Exercises
Keyboard driver

ARCOS

Operating Systems Design
Degree in Computer Engineering
University Carlos III of Madrid

Exercise
statement (1/2)

We have a uniprocessor machine and want to implement a keyboard driver for a UNIX-like operating system with a monolithic non-preemptive kernel with the following functionality:

- To manage interrupts when keyboard is pressed.
- To offer users a way to get pressed keys (blocking if there are not).
- The driver can be loaded and unloaded at run time.

The driver must store the keys temporarily if no processes requested them.

Exercise
statement (2/2)

You are asked to:

a) Design both the internal (kernel) and the external interface, system calls, etc. required in the driver.

b) Define the data structures needed to perform the required functionality.

c) Implement in the pseudocode used in the class exercises the functionality for getting the keys from the keyboard and send them to user processes. Which events are involved?

Exercise
solution

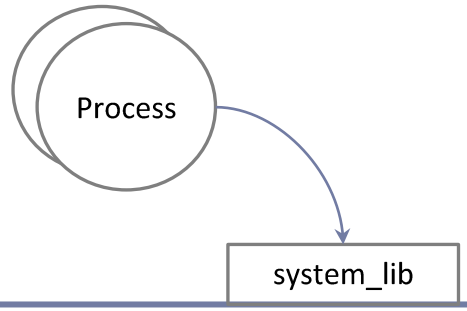
1. Initial approach
   A. Initial state of the system
   B. Study what must be modified
2. Answer the questions
3. Review the answers

Exercise
solution

1. Initial approach
   A. Initial state of the system
   B. Study what must be modified
2. Answer the questions
3. Review the answers

Exercise
solution

In user space (U) we have the processes that make system calls through the system_lib or cause exceptions, which causes the execution of the kernel (K)

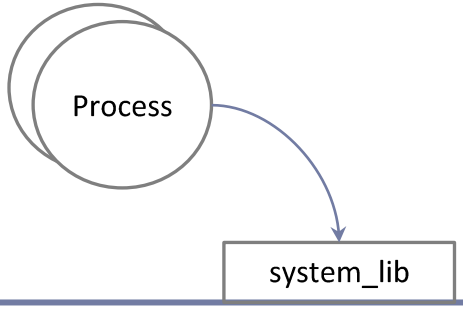


U

K

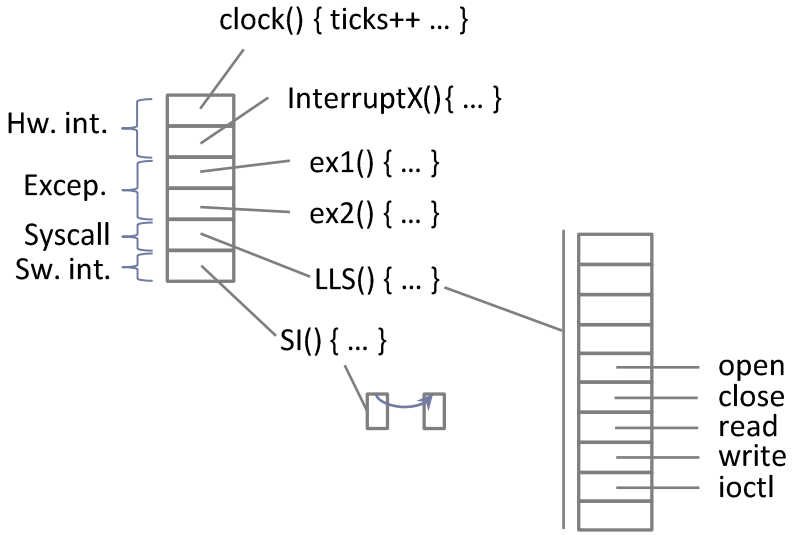
Exercise
solution

In lesson 2, the internal operation of the operating system kernel was introduced: software interrupts, system calls, hardware exceptions and interruptions



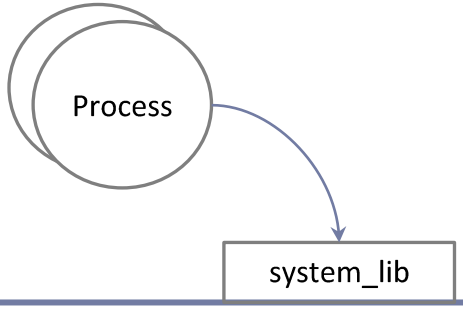
U

K



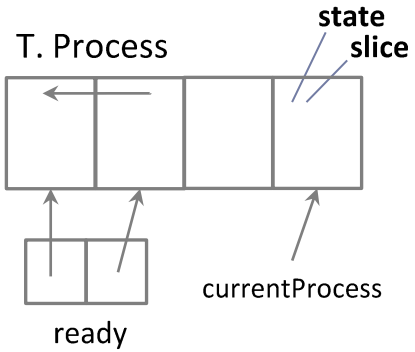
Exercise
solution

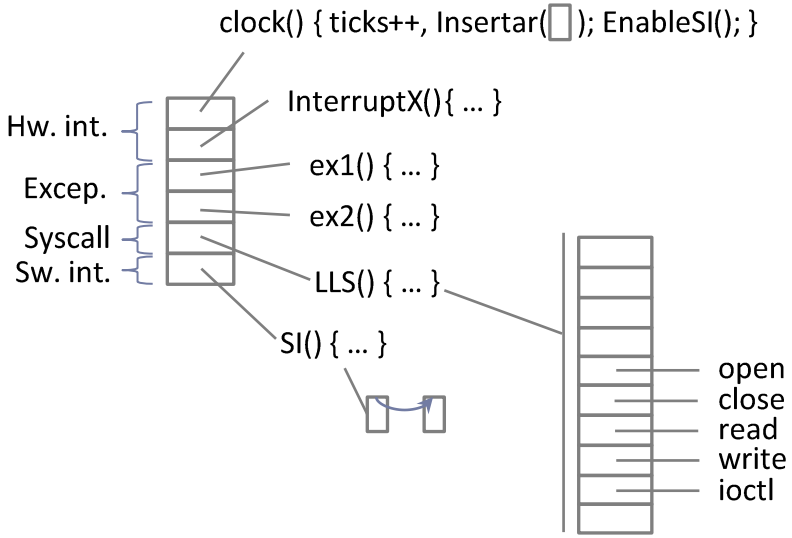
In lesson 3, the internal structures and functions for the process management were introduced, such as the process table, the ready to execute queue, the scheduler, etc.



U

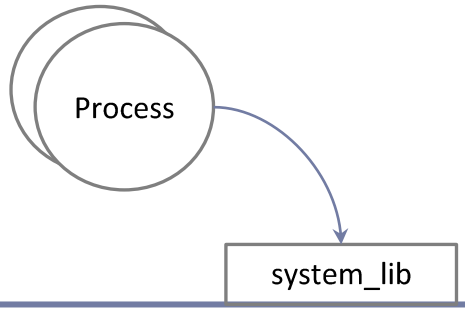
K





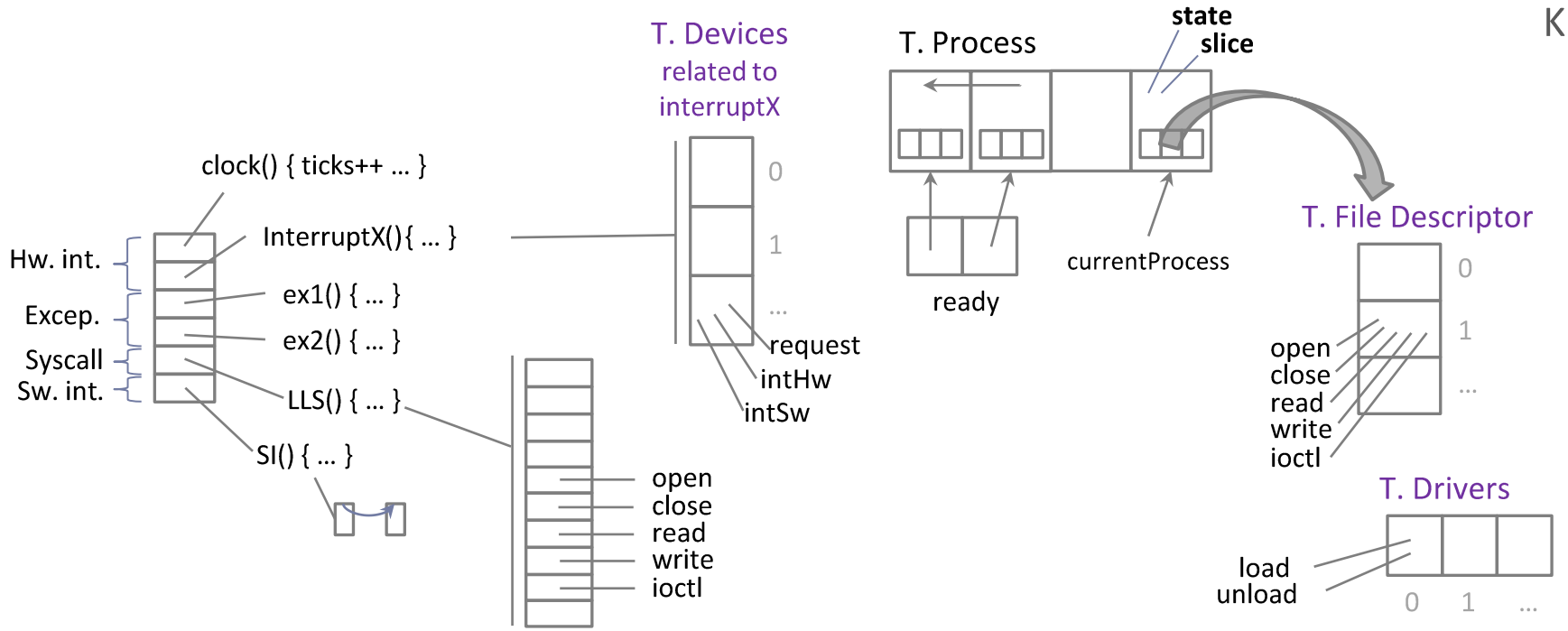
Exercise
solution

In lesson 4 we add three tables:
- Devices: associated with interruptX.
- File descriptor: upper device interface (1 table per process, in each PCB).
- Drivers: load and unload.



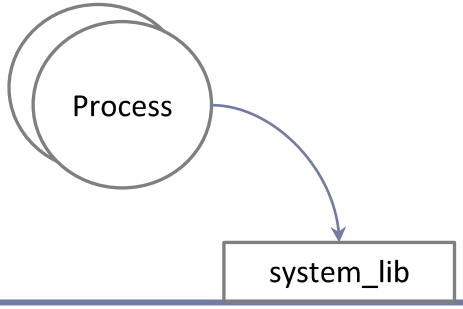
U

K



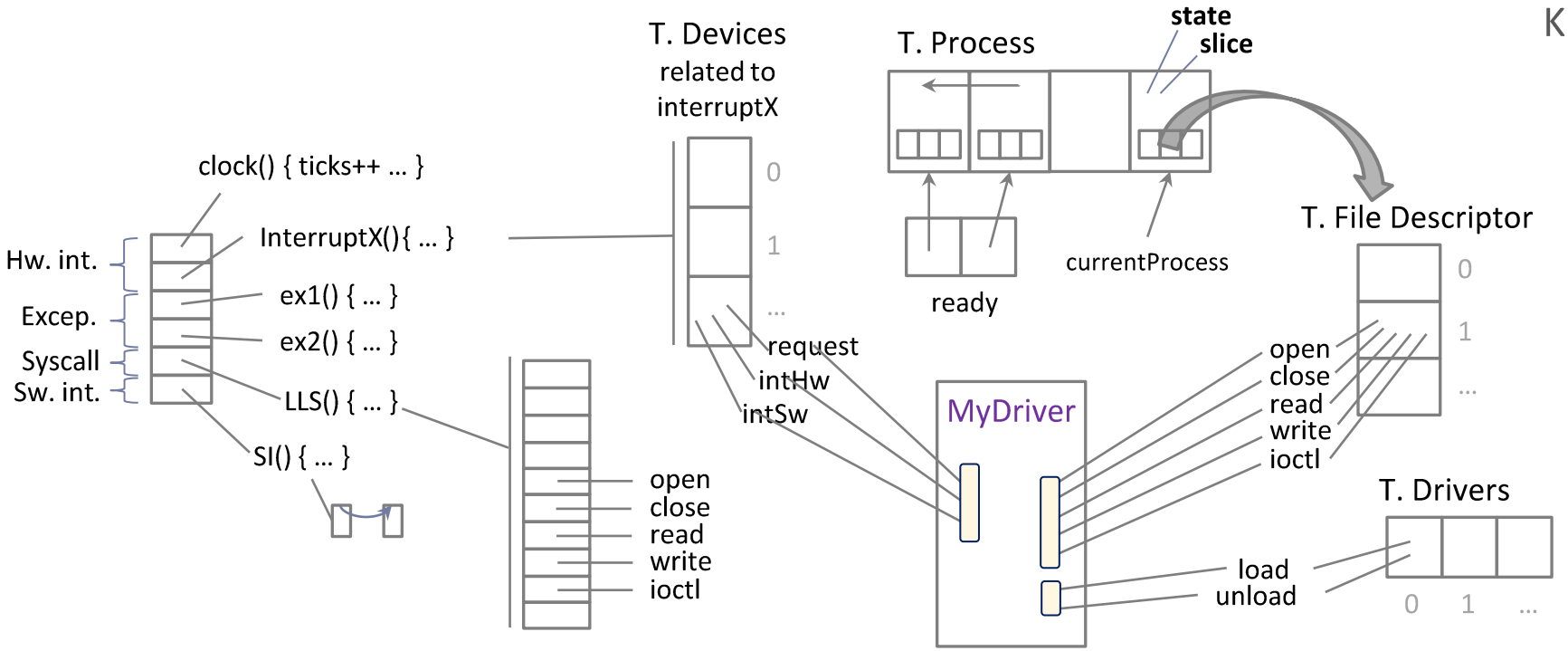
Exercise
solution

In lesson 4, we will create a driver with at least three sets of functions. Each set is related to one of the three tables discussed before.



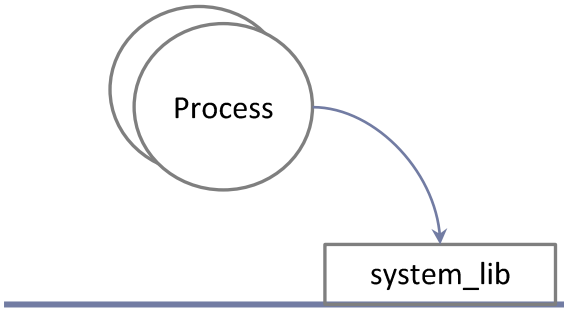
U

K



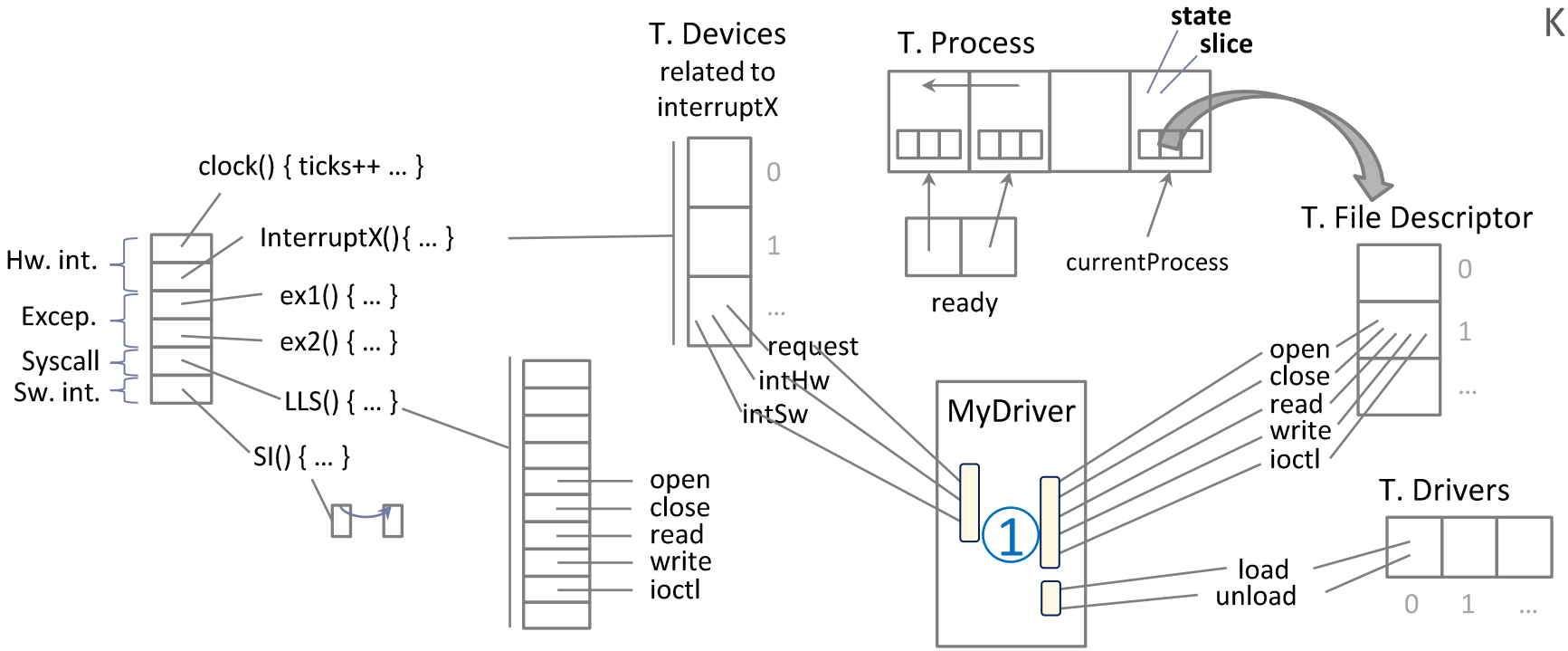
Exercise
solution

Section a)
It is necessary to detail the three sets of functions to be implemented in the keyboard driver.



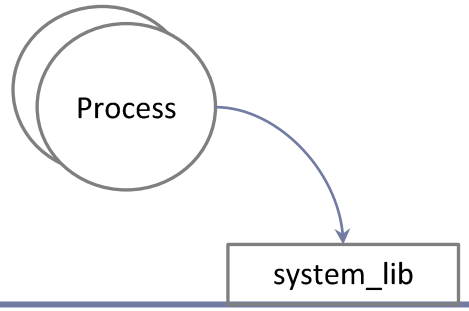
U

K



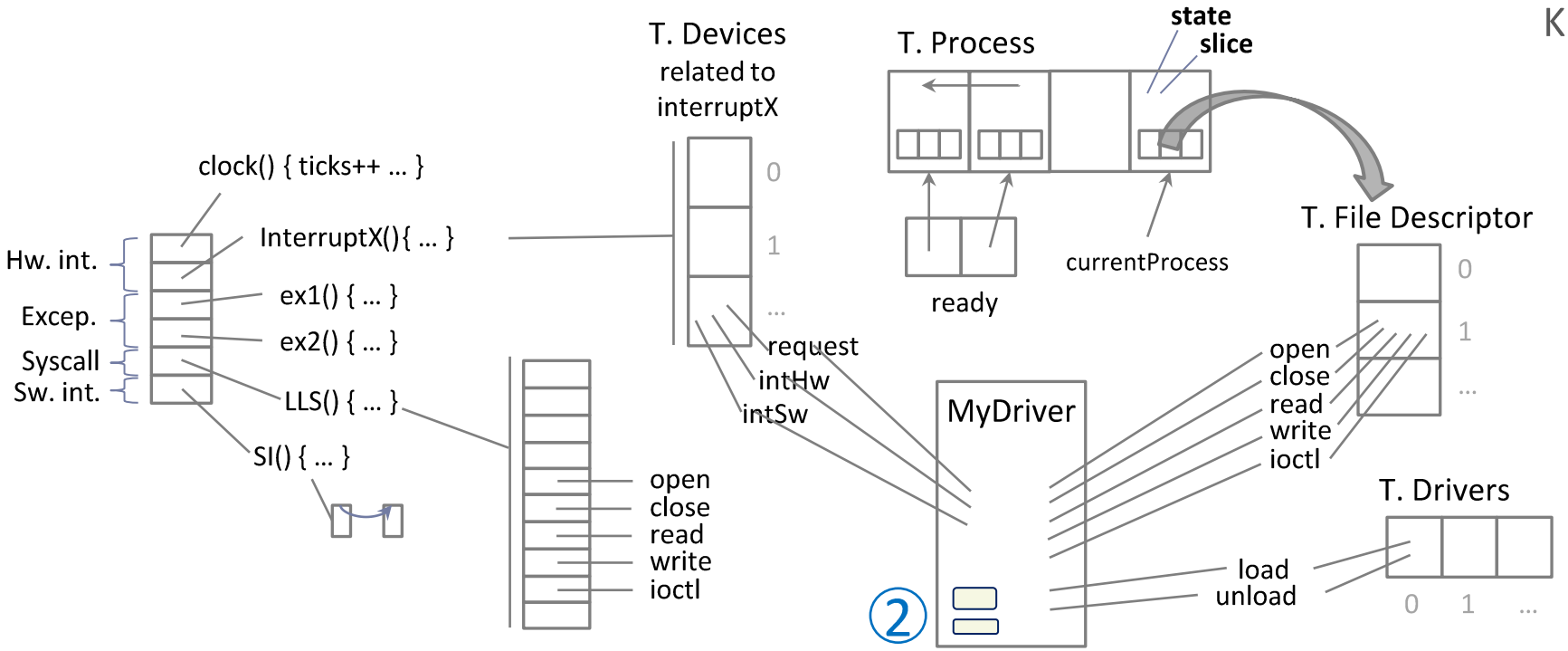
Exercise
solution

Section b)
For this exercise it is necessary to include two lists:
- List of stored keys
- List of blocked processes



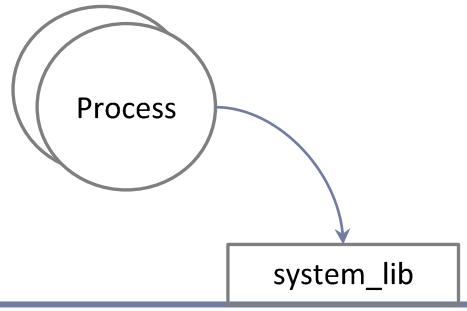
U

K



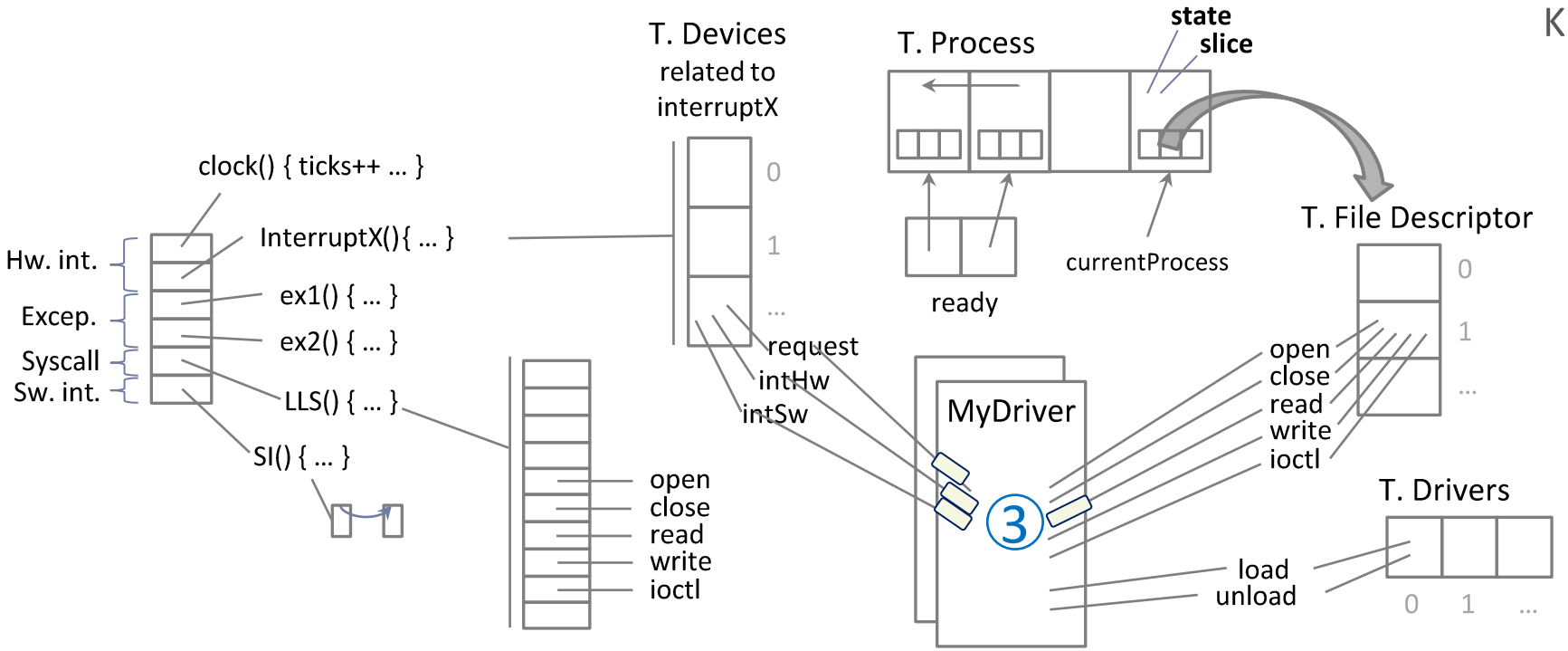
Exercise
solution

Section c)
The affected events of keyboard are:
- System call read
- Keyboard interrupt
- Request function



U

K



Exercise
solution

1. Initial approach
   A. Initial state of the system
   B. Study what must be modified
2. Answer the questions
3. Review the answers

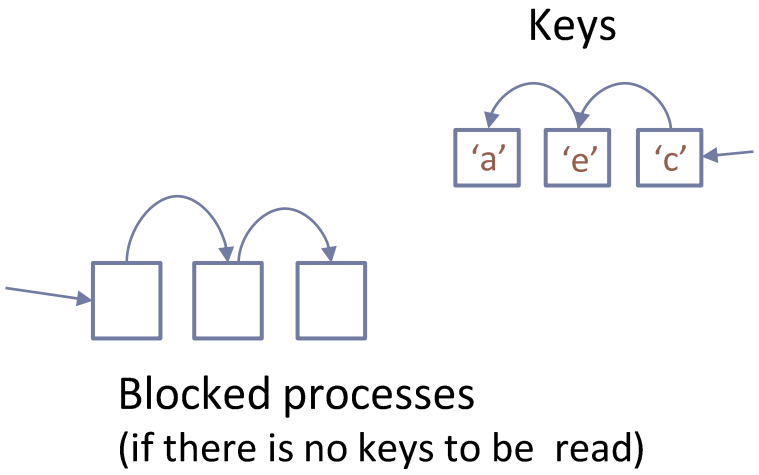
Exercise
solution a)

We look up the previous summary, and we answer the questions.

- Manage the keyboard interrupt and the request function:
  - **keyboard_int_handler();**
  - **keyboard_get_char()**
- Manage system calls for the driver (using the UNIX standard):
  - **Desc = keyboard_open (kbd_name, flags)**
    - Allows to allocate the access to the keyboard
  - **Res = keyboard_close (Desc)**
    - Release the access to the keyboard
  - **Res = keyboard_read (Desc, buffer, size)**
    - Insert the readed key in the buffer and return the number of readed bytes.
- Manage load and unload the driver at run time:
  - **keyboard_load();**
    - Insert the driver structure (variables and operations) in the corresponding tables.
    - Link the driver object file in the kernel memory space.
  - **keyboard_unload();**
    - Remove the driver structure from the corresponding tables.

Exercise
solution b)

- Data structures required:
  - List of stored keys (keyboard.BufferKeys)
  - List of blocked process (keyboard.Blocked)



Exercise
solution c)

keyboard_int_handler ():

- key = In("device id")
- Insert (key, keyboard.BufferKeys);
- Insert_Software_Interrupt(keyboard_sw_int);
- Enable_Software_Interrupt();

keyboard_sw_int ():

- Proc = getFirstProcess (keyboard.Blocked);
- Si Proc != NULL
  - Proc->state = READY
  - InsertAtEnd (ReadyList, Proc)

Exercise
solution c)

keyboard_read(fd, buffer, size):

- for (int i=0; i<size; i++)
  - Buffer[i] = keyboard_get_char()
- Return size;

keyboard_get_char():

- If (isEmpty(keyboard.BufferKeys))
  - Insert_current_process (keyboard.Blocked);
  - Change the status of the current process from executing to blocked.
  - Get the BCP process first list ready.
  - Put this BCP in the list of running.
  - Change between current process context and new process running.
- return extract_key(keyboard.BufferKeys);

Exercise
solution

1. Initial approach
   A. Initial state of the system
   B. Study what must be modified
2. Answer the questions
3. Review the answers

Typical failures

1) Answer the first question of a section only.
2) Answer another question (not requested).
3) Answer more than what is asked:
    1) If the extra part is wrong, it may be evaluated…
4) To use the initial/informal approach as answer.

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

Exercises

Operating system Internals

Operating System Design
Grado en Ingeniería Informática y Doble Grado I.I. y A.D.E.

cc BY NC SA