# SO2

Nombre:
DNI:

# Primer control de teoría

Todas las respuestas se tienen que justificar brevemente. **Una respuesta sin justificar se dará como no contestada.**

## 1. *(4 puntos) Preguntas cortas*

a) Al arrancar el ordenador se ejecuta un código de enumeración y comprobación de dispositivos, ¿cómo sabe en qué posición de memoria se encuentra este código?

b) ¿En qué momento(o momentos) **se usa** la TSS en un sistema operativo tipo ZeOS?

c) ¿Al hacer un fork por qué hay que mapear las páginas del hijo en el espacio de direcciones del padre?

d) Si el mapeo de las páginas de kernel siempre es lineal (las lógicas coinciden con las físicas) ¿por qué es necesario poner las traducciones en la tabla de páginas?

e) Supón un proceso cuyo espacio de direcciones empieza en la dirección 0x100000 y acaba en la 0x200000, incluyendo código, datos y sistema. Indica en lenguaje C un trozo de código para producir un acceso inválido a memoria.

Nombre:
DNI:

f) En un sistema tipo ZeOS con planificación Round Robin, ¿es posible que se entre en modo sistema a traves del handler de una llamada a sistema y se vuelva a modo usuario a través del handler de una interrupción?

g) En un sistema con Round Robin con el siguiente grafo de estados para sus procesos:


¿Qué situaciones pueden provocar la ejecución de la política de planificación?¿Y si cambiamos la política para que sea Round Robin con prioridades?

h) Dibuja el contenido de la pila de sistema justo antes de ejecutar la 1ª instrucción de la rutina de atención a la interrupción de reloj.

## 2. *(1 punto) Espacio de direcciones*

Suponiendo un sistema operativo tipo ZeOS en el que toda la memoria física está inicialmente inicializada a 0, que la TLB está vacia, que la tabla de páginas activa sólo tiene inicializada la zona de código y donde A es una variable de tipo entero situada en la dirección lógica 0x108008. Dado el siguiente fragmento de código:

```
...(1)...
A++;
...(2)...
A++;
...(3)...
A++;
```

Nombre:
DNI:

Tienes que completar el código de los puntos 1, 2 y 3 añadiendo 0, 1 o 2 instrucciones de gestión de memoria (set_cr3, set_ss_page, del_ss_page) para que después de ejecutar el último incremento de A el contenido de las direcciones físicas 0x108008, 0x109008 y 0x10A008 contengan los valores 2, 0 y 1 respectivamente. Todos los frames físicos implicados tienen que estar mapeados al menos una vez en la tabla de páginas. Se valorará minimizar el número de instrucciones añadidas.

Código (1)

Código (2)

Código (3)

Justificación:

***set_ss_page(int logic, int frame)*** : Modifica la tabla de páginas activa para mapear la página lógica *logic* en el frame físico *frame* .
***del_ss_page(int logic***): Elimina el mapeo de la página lógica *logic.*
***set_cr3()***: Elimina las entradas de la TLB.

Nombre:
DNI:

### 3. *(5 puntos) Gestión de procesos*

Queremos modificar el código de ZeOS para que la llamada *exit* pueda devolver un valor al padre del proceso que la ejecuta tal y como ocurre en Unix. Para ello modificaremos la implementación de la llamada a sistema *exit* y añadiremos la llamada a sistema *wait*. Los nuevos interfaces serán:

*void exit(int codigo_fin)*
*int wait(int *código_fin_hijo)*

El comportamiento será el siguiente.

**Llamada a sistema *exit***: cuando un proceso acabe, el sistema comprobará si su padre sigue vivo o si ya ha acabado la ejecución, y si el proceso también tiene hijos.

1. Si el padre sigue en ejecución, podemos distinguir dos situaciones.
   1.1. La primera es que el padre **no esté esperando el fin de un hijo** en la llamada a sistema wait. En este caso, el proceso se marcará en estado ZOMBIE, y se liberarán todos sus recursos excepto el PCB.
   1.2. La segunda es que el padre **esté esperando el fin de un hijo**. En este último caso se liberan todos los recursos del hijo como la actual implementación de *exit*, y se completa la ejecución de la llamada *wait*:
      - Dejar en la dirección que se había pasado como parámetro al *wait* el parámetro del *exit*
      - Hacer que *wait* devuelva el *pid* del hijo
      - Y desbloquear al proceso padre.
2. Si por el contrario, el padre ha acabado la ejecución, el sistema liberará los recursos asignados al proceso y marcará que el PCB del proceso está libre.
3. Si el proceso tiene algún hijo (vivo o ZOMBIE) el sistema liberará el PCB de los procesos hijo ZOMBIE y marcará el resto de procesos hijos como que su padre ya ha acabado la ejecución sin esperarlos.

**Llamada a sistema *wait***: Cuando un proceso ejecute la llamada a sistema *wait*, el sistema comprobará si tiene algún hijo ya sea en estado ZOMBIE o en ejecución.

1. Si no tiene ningún hijo, la llamada a sistema devolverá error y el proceso continuará con la ejecución.
2. Si tiene algún hijo, el comportamiento depende del estado de los hijos.
   2.1. Si algún hijo está ZOMBIE, el sistema tratará el caso sólo de uno de esos hijos de la siguiente manera:
      - Dejará en en la dirección que se ha pasado como parámetro al *wait* el valor que el hijo había puesto como parámetro del *exit*
      - Liberará el PCB del hijo
      - Y devolverá como resultado del *wait* el *pid* del hijo ZOMBIE
   2.2. En caso de que todos los hijos del proceso se encuentren en ejecución el proceso padre se bloqueará hasta que uno de los hijos acabe.

Contesta a las siguientes preguntas **justificando tus respuestas** (cualquier respuesta sin justificación se considerará errónea).

Nombre:
DNI:

a) (*0,75 puntos*) Indica cuáles de los siguientes componentes de ZeOS será necesario modificar para adaptarlos a esta nueva funcionalidad.

| Componente | Se modifica/no se modifica | ¿Cómo se modifica?/¿Por qué no se modifica? |
|---|---|---|
| PCB | | |
| sys_fork | | |
| exit (wrapper) | | |

b) (*0,5 puntos*) ¿Crees que es necesario añadir alguna estructura de datos para adaptar la implementación de la planificación de procesos? En caso afirmativo describe qué estructura haría falta y cómo se utilizaría.

c) (*1 punto*) Para implementar el caso 1.2 del *exit*, ¿será necesario modificar la tabla de páginas del proceso hijo? Justifica tu respuesta.

1. Sí, porque necesitamos darle acceso a la zona de pila de kernel del padre
2. Sí, porque necesitamos darle acceso a la zona de datos de usuario del padre
3. Sí, porque necesitamos darle acceso a la zona de pila de kernel del padre y a la zona de datos de usuario del padre
4. No, porque el hijo ya tiene acceso a toda la memoria a la que necesita acceder

d) (*1 punto*) Para implementar el caso 2.1 del *wait*, ¿será necesario modificar la tabla de páginas del proceso padre?

1. Sí, porque necesitamos darle acceso a la zona de pila de kernel del hijo
2. Sí, porque necesitamos darle acceso a la zona de datos de usuario del hijo

3. Sí, porque necesitamos darle acceso a la zona de pila de kernel del hijo y a la zona de datos del hijo
4. No, porque el padre ya tiene acceso a toda la memoria a la que necesita acceder

e) (*1 punto*) Cuándo un proceso hace *exit*, es necesario detectar si tiene algún hijo y localizarlo. Se proponen las siguientes alternativas

- Opción 1: Usar un nuevo campo en el PCB que contiene el pid del padre y recorrer la readyqueue buscando los procesos hijos
- Opción 2: Añadir al PCB una lista de los procesos hijos y comprobar si esta lista está vacía

¿Cuál de las 2 opciones seleccionarías? ¿Por qué?

f) (*0,75 puntos*) ¿Con esta nueva funcionalidad, puede ocurrir que se disminuya el grado de concurrencia posible en ZeOS (número de procesos simultáneos en ejecución)?