Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 1

Una compañía de accesorios informáticos ha creado un ratón y su correspondiente driver para sistema operativo básico (como el que está siendo presentado en pseudocódigo en la asignatura) cuya funcionalidad definida por su interfaz al usuario es:

- Funciones *open/close*: Para establecer el acceso al ratón o liberarlo
- Función *read*: Para obtener la posición actual del ratón.

Se ha fabricado una nueva versión del ratón que permite configurar la precisión del mismo indicando la distancia entre posiciones consecutivas. Por tanto, resulta necesario modificar el driver del ratón para añadir dicha funcionalidad.

Para realizar esto disponemos de la función:

```
Modificar_precision (int valor);
```

Esta función configura el hardware del ratón con el nuevo valor de la precisión:

Se pide:

a) Indicar que estructuras de datos hay que añadir o modificar y que funciones (interfaz y pseudocódigo) hay que añadir en el *kernel* y fuera de él para añadir una llamada al sistema que permita esta nueva funcionalidad.

b) Realizar el apartado anterior pero esta vez sin añadir nuevas llamadas al sistema.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Para añadir una nueva llamada al sistema es necesario:
- Utilizar una entrada libre en la tabla de llamadas al sistema
  - posición de la tabla: código de llamada
  - valor de la tabla puntero a la función de gestión de la llamada.
- Una función para gestionar el evento de la llamada al sistema.
- Una función de biblioteca que solicite la llamada al sistema.

Función de biblioteca Modificar_precision (valor):

- Guardar en REG_2 el valor
- Guardar en REG_1 el código de llamada
- Generar interrupción de llamadas al sistema
- Devolver estado de la operación contenido en REG_3;

LLamada al sistema Modificar_precision ():

- Valor = REG_2
- Res = Modificar_precision(valor);
- REG_3 = Res.

b) Hay dos posibilidades:

I. Utilizar la llamada *write* en desuso:

LLamada al sistema_Write (desc: R1, buffer: R2, size: R3)

- Valor = buffer
- Res = Modificar_precision(int valor);
- Devolver (Res);

II. Utilizar la llamada *ioctl*

LLamada al sistema_Ioctl (desc: R1, cod_operacion: R2, punt_parametros: R3)

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 2

Se dispone de un computador que dispone de una pantalla táctil como dispositivo hardware que realiza las siguientes tareas:

- Muestra continuamente lo que está escrito en la memoria de video, una zona de memoria especial dedicada solo a almacenar la información de pantalla.
- Envía una interrupción cada vez que se pulsa sobre ella y devuelve la posición de la pulsación en los registros de E/S.

El sistema operativo diseñado para esta máquina contempla, al menos, la siguiente funcionalidad:

- Modificar la información que se muestra en pantalla.
- Obtener las pulsaciones realizadas (dando su posición).

Se pide:

a) Obtener cuales son los eventos involucrados (explicando brevemente su función) y las estructuras de datos necesarias para implementar la funcionalidad requerida.
b) Codifique las funciones para manejar los eventos involucrados.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Los eventos involucrados son:
- Interrupción del teclado.
  - Se encarga de almacenar las pulsaciones en el buffer de teclado y de despertar a los procesos bloqueados en la lista asociada al teclado (quitándolo de la lista de dormidos y lo incluye en la de listos)
- Llamada al sistema para leer el teclado.
  - Si el buffer está vacío bloquea el proceso en la lista del buffer de teclado, si no devuelve una tecla.
- Llamada al sistema para modificar datos de pantalla.
  - Se modifica la memoria de video y termina.

Estructuras de datos involucradas:
- Datos propios del teclado, que incluye:
  - Lista de pulsaciones almacenadas (TECLADO-> Buffer)
  - Lista de procesos bloqueados (TECLADO ->lista_procesos_bloqueados)
- Datos propios de la pantalla que incluye:
  - Buffer de memoria de video.

b)

Manejador_interrupción_teclado ():
- Insertar_Pulsacion(pulsación, TECLADO-> Buffer);
- Insertar_Interrupcion_Software(Despertar_bloqueados_teclado);,
- Generar_Interrupcion_Software();

Despertar_bloqueados_teclado ():
- Proc = PrimerProceso (TECLADO ->lista_procesos_bloqueados);
- If (Proc != NULL)
  - Proc = BorrarProceso (TECLADO ->lista_procesos_bloqueados, Proc);
  - Proc->estado = LISTO;
  - InsertarProceso(lista_procesos_listos, Proc);

LLamada_al_Sistema_Leer_Teclado():
- Si (estaVacio(TECLADO->Buffer))
  - Insertar_proceso_actual(TECLADO ->lista_procesos_bloqueados);
  - procesoActual->estado = BLOQUEADO; // cambiar el estado
  - procesoAnterior = procesoActual;
  - procesoActual=planificador() ; // Obtener BCP de la lista de listos.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 3

Se dispone de un computador con los siguientes dispositivos hardware:

- Un disco de almacenamiento secundario que recibe peticiones mediante los registros de E/S y que lanza una interrupción al finalizar cada petición.
- Un teclado que envía una interrupción cada vez que se pulsa una tecla y devuelve cual es en los registros de E/S.

El sistema operativo multiproceso diseñado para esta máquina contempla, al menos, los siguientes eventos:

- Interrupción del disco
- Interrupción del teclado
- Llamada al sistema para leer el teclado.
- Llamada al sistema para realizar peticiones al disco.

Se pide:

a) Diseñar las modificaciones del sistema operativo, indicando las estructuras necesarias, las funciones requeridas (interfaz y pseudocódigo) y los eventos involucrados.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Los eventos involucrados son:
- Interrupción del disco
- Interrupción del teclado
- Llamada al sistema para leer el teclado.
- Llamada al sistema para realizar peticiones al disco.

Las variables globales necesarias son:
- Datos internos del teclado (estructura TECLADO1) compuestos por:
  - Lista de teclas almacenadas (TECLADO1.buffer_teclado)
  - Lista de procesos bloqueados (TECLADO1.lista_procesos_bloqueados)

- Datos internos del disco (estructura DISCO1) compuesto por:
  - Lista de peticiones de bloques de disco (DISCO1->lista_peticiones_disco), donde cada petición contiene:
    - Identificador del bloque de datos (petición->bloque_id)
    - Datos del buffer (petición->bloqueDatos)
    - Tipo de petición (petición->tipo: READ/WRITE)
    - Lista de procesos bloqueados (petición->lista_procesos_bloqueados)

El pseudocódigo de las funciones involucradas es:

Manejador_interrupción_disco ():
- Si (peticionActual->tipo == READ)
  - CopiarDeDisco(peticionActual-> bloqueDatos);
- Insertar_Interrupcion_Software(Despertar_bloqueados_disco, peticionActual->bloque_id);
- Generar_Interrupcion_Software();

Despertar_bloqueados_disco (bloque_id):
- Mientras (NoVacia(peticionActual->lista_procesos_bloqueados))
  - Proc=ExtraerPrimero (peticionActual->lista_procesos_bloqueados);
  - Proc->estado=LISTO;
  - Insertar(lista_procesos_listos,Proc);
- peticionAnterior = peticionActual ;
- peticionActual = pedirSiguientePetición(peticionAnterior, DISCO1->lista_peticiones_disco);
- BorrarPetición(peticionAnterior, DISCO1->lista_peticiones_disco);

Grupo: …… NIA: …… Nombre y apellidos: ……..

Despertar_bloqueados_teclado ():

- Proc = ExtraerPrimero (TECLADO1->lista_procesos_bloqueados);
- Si Proc != NULL
  - Proc->estado = LISTO;
  - Insertar(lista_procesos_listos, Proc);

Petición_Bloque_Disco (bloque_id, operacion, buffer):

- petición = BuscarPetición(DISCO1->lista_peticiones_disco, bloque_id);
- SI (petición == NULL)
  - petición = crearPetición(DISCO1->lista_peticiones_disco, bloque_id, operacion);
  - Insertar(DISCO1->lista_peticiones_disco, petición) ;
- Insertar(petición-> lista_procesos_bloqueados, procesoActual);
- si (operacion == WRITE)
  - Copiar buffer a petición->bloqueDatos
- Si (DISCO1.estado == libre)
  - Poner petición como peticiónActual y pedirla al disco
- procesoActual->estado = BLOQUEADO;
- procesoAnterior = procesoActual;
- procesoActual = planificador();
- procesoActual->estado = EJECUTANDO;
- cambioContexto(procesoAnterior->ctxt, procesoActual->ctxt) ;
- si (operacion == READ)
  - Copiar petición->bloqueDatos a buffer

LLamada_al_Sistema_Read_Disco(descriptor, buffer, tamaño):

- listaBloques = bloquesAsociados(posicionLE[descriptor], tamaño)
- para cada bloque i en listaBloques
  - Petición_Bloque_Disco(listaBloques[i], READ, buffer+BLOCK_SIZE*i)

LLamada_al_Sistema_Write_Disco(descriptor, buffer, tamaño):

- listaBloques = bloquesAsociados(posicionLE[descriptor], tamaño)
- para cada bloque i en listaBloques
  - Petición_Bloque_Disco(listaBloques[i], WRITE, buffer+BLOCK_SIZE*i)

LLamada_al_Sistema_Leer_Teclado():

- Si (estaVacio(TECLADO1->buffer_teclado))
  - Insertar_proceso_actual(TECLADO1->lista_procesos_bloqueados);

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 4

Disponemos de una maquina monoprocesador y queremos implementar un driver de teclado para un sistema operativo básico con un *kernel* monolítico no expulsivo con la siguiente funcionalidad:

- Gestionar las interrupciones del teclado al pulsar una tecla.
- Ofrecer a los procesos usuario la forma de obtener las teclas pulsadas (bloqueándose si no hay ninguna).
- Que el driver pueda ser cargado y descargado en tiempo de ejecución.

El driver debe almacenar las teclas de forma temporal mientras que ningún proceso las solicita.

Se pide:

a) Diseñar un interfaz tanto interno (*kernel*) como externo, llamadas al sistema para las funciones que requiere el driver.

b) Definir las estructuras de datos necesarias para realizar la funcionalidad requerida.

c) Implementar en pseudocódigo la funcionalidad para obtener las teclas del teclado y para enviarlas a los procesos usuarios. ¿En qué eventos debe incluirse?

d) ¿Qué cambios hay que realizar si el *kernel* del sistema operativo fuera expulsivo? ¿Y si la computadora tuviera dos procesadores y el *kernel* fuera SMP?

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestionar la interrupción de teclado.
  - **Manejador_Interrupcion_teclado;**
- No tiene ningún parámetro
  Gestionar las llamadas al sistemas del driver
  - Mínimo, leer una tecla
  - Una opción es utilizar el estándar UNIX.
    - **Desc = Open (nombre_teclado, flags)**
      - Permite reservar el acceso al teclado
    - **Res = Close (Desc)**
      - libera la reservar para el acceso al teclado
    - **Res = Read (Desc, buffer, size)**
- Devuelve en *buffer* la tecla leída
  Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_teclado;**
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
  - **Descargar_driver_teclado;**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del kernel.

b) Las estructuras de datos requeridas son:
- Los datos del driver de teclado en una estructura, con los siguientes campos:
  - Lista de teclas almacenadas (KB1.BufferTeclas)
  - Lista de procesos bloqueados (KB1.Bloqueados)

c) Los eventos involucrados en la lectura de teclas son:
- Interrupción de teclado
- Llamada al sistema read (para leer una tecla)

Manejador de interrupción_teclado ():
- Insertar_tecla(tecla, KB1.BufferTeclas);
- Proc = ObtenerPrimerProceso (KB1.Bloqueados);

Grupo: …… NIA: …… Nombre y apellidos: ……..

Llamada al sistema read(fd, buffer, size):

- Si (estaVacio(KB1.BufferTeclas))
  - Insertar_proceso_actual(KB1.Bloqueados);
  - Cambiar el estado del proceso actual de ejecutando a bloqueado.
  - Guardar el proceso actual como proceso anterior.
  - Obtener el primer proceso de la lista de listos, como proceso actual.
  - Cambiar el estado del proceso actual a ejecutando.
  - Cambiar contexto entre proceso anterior y el proceso actual.
- buffer[0] = extraer_tecla(KB1.BufferTeclas);
- devolver 1;

d) Hay que evitar que los eventos que acceden a la lista puedan interferirse.
Las interrupciones hardware y las excepciones siempre interferirán entre si y con las llamadas al sistema e interrupciones software. Luego es necesario modificar el código anterior para añadir una interrupción software:

Manejador de interrupción_teclado ():

- Insertar_tecla(tecla, KB1.BufferTeclas);
- Insertar_Interrupcion_Software(Despertar_bloqueados_teclado);
- Generar_Interrupcion_Software();

Despertar_bloqueados_teclado ():

- Proc = ObtenerPrimerProceso (KB1.Bloqueados);
- Cambiar su estado de bloqueado a listo.
- Incluirlo en la lista de procesos listos para ejecutar al final.

Llamada al sistema read(fd: R1, buffer: R2, size: R3):

- Si (estaVacio(KB1.BufferTeclas))
  - Insertar_proceso_actual(KB1.Bloqueados);
  - Cambiar el estado del proceso actual de ejecutando a bloqueado.
  - Guardar el proceso actual como proceso anterior.
  - Obtener el primer proceso de la lista de listos, como proceso actual.
  - Cambiar el estado del proceso actual a ejecutando.
  - Cambiar contexto entre proceso anterior y el proceso actual.
- buffer[0] = extraer_tecla(KB1.BufferTeclas);
- devolver 1;

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 5

Disponemos de una maquina monoprocesador y se quiere implementar el driver para un modelo de impresora con escáner (con interrupciones pero sin DMA) para un sistema operativo básico con un *kernel* monolítico no expulsivo con la siguiente funcionalidad:

- El computador solicita a la impresora-escáner el escaneo de un documento recibiendo los datos escaneados en formato PDF.
- El computador solicita a la impresora-escáner la impresión de un documento enviando los datos a imprimir en formato PDF.
- El computador solicitar a la impresora-escáner que realice fotocopias, es decir, que escanee el documento y lo imprima a continuación sin mediación del computador.
- La impresora-escáner solo dispone de una interrupción asociada que sirve para notificar la finalización de alguna de las tres operaciones anteriores.

Se pide además que el driver se pueda cargar y descargar en tiempo de ejecución:

a) Diseñar el interfaz completo del driver (funciones internas del *kernel* y llamadas al sistema) para ofrecer la funcionalidad que requiere el driver. Justifique las razones para el diseño realizado.
b) Definir las estructuras de datos necesarias (nuevas o modificadas) del SSOO y del driver para realizar la funcionalidad requerida.
c) Implementar en pseudocódigo la funcionalidad requerida del driver. ¿En qué eventos debe incluirse?
d) Describa en pocas palabras que cambios habría que realizar si la impresora-escáner utilizara E/S con DMA.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestionar la interrupción de la impresora escáner.
  - **Manejador_Interrupcion_impresora_escaner;**
    - No tiene ningún parámetro)

- Gestionar las llamadas al sistema del driver.
  Una opción es utilizar el estándar UNIX:
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite reservar el acceso a la impresora-escáner
  - **Res = Close (Desc)**
    - libera la reservar para el acceso a la impresora-escáner
  - **Res = Read (Desc, buffer, size)**
    - Indica a la impresora-escáner que realice un escaneo y devuelve el resultado en PDF en *buffer*.
  - **Res = Write (Desc, buffer, size)**
    - Indica a la impresora-escáner que imprima los datos de buffer (en PDF).
  - **Res = Ioctl (Desc, operacion, punt_param)**
    - Si operación = OP_FOTOCOPIAR le dice a la impresora que fotocopie (no hay parámetros)
    - NOTA: SUPONEMOS QUE EL PROCESO NO SE BLOQUEA (si no hay que modificar el resto para permitir dicho bloqueo)

- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_impresora-escaner;**
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
  - **Descargar_driver_ impresora-escaner;**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del kernel.

b) Las estructuras de datos requeridas son:
- Estructura con los datos necesarios para el driver de impresora:
  - Peticiones a imprimir (PRINT.peticiones) y petición en curso (PRINT.pecu)
  - Lista de procesos bloqueados (PRINT.lista_bloqueados)
- Buffer de escáner: compuesto por:

Grupo: …… NIA: …… Nombre y apellidos: ……..

Manejador_interrupción_ impresora_escaner ():

- Obtener operación realizada y resultado de los registros de E/S
- SI operación == impresión.
  - Proc = ObtenerPrimerProceso (PRINT.lista_bloqueados);
  - Cambiar su estado de bloqueado a listo.
  - Incluirlo en la lista de procesos listos para ejecutar al final
  - SI hay más peticiones de impresión (PRINT.peticiones)
    - Actualizar la petición actual (PRINT.pecu)
    - Copiar datos del buffer de usuario al buffer de impresión
    - Enviar la orden de impresión con los registros de control.
- SI operación == escaneo.
  - Copiar datos del escáner al buffer de la petición actual (SCAN.pecu.buffer)
  - Proc = ObtenerPrimerProceso (SCAN.lista_bloqueados);
  - Cambiar su estado de bloqueado a listo.
  - Incluirlo en la lista de procesos listos para ejecutar al final.
  - SI hay más peticiones de escaneo (SCAN.peticiones)
    - Actualizar la petición actual (SCAN.pecu)
    - Enviar la orden de escaneo con los registros de control.

LLamada_al_Sistema_write():

- Crear una nueva petición de impresión (petición)
- Insertar la nueva petición en la lista de peticiones (PRINT.peticiones)
- Poner los datos a imprimir en la nueva petición (petición.buffer)
- SI el dispositivo está libre
  - Copiar datos del buffer de usuario al buffer de impresión
  - Enviar la orden de impresión con los registros de control.
- Insertar_proceso_actual(PRINT.lista_bloqueados);
- Cambiar el estado del proceso actual de ejecutando a bloqueado.
- Guardar como proceso anterior el proceso actual.
- Obtener el BCP del primer proceso de la lista de listos, como proceso actual.
- Cambiar el estado del proceso actual a ejecutando.
- Cambiar contexto entre proceso anterior y el proceso actual.

Grupo: …… NIA: …… Nombre y apellidos: ……..

LLamada_al_Sistema_read(fd: R1, buffer: R2, size: R3):

- Crear una nueva petición de escaneo (petición)
- Insertar la nueva petición en la lista de peticiones (SCAN.peticiones)
- SI el dispositivo está libre
  - Enviar la orden de escaneo con los registros de control.
- Insertar_proceso_actual(SCAN.lista_bloqueados);
- Cambiar el estado del proceso actual de ejecutando a bloqueado.
- Obtener el BCP del primer proceso de la lista de listos.
- Poner dicho BCP en la lista de ejecutando
- Cambiar contexto entre proceso actual y nuevo proceso ejecutando.
- Copiar datos de la petición (SCAN.pecu.buffer) al buffer de usuario.
- Devolver los bytes copiados al buffer de usuario.

LLamada_al_Sistema_ioctl():

- Si operación = OP_FOTOCOPIAR
  - Enviar la orden de fotocopiar con los registros de control.

d) En caso de utilizar DMA los datos no se envían o recogen por los registros de datos, en su lugar al preparar la operación en los registros de control se indica la dirección de memoria donde están los datos o donde se deben poner (según el caso).

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 6

Una compañía de accesorios informáticos Dispone de un catálogo de tarjetas de sonido y ha implementado un driver tipo UNIX genérico para todas ellas cuya funcionalidad (definida por su interfaz al usuario) es:

- Funciones *open* / *close*: Para establecer acceso exclusivo a la tarjeta de sonido o liberarla
- Función *write*: Para enviar la música a reproducir en formato *wav*

Se ha actualizado el catalogo, el cual presenta únicamente los siguientes modelos:

- TS100: Incluye la funcionalidad básica explicada antes.
- TS200: incluye la funcionalidad del TS100 e incorpora un micrófono para grabar música en formato *wav*.
- TS300: Incluye la funcionalidad del TS200 más la posibilidad de modificar el formato de reproducción y grabación entre *wav*, *mp3* y *ogg*.

La versión de la tarjeta correspondiente puede obtenerse consultando el hardware de la misma.
Se pide:

a) Diseñar un único driver genérico válido para las tres tarjetas anteriores que incluya:

   - Estructuras de datos que haya que añadir /modificar en el *kernel*.
   - Interfaz completo del driver (funciones internas del *kernel* y llamadas al sistema) y un pseudocódigo de las mismas explicando en que eventos se realizan.
   - Código de apoyo necesario (código externo al *kernel* del SSOO).

   Realizar este diseño sin incluir nuevas llamadas al sistema.

b) Modificar el diseño del apartado a) todo lo que sea necesario para realizar la misma funcionalidad pero esta vez incluyendo nuevas llamadas al sistema.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestionar la interrupción de la tarjeta de sonido.
  - **Manejador_Interrupcion_tarjeta_sonido;**
    - No tiene ningún parámetro)
- Gestionar las llamadas al sistemas del driver:
  Una opción es utilizar el estándar UNIX.
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite reservar el acceso a la tarjeta de sonido
  - **Res = Close (Desc)**
    - libera la reservar para el acceso a la tarjeta de sonido
  - **Res = Read (Desc, buffer, size)**
    - Indica a la tarjeta de sonido que grabe sonido y devuelva el resultado en *buffer*.
  - **Res = write (Desc, buffer, size)**
    - Indica a la tarjeta de sonido que reproduzca los datos de buffer.
  - **Res = Ioctl (Desc, operacion, punt_param)**
    - Si operación = OP_CAMBIAR_FORMATO le dice a la tarjeta de sonido que cambie el formato al indicado en punt_param

- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_ tarjeta_sonido;**
    - Consulta el modelo de tarjeta en los registros de HW y lo apunta en la variable TS_TYPE de la estructura del driver.
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
  - **Descargar_driver_ tarjeta_sonido;**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Las estructuras de datos requeridas son:

- Estructura de datos del driver. Incluye punteros al resto de variables del driver y a las funciones que lo componen.
- Variable TS_TYPE
- Buffer de reproducción: compuesto por:
  - Datos a reproducir.
  - Lista de procesos bloqueados (cada entrada un puntero a un BCP)
- Buffer de grabación: compuesto por:
  - Datos grabados.
  - Lista de procesos bloqueados (cada entrada un puntero a un BCP)

Los eventos involucrados son:

- Interrupción de tarjeta de sonido
- Llamada_al_sistema_write (para reproducir)
- Llamada_al_sistema_read (para grabar)
- Llamada_al_sistema_ioctl (para cambiar el formato)

Manejador_interrupción_ tarjeta_sonido ():

- Obtener operación realizada y resultado de los registros de E/S
- SI operación == reproducción.
  - Proc = ObtenerPrimerProceso (Buffer_reproducción.lista_bloqueados);
  - Cambiar su estado de bloqueado a listo.
  - Incluirlo en la lista de procesos listos para ejecutar al final
  - SI hay más peticiones de reproducción (Buffer_reproducción.peticiones)
    - Tomar siguiente orden de reproducción del Buffer_reproducción
    - Enviar esa orden de reproducción a los registros de control y de datos del dispositivo
- SI operación == grabación.
  - Copiar datos de grabación (regs. Datos) al buffer de grabación
  - Proc = ObtenerPrimerProceso (Buffer_grabación.lista_bloqueados);
  - Cambiar su estado de bloqueado a listo.
  - Incluirlo en la lista de procesos listos para ejecutar al final
  - SI hay más peticiones de grabación (Buffer_ grabación.peticiones)
    - Tomar siguiente orden de reproducción del Buffer_ grabación
    - Enviar esa orden de grabación a los registros de control y de datos

Grupo: …… NIA: …… Nombre y apellidos: ……..

LLamada_al_Sistema_write():

- Crear una nueva petición de reproducción (petición)
- Insertar la nueva petición en la lista de peticiones (Buffer_reproducción.peticiones)
- SI el dispositivo está libre
    - Enviar la orden de reproducción a los registros de control y de datos del dispositivo
- Insertar_proceso_actual(Buffer_reproducción.lista_bloqueados);
- Cambiar el estado del proceso actual de ejecutando a bloqueado.
- Obtener el BCP del primer proceso de la lista de listos.
- Poner dicho BCP en la lista de ejecutando
- Cambiar contexto entre proceso actual y nuevo proceso ejecutando.

LLamada_al_Sistema_read():

- Si (TS_TYPE = TS200) or (TS_TYPE = TS300)
    - Crear una nueva petición de grabación (petición)
    - Insertar la nueva petición en la lista de peticiones (Buffer_grabación.peticiones)
    - SI el dispositivo está libre
        - Enviar la orden de grabación a los registros de control y de datos del dispositivo
    - Insertar_proceso_actual(Buffer_grabación.lista_procesos_bloqueado)
    - Cambiar el estado del proceso actual de ejecutando a bloqueado.
    - Obtener el BCP del primer proceso de la lista de listos.
    - Poner dicho BCP en la lista de ejecutando
    - Cambiar contexto entre proceso actual y nuevo proceso ejecutando.
      /* Aquí se bloquea el proceso, hasta un cambio de contexto de vuelta */
    - Copiar datos del buffer de grabación al buffer de usuario

LLamada_al_Sistema_ioctl():

- Si (operación = OP_ CAMBIAR_FORMATO) y (TS_TYPE = TS300)
    - Si el dispositivo está libre
        - Enviar la orden de cambiar formato al indicado en los parámetros con los registros de control.
    - Sino
        - Rechazar petición

Grupo: …… NIA: …… Nombre y apellidos: ……..

b) En caso de insertar nuevas llamadas al sistemas hay que modificar lo siguiente:

- La función read se cambia por la función interna:
  Res = grabar (buffer, size)
- La función write se cambia por la función interna:
  Res = Reproduccir (buffer, size)
- La función ioctl se cambia por la función interna:
  Res = cambiar_formato (formato)

- Se incluye 3 nuevas entradas en la tabla de llamadas al sistema para cada uno de estas funciones.
- Se incluye tres nuevas funciones en la librería de acceso al *kernel* (que es externa al *kernel* que lanza las llamadas (se llamaran igual):

Funcion_de_biblioteca_grabar (buffer, size)

- Guardar en REG_2 el valor de buffer
- Guardar en REG_3 el valor de size
- Guardar en REG_1 el código de llamada (grabar)
- Generar interrupción de llamadas al sistema (trap)
- Devolver_estado(REG_9);

Funcion_de_biblioteca_Reproduccir (buffer, size)

- Guardar en REG_2 el valor de buffer
- Guardar en REG_3 el valor de size
- Guardar en REG_1 el código de llamada (reproducir)
- Generar interrupción de llamadas al sistema (trap)
- Devolver_estado(REG_9);

Funcion_de_biblioteca_cambiar_formato (formato)

- Guardar en REG_2 el formato
- Guardar en REG_1 el código de llamada (cambiar_formato)
- Generar interrupción de llamadas al sistema (trap)

Grupo: …… NIA: …… Nombre y apellidos: ……..

LLamada al sistema_Grabar ()

- buffer = REG_2
- comprobar memoria correcta en buffer
- size = REG_3
- Res = grabar(buffer, size);
- REG_9 = Res.

LLamada al sistema_Reproducir ()

- buffer = REG_2
- comprobar memoria correcta en buffer
- size = REG_3
- Res = reproducir(buffer, size);
- REG_9 = Res.

LLamada al sistema_cambiar formato ()

- formato = REG_2
- Res = cambiar_formato(formato);
- REG_9 = Res.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 7

Una compañía de domótica ha desarrollado un sistema de control de la calefacción basada en un dispositivo que incorpora un sensor de temperatura y un sistema de activación de la calefacción. El funcionamiento es el siguiente:

- El sensor detecta y devuelve la temperatura de la casa
- El sistema activación permite establecer el nivel de la calefacción (0=apagado, 1, 2, 3, 4 y 5)
- Se pretende que el driver devuelva el valor del estado actual de encendido de la calefacción a pesar de que el hardware no permite conocer dicho valor.

Para facilitar el desarrollo del driver se ofrecen las siguientes funciones:

- **DevolverTemperatura (temp)**
  Esta función accede al HW del dispositivo y devuelve la temperatura actual que detecta el sensor.

- **ActivarCalefacción (nivelActivación)**
  Esta función permite apagar o encender la calefacción y situarla en uno de los niveles indicados anteriormente (0, 1, 2, 3, 4 ò 5)

Se pide:

a) Diseñar un driver para el sistema operativo UNIX del dispositivo anterior que incluya:

- Estructuras de datos que haya que añadir /modificar en el *kernel*.
- Interfaz completo del driver (funciones internas del *kernel* y llamadas al sistema) y un pseudocódigo de las mismas explicando en que eventos se realizan. (1.5 puntos)

b) Una versión más moderna del hardware permite lanzar una interrupción en el momento que la temperatura alcanza un nivel un poco más bajo o más alto que una temperatura umbral indicada con antelación. Esta nueva versión ofrece una función auxiliar más para facilitar el desarrollo del driver.

- **EstablecerTempUmbral (temp)**
  Esta función permite indicar la temperatura umbral que provocara que cuando se cruce hacia arriba o abajo se lance una interrupción.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestionar las llamadas al sistemas del driver
  - Una opción es utilizar el estándar UNIX.
    - **Desc = Open (nombre_dispositivo, flags)**
      - Permite reservar el acceso al controlador de la calefacción.
    - **Res = Close (Desc)**
      - libera la reservar para el acceso al controlador de la calefacción.
    - **Res = Read (Desc, buffer, size)**
      - Indica al controlador de calefacción que devuelva en buffer la temperatura que devuelve el sensor.
    - **Res = write (Desc, buffer, size)**
      - Indica al controlador de calefacción que cambie el nivel de la calefacción al indicado en el buffer.
    - **Res = Ioctl (Desc, operacion, punt_param)**
      - Si operación = OP_DARNIVELCALEFACION le dice al controlador de calefacción que devuelve el valor de activación de la calefacción en punt_param

- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_calefactor;**
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
  - **Descargar_driver_calefactor;**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

Las estructuras de datos requeridas son:

- Estructura de datos del driver. Incluye punteros al resto de variables del driver y a las funciones que lo componen.
  - Variable **temperatura**: indica la última temperatura obtenida del calefactor.
  - Variable **activación**: indica el valor de activación actual del calefactor

Grupo: …… NIA: …… Nombre y apellidos: ……..

LLamada_al_Sistema_write(Desc, buffer, size):

- activacion = write[1]
- ActivarCalefacción (activacion);

LLamada_al_Sistema_read(Desc, buffer, size):

- DevolverTemperatura (temperatura)
- Buffer[1]= temperatura;
- Size = 1

LLamada_al_Sistema_ioctl(Desc, operacion, punt_param):

- Si (operación = OP_ DARNIVELCALEFACION)
  - Punt_param=activación

b) Modificaciones a realizar:
Incluir la siguiente llamada del interfaz

Res = Ioctl (Desc, operacion, punt_param)

- Si operación = OP_DARTEMPUMBRAL
  - cambia la temperatura umbral por la que recibe en punt_param

Incluir la siguiente variable en la estructura del driver

- Variable **umbral**: indica la temperatura umbral del calefactor.

Modificaciones en el pseudocódigo

LLamada_al_Sistema_ioctl(Desc, operacion, punt_param):

- **……..**
- Si (operación = OP_ DARTEMPUMBRAL)
  - umbral=punt_param
  - EstablecerTempUmbral (umbral)

Manejador_interrupción_ calefactor ():

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 8

Se nos pide hacer el driver de una cámara web para un sistema operativo. Dicha cámara web envía una imagen cada vez que se le solicita escribiendo un comando en su puerto de E/S. Dicha imagen se envía por DMA sin intervención del S.O. y cuando termina la operación de DMA la cámara avisa con una interrupción hardware. Además dicha cámara se puede apagar y encender escribiendo otro comando en su puerto de E/S

El driver debe seguir el interfaz POSIX (open, read, write, ioctl, etc). La funcionalidad del driver debe incluir.

- Encender y apagar la cámara,
- Solicitar y recoger la imagen de la cámara.

Se pide:

a) Diseñar la interfaz de llamadas POSIX del driver, indicando en una línea cual será la labor de cada llamada.
b) Indicar qué eventos y estructuras de datos serán necesarias usar, añadir o modificar para implementar únicamente dicha funcionalidad.
c) Implementar en pseudocódigo las estructuras de datos y funciones descritas en el apartado anterior (usando interrupciones software si es necesario).

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Un posible uso de la interfaz posix es el siguiente
- **Desc = Open (nombre_dispositivo, flags)**
  - Permite acceder a la cámara web
- **Res = Close (Desc)**
  - libera el acceso a la cámara web
- **Res = Read (Desc, buffer, size)**
  - Indica a la cámara web que mande una imagen y, cuando llegue, la guarda en buffer.
- **Res = Write (Desc, buffer, size)**
  - Indica a la cámara web que encienda o apague la cámara según el valor del primer byte de buffer

b) Los eventos involucrados son:
- Llamadas al sistema read y write
- Interrupción hardware de la cámara web.
- Interrupción software de la cámara web.

Las estructuras de datos:

- Tabla de procesos, Lista de listos y proceso actual.
- Lista de proceso bloqueados en la cámara web.
- Zona de memoria para la operación de DMA donde almacenar la imagen.

c) Los eventos involucrados son:

Pseudocódigo Manejador_interrupción_hw_camara web()
- Insertar_Interrupción_Software(Int_Sw_camara_web)
- Generar_Interrupció_Software()

Pseudocódigo Int_Sw_camara_web ()
- ProcAux = primero(CAMARA.ListaProcBloqueados)
- Borrar(CAMARA.ListaProcBloqueados, ProcAux)
- ProcAux->estado = LISTO
- InsertarAlFinal(ListaListos, ProcAux)

Grupo: …… NIA: …… Nombre y apellidos: ……..

LLamada_al_Sistema_read(desc, buffer, tamanyo):

- Enviar (puerto_Camara_web, Cmd_enviar_imagen)
- Enviar (puerto_Camara_web, CAMARA.zona_mem_DMA)
- Enviar (puerto_Camara_web, tamaño_imagen)
- ProcAux = procesoActual
- InsertarAlFinal(CAMARA.ListaProcBloqueados, ProcAux)
- ProcAux ->estado = BLOQUEADO
- procesoActual = Planificador()
- procesoActual->estado = EJECUTANDO
- CambioContexto(ProcAux, procesoActual)
- Tamaño_copiado = Copiar (buffer, tamaño,
  CAMARA.zona_mem_DMA, tamaño_imagen).
- Return (Tamaño_copiado)

LLamada_al_Sistema_write(desc, buffer, tamanyo):

- Si (tamaño > 1)
  - Si (buffer[0] == 0)
    - Enviar (puerto_Camara_web, Cmd_apagar)
  - Si no
    - Enviar (puerto_Camara_web, Cmd_encender)

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 9

Una compañía nos contrata para hacer el driver de una tarjeta digitalizadora de sonido. Nos pide para un sistema operativo básico (como el que está siendo visto en clase) que usa un kernel monolítico con planificación *Round-Robin* que cumpla la siguiente interfaz de usuario:

- Funciones *open(nombre, flags) -> descriptor* y *close(descriptor)*: Para establecer acceso exclusivo a la tarjeta o liberarla.
- Función *read(descriptor, buffer, tamaño) -> bytes leídos*:
  Para leer un buffer con música digitalizada en formato *mp3*.

Se pide:

a) Indicar la interfaz completa que tendrá el driver, así como las estructuras de datos que se necesiten.
b) Indicar los eventos involucrados así como implementar en pseudocódigo dichos eventos, minimizando el número de copias en memoria de los datos hasta llegar al proceso de usuario que los solicitó.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) La interfaz se divide en, al menos, tres bloques:

- Gestionar la interrupción de la tarjeta de sonido:
  - **Manejador_Interrupcion_hw_tarjeta_sonido()**

- Gestionar las llamadas a sistema del driver:
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite reservar el acceso a la tarjeta de sonido
  - **Res = Close (Desc)**
    - libera la reservar para el acceso a la tarjeta de sonido
  - **Res = Read (Desc, buffer, size)**
    - Indica a la tarjeta de sonido que grabe sonido y devuelva el resultado en *buffer*.

- Gestionar la carga y descarga del driver en tiempo de ejecución:
  - **Cargar_driver_ tarjeta_sonido()**
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
  - **Descargar_driver_ tarjeta_sonido()**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

Las estructuras de datos requeridas son:

- Operaciones del driver: Incluye punteros a las funciones de interfaz.
- Lista enlazada de peticiones, donde cada petición incluye:
  - Dirección donde grabar los datos.
  - Proceso bloqueado (puntero a un BCP)

Grupo: …… NIA: …… Nombre y apellidos: ……..

b) Los eventos involucrados son:
- Interrupción hardware/software de tarjeta de sonido
- Llamada_al_sistema_read (para grabar)

Manejador_interrupción_ tarjeta_sonido ():

- Obtener resultado de los registros de E/S
- peticionEnCurso = primera(SONIDO.ListaPeticiones)
- Copiar(registro_de_dato, peticionEnCurso->buffer)
- Si quedan_más_datos
  - Return;
- Borrar(SONIDO.ListaPeticiones, peticionEnCurso)
- peticionEnCurso->BCP->estado = LISTO
- Insertar(ListaListos, peticionEnCurso->BCP)
- peticionEnCurso = primera(SONIDO.ListaPeticiones)
- Si peticionEnCurso != NULL
  - Enviar la orden de grabación a los registros de control y de datos del dispositivo

LLamada_al_Sistema_read(desc, buffer, tamanyo):

- Crear una nueva petición de grabación (petición)
  - petición->BCP = procesoActual
  - petición->Buffer = buffer
- Insertar(SONIDO.ListaPeticiones, petición)
- Si el dispositivo está libre
  - Enviar la orden de grabación a los registros de control y de datos del dispositivo
- procesoActual->estado = BLOQUEADO
- procesoAnterior = procesoActual
- procesoActual = Planificador()
- procesoActual->estado = EJECUTANDO
- CambioContexto(procesoAnterior, procesoActual)
  /* Aquí se bloquea el proceso, hasta un cambio de contexto de vuelta */

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 10

Se dispone de una máquina monoprocesador y queremos implementar un driver para un mando a distancia que será conectado a un sistema operativo básico con un *kernel* monolítico no expulsivo con la siguiente funcionalidad:

- Gestionar las interrupciones del teclado del mando a distancia al pulsar una tecla.
  - Si no hay proceso que solicite la tecla en el momento de ser pulsada el driver deberá almacenar las mismas de forma temporal.
- Ofrecer a los procesos usuario la forma de obtener las teclas pulsadas.
  - Si no hay tecla en el momento que la solicita el proceso se bloqueará el mismo.

El mando a distancia tiene un botón de cancelar que borra todas las teclas pulsadas anteriormente por si el televisor deja de responder.

Se pide:

a) Diseñar una **interfaz** para las funciones que requiere el driver (que incluya llamadas al sistema y funcionalidad propia del driver).

b) Definir las **estructuras de datos** necesarias para realizar la funcionalidad requerida.

c) Implementar en **pseudocódigo** la funcionalidad para las funciones del manejador de interrupción y del evento para obtener la tecla por parte del proceso.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestionar la interrupción de tecla pulsada:
  - **void manejador_Interrupcion_mando (void);**
    - Trata la interrupción hardware del piano
- Gestionar las llamadas al sistemas del driver:
  - Mínimo, leer una tecla
  - Una opción es utilizar el estándar UNIX:
    - **int desc = open (char *nombre, int flags)**
      - Permite reservar el acceso al mando
    - **int res = close (int desc)**
      - Libera la reservar para el acceso al mando
    - **int res = read (int desc, char *buffer, int size)**
      - Devuelve en *buffer* la tecla leída
- Gestionar la carga/descarga del driver.

b) Las estructuras de datos requeridas son:
- Estructura de datos del driver:
  incluye punteros al resto de variables del driver y a las funciones que lo componen.
- Buffer de teclado:
  compuesto por la lista de teclas almacenadas.
- Lista de procesos bloqueados (cada entrada un puntero a un BCP)

c) Una implementación de las funciones sería:

void manejador_interrupción_mando (void):

- Leer tecla del mando.
- SI (tecla leída es cancelar)
  - Borrar todos los caracteres de mando.buffer
- En caso contrario
  - Insertar la tecla leída en el mando.buffer
  - Si (no está vacío mando.lista_procesos_bloqueados)

Grupo: …… NIA: …… Nombre y apellidos: ……..

int res = read (int desc, char *buffer, int size):

- For (i=0; i<size; i++)
  - Si (está vacío el mando.buffer)
    - Eliminar el proceso actual de la lista de ejecutando
    - Insertar el proceso actual en mando.lista_procesos_bloqueados
    - Cambiar el estado del proceso actual de ejecutando a bloqueado

    - Obtener el BCP del primer proceso de la lista de listos
    - Poner dicho BCP en la lista de ejecutando
    - Cambiar contexto entre proceso actual y nuevo proceso ejecutando

    /* Aquí se bloquea el proceso */

  - Extraer la primera tecla de mando.buffer_teclado y se inserta en buffer[i]
- Devolver el número de teclas leídas

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 11

Disponemos de un sistema empotrado con CPU monoprocesador y sistema operativo básico con un *kernel* monolítico expulsivo (como el que está siendo presentado en pseudocódigo en la asignatura). Queremos implementar un driver para los teclados asociados a un ascensor (de un edificio con 5 pisos) con la siguiente funcionalidad:

- Gestionar las interrupciones del teclado al pulsar una tecla.
- Ofrecer a los procesos usuario la forma de obtener las teclas pulsadas (bloqueándose si no hay ninguna).

El teclado interior del ascensor emite una interrupción HW1 cada vez que se pulsa una tecla. La gestión del sistema operativo se encargará de leer del registro de datos del teclado la tecla pulsada que será un número de planta.

Cada planta tiene una tecla (para la llamada al ascensor) que están conectadas todas a un teclado externo. Cuando se pulsa una tecla en el teclado externo se emite la interrupción HW2, y el sistema operativo deberá conocer la planta desde la que fue llamado leyendo el registro de datos (de igual manera que antes, se almacena el número de planta).

El driver debe almacenar las teclas de forma temporal mientras que ningún proceso las solicita.

Se pide:

a. Diseñar la interfaz utilizando el estándar Unix (tanto interna como externa, incluidas las llamadas al sistema a ser implementadas) que requiere dicho driver en el caso de ser añadido al sistema operativo básico, indicando qué funcionalidad está asociada a cada elemento de la interfaz.
b. Definir la estructura de datos necesaria que define el driver para realizar la funcionalidad requerida en el sistema operativo básico.
c. Implementar en pseudocódigo la funcionalidad para obtener las teclas del teclado y para enviarlas a los procesos usuarios utilizando interrupciones software dentro del sistema operativo básico.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestión de carga y descarga del driver.
- Gestionar el dispositivo:
  - **Manejador_Interrupcion_tecladoHW1();**
  - **Manejador_Interrupcion_tecladoHW2();**
  - **PeticionTecla();**
- Gestionar la interfaz del driver (usando el estándar UNIX):
  - **Desc = Open (nombre_teclado, flags)**
    - Permite reservar el acceso al teclado
  - **Res = Close (Desc)**
    - Libera la reservar para el acceso al teclado
  - **Res = Read (Desc, buffer, size)**
    - Devuelve en *buffer* la tecla leída

b) Las estructuras de datos requeridas son:
- Estructura de datos del driver. Incluye punteros al resto de variables del driver y a las funciones que lo componen.
- **Buffer de teclado**:
  - Lista de teclas almacenadas (una lista de elementos que son caracteres)
- **Lista de procesos esperando por una tecla**:
  - Lista de procesos bloqueados (cada entrada un puntero a un BCP)

c) Los eventos involucrados en la lectura de teclas son:
- Interrupciones de teclado
- Función de petición de tecla
- Llamada_al_sistema_read (para leer una tecla)

El pseudo-código de las funciones es el siguiente:

Manejador_interrupción_tecladoHW1 ():
- Insertar_tecla(tecla, teclado.buffer);
- Insertar_Interrupcion_Software(Despertar_bloqueados_teclado);
- Generar_Interrupcion_Software();

Grupo: …… NIA: …… Nombre y apellidos: ……..

Despertar_bloqueados_teclado ():

- Proc = ObtenerPrimerProceso (teclado.lista_procesos_bloqueados);
- Cambiar su estado de bloqueado a listo.
- Incluirlo en la lista de procesos listos para ejecutar al final.

PeticionTecla():

- Si (estaVacio(teclado.buffer))
  - Insertar_proceso_actual(teclado.lista_procesos_bloqueados);
  - Cambiar el estado del proceso actual de ejecutando a bloqueado.
  - Establecer como proceso anterior el proceso actual.
  - Obtener el BCP del primer proceso de la lista de listos como proceso actual
  - Poner el estado del proceso actual a ejecutando.
  - Cambiar contexto entre proceso anterior y el proceso actual.
  - /* aquí se bloquea el proceso */
- devolver (extraer_tecla(teclado.buffer));

LLamada_al_Sistema_read(int fd, char *buffer, int size):

- for (i=0; i<size; i++)
  - buffer[i] = PeticionTecla() ;
- return size;

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 12

Disponemos de un sistema empotrado con CPU monoprocesador y sistema operativo básico con un *kernel* monolítico expulsivo (como el que está siendo presentado en pseudocódigo en la asignatura). Queremos implementar un driver para un ratón para poder leer la posición en la que se encuentra el mismo.

El dispositivo físico del ratón actualiza su posición (x e y) en registros internos. Cada vez que el ratón modifica su posición envía una interrupción HW a la CPU. La gestión del sistema operativo se encargará de leer de los registros de datos del ratón la posición nueva (se usará instrucciones in/out para ello)

Los procesos que desean conocer la posición del ratón deben realizar una llamada explicita al sistema operativo para ello y quedaran bloqueados a la espera de que se produzca una modificación en la posición del ratón.

El driver debe almacenar la posición actual del ratón mientras que ningún proceso la solicite y debe devolver la posición a todo proceso que se la solicite.

Se pide:

a. Diseñar la interfaz utilizando el estándar Unix (tanto interna como externa, incluidas las llamadas al sistema a ser implementadas) que requiere dicho driver en el caso de ser añadido al sistema operativo básico, indicando qué funcionalidad está asociada a cada elemento de la interfaz.
b. Definir la estructura de datos necesaria que define el driver para realizar la funcionalidad requerida en el sistema operativo básico.
c. Implementar en pseudocódigo la funcionalidad para obtener la posición del ratón y para comunicarla a los procesos usuarios utilizando interrupciones software dentro del sistema operativo básico.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestionar el dispositivo:
  - **manejadorInterrupcionHW_Ratón();**
  - **PeticionTecla();**
- Gestionar la interfaz del driver (usando el estándar UNIX):
  - **Desc = Open (nombre_teclado, flags)**
    - Permite reservar el acceso al ratón
  - **Res = Close (Desc)**
    - Libera la reservar para el acceso al ratón
  - **Res = Read (Desc, buffer, size)**
    - Devuelve en *buffer* un array con 2 enteros que tienen las posiciones x e y del ratón.
- Cargar y descargar driver

b) Las estructuras de datos requeridas son:
- Estructura de datos del driver. Incluye punteros al resto de variables del driver y a las funciones que lo componen.
- **posiciónRatón**:
  - Variable global array con 2 enteros con la posición actual del ratón
- **Lista de procesos esperando por eventos en el ratón**: ListaBloqueadosRatón
  - Lista de procesos bloqueados (un puntero a BCP que tendrá el inicio de la lista de bloqueados esperando eventos del ratón. Los demás estarán enlazados a partir de ese)

c) Los eventos involucrados en la lectura de posición del ratón son:
- Interrupción de ratón
- Función de petición de posición de ratón
- Llamada_al_sistema_read (para leer la posición)

El pseudo-código de las funciones es el siguiente:

manejadorInterrupcionHW_Ratón();
- posX= leerDeDispositivoRatón(posición X) (Operación IN/OUT sobre el dispositivo)
- posY= leerDeDispositivoRatón(posición Y) (Operación IN/OUT sobre el dispositivo)
- // si sólo se genera la interrupción cuando se mueve el ratón no hace falta el if

Grupo: …… NIA: …… Nombre y apellidos: ……..

Despertar_bloqueados_ratón ():

- Para cada proceso de ListaBloqueadosRatón hacer
  - Proc = ObtenerPrimerProceso (ListaBloqueadosRatón);
  - Cambiar su estado de bloqueado a listo.
  - Incluirlo en la lista de procesos listos para ejecutar al final.

LLamada_al_Sistema_read(int fd, void *buffer, int size):

- *//La llamada siempre es bloqueante, hasta que se mueva el ratón*
- Insertar_proceso_actual(ListaBloqueadosRatón);
- Cambiar el estado del proceso actual de ejecutando a bloqueado.
- Obtener el BCP del primer proceso de la lista de listos.
- Poner dicho BCP en la lista de ejecutando
- Cambiar contexto entre proceso actual y nuevo proceso ejecutando.
  // aquí se bloquea el proceso
- peticiónPos ((int *) buffer);
- devolver (8); // 2 enteros * 4 bytes por entero

PeticionPos (int *buffer):

- buffer[0]= posiciónRatón[0]
- buffer[1]= posiciónRatón[1]

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 13

Disponemos de un sistema móvil con CPU monoprocesador y sistema operativo con un *kernel* monolítico expulsivo del estilo del minikernel. Queremos implementar un driver para el manejo de las tres teclas de control que tiene el dispositivo, lo que supondrá:

- Gestionar las interrupciones del teclado al pulsar una de estas teclas.
- Ofrecer al sistema operativo la forma de obtener las teclas pulsadas (el sistema operativo indicará el número de teclas que quiere obtener) sin quedarse bloqueado si no hay ninguna.
- Almacenar las pulsaciones de las teclas hasta que el sistema operativo las solicite.

Las tres teclas suponen la funcionalidad de activar la retroiluminación de la pantalla, desactivar el sonido y cambiar el turbo de la CPU. El sistema operativo tiene unas rutinas internas para poder gestionar cada una de ellas: cambiar_retroiluminacion (), cambiar_sonido () y cambiar_turbo (). La ejecución de una de estas funciones se encarga de cambiar el estado en el que esté el dispositivo al opuesto.

Se pide:

a. Diseñar la interfaz completa que requiere dicho driver en el caso de ser añadido al sistema operativo básico, indicando qué funcionalidad está asociada a cada elemento de la interfaz.
b. Definir la estructura de datos necesaria que define el driver para realizar la funcionalidad requerida en el sistema operativo básico.
c. Implementar en pseudocódigo la funcionalidad para obtener las teclas del teclado y para enviarlas a los procesos usuarios utilizando interrupciones software dentro del sistema operativo básico.
d. Implementar en pseudocódigo cómo sería la parte del sistema operativo que se encargaría de pedir las teclas y llamar a las anteriores funciones según corresponda.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestión de carga y descarga.
- Gestionar el dispositivo:
  - **Manejador_Interrupcion_tecladoHW();**
  - **PeticionTecla();**
- Gestionar la interfaz del driver (si se usa el estándar UNIX):
  - **Desc = Open (nombre_teclado, flags)**
    - Permite reservar el acceso al teclado
  - **Res = Close (Desc)**
    - Libera la reservar para el acceso al teclado
  - **Res = Ioctl (Desc, comando, opciones)**
    - Devuelve la tecla leída
- Gestionar la interfaz del driver (si no se usa el estándar UNIX):
  - **Res = obtener_tecla_nobloqueante (buffer, tamaño)**
    - Devuelve el número de teclas leídas, y almacena en el buffer las mismas.

b) Las estructuras de datos requeridas son:
- Estructura de datos del driver. Incluye punteros al resto de variables del driver y a las funciones que lo componen.
- **Buffer de teclado**:
  - Lista de teclas almacenadas (una lista de elementos que son caracteres)

No se precisa una lista de procesos esperando por una tecla puesto que el solicitante (en este caso el sistema operativo y no los procesos) no tiene que esperar por teclas.

c) Los eventos involucrados en la lectura de teclas son:
- Interrupciones de teclado
- Función de petición de tecla
- Interfaz para leer una tecla

El pseudo-código de las funciones es el siguiente:

Manejador_interrupción_tecladoHW ():
- Insertar_tecla(tecla, teclado.buffer);

Grupo: …… NIA: …… Nombre y apellidos: ……..

PeticionTecla (tecla):

- Si está vacío (teclado.buffer)
  - Devolver false
- tecla <- extraer_tecla(teclado.buffer)
- Devolver true

obtener_teclas_nobloqueante (char *buffer, int size):

- ok = true
- for (i=0; (i<size) && (ok); i++)
  - ok = PeticionTecla(& buffer[i]) ;
- return i;

d) La parte del sistema operativo tendría una estructura como se indica a continuación:

- char buffer[10];
- …
- obtener_teclas_nobloqueante(buffer,10);
- for (i=0; i<10; i++)
  - switch(buffer[i])
    - case retro: cambiar_retroiluminacion()
    - case turbo: cambiar_turbo()
    - case mute: cambiar_sonido()
- …

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 14

Disponemos de un sistema empotrado con CPU monoprocesador y sistema operativo con un *kernel* monolítico expulsivo del estilo del sistema operativo básico. Queremos implementar un driver para un ratón que controle la pulsación de las teclas del mismo con la siguiente funcionalidad:

- Gestionar las interrupciones del ratón al pulsar una tecla.
- Ofrecer a los procesos usuario la forma de obtener la tecla pulsada (bloqueándose hasta que se pulse una).

El ratón tiene 3 teclas y emite una interrupción HW1 cada vez que se pulsa una de ellas (la misma independientemente de la tecla pulsada). Además, almacena en un registro interno cuál es la tecla que se ha pulsado (1-Derecha, 2-Izquierda o 3-Centro), debiendo el sistema operativo acceder al dispositivo para recuperar el valor del registro con el dato de la tecla pulsada. La gestión del sistema operativo se encargará de leer del registro de datos del ratón la tecla pulsada (se usará instrucciones in/out para ello) que será un número entre 1 y 3.

El ratón también emite una interrupción HW2 cuando la tecla deja de estar pulsada. Dada la escasa velocidad de reacción de los humanos se supondrá que desde que se recibe la interrupción HW1 hasta que se recibe la interrupción HW2 da tiempo para que el Sistema Operativo procese la petición y lea la tecla que se ha pulsado.

La pulsación de las teclas es excluyente así que podemos estar seguros de que cuando se pulsa una tecla no se va a pulsar otra hasta que no se haya liberado la anterior.

Cada vez que un proceso solicita una tecla se le deberá enviar la tecla pulsada en ese momento o dejarlo en espera hasta que se pulse alguna.

El driver debe almacenar las teclas de forma temporal mientras que ningún proceso las solicita.

Se pide:

a. Diseñar la interfaz utilizando el estándar Unix (tanto interna como externa, incluidas las llamadas al sistema a ser implementadas) que requiere dicho driver en el caso de ser añadido al sistema operativo básico, indicando qué funcionalidad está asociada a

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Primero hay que ver la funcionalidad requerida, que es:
- Gestionar el dispositivo:
  - **Manejador_Interrupcion_ratónHW1(); //Pulsación de tecla**
  - **Manejador_Interrupcion_ratónHW2(); //Liberación de tecla**
  - **PeticionTecla();**
- Gestionar la interfaz del driver (usando el estándar UNIX):
  - **Desc = Open (nombre_ratón, flags)**
    - Permite reservar el acceso al ratón
  - **Res = Close (Desc)**
    - Libera la reservar para el acceso al ratón
  - **Res = Read (Desc, buffer, size)**
    - Devuelve en *buffer* la tecla leída

b) Las estructuras de datos requeridas son:
- Estructura de datos del driver. Incluye punteros al resto de variables del driver y a las funciones que lo componen.
- **Tecla pulsada en el ratón: teclaPulsada inicializada a 0 al cargar el driver**
  - Tecla actualmente pulsada : 1-Derecha, 2-Izquierda, 3-Centro o 0 si no se ha pulsado ninguna
- **Pulsando.** Valor 1 si está pulsada en ese momento y 0 si no lo está. Inicializada a 0 en la inicialización del driver
- **Lista de procesos esperando por eventos en el ratón**: ListaBloqueadosRatón
  - Lista de procesos bloqueados (un puntero a BCP que tendrá el inicio de la lista de bloqueados esperando eventos del ratón. Los demás estarán enlazados a partir de ese)

c) Los eventos involucrados en la lectura de teclas son:
- Interrupciones de ratón
- Función de petición de tecla
- Llamada_al_sistema_read (para leer una tecla)

El pseudo-código de las funciones es el siguiente:

Grupo: …… NIA: …… Nombre y apellidos: ……..

Manejador_interrupción_ratónHW2 ():

- Pulsando= 0

Despertar_bloqueados_ratón ():

- Para cada proceso de ListaBloqueadosRatón hacer
  - Proc = ObtenerPrimerProceso (ListaBloqueadosRatón);
  - Cambiar su estado de bloqueado a listo.
  - Incluirlo en la lista de procesos listos para ejecutar al final.

int PeticionTecla():

- Si (Pulsando== 0)
  - Insertar_proceso_actual(ratón.lista_procesos_bloqueados);
  - Cambiar el estado del proceso actual de ejecutando a bloqueado.
  - Establecer como proceso anterior el proceso actual.
  - Obtener el BCP del primer proceso de la lista de listos como proceso actual
  - Cambiar el estado del proceso actual a ejecutando.
  - Cambiar contexto entre proceso anterior y el proceso actual.
    /* aquí se bloquea el proceso */
- return teclaPulsada;

LLamada_al_Sistema_read(int fd, void *buffer, int size):

- *buffer= PeticionTecla() ;
- return 4 // 4 es el tamaño de un entero que es lo que devuelve el read

Opción mejorada no considerada para evaluar el ejercicio:
Si hubiera que devolver la 1º tecla pulsada desde que se realizó la llamada al read habría que tener una lista de teclas pulsadas asociada a un contador de procesos bloqueados.
struct{ int tecla; int contadorBloqueados; void *siguiente}
En el campo contadorBloqueados almacenaría el número de procesos que estaban bloqueados esperando esa tecla. Se incrementará en DespertarBloqueadosRatón 1 vez por cada proceso que se despierte). Se decrementará en PeticiónTecla justo después de que se desbloque el proceso y si vale llega a 0 se eliminará el nodo de la lista. La tecla que se devolverá será la asociada al contadorBloqueados o la actualmente pulsada si no se ha bloqueado el proceso porque justo cuando se invocó al read se estaba pulsando una tecla

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 15

Una compañía nos contrata para **hacer el driver de un lector de códigos de barras**. El lector consta de un botón que pulsa el empleado para proceder a la lectura del código de barras, de un láser que se activa al pulsar el botón y lee el código de barras y de un led.
Nos piden, para un sistema operativo básico (como el que está siendo visto en clase) que usa un kernel monolítico con planificación *Round-Robin*, que creemos un driver que permita ofrecer a los procesos de usuario una interfaz para leer dichos códigos de barras **utilizando el estándar POSIX**.
Para ello se dispone de las siguientes funciones proporcionadas por el fabricante del lector:

long obtenerCodigoBarrasLeido(): función que envía al computador un número entero con el último código de barras leído y almacenado en los registros del dispositivo lector. Una vez leído el registro del dispositivo éste se coloca a 0 y si se invoca esta función sin haber realizado ninguna lectura posterior ésta devolverá 0

void encenderLed (): Enciende el led del dispositivo durante 1 segundo Su utilidad podría ser indicar al operario que el código de barras ha sido correctamente procesado por el computador.

Además **el dispositivo envía la interrupción INT_LECTOR** para indicar que se ha leído un código de barras y que éste ha sido almacenado en el registro al efecto del dispositivo.

Se desea que no se pierda ningún código de barras leído por el dispositivo a no ser que el programador de la aplicación los descarte expresamente y que se avise al operador, encendiendo el led, cada vez que una aplicación haya recuperado correctamente un código leído.
Además se desea proporcionar a los programas de usuario un método para eliminar todos los códigos de barras leídos por el driver pero aún no procesados por procesos de usuario.
En resumen se deben proporcionar la siguiente funcionalidad a los procesos de usuario:

- **Leer códigos de barras**. Llamada bloqueante hasta que se recupere un código de barras sino hay ninguno previamente leído.
- **Encender el led** para indicar que el código de barras ha sido obtenido por un proceso de usuario
- **Eliminar la lista de códigos** de barras no procesados (aún no entregados a aplicaciones de usuarios)

Se pide:

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) La interfaz se divide en, al menos, tres bloques:

- Gestionar la interrupción del lector.
  - **Manejador_Interrupcion_lector()**

- Gestionar las llamadas al sistemas del driver
  - **Desc = open (nombre_dispositivo, flags)**
    - Permite reservar el acceso al lector
  - **Res = close (Desc)**
    - libera la reserva para el lector
  - **Res = read (Desc, buffer, size)**
    - Devuelve un código de barras leído.
  - **Res = write (Desc, buffer, size)**
    - Indica al lector que encienda el led
  - **ioctl (Desc, codigoOperacion, puntParametros)**
    - Indica al driver que debe borrar la lista de códigos de barras leídos pero no procesados cuando el códigoOperación es BORRAR

- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_ lector()**
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
  - **Descargar_driver_lector()**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

Las estructuras de datos requeridas son:

- Operaciones del driver: Incluye punteros a las funciones de interfaz.
- Lista enlazada de códigos de barras leídos, donde cada petición incluye:
  - Código de barras leído.

b) Los eventos involucrados son:

Grupo: …… NIA: …… Nombre y apellidos: ……..

Variables Globales:

ColaCodigosBarras Cola con los códigos de barras leídos

ColaBloqueadosEsperaCodigos Cola con los procesos bloqueados esperando códigos de barras.

Manejador_interrupción_lector ():

- codBarras=obtenerCodigoBarrasLeido()
- añadir(codBarras, colaCodigosBarras)
- Insertar_Interrupción_Software(DespertarColaBloqueadosBarras)
- Generar_Interrupció_Software()

DespertarColaBloqueadosBarras ()

- Si colaBloqueadosEsperaCodigos != NULL
  - proc=SacarPrimero(colaBloqueadosEsperaCodigos)
  - proc.estado = LISTO
  - InsertarUltimo(ColaListos, proc)

LLamada_al_Sistema_read(desc, buffer, tamanyo):

- while (colaCodigosBarras == null)
  - procesoActual->estado = BLOQUEADO
  - procesoAnterior = procesoActual
  - procesoActual = Planificador()
  - Borrar(ColaListos, procesoActual)
  - procesoActual->estado = EJECUTANDO
  - CambioContexto(procesoAnterior, procesoActual)
  - /* Se bloquea el proceso, hasta un cambio de contexto de vuelta */
- codBarras=sacarPrimero(colaCodigosBarras)
- Copiar (buffer, codBarras)
- Return (sizeof(long))

LLamada_al_Sistema_write(desc, buffer, tamanyo):

- encenderLed()
- return 1

LLamada_al_Sistema_ioctl(desc, codOperacion, puntDatos):

- if (codOperacion == BORRAR)
  - EliminarTodasPeticiones (colaCodigosBarras)

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 16

Disponemos de una maquina monoprocesador y nos pagan por implementar un driver de teclado para un sistema operativo UNIX con un *kernel* monolítico expulsivo (como el descrito en el material de la asignatura) con la siguiente funcionalidad:

- Gestionar las interrupciones del teclado al pulsar una tecla.
- Ofrecer a los procesos usuario la forma de obtener las teclas pulsadas (bloqueándose si no hay ninguna) con una nueva llamada obtenerTecla.
- Cargar y descargar el driver en tiempo de ejecución del núcleo.
- En caso de que un proceso solicite una tecla, si no hay se bloqueará el proceso y se ejecutará otro, hasta que llegue la tecla solicitada tecla, donde el proceso anteriormente en espera pasará directamente a ejecutarse de nuevo.
- El driver debe almacenar las teclas de forma temporal mientras que ningún proceso las solicita.
- Si el teclado lleva más de 1.000 ticks de reloj sin pulsarse una tecla entonces hay que ejecutar 'out(0x100,200)' para seguir recibiendo teclas.

Se pide:

a) Diseñar una interfaz completa para las funciones que requiere el *driver* descrito. Use las interrupciones software necesarias, e incluya función es espacio de usuario.
b) Definir las estructuras de datos necesarias para realizar la funcionalidad requerida.
c) Implementar en pseudocódigo la funcionalidad pedida.
   ¿En qué eventos debe incluirse esta funcionalidad?

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) Interfaz del driver:

- Gestionar la interrupción de teclado.
  - **Manejador_Interrupcion_HW_teclado();**
  - **Manejador_Interrupcion_SW1_teclado();**
  - **Manejador_Interrupcion_SW2_teclado();**
- Gestionar las llamadas al sistemas del driver (sin usar estándar UNIX):
  - **Res = obtenerTecla()**
    - Pide una tecla.
- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_teclado();**
    - Inserta la estructura del driver (variables y operaciones) en las tablas correspondientes.
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
  - **Descargar_driver_teclado();**
    - Elimina la estructura del driver de las tablas correspondientes.

b) Las estructuras de datos necesarias serían:

- Los datos del driver de teclado en una estructura para registrar con el núcleo.
- Como variables internas del driver:
  - **Lista de teclas almacenadas (KB1.BufferTeclas)**
  - **Lista de procesos bloqueados (KB1.Bloqueados)**
  - **Tiempo desde la última vez pulsada una tecla (KB1.Tiempo)**

c) El pseudocódigo de la funcionalidad pedida sería:

Grupo: …… NIA: …… Nombre y apellidos: ……..

Manejador interrupción_SW1_teclado ():

- Proc = ObtenerPrimerProceso (KB1.Bloqueados)
- Si Proc != NULL
  - **Cambiar el estado del proceso actual de ejecutando a LISTO**
  - **Insertar el proceso actual en la lista de listos**
  - **Quitar el proceso Proc de la lista de bloqueados.**
  - **Cambiar su estado de bloqueado a ejecutando.**
  - **Cambiar de contexto entre el proceso anterior y el proceso actual.**

Manejador interrupción_HW_reloj():

- <<Lo que antes se hiciera>>
- **Insertar_Interrupcion_Software(interrupción_SW2_teclado);**
- **Generar_Interrupcion_Software();**

Manejador interrupción_SW2_teclado ():

- **If (ticks - KB1.Tiempo > 1.000)**
  - **out(0x100, 200);**

Llamada al sistema obtenerTecla():

- R0 = código OBTENERTECLA
- SYSCALL
- Devolver R0

Llamada al sistema KERNEL_obtenerTecla():

- Si (estaVacio(KB1.BufferTeclas))
  - Insertar_proceso_actual(KB1.Bloqueados);
  - Cambiar el estado del proceso actual de ejecutando a bloqueado.
  - Establecer como proceso anterior el proceso actual.
  - Sacar el BCP del primer proceso de la lista de listos, como proceso actual.
  - Cambiar el estado del proceso actual a ejecutando.
  - Cambiar de contexto entre el proceso anterior y el proceso actual.

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 17

Una compañía nos contrata para hacer el driver de una cámara de video conectada a un ordenador. El hardware de la cámara mantiene en una memoria propia el último minuto de grabación. El fabricante del HW de la cámara nos ha proporcionado la función de bajo nivel: *ObtenerUltimoMinutoFabHW ()*
Cuando termina la operación de transmitir el video la cámara envía la interrupción INTCAMARA. La función utiliza DMA para la comunicación de la cámara al ordenador.

Se debe diseñar el driver de la cámara para un sistema operativo básico (como el que está siendo visto en clase) que usa un kernel monolítico con planificación *Round-Robin. Se deberá proporcionar la siguiente funcionalidad* utilizando llamadas del estándar POSIX:

- Obtener el último minuto de grabación. Si hay varios procesos que solicitan el último minuto de grabación mientras una petición de este tipo está pendiente se les deberá devolver la misma secuencia sin solicitar a la cámara la transmisión de una nueva.

NOTA: Las peticiones de los procesos serán bloqueantes y el dispositivo no permite conocer si está siendo usado.

Se pide:

a) Indicar la interfaz completa que tendrá el driver, así como las estructuras de datos que se necesiten.
b) Indicar los eventos involucrados así como implementar en pseudocódigo dichos eventos.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) La interfaz se divide en, al menos, tres bloques:

- Gestionar la interrupción de la cámara de fotos.
  - **Manejador_InterrupcionHW_Camara ()**

- Gestionar las llamadas al sistemas del driver
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite reservar el acceso a la cámara
  - **Res = Close (Desc)**
    - libera la reservar para el acceso a la cámara
  - **Res = Read (Desc, buffer, size)**
    - Solicita el último minuto de grabación grabado por la cámara guardando el resultado en *buffer*.

- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **CargarDriverCamara ()**
    - Carga el driver en el espacio de memoria del kernel
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
  - **DescargarDriverCamara ()**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

Las estructuras de datos requeridas son:

- Operaciones del driver: Incluye punteros a las funciones de interfaz.
- Zona de memoria DMAVideo: donde la cámara vuelca el último minuto de grabación usando DMA.
- Variable peticionEnCurso que es un puntero a la petición en curso.
  Una petición incluye una lista de elementos.
  Un elemento incluye:
  - Dirección del buffer del proceso que solicitó la operación.

Grupo: …… NIA: …… Nombre y apellidos: ……..

b) Los eventos involucrados son:
- Interrupción hardware/software de cámara de fotos
- Llamada_al_sistema_read (para grabar)

Manejador_InterrupciónHW_Camara (): //Se ejecuta al recibir INTCAMARA
- // el DMA se encarga de copiar los datos a DMAVideo
- Insertar_Interrupcion_Software(IntSWCamara)
- Generar_Interrupcion_Software();

IntSWCamara ():
- Elto = extraerElemento(peticionEnCurso)
- mientras (Elto != NULL)
  - Copiar (DMAVideo, Elto->buffer)
  - proc = elto->proc
  - proc->estado = LISTO
  - Insertar(ListaListos, proc)
  - Elto = extraerElemento (peticionEnCurso)
- Liberar(peticiónEnCurso)
- peticiónEnCurso = NULL

LLamada_al_Sistema read (desc, buffer, tamanyo):
- Si peticionEnCurso == NULL
  - peticionEnCurso == nuevaPetición()
  - Insertar(peticionEnCurso, procesoActual, buffer)
  - ObtenerUltimaFotoFabHW()
- En otro caso
  - Insertar(peticionEnCurso, procesoActual, buffer)
- procesoActual->estado = BLOQUEADO
- procesoAnterior = procesoActual
- procesoActual = Planificador()

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 18

Empresas fotos 360 nos contrata para hacer el driver de una cámara de fotos que llevará un vehículo de conducción automática.

Nos pide para un sistema operativo básico (como el que está siendo visto en clase) que usa un kernel monolítico con planificación *Round-Robin* que cumpla la siguiente interfaz de usuario:

- Funciones *open (nombre, flags) -> descriptor* y *close(descriptor)*:
  Para establecer acceso exclusivo a la cámara o liberarla.
- Función *read (descriptor, buffer, tamaño) -> bytes leídos*:
  Para leer un buffer con la foto en formato PNG.

Se pide:

a) Indicar la interfaz completa que tendrá el driver, así como las estructuras de datos que se necesiten.
b) Indicar los eventos involucrados para hacer una foto así como implementar en pseudocódigo dichos eventos, minimizando el número de copias en memoria de los datos hasta llegar al proceso de usuario que los solicitó.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) La interfaz se divide en, al menos, tres bloques:

- Gestionar la interrupción de la cámara de fotos.
  - **Manejador_Interrupcion_hw_camara_fotos()**
  - **Manejador_Interrupcion_sw_camara_fotos()**

- Gestionar las llamadas al sistemas del driver
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite reservar el acceso a la cámara de fotos
  - **Res = Close (Desc)**
    - libera la reservar para el acceso a la cámara de fotos
  - **Res = Read (Desc, buffer, size)**
    - Indica a la cámara de fotos que tome foto y devuelva el resultado en *buffer*.

- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_ cámara_fotos()**
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
  - **Descargar_driver_ camara_fotos()**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

Las estructuras de datos requeridas son:

- Operaciones del driver: Incluye punteros a las funciones de interfaz.
- Lista enlazada de peticiones, donde cada petición incluye:
  - Dirección del buffer donde guardar los datos.
  - Proceso bloqueado (puntero a un BCP)
- Petición en curso: puntero a la petición en curso.

Grupo: …… NIA: …… Nombre y apellidos: ……..

b) Los eventos involucrados son:
- Interrupción hardware/software de cámara de fotos
- Llamada_al_sistema_read (para grabar)

Manejador_interrupcion_ hw_camara_fotos ():
- Obtener resultado de los registros de E/S
- Copiar(registro_de_dato, peticionEnCurso->buffer)
- InsertarTareaPendiente(Manejador_interrupcion_sw_camara_fotos)
- ActivarInterrupciónSoftware()

Manejador_interrupcion_ hw_camara_fotos ():
- peticionEnCurso->BCP->estado = LISTO
- Insertar(ListaListos, peticionEnCurso->BCP)
- peticionEnCurso = primera(CAMARA.ListaPeticiones)
- Si peticionEnCurso != NULL
  - Enviar la orden de toma de foto a los registros de control y de datos del dispositivo

LLamada_al_Sistema_read(desc, buffer, tamanyo):
- Crear una nueva petición de toma de fotos (petición)
  - petición->BCP = procesoActual
  - **petición->buffer = buffer**
- Si el dispositivo está libre
  - peticionEnCurso = petición
  - Enviar la orden de toma de foto a los registros de control y de datos del dispositivo
- En otro caso
  - Insertar(CAMARA.ListaPeticiones, petición)
- procesoActual->estado = BLOQUEADO
- procesoAnterior = procesoActual
- procesoActual = Planificador()
- Borrar(ListaListos, procesoActual)

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 19

Una compañía nos contrata para hacer el driver de una minitarjeta de red que se venderá como parte de un *smartwatch* verde que quieren sacar a la venta.

Nos pide para un sistema operativo básico (como el que está siendo visto en clase) que usa un kernel monolítico con planificación *Round-Robin* que cumpla la siguiente interfaz de usuario:

- Funciones *open(nombre, flags) -> descriptor* y *close(descriptor)*:
  Para establecer acceso exclusivo a la tarjeta o liberarla.
- Función *read(descriptor, buffer, tamaño) -> bytes leídos*:
  Para leer un buffer con los datos de un paquete de red por parte de un proceso solicitante.

Se pide:

a) Indicar la interfaz completa que tendrá el driver, así como las estructuras de datos que se necesiten.
b) Indicar los eventos involucrados así como implementar en pseudocódigo dichos eventos.

   Dos requisitos a cumplir: minimizar el número de copias en memoria de los datos hasta llegar al proceso de usuario que los solicitó, e insertar siempre la petición en la lista de peticiones del dispositivo.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) La interfaz se divide en, al menos, tres grandes bloques:

- Gestionar la interrupción de la tarjeta de minired.
  - **Manejador_Interrupcion_hw_tarjeta_minired()**
  - **Manejador_Interrupcion_ww_tarjeta_minired()**

- Gestionar las llamadas al sistemas del driver
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite reservar el acceso a la tarjeta de minired
  - **Res = Close (Desc)**
    - libera la reservar para el acceso a la tarjeta de minired
  - **Res = Read (Desc, buffer, size)**
    - Indica a la tarjeta de minired que capture un paquete de red y devuelva los datos en *buffer*.

- Gestionar la carga y descarga del driver en tiempo de ejecución.
  - **Cargar_driver_ tarjeta_minired()**
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
  - **Descargar_driver_ tarjeta_minired()**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

Las estructuras de datos requeridas son:

- Operaciones del driver: Incluye punteros a las funciones de interfaz.
- Lista enlazada de peticiones, donde cada petición incluye:
  - Datos grabados.
  - Proceso bloqueado (puntero a un BCP)

Grupo: …… NIA: …… Nombre y apellidos: ……..

b) Los eventos involucrados son:
   - Interrupción hardware/software de tarjeta de minired
   - Llamada_al_sistema_read (para capturar un paquete)

Manejador_interrupción_ hw_tarjeta_minired ():

- Obtener resultado de los registros de E/S
- peticionEnCurso = primera(MINIRED.ListaPeticiones)
- Borrar(MINIRED.ListaPeticiones, peticionEnCurso)
- Copiar(registro_de_dato, peticionEnCurso->buffer)

Manejador_interrupción_ sw_tarjeta_minired ():

- peticionEnCurso->BCP->estado = LISTO
- Insertar(ListaListos, peticionEnCurso->BCP)
- peticionEnCurso = primera(MINIRED.ListaPeticiones)
- Si peticionEnCurso != NULL
  - Enviar la orden de grabación a
    los registros de control y de datos del dispositivo

LLamada_al_Sistema_sys_read(desc: R1, buffer: R2, tamaño: R3):

- Crear una nueva petición de captura (petición)
  - petición->BCP = procesoActual
  - petición->Buffer = buffer
- Insertar(MINIRED.ListaPeticiones, petición)

- Si el dispositivo está libre
  - Enviar la orden de captura de paquete de red
    a los registros de control y de datos del dispositivo

- procesoActual->estado = BLOQUEADO
- procesoAnterior = procesoActual
- procesoActual = Planificador()
- Borrar(ListaListos, procesoActual)
- procesoActual->estado = EJECUTANDO

Grupo: …… NIA: …… Nombre y apellidos: ……..

Ejercicio 20

Una empresa nos contrata para hacer el driver de un detector de movimiento. Dicho dispositivo proporcionará cada 300 ms la posición GPS en la que se encuentra. Los procesos podrán solicitar un número determinado de posiciones para para dibujar la trayectoria del dispositivo.
Por ejemplo, el proceso 1 pide 10 posiciones y el proceso 2 solicita 5 en el mismo instante; cada posición que vaya llegando se servirá al primer proceso hasta satisfacer el número de posiciones momento en el cual se desbloqueará y se pasará al siguiente.

Se cuenta con un sistema operativo básico (como el que está siendo visto en clase) que usa un kernel monolítico con planificación *Round-Robin* denominado SO1.

Se pide:

a) Indicar la interfaz completa que tendrá el driver de dicho dispositivo.
b) Indique las estructuras de datos que se necesitan para cumplir la funcionalidad descrita en el enunciado.
c) Implementar en pseudocódigo:
    a. La llamada al sistema de petición.
    b. La interrupción hardware del dispositivo. Tenga en cuenta que dicha interrupción se ejecutará en menos de 100 ms y no será necesaria la implementación de una Interrupción Software.

Grupo: …… NIA: …… Nombre y apellidos: ……..

SOLUCIÓN

a) La interfaz se divide en, al menos, tres grandes bloques:

- Gestión del controlador.
  - **Manejador_Interrupcion_hw ()**
- Gestionar las llamadas al sistemas del driver:
  - **Desc = Open (nombre_dispositivo, flags)**
    - Permite reservar el acceso al dispositivo
  - **Res = Close (Desc)**
    - Libera la reservar para el acceso al dispositivo
  - **Res = Read (Desc, buffer, size)**
    - Indica al dispositivo que se quiere capturar tantas posiciones como valor de size, y los datos se guardarán en *buffer*.
- Gestionar la carga y descarga del driver:
  - **Cargar_driver ()**
    - Enlaza el fichero objeto del driver en el espacio de memoria del kernel
    - Inserta la estructura del driver (variables y operaciones) en la tabla de drivers.
  - **Descargar_driver ()**
    - Elimina la estructura del driver de la tabla de drivers.
    - Borra el código del driver del espacio de memoria del *kernel*.

b) Las estructuras de datos requeridas son:

- Operaciones del driver: Incluye punteros a las funciones de interfaz.
- Variable peticiones con una lista enlazada de peticiones, donde cada petición incluye:
  - npos: Número de posiciones a guardar.
  - pos: Número de posiciones guardadas
  - buffer: Dirección del buffer del proceso solicitante.
  - bloqueado: Puntero del BCP del proceso.

Grupo: …… NIA: …… Nombre y apellidos: ……..

c) Implementaciones pedidas:

LLamada_al_Sistema_usr_read (fd: R1, buffer: R2, size: R3):

- R0 <- READ
- R1 <- fd
- R2 <- buffer
- R3 <- size
- Trap
- Return R0

LLamada_al_Sistema_sys_read (fd: R1, buffer: R2, size: R3):

- Crear una nueva petición de captura: petición
  - petición->bloqueado = procesoActual
  - petición->npos = R3
  - petición->pos = 0
  - petición->buffer = R2

- Si peticionActual == NULL
  - peticionActual = petición
- en otro caso
  - Insertar(peticiones, petición)

  // no es necesario enviar la orden al dispositivo,
  // proporciona los datos cada 300 ms.

- procesoActual->estado = BLOQUEADO
- procesoAnterior = procesoActual
- procesoActual = Planificador()
- Borrar(ListaListos, procesoActual)
- procesoActual->estado = EJECUTANDO
- CambioContexto(procesoAnterior, procesoActual)

  /* Aquí se bloquea el proceso, hasta un cambio de contexto de vuelta */

Grupo: …… NIA: …… Nombre y apellidos: ……..

Manejador_interrupción_ hw ():

- Obtener posición de los registros de E/S
- Si peticionActual == NULL
  - Return

- Guardar posición en peticionActual->buffer[PX->pos]
- (peticionActual ->pos)++
- Si peticionActual ->pos == peticionActual ->npos
  - Proc = peticionActual ->bloqueado
  - Proc->estado = LISTO
  - Insertar(ListaListos, proc)

- peticionActual = Extraer(peticiones)