Ejercicios

drivers y servicios ampliados

Grupo ARCOS

Diseño de Sistemas Operativos

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Ejercicio
enunciado (1/2)

Disponemos de una maquina monoprocesador y queremos implementar un driver de teclado para un sistema operativo UNIX 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.

Ejercicio
enunciado (2/2)

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?

Ejercicio
solución

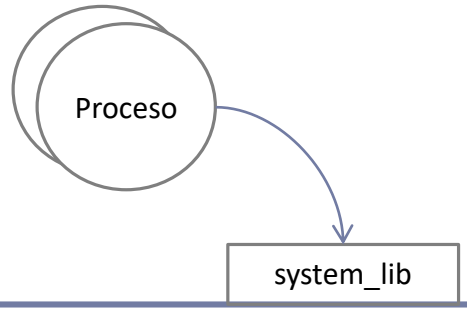
1. Planteamiento inicial
    1. Estado inicial del sistema
    2. Estudio de qué hay que modificar
2. Responder a las preguntas
3. Revisar las respuestas

Ejercicio
solución

1. Planteamiento inicial
    1. Estado inicial del sistema
    2. Estudio de qué hay que modificar
2. Responder a las preguntas
3. Revisar las respuestas

Ejercicio
solución

En espacio de usuario (U) tenemos los procesos que hacen llamadas al sistema a través de system_lib o provocan excepciones, lo que provoca la ejecución del núcleo (K)

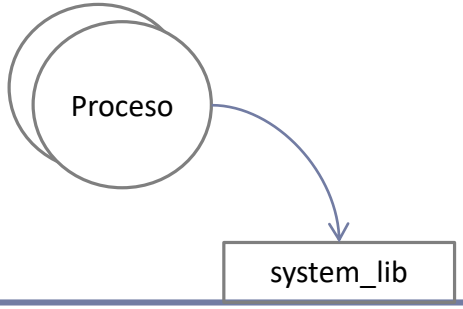


U

K

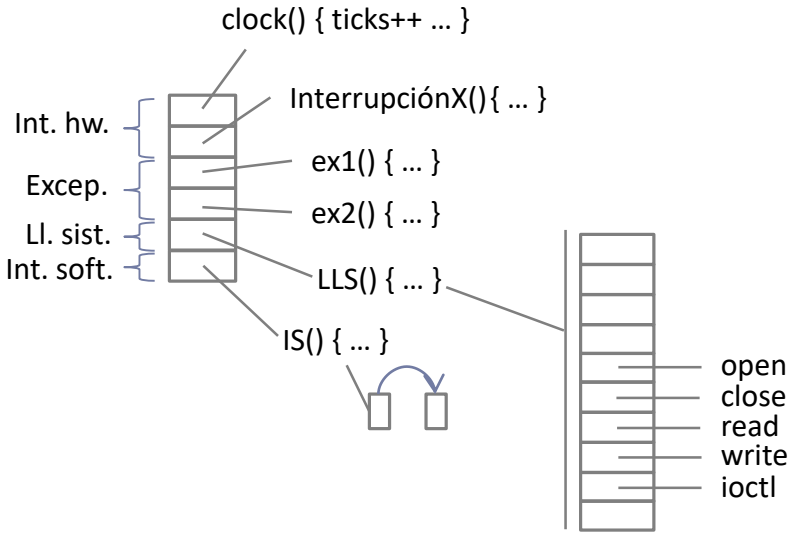
Ejercicio
solución

En el tema 2 se introducía el funcionamiento interno del núcleo del sistema operativo: interrupciones software, llamadas al sistema, excepciones e interrupciones hardware



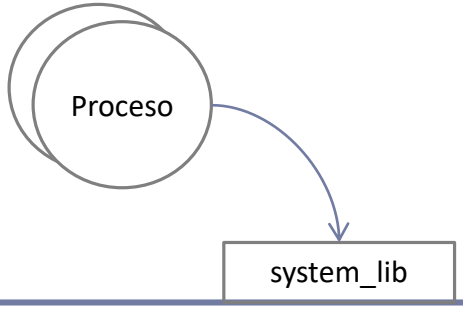
U

K



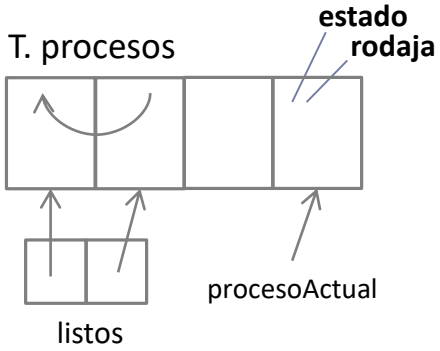
Ejercicio
solución

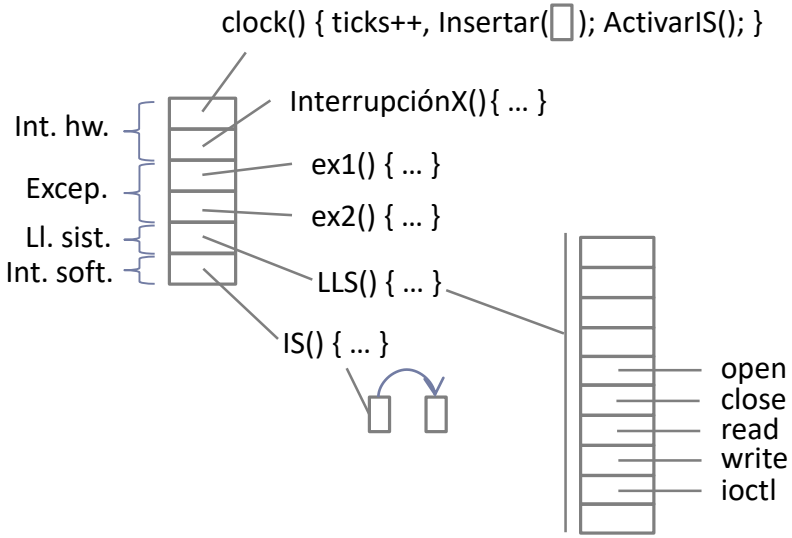
En el tema 3b se introducía las estructuras y funciones internas para la gestión de procesos, como la tabla de procesos, la cola de listos para ejecutar, el planificador, etc.



U

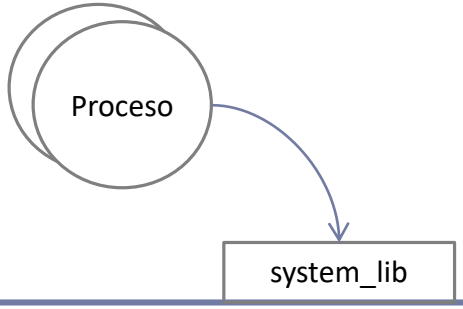
K





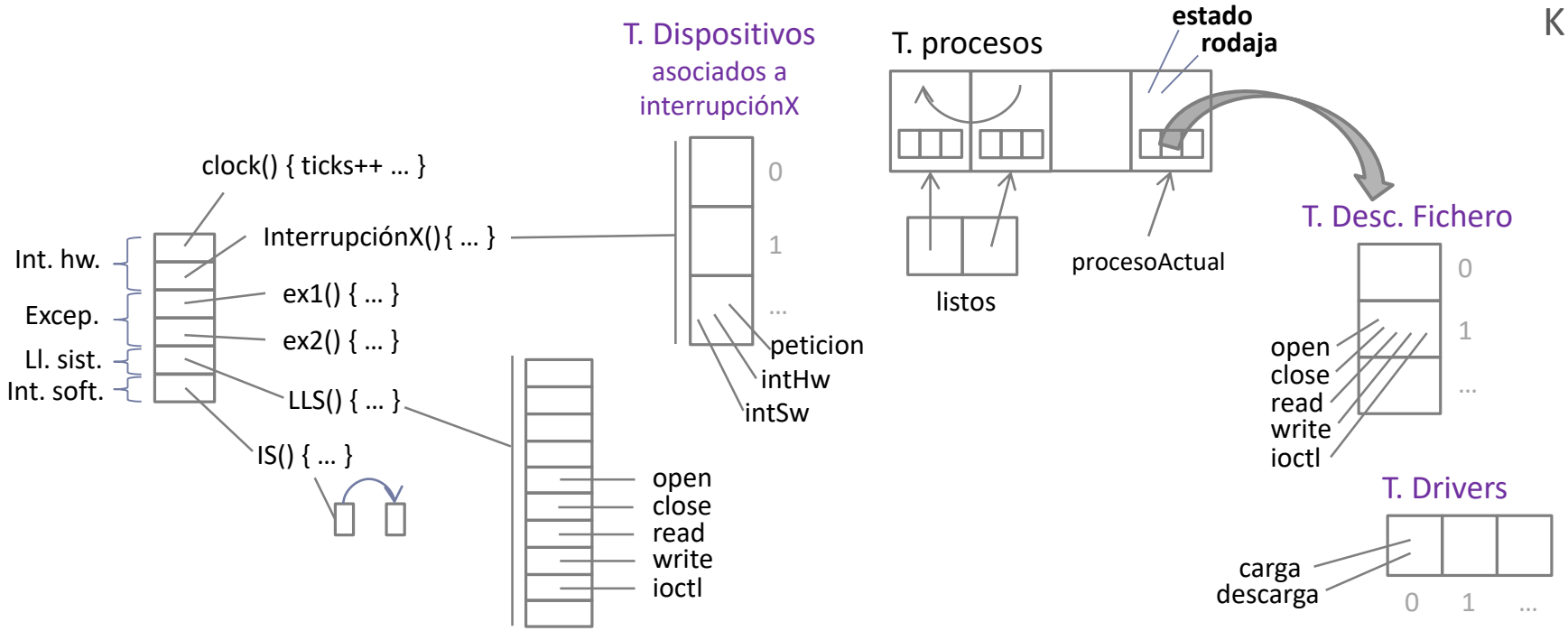
Ejercicio
solución

En el tema 3c añadimos tres tablas:
- Dispositivos: asociados a interrupciónX.
- Desc. de fichero: interfaz del dispositivo (1 tabla por proceso, en cada BCP).
- Drivers: carga y descarga.



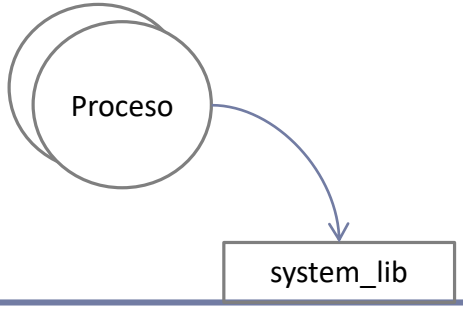
U

K



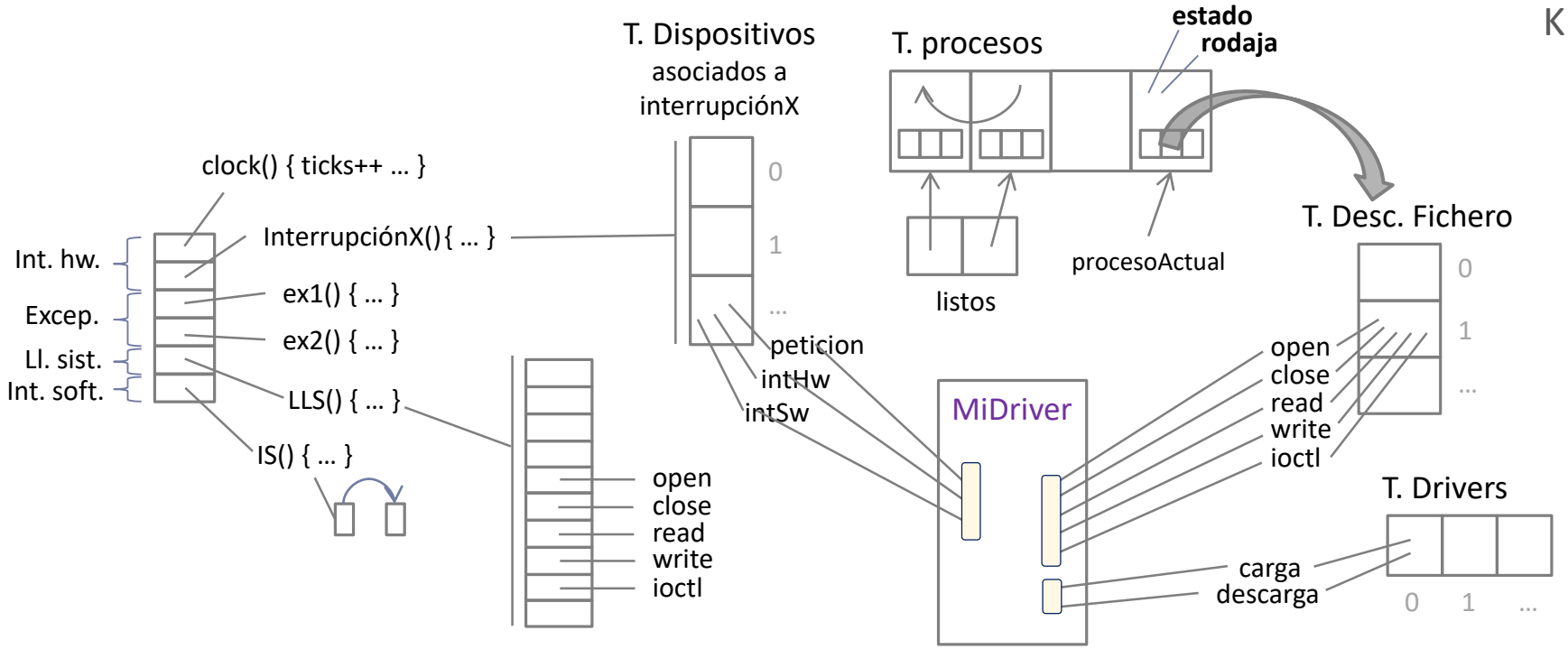
Ejercicio
solución

En el tema 3c creamos un driver como un fichero con, al menos, tres conjuntos de funciones.
Cada conjunto está asociado a una de las tres tablas comentadas antes.



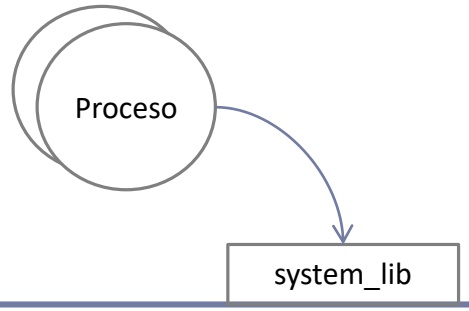
U

K



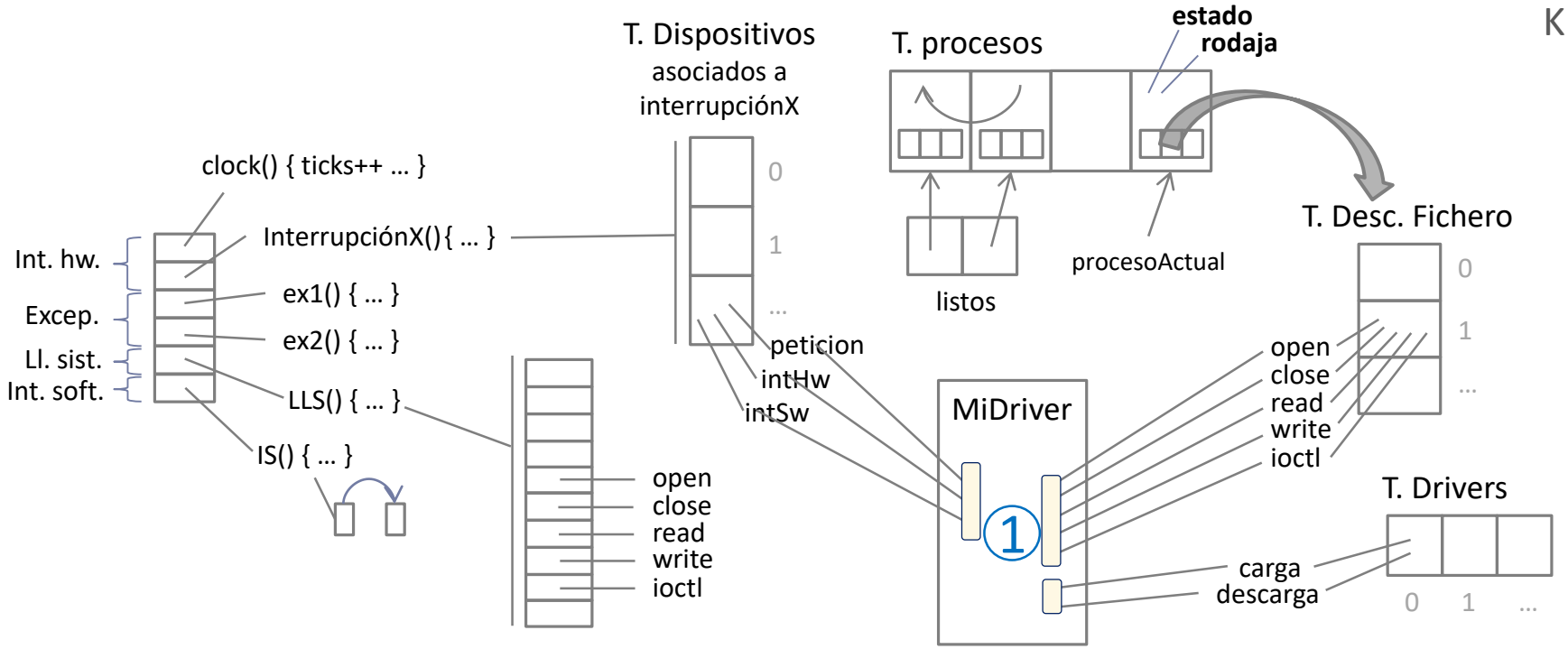
Ejercicio
solución

Apartado a)
Hay que detallar los tres conjuntos de funciones a implementar en el driver de teclado.



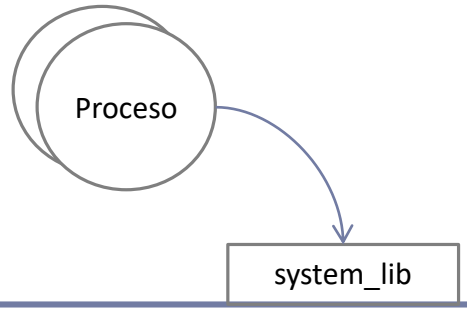
U

K



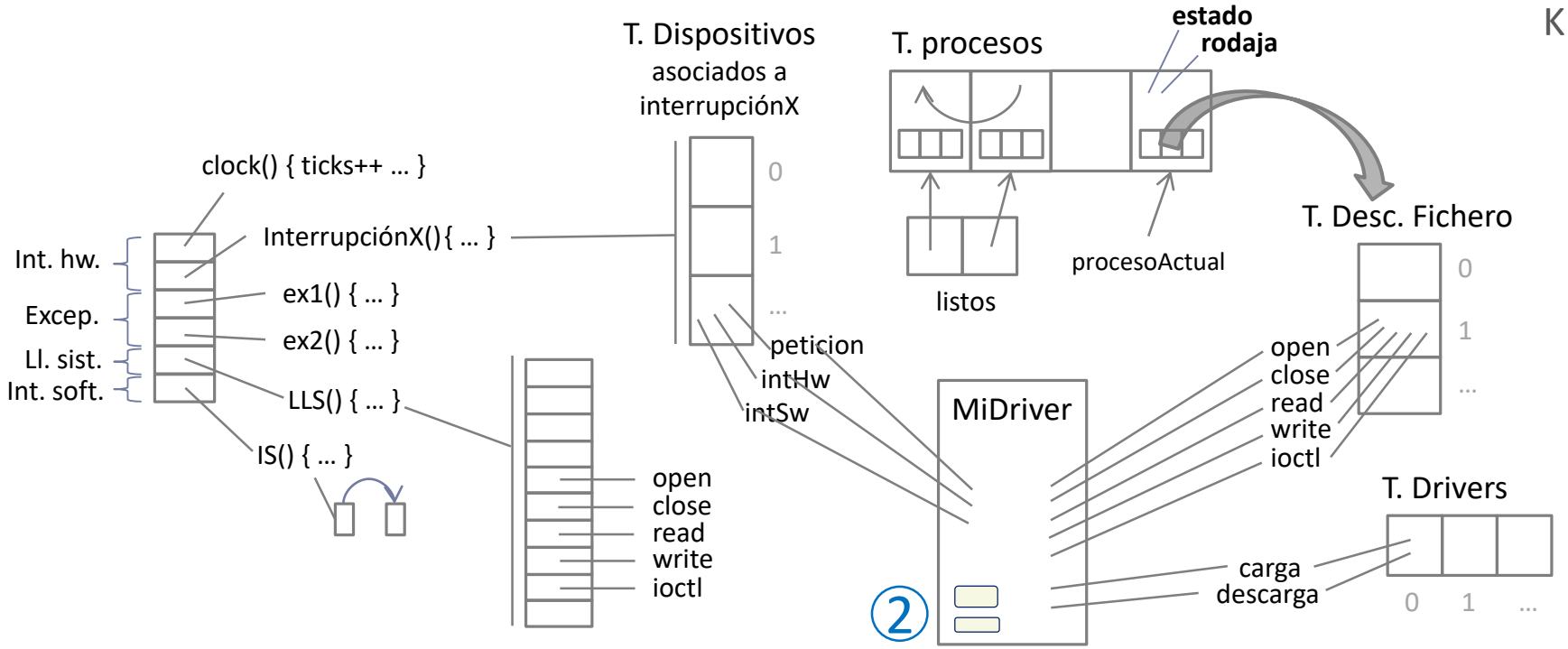
Ejercicio
solución

Apartado b)
Para este enunciado se precisa:
- Lista de teclas almacenadas
- Lista de procesos bloqueados



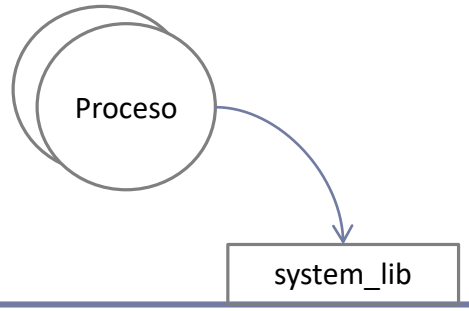
U

K



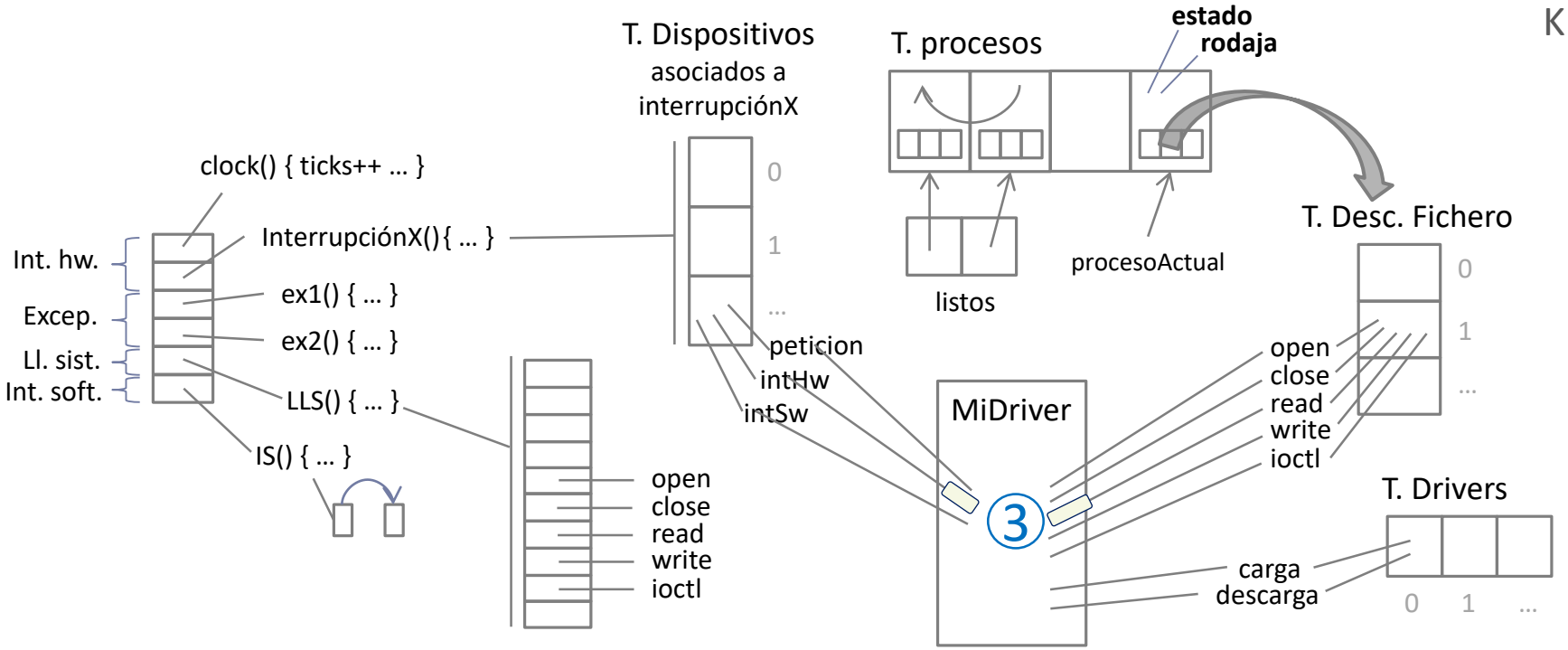
Ejercicio
solución

Apartado c)
El tratamiento de eventos afectado es:
- Llamada al sistema read
- Interrupción de teclado



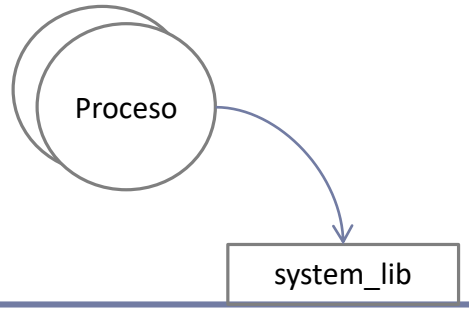
U

K



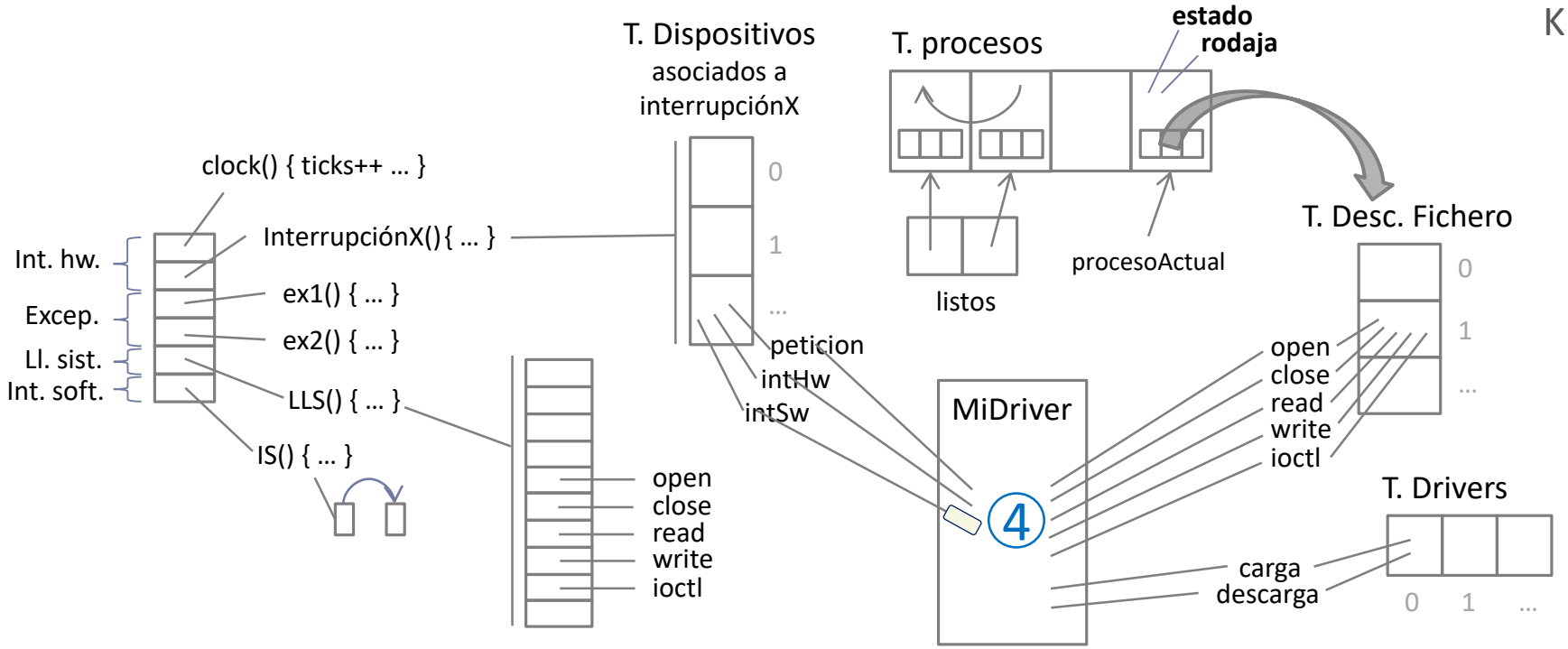
Ejercicio
solución

Apartado d)
- Necesario el uso de interrupción software
- Si kernel expulsivo, bloqueo de estructuras de datos (bloqueo interrupciones)
- Si kernel SMP, uso de *spinlocks*



U

K



Ejercicio
solución

1. Planteamiento inicial
    1. Estado inicial del sistema
    2. Estudio de qué hay que modificar
2. Responder a las preguntas
3. Revisar las respuestas

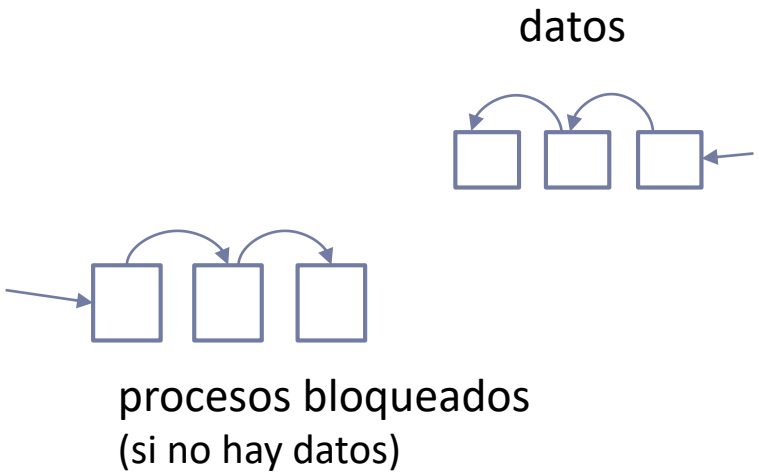
Ejercicio
solución a)

Mirando el planteamiento realizado, contestamos a las preguntas

- Gestionar la interrupción de teclado.
  - **Manejador_Interrupcion_teclado();**
- Gestionar las llamadas al sistemas del driver:
  - Uso de la opción de 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)**
      - Pone en el buffer la tecla leída y devuelve el número de bytes leídos.
- 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.

Ejercicio
solución b)

- 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)



Ejercicio
solución c)

Manejador de interrupción_teclado ():

- Insertar_tecla(tecla, KB1.BufferTeclas);
- Proc = ObtenerPrimerProceso (KB1.Bloqueados);
- Si Proc != NULL
  - Proc->estado = LISTO
  - InsertarAlFinal(Listos, Proc)

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 como proceso anterior el proceso actual.
  - Obtener el BCP del primer proceso de la lista de listos y asignarlo al 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;

Ejercicio
solución d)

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)
- Si Proc != NULL
  - Cambiar su estado de bloqueado a listo.
  - Incluirlo en la lista de procesos listos para ejecutar al final.

Llamada al sistema read():

- <misma que antes>

Ejercicio
solución d) (alternativa para read)

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

- For (i=0; i<size; i++)
  - Buffer[i] = PeticiónTecla()

Llamada al sistema PeticiónTela():

- ~~Si~~ Mientras (estaVacio(KB1.BufferTeclas))
  - Insertar_proceso_actual(KB1.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 y asignarlo al proceso actual.
  - Cambiar el estado del proceso actual a ejecutando.
  - Cambiar contexto entre proceso anterior y el proceso actual.
- Devolver extraer_tecla(KB1.BufferTeclas);

Ejercicio
solución

1. Planteamiento inicial
    1. Estado inicial del sistema
    2. Estudio de qué hay que modificar
2. Responder a las preguntas
3. Revisar las respuestas

Fallos a evitar

1) Contestar a la primera pregunta de un apartado únicamente (y no contestar al resto de preguntas/peticiones)
2) Contestar a otra pregunta de la pedida.
3) Respuestas largas:
   1) Quitan tiempo para realizar el resto del examen.
   2) Contestar más de lo pedido puede suponer fallos extra.
   3) Importante que las partes claves del ejercicio estén correctas.
4) Usar el planteamiento del problema como respuesta.

Ejercicios

drivers y servicios ampliados

Grupo ARCOS

Diseño de Sistemas Operativos

Grado en Ingeniería Informática

Universidad Carlos III de Madrid