SOA

Nombre:
DNI:

Examen final de teoría

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

1. *(3 puntos) Preguntas cortas*

Contesta si son ciertas o falsas las siguientes afirmaciones justificando brevemente tu respuesta:

1. Cualquier llamada a función desde modo usuario cambia el nivel de privilegio en un procesador de Intel.

2. La estructura que nos permita encontrar la pila a usar cuando entramos a ejecutar en modo sistema es el PCB.

3. En una llamada al sistema, el contexto hardware lo guarda automáticamente el procesador mientras que el contexto software se tiene que guardar en el handler de la llamada al sistema.

4. La pila usada para guardar el contexto hardware podria ser la misma que la pila usada en modo usuario.

5. En un procesador de 32 bits, con un sistema operativo que utiliza páginas de 4 KBs, el TLB utiliza los 20 bits de menos peso de la dirección lógica como identificador de página lógica.

Nombre:
DNI:

6. Durante un cambio de contexto se tiene que modificar el registro esp del procesador para que apunte a la cima de la pila de sistema del siguiente proceso a ejecutar.

7. Al crear un proceso nuevo con *fork* no hace falta copiar la pila de sistema del proceso padre al proceso hijo, ya que esta copia se realiza cuando se crea el espacio de direcciones logico del proceso hijo.

8. La finalidad del Inodo es almacenar el número de referencias a un fichero en un sistema de ficheros con una estructura de directorios en árbol.

9. Gracias al Major y al minor, se consigue que el sistema operativo sea independiente del dispositivo físico.

10. En un sistema operativo actual, la copia del espacio de direcciones del kernel de la tabla de páginas del proceso padre al hijo representa la mayor parte del tiempo de ejecución de fork.

11. Al crear el proceso *idle* el valor asignado como EBP es 0 pero podria ser cualquier valor.

Nombre:
DNI:

12. La página 1 contiene la dirección de memoria 0x1234.

13. En Linux puedo usar el registro ESP para encontrar el PCB del proceso que está en ejecución.

14. La tabla de canales guarda los dispositivos virtuales accesibles para el proceso.

15. El sector es la unidad mínima de transferencia del sistema de ficheros

16. La asignación contigua de bloques a ficheros tiene un mejor tiempo de acceso aleatorio que la asignación indexada multinivel.

17. La buffer cache de un sistema de ficheros es solamente necesaria si existe un virtual file system.

Nombre:
DNI:

18. El descriptor de dispositivos contiene, entre otras cosas, el código dependiente de un dispositivo físico.

19. En un sistema operativo que permite varios threads por proceso, se tiene que implementar, necesariamente, un planificador a dos niveles.

20. La implementación del wrapper de una llamada al sistema depende de como se ha implementado el handler de las llamadas a sistema.

2. *(2 puntos) Gestión de procesos*

1. Supon un espacio de direcciones similar al de ZeOS en el que usamos un único directorio de páginas y una única tabla de páginas (1ª entrada del directorio) ya inicializados. Dada la siguiente salida del comando **nm** :

```
0x1004        A
0x1008        pA
```

mostrando las direcciones asignadas a las variables globales A y pA respectivamente; y el siguiente código:

```
1.  int A;
2.  char *pA;
3.  set_ss_pag(1, 1);
4.  A = 1;
5.  pA = &A;
6.  *pA = *pA + 1;
7.  set_ss_pag(1, 2);
8.  set_CR3(); //flush TLB
9.  /* A? == ............................ */
10. *pA = *pA + 1;
11. del_ss_pag(1);
12. *pA = *pA + 1;
13. /* A? == ............................ */
```

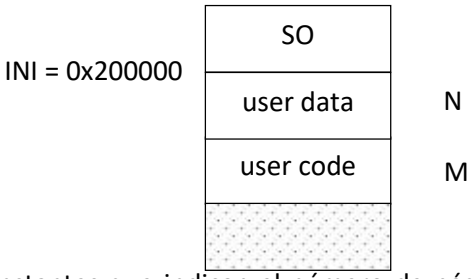
Nombre:
DNI:

Donde *set_ss_pag(pagina, frame)* asigna un *frame* a la entrada *pagina* de la tabla de páginas y *del_ss_pag(pagina)* que pone todos los bits de la entrada a zero. Suponiendo que una posición de memoria NO modificada tiene un valor 0 por defecto y que NO se ha accedido nunca a memoria, indica en las líneas de puntos cual sería el valor de A si se consultara en ese momento.

2. Indica para cada línea indicada del código anterior si produce un HIT/MISS en la TLB o una excepción. Nota: Puedes ignorar los accesos a memoria debidos a instrucciones.

	HIT	MISS	EXCEPCION
4			
5			
6			
10			
12			

3. Dado un sistema operativo donde el espacio lógico de sus procesos es como indica la siguiente figura:



Donde N i M son constantes que indican el número de páginas que ocupan la zona de datos y código de usuario respectivamente. Se quiere implementar la llamada a sistema para crear un nuevo proceso, concretamente, **la parte de herencia de la zona de datos de usuario** (el código de usuario se compartirá para todos los procesos). Puedes considerar que el tamaño en bytes que ocupa una página está definido en la constante PAGESIZE. Suponiendo que dispones de un vector frames (`int frames[N]`) que contiene los frames asignados al nuevo proceso, responde a las siguientes preguntas:

a) ¿Cuál es la 1ª dirección lógica sin asignar que se podría usar para implementar esta herencia? Indica una fórmula para calcularla

Nombre:
DNI:

b) Llama P a éste valor inicial y úsalo para implementar en C la herencia de de la zona de datos de usuario:

```
for (i=0; i < ……; i++) {

        copy_data(

}
```

3. (3 puntos) Creación de procesos

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 padre para crear la imagen del hijo
5.-Copiar el contenido de la memoria del proceso padre a esta memoria asignada
6.-Asignar/mapear esta memoria física al hijo
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

a.- Indica qué estructuras de datos se acceden para cada uno de estos pasos

b.- Indica qué modificaciones se tienen que hacer a este algoritmo para que el proceso hijo se ejecute a continuación y el padre pase al estado de Ready.

c.- Si el sistema operativo ocupa 128KBs y el código, los datos y la pila del usuario del proceso padre ocupan, respectivamente, 24KBs, 256KBs y 4KBs, y las páginas tienen un tamaño de 4KBs, indica cuantas entradas dentro de la tabla de páginas del proceso padre están asignadas a páginas físicas

d.- Con los datos del apartado anterior, indica cual es el número total de páginas físicas que se necesitan para poder tener creados los dos procesos en un sistema operativo basado en Linux (no en ZeOS) teniendo en cuenta que no existe ninguna optimización de memoria.

e.- Con los datos del apartado anterior, indica cual es el número total de páginas físicas que se necesitan para poder tener creados los dos procesos en ZeOS

4. Gestores (2 puntos)

Disponemos en nuestro sistema de un sistema de ficheros virtual que abstrae dos sistemas de ficheros: EXT4 y EXT2. Para ello, tanto el sistema de ficheros virtual como los sistemas de ficheros EXT4 y EXT2 disponen, cada uno de ellos, de un gestor que realiza las operaciones de entrada/salida. El gestor del sistema de ficheros virtual (VFS) traduce las peticiones que se realizan a peticiones a los gestores de EXT4 y EXT2 que son los que acaban accediendo a las particiones. Los gestores de EXT4 y EXT2 pueden trabajar concurrentemente. El gestor de EXT4 atiende las peticiones de forma secuencial, mientras que el gestor de EXT2 puede procesar más de una petición a la vez (es multiprogramado). Los gestores de EXT4 y EXT2 solamente son accesibles a través del gestor de VFS.

a) ¿Qué estructuras de datos se necesitan para poder enviar peticiones al gestor de VFS?

b) ¿Dentro de qué función(es) se le envían peticiones al gestor de VFS?

c) ¿Qué estructuras de datos se necesitan para poder enviar peticiones entre el gestor VFS y los gestores EXT4 y EXT2?

d) ¿Cuántos semáforos se necesitan para las peticiones entre el gestor de VFS y los gestores de EXT4 y EXT2?

e) Escribe el pseudo-código del gestor de EXT4

Nombre:
DNI:

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.
- **i*nt 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.
- ***int copy_data(void * src, void * dest, int size):*** Copia size bytes desde la posición de memoria src a la posición dest.
- ***void * getAddrFaulted():*** Retorna la dirección lógica que ha producido la última excepción de fallo de página.
- ***int validAddr (struct task_struct *proces, void *address):*** Retorna 1 si la dirección está contenida dentro del espacio de direcciones del proceso.