Semáforo usuarios = 0;
Semáforo limpiador = 0;

void usar_baño() {
      P(mutex);
      // Bloquearse si el limpiador ha puesto
      // el letrero
      while (letrero_puesto) {
            usuarios_esperando++;
            V(mutex);
            P(usuarios);
            P(mutex);
      }
      usuarios_dentro++;
      V(mutex);

      // ... usar el baño
      // nótese que se hace fuera del mutex
      // para permitir usuarios concurrentes

      P(mutex);
      usuarios_dentro--;
      // Si es el ultimo usuario en salir
      // y el letrero está puesto, avisa al limpiador
      if (usuarios_dentro==0 and letrero_puesto) {
            V(limpiador);
      }
      V(mutex);
}

void limpiar_baño() {
      P(mutex);
      letrero_puesto = true;
      // Bloquearse mientras haya usuarios dentro del baño
      if (usuarios_dentro>0) {
            V(mutex);
            P(limpiador);
            P(mutex);
      }

      // ... limpiar el baño

      letrero_puesto = false;
      // Si hay usuarios esperando, hay que desbloquearlos
      while (usuarios_esperando>0) {
            usuarios_esperando--;
            V(usuarios);
      }
      V(mutex);
}
```