SOA/SO2

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:

a) 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.

b) La estructura TSS solamente se modifica en el BOOT del sistema para apuntar a la cima de la pila del proceso *Init*.

c) El *wrapper* de una llamada al sistema tiene como finalidad asegurar la portabilidad del código.

d) La IDT es la estructura hardware en la que se almacena el código de las llamadas al sistema.

e) Durante la generación de un ejecutable, todas las direcciones que se crean son lógicas ya que indican la dirección, dentro del fichero ejecutable, donde se pueden encontrar las variables y el código.

f) La MMU es un mecanismo hardware cuyo único propósito es traducir direcciones lógicas a físicas.

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

h) El número de frames disponibles en un ordenador depende, exclusivamente, del tamaño de la tabla de páginas de un proceso.

i) El objectivo de tener sistemas operativos multiprogramados es solapar el tiempo de E/S de unos procesos con la ejecución de otros procesos.

j) En un sistema operativo multiprogramado se reserva espacio para tantas pilas de sistema como procesos estén en estado de RUN.

Comentado [JJC1]: Esta puede ser cierta y falsa, no? En Zeos usamos 1 pila para cada proceso, pero podriamos usar una unica para todos, no?

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

Nombre:
DNI:

l) La página 1 contiene la dirección de memoria 1234.

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

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

o) El sector es la unidad mínima de trabajo dentro del sistema de ficheros.

p) Dado el siguiente código de usuario:

```
1:  void th(void) {
2:      sem_signal(1);
3:      exit();
4:  }
5:  int main() {
6:      sem_init(1, 0);
7:      ...
8:      int p = clone(th, pila1);
9:      if (p>0) clone(th, pila2);
10:     sem_wait(1);
11:     sem_wait(1);
12:     ...
13: }
```

Nombre:
DNI:

Se puede dar el caso que el proceso que ejecuta el *main* se bloquee en la linea 11 pero no en la 10.

Sí, en el caso que el padre hiciera un cambio de contexto entre la linea 8 y la 9 (justo despues de crear el 1r thread), se ejecutara primero thread y despues el otro.

q) El siguiente código de usuario usa el *malloc* de Doug Lea visto en clase:

```
int main() {
        char * p = malloc(16);
        read(0, p, 20);
}
```

Al ejectuarse generará una excepción de fallo de página.

NO

r) La estructura *file_operations* de Linux contiene información de uso de los ficheros.

s) El gestor de memoria virtual retorna error cuando no may más *frames* disponibles.

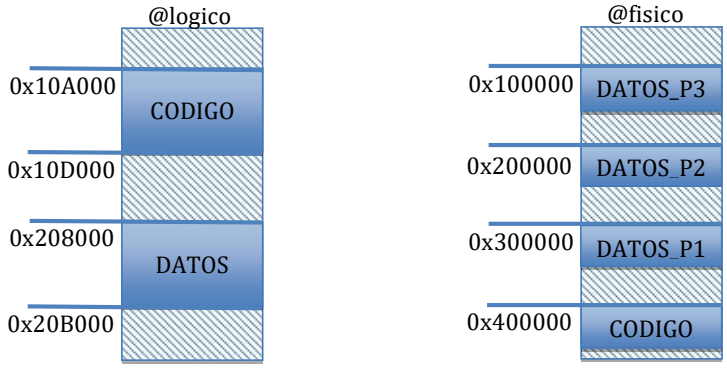
t) El coste de hacer el splitting del Doug Lea malloc al hacer una reserva de memoria es lineal con el número de bloques libres disponibles.

constante, dado un bloque se divide en 2 trozos, el que deseamos y el nuevo libre, y el nuevo libre se coloca en la posición del vector que le toque.

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) y en un momento determinado encontramos el siguiente contenido en el espacio de direcciones físico correspondiente a 3 procesos distintos:



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

4KB = 0x1000
que es representa amb 12bits
32-12 = 20
2^20 == 1024*1024 entradas (1048576)

b) Indica el contenido de la tabla de páginas (posición->valor) para mapear la zona de código del proceso 2 en este espacio de direcciones.

0x10A -> 0x400 (codigo)
0x10B -> 0x401 (codigo)
0x10C -> 0x402 (código)

c) Indica el código necesario para mapear la zona física de datos del proceso 3 en alguna zona libre del espacio lógico del proceso 2. NOTA: Puedes usar las rutinas del final del ejercicio que creas convenientes.

Por ejemplo en la 0x200000
set_ss_page(0x200, 0x100)
set_ss_page(0x201, 0x101)
set_ss_page(0x202, 0x102)

d) Completa el código siguiente para copiar la zona de datos del proceso actual a la zona de datos del proceso 3.

copy_data(0x208000, 0x200000, 3*4096)

e) ¿Cuantos fallos de TLB generará el código del apartado anterior como mínimo?

3 fallos de las 3 paginas a copiar, puede haber más si tenemos en cuenta las páginas de código

3. (3 puntos) Sistemas de ficheros

Supón que en una maquina de 32bits tienes un sistema operativo como ZeOS y un fabricante muy conocido te deja un dispositivo de almacenamiento por bloques. Este fabricante te implementa las siguientes rutinas para acceder a este dispositivo:

int put (void *logical_adress, int block): Guarda 4096 bytes a partir de la dirección pasada como parámetro en el bloque del dispositivo pasado como segundo parámetro. El bloque es un identificador útil para su posterior recuperación.

int get (int block, void *logical_address): Recupera el bloque del dispositivo identificado por el parámetro *block* y copia su contenido (4096 bytes) en la dirección lógica pasada como segundo parámetro.

Tienes que diseñar un sistema de ficheros que utilice este dispositivo. Este sistema de ficheros debe poder crear ficheros de diferentes tamaños (aunque no muy grandes), modificarlos y borrarlos, así como acceder de forma secuencial y/o directa a cualquier posición de estos ficheros de forma eficiente (minimizando el número de accesos al dispositivo). También debe poder organizar los ficheros en una estructura en grafo mediante directorios que pueden contener ficheros y/o otros directorios. Para ello decides que lo mejor es usar un **sistema de ficheros indexado**. Para cada fichero tendrás, de forma separada, (1) sus bloques de datos y (2) una estructura, que llamaremos *inodo,* para gestionar estos bloques (contiene todos los índices directos de los bloques de datos del fichero).

SOA/SO2

Nombre:
DNI:

Los procesos de usuario pueden usar la llamada a sistema *open(nombre_de_fichero)* para abrir un fichero previamente creado (o crearlo si no existia), y a continuación usar las llamadas a sistema *read* o *write* para leer o escribir el contenido del fichero. Finalmente pueden usar la llamada a sistema *close* cuando quieran dejar de usar el fichero y borrarlo con la llamada a sistema *unlink(nombre_de_fichero)*.

a) ¿Qué información mínima debes guardar en el inodo para gestionar cada fichero?

- tamaño del fichero
- puntero a los bloques de datos

b) ¿Como guardas esta información en el dispositivo? ¿Cual es el tamaño de tu inodo?

Uso un bloque del dispositivo entero (4096bytes), y lo guardo con un put

c) Si solo usáramos un inodo por fichero ¿Cual es el tamaño máximo de un fichero en este sistema?

4096 – size --> 4092bytes disponibles para punteros
4092/4 --> 1023 bloques
1023*4096 --> 4190208 Bytes-> 4092KiB

Nombre:
DNI:

d) ¿Qué información debes guardar dentro del directorio?

- nombre fichero
- num inodo

e) ¿Como guardas esta información en el dispositivo?

Un directorio es un fichero, y por lo tanto, lo guardo como un fichero, con un inodo y tantos bloques como necesite.

f) El tamaño máximo de un directorio en este sistema ¿es el mismo que el de un fichero?

Sí

g) ¿Todos los directorios son iguales o necesitas tener alguno con alguna restricción especial?

Son todos iguales, a excepción del directorio raiz (/), en el que su padre es él mismo, y tiene una posición conocida.

h) Cuando un proceso esté trabajando con un fichero ¿Qué información es indispensable tener actualizada en memoria?

La posición dentro del fichero

i) Supón el caso en que un usuario quiere escribir 4 bytes (¨hola¨) en un fichero que ya habia creado anteriormente y que esta escritura solo afecta al bloque de datos número 42, indica el pseudocódigo necesario de la rutina dependiente para implementarlo:

```
sys_write_fs( int pos, char *buff, int size) {
        b = pos/4096; // calcular bloc
        char tmp[4096];
        get (b, tmp);   // llegir bloc actual
        i = pos %4096;          //desplaçament dins bloc
        for(int j = 0; j < size; j++)
                tmp[i+j] = buff[j]; // suposant que nomes afecta 1 bloc
        write(b, tmp);

}
```

Nombre:
DNI:

4. (2 puntos) Entrada/Salida

Dada la llamada a sistema síncrona *write* de ZeOS queremos modificarla para que sea **asíncrona**. Para ello tienes que añadir un gestor.

a) ¿Qué estructuras de datos nuevas necesitas?

cola peticiones pendientes (IORBS)
cola peticiones finalizadas (IOFIN)

Un nuevo PCB para el gestor

b) Completa el pseudocódigo del gestor, indicando si hay alguna parte bloqueante:

```
for(;;) {
        X= getRequest(); // BLOQUEANTE

        R = do I/O (X)

        sendResult(X, R)

}
```

c) ¿Cómo quedaría el pseudocódigo de la llamada dependiente de escritura?

Igual, no hay que tocarlo, será llamada por el gestor

d) ¿Seria necesario añadir alguna llamada a sistema adicional?

Sí, una nueva llamada para esperar el resultado

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.