# SOA2 (17/01/2023)

Nombre:

DNI:

# Examen final de teoría

***Justifica todas tus respuestas. Una respuesta sin justificación se considerará errónea.***

## 1. *(2 puntos) Contextos*

Indica el contenido de la pila de sistema del proceso indicado en los siguientes casos:

a) Un proceso se está ejecutando en modo usuario

b) Un proceso esta en modo sistema justo antes de ejecutar la primera instrucción del handler de las llamadas al sistema.

c) El proceso idle está creado, pero aún no se ha ejecutado nunca.

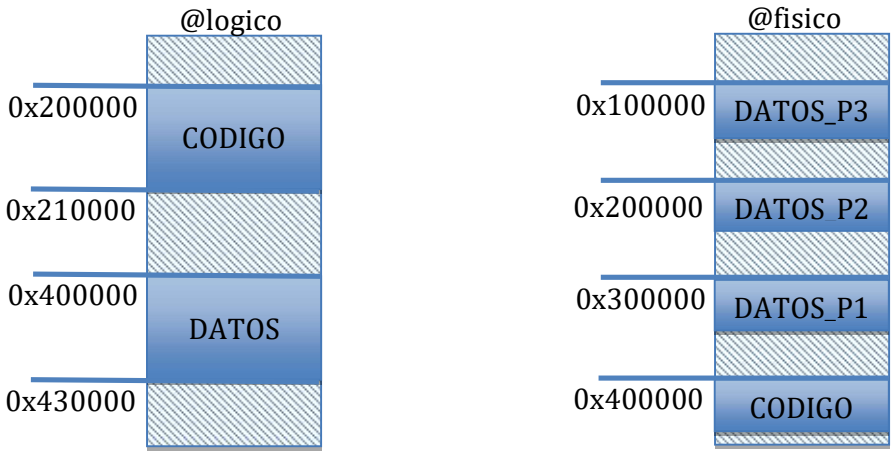
d) Un proceso está en modo sistema dentro del handler de la interrupción de reloj antes de ejecutar la instrucción iret.

e) Un proceso está en modo sistema antes de ejecutar la instrucción pop ebp de task_switch

Nombre:
DNI:

## 2. (2 puntos) Memoria

Tenemos una máquina, con arquitectura de 32 bits, que gestiona la memoria mediante una única tabla de páginas **sin directorio**, y donde las páginas ocupan 4KB.
En esta máquina instalamos un sistema operativo tipo ZeOS, en la que todos los procesos de usuario tienen el mismo espacio de direcciones lógico (como el de la siguiente figura). En un momento determinado el proceso P2 está en ejecución y encontramos el siguiente contenido en el espacio de direcciones físico correspondiente a 3 procesos distintos:




a) En esta máquina, ¿cuántas entradas tiene la tabla de páginas?

b) Indica el contenido de la tabla de páginas (posición->valor) para mapear la zona de datos del proceso actual en este espacio de direcciones (al menos la 1ª y la última dirección de la zona).

c) Completa el código siguiente para copiar la zona de datos del proceso actual a la zona de datos del proceso P3 usando una única página libre del espacio lógico. NOTA: Puedes usar las rutinas del final del ejercicio que creas convenientes.

```
for( i = 0;  i< ……..; i ++) {

        copy_data(
```

d) ¿Cuántos fallos de TLB generará el código del apartado anterior como mínimo?

### *3. (2 puntos) Planificación y fork:*

Actualmente, la llamada al sistema fork, ejecuta las siguientes acciones:

1.-Asignar PID del hijo
2.-Asignar PCB del hijo
3.-Copiar contexto de ejecución del padre al hijo
4.-Asignar memoria física al proceso hijo para crear su imagen
5.-Asignar, de forma temporal, la memoria física del proceso hijo al padre
6.-Copiar el contenido de la memoria del proceso padre a esta memoria asignada
7.-Inicializar las estructuras necesarias del proceso hijo
8.-Crear contexto de ejecución del proceso hijo
9.-Encolar PCB del hijo en la cola de ready
10.-Return PID del hijo

Queremos modificar esta implementación para minimizar el tiempo que el proceso padre está ejecutando el fork. Para ello, hacemos que la mayoría de las acciones se ejecuten cuando el proceso hijo vaya a pasar por primera vez de Ready a Run. Así, las acciones 1, 2, 3, 8, 9 y 10 las realizará el padre y el resto las ejecutará el hijo.

a. Con esta distribución de trabajo entre padre e hijo, ¿Se reduce el tiempo de ejecución de fork en el proceso padre?

b. ¿En qué parte del código, visto en clase, implementarías las acciones que tiene que realizar el hijo al pasar a ready?

c. Cuando ejecutamos el siguiente código, vemos que, a veces, la variable "a" tiene el valor 1 en el hijo y otras no. ¿Cómo es posible?

```
int main()
{
    int a=0;
    switch (fork()) {
           case 0: while(1);
           default: a=1;
    }
    while(1);
}
```

d. Sin modificar la distribución de trabajo que hemos hecho en el fork, ¿cómo solucionarías el problema del apartado anterior?

## 4. (2 puntos) /dev/null

Queremos implementar el dispositivo lógico /dev/null. Este dispositivo lógico es un "agujero negro": todo lo que se escribe en él, no se almacena en el sistema de ficheros sino que desaparece. Las lecturas a /dev/null siempre devuelven final de fichero. Por ahora supondremos que existe un gestor asociado.

La sucesión de llamadas a funciones que se hace hasta llegar al código del driver de /dev/null, para la llamada al sistema read, es:

Nombre:
DNI:

int sys_read(int fd, void *buffer, size_t size)
↓
int read_vfs(struct Inode *Inode, void *buffer, size_t size)
↓
Gestor VFS
↓
int read_gestor(struct Inode *Inode, int offset, void *buffer, size_t size)
↓
Gestor /dev/null
↓
int read_driver(struct opaque* data, int offset, void *buffer, size_t size)
↓
Driver /dev/null

a) Escribe el código del read dependiente del gestor de /dev/null.

```
int read_gestor(struct Inode* Inode, int offset, void *buffer, size_t size)
{




}
```

b) Escribe el código del write dependiente del gestor de /dev/null

```
int write_gestor(struct Inode* Inode, int offset, void *buffer, size_t size)
{




}
```

c) Escribe el código del read del driver de /dev/null

```
int read_driver(struct opaque* data, int offset, void *buffer, size_t size)
{




}
```

d) Escribe el código del write del driver de /dev/null

```
int write_driver(struct opaque* data, int offset, void *buffer, size_t size)
{




}
```

e) Dado el funcionamiento de este dispositivo, ¿sería necesario tener un gestor? En caso negativo indica qué funciones de las anteriores se tendrían que modificar y cómo se modificarían.

## 5. (2 puntos) MSDOS y Windows 95

En MsDos, un programa podía modificar el contenido de la IDT. Para mantener esta característica en Windows 95, Microsoft® decidió virtualizar la IDT, haciendo que cada proceso tuviese su propia IDT. De esta forma, cuando un programa modificaba la IDT, en realidad, modificaba su copia de la IDT haciendo que el resto de procesos no viese esta modificación. En realidad, la modificación es mucho más compleja como veremos ahora:

a) Teniendo en cuenta que MsDOS es un sistema operativo monoprogramado (solo se ejecutaba un programa a la vez) y Windows 95 era multiprogramado, ¿qué entradas de la IDT correspondientes a interrupciones hardware no se podían modificar?

b) ¿Cómo detectarías que un programa estaba intentando modificar la IDT?

c) ¿Cómo se tiene que modificar el cambio de contexto para tener en cuenta esta característica?

d) Como MsDOS no trabajaba con modos de privilegios, se podia colgar cualquier código (de usuario) de cualquier entrada de la IDT. En Windows 95 si que se trabaja ya con modos de privilegio. ¿Esto supone algún problema para ejecutar el código que se habia colgado de una entrada de la IDT en un proceso MsDOS? ¿Cómo lo solucionarias?

### *Información del Sistema Operativo*

Cada proceso contiene una estructura para guardar su espacio de direcciones, consistente en un directorio de páginas con una única entrada de válida que es su tabla de páginas. Cada una de las entradas de la tabla de páginas usada por la MMU (*page_table_entry*) contiene, entre otros, los siguientes campos (codificados en una estructura de 32 bits):

- ***present***: *Present flag*, si este bit està a 1, la pàgina està a memòria principal; si està a 0, la pàgina no està a memòria principal i llavors la resta de bits de l'entrada es poden usar pel sistema operatiu.
- ***pbase_addr***: *Address field,* camp amb els 20 bits més significatius de l'adreça física d'un marc de pàgina (frame).
- ***user***: *User/Supervisor flag*, bit per indicar si la pàgina és d'usuari o sistema
- ***rw***: *Read/Write flag*, bit per indicar els permisos d'accés de la pàgina (*Read/Write* o *Read*)

El sistema tambien dispone de las siguientes rutinas:

- ***struct task_struct *current():*** Retorna la task_struct del proceso actual.
- ***int alloc_frame()*:** Reserva un frame de memoria física.
- ***void free_frame (int frame)***: Libera un frame de memoria.
- ***page_table_entry * get_PT(struct task_struct *t):*** Retorna la tabla de paginas del proceso *t*.
- ***page_table_entry * get_DIR(struct task_struct *t):*** Retorna el directorio de páginas del proceso *t*.
- ***set_CR3 (page_table_entry * dir):*** Sobreescribe el registro CR3 con el nuevo directorio de páginas *dir*, provoca una invalidación de la TLB.
- ***int get_frame (page_table_entry *pt, int logical_page)***: Retorna el frame asignado a una página lógica en la tabla de páginas de un proceso determinado.
- ***int set_ss_page (page_table_entry *pt, int logical_page, int frame):*** Asigna un *frame* a una página lógica de la tabla de paginas *pt*.
- ***int del_ss_page (page_table_entry *pt, int logical_page)*:** Borra una entrada de la tabla de páginas.