SOA2 (27/05/2024)

Nombre:
DNI:

Segundo control de teoría

Responde brevemente a las siguientes preguntas justificando tus respuestas. **Una respuesta sin justificar no será dada como válida.**

1. (4 puntos) Threads

Queremos modificar la implementación de threads de Linux (o ZeOS) vista en clase para que, en vez de que el task_struct identifique un thread y el proceso sea abstracto, tener una estructura donde guardaremos los datos del proceso (llamada EPROCESS) y otra donde guardaremos los datos del thread (llamada ETHREAD).
Recuerda que la implementación de sys_pthread_create tiene 3 parámetros: la dirección del wrapper de usuario, la función del thread y el parámetro de la función del thread. Dentro de sys_pthread_create es donde se allocatan los recursos para el thread, entre ellos, la pila de usuario (siempre de 4KBs).

a) Indica que campos, de los vistos en clase, tienen que ir en el EPROCESS

b) Indica qué campos, de los vistos en clase y en el laboratorio, tienen que ir en el ETHREAD.

También modificamos la creación del thread para que la asignación de la pila de usuario se haga en una función llamada ret_from_thread. Ret_from_thread se ejecutará solamente una vez cuando se ejecute por primera vez el nuevo thread creado (es la misma idea que ret_from_fork).

c) Indica qué cambios se tienen que hacer el contexto de ejecución del thread dentro de sys_pthread_create para ejecutar ret_from_thread.

d) Indica qué información se tiene que guardar dentro de la estructura ETHREAD que se necesite en ret_from_thread

e) Escribe el código de la función ret_from_thread.

f) Esta nueva implementación, ¿plantea algún problema respecto a la original?

Por último, queremos evitar tener un wrapper de la función del thread. Si recuerdas la estructura de la pila de usuario que se monta en sys_pthread_create, sabemos que si no se invoca la llamada al sistema pthread_exit desde dentro de la función del thread, se producirá una excepción de page_fault ya que se intentará ejecutar la instrucción que está en la dirección lógica 0.
Para este apartado, puedes suponer que current() devuelve la dirección del task_union del thread.

g) Escribe el código de la excepción page_fault teniendo en cuenta que tiene que ser capaz de detectar este caso para saber que se ha acabado un thread pero también tiene que ser capaz de detectar si el page_fault viene por un acceso incorrecto a memoria.

Nombre:
DNI:

2. (2 puntos) Sistema de ficheros en disco

Estamos implementando un driver para poder interpretar múltiples sistemas de ficheros en disco. Para esto, la única función que nos ofrece la buffer cache del sistema para poder acceder a disco es:

char *GetBlock(unsigned long BlockID);

que dado un identificador de bloque de disco, *BlockID*, nos devuelve un puntero a la zona de memoria donde ha almacenado el contenido de ese bloque de disco.

Los identificadores de bloque son de 4 bytes, el tamaño de bloque de disco es BLOCK_SIZE. A no ser que se diga lo contrario en el enunciado, puedes suponer que dentro de un Inodo existe un campo llamado FirstBlockID que contiene el identificador del primer bloque de datos de ese fichero y un campo FileSize, también de 4 bytes, que contiene el tamaño, en bytes, del fichero.

a) Queremos implementar la siguiente función:

unsigned int GetBlockID(struct i_node * inode, unsigned long long offset);

que dado un fichero identificado por su Inodo (parámetro inode) y un offset dentro del fichero, nos devuelve el identificador del bloque que contiene ese offset. Para los siguientes apartados no hace falta incluir el código de comprobación de errores. Puedes suponer que el offset siempre estará entre 0 y FileSize.

a.1) Escribe el código de la función GetBlockID si la asignación de bloques a ficheros es una asignación contigúa.

a.2) Escribe el código de la función si la asignación de bloques a ficheros es una asignación encadenada (sin tabla).

b) Suponiendo que ahora implementamos una asignación encadenada en tabla.

b.1) Escribe en C la declaración de la variable global FAT en la que se guardará en memoria una copia de la FAT de disco suponiendo que hay TOTAL_BLOCKS bloques de disco.

b.2) Escribe el código de la función load_FAT que carga una FAT de disco dentro de la variable declarada en el apartado anterior, suponiendo que esta FAT siempre se encuentra a partir del bloque 35 y el disco tiene TOTAL_BLOCKS bloques de disco para datos de usuario.

3. (4 puntos) Gestores

El señor Bakabaka ha salido de su clandestinidad para intentar que ZeOS vuelva a brillar como el mejor sistema operativo del mundo con una nueva característica. En este caso, ha implementado un sistema de ficheros virtual, con gestor monohilo, y con un dispositivo de eco (sin gestor). Este dispositivo sirve para que todo lo que escriba un proceso en este dispositivo puede ser leído por ese proceso, pero por ningún otro proceso. Varios procesos pueden utilizar este dispositivo a la vez sin que se mezclen sus datos. También hay otros dispositivos en el sistema que solo el señor Bakabaka entiende.

Nombre:
DNI:

La secuencia de funciones que se ejecutan hasta llegar al driver de ese dispositivo es:

sys_read -> read_vfs -> gestor VFS -> read_eco

Cuando se hace un open del dispositivo, se allocata si no tenía ya uno, para ese proceso, un buffer circular infinito (es el señor Bakabaka quien ha implementado esto, él es capaz de hacer estas cosas) para guardar los datos que escriba ese proceso.

a) Indica cuantas colas de iorbs e iofin se necesitan en esta implementación.

b) Indica cuantos semáforos se necesitan en esta implementación

c) Escribe el código de read_vfs.

d) Escribe el código del gestor VFS

Open_eco es la función que allocata un buffer para el proceso en el caso de que no tenga ninguno. Para esto, existe una función llamada GetCircularBufferFromTaskStruct que dado el task_struct * de un proceso, devuelve un puntero al buffer circular de ese proceso o -1 en el caso de que no tenga ninguno. El pseudocódigo de open_eco, sin comprobación de errores, podría ser:

```
int open_eco(struct IORB* iorb)
{
        CircularBuffer *circularBuffer;

        circularBuffer = GetCircularBufferFromTaskStruct(current());
        if (circularBuffer == (void*)-1)
                circularBuffer = AllocateCircularBufferFromTaskStruct(current());
        // Data initilization
        …
        return 0;
}
```

e) Cuando el señor Bakababa prueba el código, se da cuenta de que solamente se crea un buffer compartido por todos los procesos. ¿a qué se debe esto?

f) ¿Cómo solucionarías el problema anterior?