UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

**Fundamentos de los Sistemas Operativos**
**Convocatoria ordinaria - 20 de mayo de 2019**

| 1 | 2 | 3 | test | extra | NOTA |
|---|---|---|---|---|---|
| | | | | | |

| **Nombre y apellidos** | **DNI/NIE** |
|---|---|
| | |

***DURACIÓN:*** *Dispone de 105 minutos para realizar el examen.*

*En los ejercicios justifique TODAS sus respuestas. Las respuestas que sólo den resultados sin justificar se considerarán nulas.*

*Lea las instrucciones para el test en la hoja correspondiente.*

---

**1 (1,75 puntos)** El sistema operativo ofrece funciones y servicios que persiguen mejorar aspectos bien distintos del sistema, tales como la seguridad, el rendimiento y la usabilidad. Por cada uno de esos tres aspectos, tiene usted que dar un ejemplo de un servicio del sistema operativo cuyo objetivo principal consista en mejorar ese aspecto particular. En cada caso, justifique por qué ha elegido ese servicio.

Estos son los tres aspectos para los que tiene que encontrar un servicio del S.O. que sirva de ejemplo:

- Mejorar la protección y seguridad del sistema.
- Aumentar el rendimiento del sistema (velocidad, aprovechamiento de los recursos, etc.).
- Mejorar la usabilidad del sistema (facilitar a los usuarios la interacción con el sistema).

En cualquier texto introductorio sobre sistemas operativos pueden encontrarse referencias a servicios que destacan por alguno de los aspectos mencionados. Por ejemplo, la *multiprogramación* se concibió como un mecanismo para aumentar el rendimiento del procesador. Los *sistemas de archivos* y el propio concepto de archivo pretende crear una visión del almacenamiento más cercana a la forma en la que los usuarios trabajan con la información (usabilidad). Ejemplos de funciones que tienen que ver con la seguridad: *protección de zonas de memoria*, la distinción entre niveles de privilegio del código de usuario y el código del núcleo, o el concepto de *sistema multiusuario* con distintos privilegios para distintas personas.

---

**2 (1,25 puntos)** En una máquina con un solo procesador tenemos un planificador de procesos que utiliza el algoritmo Round Robin con Q=1 milisegundo. El tiempo de cambio de contexto se considera despreciable. En la cola de preparados tenemos un conjunto de 20 procesos recién llegados, todos ellos con intención de ejecutar una ráfaga de 2 milisegundos de duración. Bajo estas condiciones, obtenga los siguientes datos:

- ¿Cuál será el tiempo de retorno del primer proceso en finalizar?
  21 mseg (sus 2 mseg. más 19 ciclos de los otros procesos)
- ¿Cuánto tardará en vaciarse toda la cola?
  2·20 = 40 mseg (la suma de las ráfagas de todos los procesos. Los procesos se ejecutan uno tras otro sin tiempos muertos en medio).
- ¿Cuántos cambios de contexto serán necesarios para planificar toda esta carga?
  Cada proceso entra y sale de la CPU dos veces. Como son 20 procesos, serán unos 40 cambios de contexto. Si no contamos el cambio al entrar el primer proceso y/o al salir el último, podemos considerar válidas las respuestas de 38 y 39 cambios.
- ¿Cuál será el tiempo medio de espera para los procesos de la cola?
  El primer proceso tiene que esperar 19 mseg hasta terminar (la primera ronda de los otros 19 procesos). El segundo proceso también tiene que esperar esos 19 mseg, más 1mseg en la primera ronda, esperando a que le toque su primer turno. El tercer proceso espera 19meg más 2 mseg en su primer turno. En general, el proceso K espera 19+(K-1) mseg. Por tanto, la suma de tiempos de espera es 0+1+2+…+19 + (20·19) = 190 + 380 = 570 mseg. El promedio es 570/20 = **28,5 mseg**.

**3 (1 punto)** Se propone el siguiente algoritmo para gestionar el acceso a una sección crítica para dos procesos. Analice si verifica todas las propiedades que debe cumplir una solución aceptable y por tanto puede considerarse un algoritmo correcto.

```
// variables globales
bool sc1 = false, sc2 = false;
int turno = 1;
```

Código del proceso 1

```
11  while ( true ) {
12    … Sección no crítica …
13    sc1 = true;
14    while ( turno==2 && sc2 ) {}
15    … SECCIÓN CRÍTICA …
16    turno = 2;
17    sc1 = false;
18  }
```

Código del proceso 2

```
21  while ( true ) {
22    … Sección no crítica …
23    sc2 = true;
24    while ( turno==1 && sc1 ) {}
25    … SECCIÓN CRÍTICA …
26    turno = 1;
27    sc2 = false;
28  }
```

Este algoritmo es muy similar a la solución de Peterson (ver libro de los dinosaurios y diapositivas del Tema 3). Pero tiene una diferencia importante, y es que no aparece la cesión del turno al otro proceso *antes* de intentar entrar en sección crítica. Como veremos, esto invalida el algoritmo, ya que no asegura la exclusión mutua.

La idea del algoritmo parece ser que cada proceso avise mediante un indicador (**sc1**, **sc2**) si está interesado en entrar en sección crítica. En caso de que ambos estén interesados, la variable **turno** sirve para desempatar (líneas 14 y 24). Si ambos procesos tratan de entrar a la vez en la sección crítica, tanto **sc1** como **sc2** estarán a **true**, pero solo uno de los while mantendrá al proceso bloqueado, que será aquel cuyo ID no coincida con el valor de **turno**.

Pero esta idea está mal ejecutada en el algoritmo: si un proceso observa que el otro proceso no está interesado en entrar en sección crítica, el proceso entra, sea cual sea el valor de **turno**. Esto aparentemente es correcto. Pero supongamos este caso: el proceso 1 ha entrado en sección crítica siendo **turno==2** y **sc2==false**. Si mientras el proceso 1 está en su sección crítica (línea 15), el proceso 2 quisiera entrar, evaluará en la línea 24 la condición de bloqueo y, como **turno==2**, no se mantendrá en el **while** y avanzará a la sección crítica. En este caso, tendríamos a los dos procesos en su sección crítica (líneas 15 y 25), violándose con ello la propiedad de exclusión mutua. Este escenario puede repetirse invirtiendo los papeles del proceso 1 y el proceso 2.