Sistema Operativos Practica 5

Ejercicio 1

¿Cuáles de las siguientes opciones describen el concepto de driver? Seleccione las correctas y justifique.

a) Es una pieza de software

Si se implementa por sw (es un .c)

b) Es una pieza de hardware.

No es una pieza de hw, es la interfaz para una pieza de hw.

c) Es parte del SO.

Si, es parte del SO. Es el encargado de que puedan hablar dispositivos externos con el sw.

d) Dado que el usuario puede cambiarlo, es una aplicación de usuario.

No es una aplicación de usuario, corre a nivel kernel.

e) Es un gestor de interrupciones.

No. A veces se vale de las interrupciones para implementar drivers.

f) Tiene conocimiento del dispositivo que controla pero no del SO en el que corre.

Debe conocerlo porque cada SO tiene una API distinta.

g) Tiene conocimiento del SO en el que corre y del tipo de dispositivo que controla, pero no de las particularidades del modelo específico.

Si cononce las particularidades del modelo específico. La idea es poder comunicar al SO con cualquier dispositivo. (cada grafica distinta por ejemplo tiene su propio driver)

Ejercicio 2

Un cronómetro posee 2 registros de E/S:

- CHRONO_CURRENT_TIME que permite leer el tiempo medido,
- CHRONO_CTRL que permite ordenar al dispositivo que reinicie el contador.

El cronómetro reinicia su contador escribiendo la constante CHRONO_RESET en el registro de control. Escribir un driver para manejar este cronómetro. Este driver debe devolver el tiempo actual cuando invoca la operación read(). Si el usuario invoca la operación write(), el cronómetro debe reiniciarse.

Para el código voy a usar la API que hay al final de la práctica. Estimo que tengo que tener cuidado con las race condition del cronometro así que voy a usar mutex.

```
sema mutex;
int driver_init(){
    sema_init(&mutex,1);
}

int driver_read(int *data){
    sema_wait(&mutex);
    int valor = IN(CHRONO_CURRENT_TIME);
    copy_to_user(data,&valor,sizeof(data));
    sema_wait(&mutex);
    return IO_OK;
}

int driver_write(int *data){
    sema_wait(&mutex);
    OUT(CHRONO_CTRL,CHRONO_RESET);
    sema_signal(&mutex);
    return IO_OK
}
```

Ejercicio 3

Una tecla posee un único registro de E/S : BTN_STATUS. Solo el bit menos significativo y el segundo bit menos significativo son de interés:

- BTN_STATUS0: vale 0 si la tecla no fue pulsada, 1 si fue pulsada.
- BTN_STATUS1: escribir 0 en este bit para limpiar la memoria de la tecla.

Escribir un driver para manejar este dispositivo de E/S. El driver debe retornar la constante BTN_PRESSED cuando se presiona la tecla. Usar busy waiting.

```
sema mutex;
int driver_init(){
    sema_init(&mutex,1);
    return IO_OK
}

int driver_read(int *data){
    sema_wait(&mutex);
    while(IN(BTN_STATUS) & 1 != 1); // esperar mientras no haya sido pulsada
    OUT(BTN_STATUS, ALGUNA MASCARA PARA ESCRIBIR 0);
    sema_signal(&mutex);
    int valor = BTN_PRESSED;
    copy_to_user(data,&valor,sizeof(data));
    return IO_OK;
}
```

Ejercicio 4

Reescribir el driver del ejercicio anterior para que utilice interrupciones en lugar de busy waiting. Para ello, aprovechar que la tecla ha sido conectada a la línea de interrupción número 7. Para indicar al dispositivo que debe efectuar una nueva interrupción al detectar una nueva pulsación de la tecla, debe guardar la constante BTN_INT en el registro de la tecla. Ayuda: usar semáforos.

```
sema mutex;
sema int_7

void pressed_handler(){
    sema_signal(&int_7)
}
int driver_init(){
    sema_init(&mutex,1);
    sema_init(&int_7,0);
    request_irq(7,pressed_handler)
}

int driver_read(*data){
    sema_wait(&mutex);
    sema_wait(&int_7);

    OUT(BTN_STATUS,BTN_INT);
    sema_signal(&mutex);
    int valor = BTN_PRESSED;
    copy_to_user(data,&valor,sizeof(data));
    return IO_OK
}
```

Ejercicio 6

¿Cuál debería ser el nivel de acceso para las syscalls IN y OUT? ¿Por qué?

Deben ser de nivel de kernel. Los usuarios no tienen que poder leer/escribir cualquier registro. Por eso los drivers corren a nivel de kernel.

Ejercicio 7

no voy a copiar la consigna, es el de los discos.

a

Implementar la función write(int sector, void *data) del driver, que escriba los datos apuntados por data en el sector en formato LBA indicado por sector. Para esta primera implementación, no usar interrupciones.

```
int driver_write(int sector, void *data){
    sema_wait(&mutex);
    OUT(DOR_IO,1); //prendemos el bicho
    while(!IN(DOR_STATUS)) // esperamos a que se prenda
    valor;
    copy_from_user(&valor,data, sizeof(data));
    sleep(50); // hay q esperar 50ms antes de cualquier operacion despues de prendido
    // segun mi amigo gpt asi se calcula la pista y sector:
    int pista = sector / cantidad_de_sectores_por_pista();
    int s = sector % cantidad_de_sectores_por_pista();
    OUT(ARM,pista);
    OUT(SEEK_SECTOR,s);
    while(!IN(ARM_STATUS)); //esperamos que el brazo este ubicado
    escribir_datos(&valor);
    while(!IN(DATA_READY)); //esperamos a que termine de escribir el dato
    IN(DOR_IO,0); //apagamos el bicho
    sleep(200); //apagar puede demorar 200ms lo hacemos esperar
    sema_signal(&mutex);
    return IO_OK;
}
```

b)

Modificar la función del inciso anterior utilizando interrupciones. La controladora del disco realiza una interrupción en el IRQ 6 cada vez que los registros ARM_STATUS o DATA_READY toman el valor 1. Además, el sistema ofrece un timer que realiza una interrupción en el IRQ 7 una vez cada 50 ms. Para este inciso, no se puede utilizar la función sleep.

```
int looking;
int writing;
sema mutex;
sema timer;
sema arm;
sema finished;
int ciclos;
```

```
void arm_handler(){
    if(looking){
        sema_signal(&arm);
    }
    if(writing){
        sema_signal(&finished);
    }
}

void timer_handler(){
    if(ciclos > 0){
        if(ciclos == 1){
            sema_signal(&timer)
        } else{
            ciclos --;
        }
    }
}

int driver_init(){
    sema_init(&mutex,1);
    //inicialzar a todo el resto de semaforos en 0...
    looking = 0;
    writing = 0;
    request_irq(6,arm_handler);
    request_irq(7,timer_handler);
}

int driver_write(int sector, void *data){
    sema_wait(&mutex);
    valor;
    copy_from_user(&valor,data,sizeof(data));
    OUT(DOR_IO,1);
    while(!IN(DOR_STATUS)) // esperamos a que se prenda
    ciclos = 2; //hay que esperar 50 ms con un ciclo del timer no alcanza si lo enganchamos
    sema_wait(&timer); // esperamos la interrupcion
    int pista = sector / cantidad_de_sectores_por_pista();
    int s = sector % cantidad_de_sectores_por_pista();
    OUT(ARM,pista);
    OUT(SEEK_SECTOR,s);
    looking = 1; //indico que estoy acomodando el brazo
    sema_wait(&arm); //espero la int de que se acomodo
    escribir_dato(valor);
    writing = 1; //indico que estoy escribiendo
    sema_wait(&finished); // espero la int que indica que termino
    IN(DOR_IO,0) // apago el bicho
```

```
    ciclos = 5;
    sema_wait(&timer) // espero 5 vueltas del timer (entre 200 y 250 ms)
    sema_signal(&mutex);
    return IO_OK;
}
```