Exercises 2

Process scheduling

ARCOS

Operating Systems Design

Degree in Computer Engineering

University Carlos III of Madrid

Exercise
statement (1/3)

The aim is to develop a priority-based scheduler. Processes can own two different priorities:

- High priority.
- Low priority.

Each priority has its own scheduling policy:

- **High priority** processes will be scheduled following a **FIFO** policy.
- **Low priority** processes will be scheduled following a **Round-Robin** policy. Time slice will be **100 milliseconds**.

Exercise

statement (2/3)

Processes of **high priority** will be executed following an strict order of arrival (FIFO). A high priority process executes until:

- It finishes.
- It sleeps (through *sleep()* syscall).
- It gets blocked (due to an I/O operation).

Processes of **low priority** abandon the CPU when:

- Its time slice ends.
- It finishes.
- It sleeps (through *sleep()* syscall).
- It gets blocked (due to an I/O operation).

Exercise

statement (3/3)

Requirements:

a) Design a solution indicating what functions and structures are necessary to implement the requested scheduler.

Exercise
solution

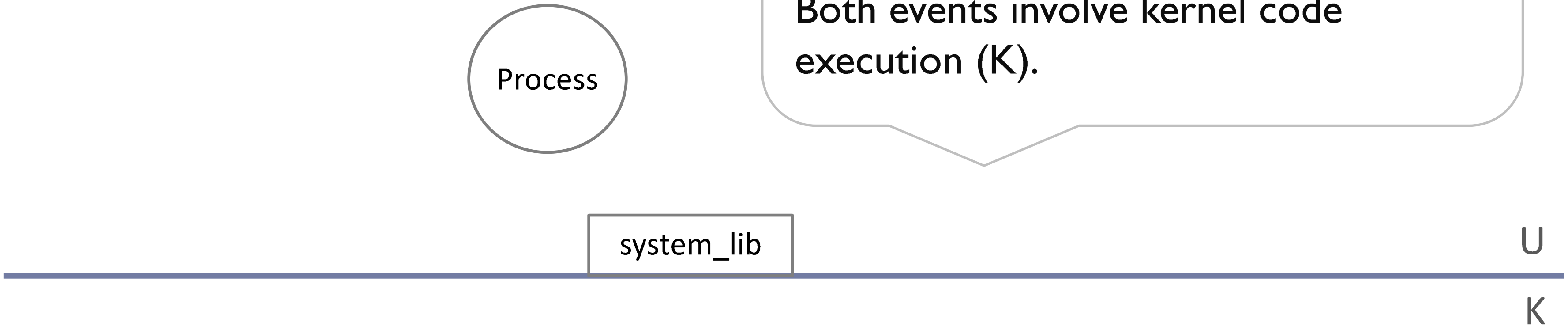
1. Starting approach
   1. Operating system structure
   2. Analysis of modifications
2. Answer the questions
3. Review the answers

Exercise
solution

1. Starting approach
   1. Operating system structure
   2. Analysis of modifications
2. Answer the questions
3. Review the answers

Exercise
solution

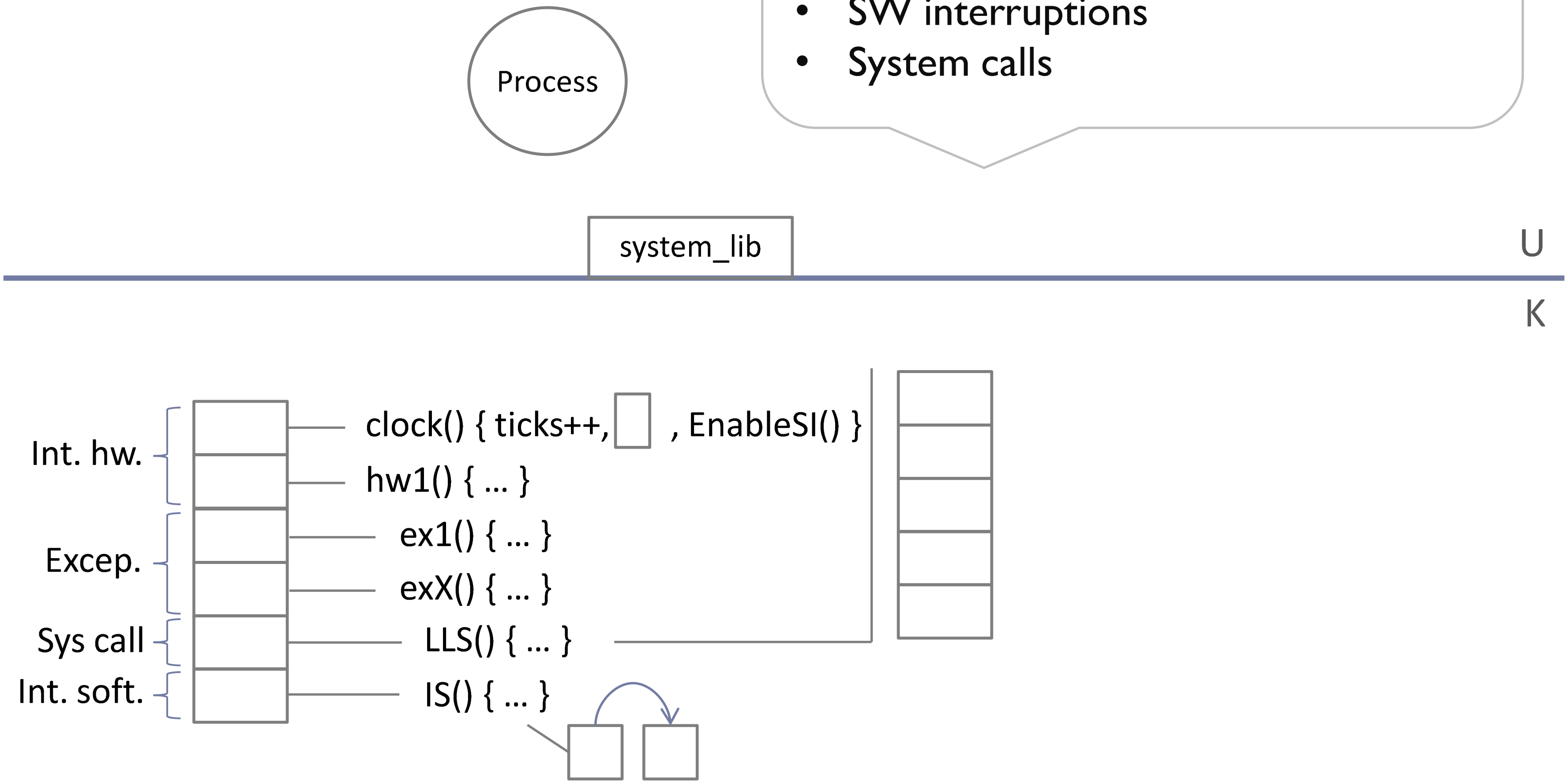
In user space (U) processes perform system calls through *system_lib* or provoke exceptions.
Both events involve kernel code execution (K).



Exercise
solution

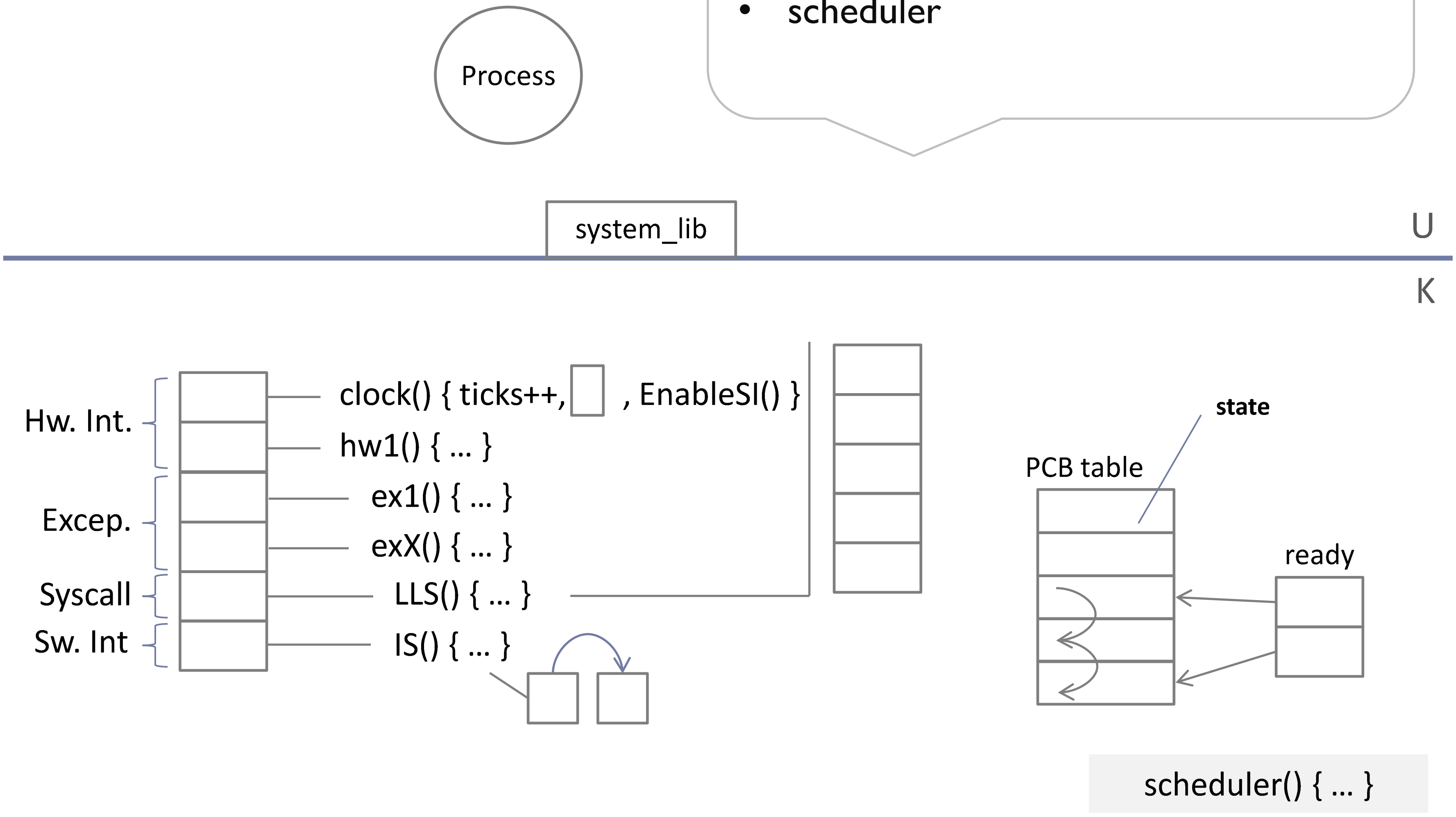
Topic 2: operating system working

- HW interruptions
- Exceptions
- SW interruptions
- System calls



Exercise
solution

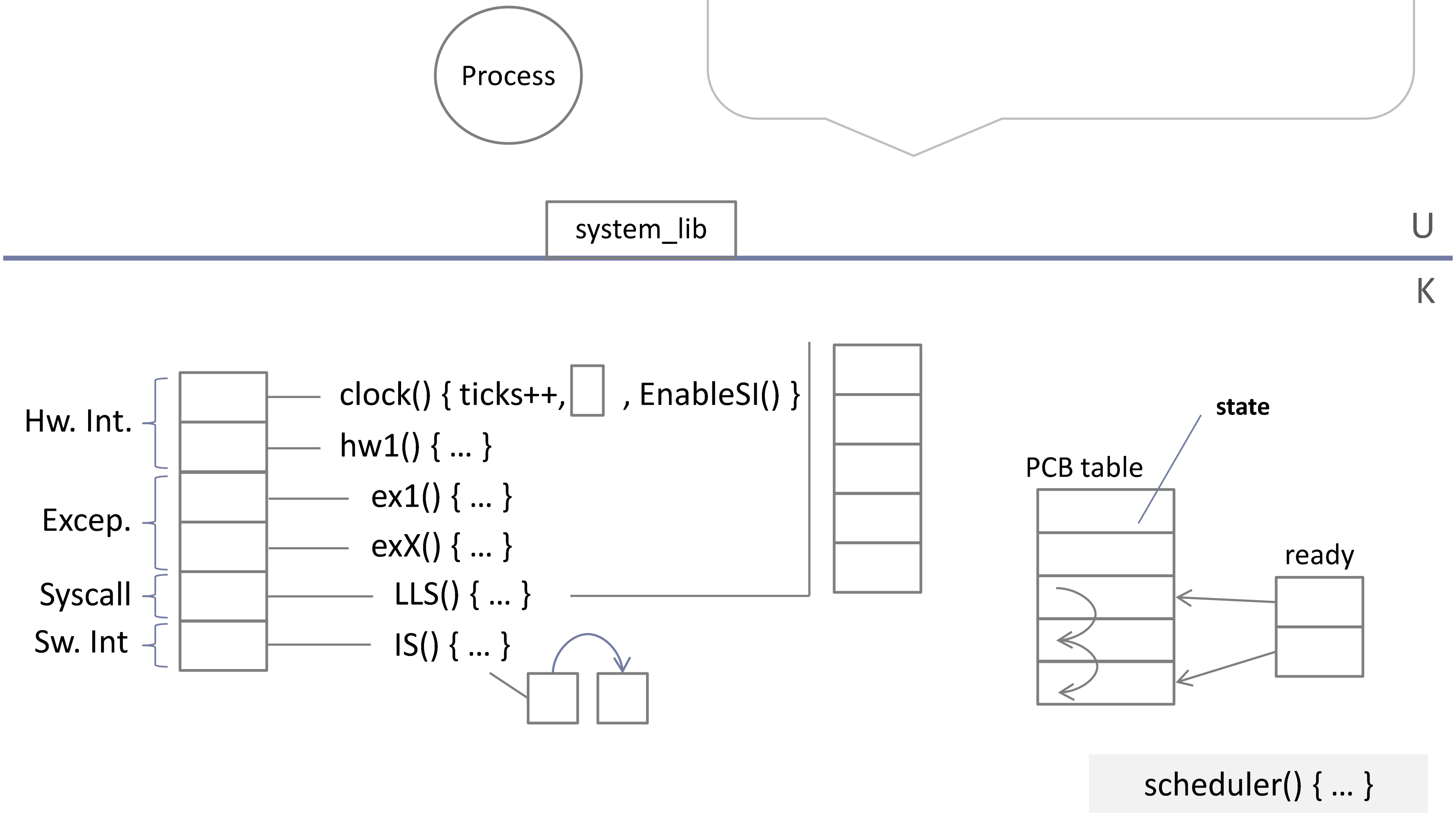
Topic 3: process management
- PCB table
- Ready-state queues
- scheduler



Exercise

solution

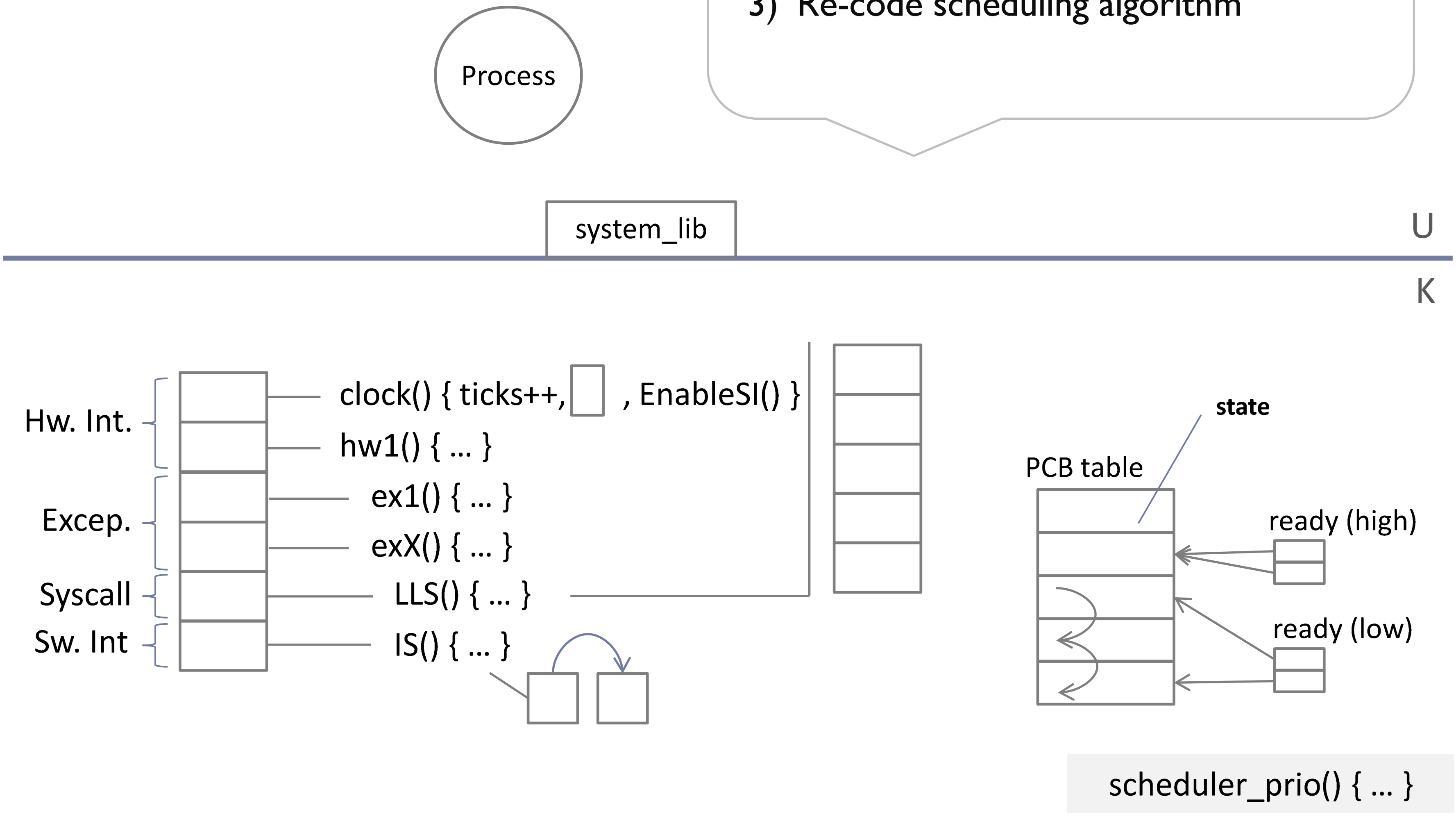
Initial structure completed



Exercise
solution

Adding priorities to the kernel:

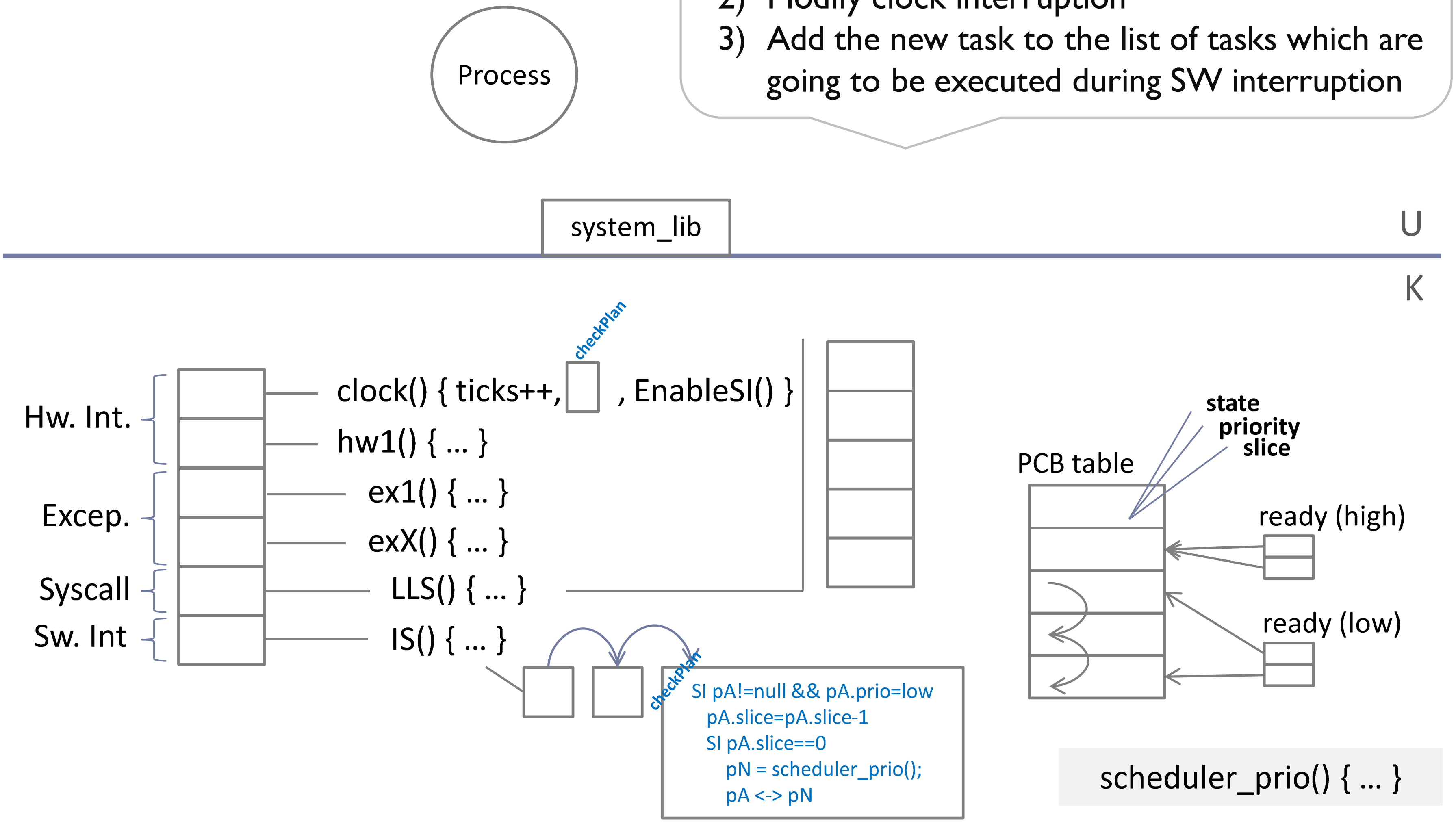
1) Add 'priority' field to PCB
2) Two ready state queues instead of one
3) Re-code scheduling algorithm



Exercise
solution

Adding Round-Robin:

1) Add 'slice' field to PCB
2) Modify clock interruption
3) Add the new task to the list of tasks which are going to be executed during SW interruption



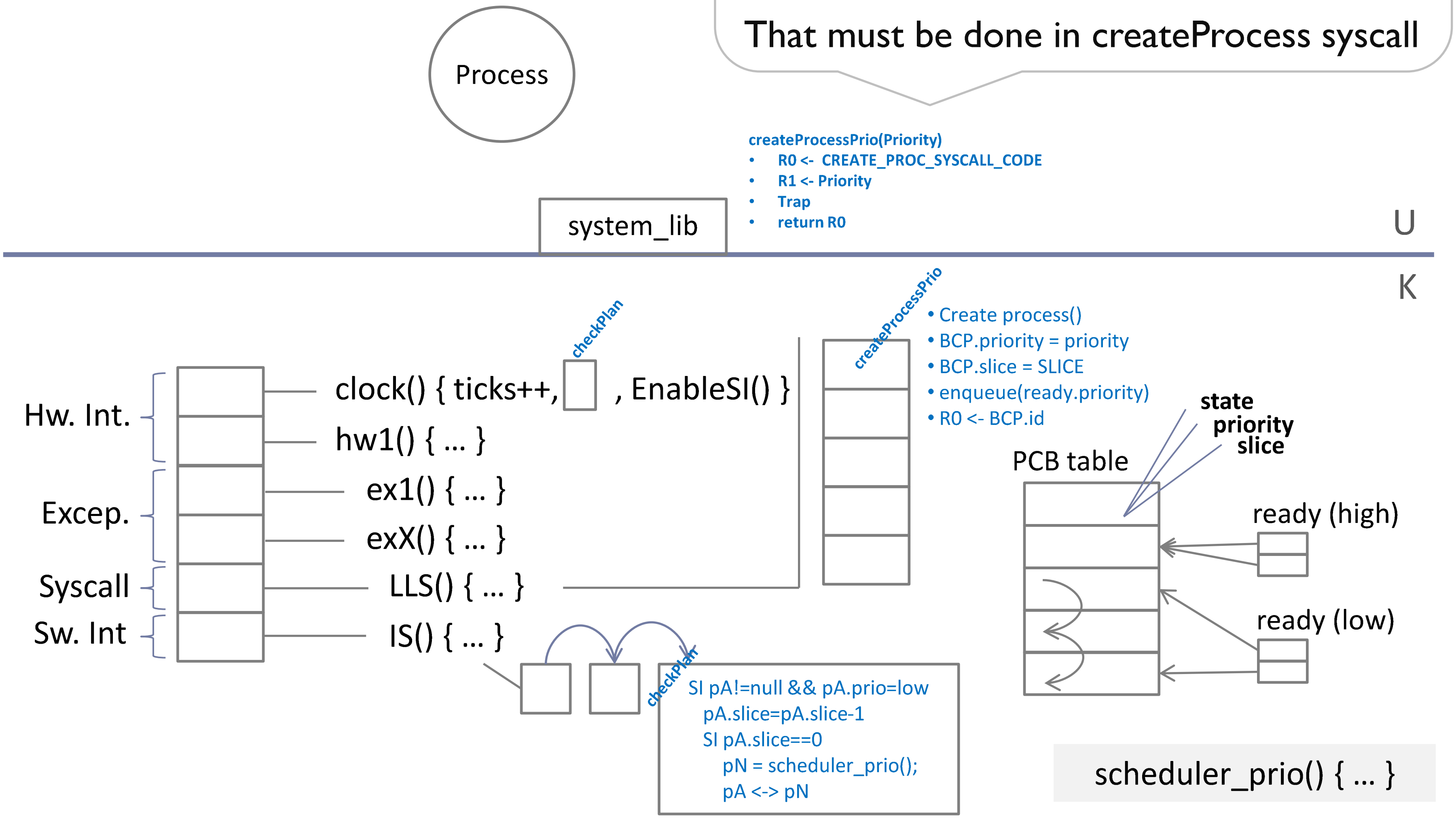
```
SI pA!=null && pA.prio=low
  pA.slice=pA.slice-1
  SI pA.slice==0
     pN = scheduler_prio();
     pA <-> pN
```

Exercise
solution

Process creation:

- Setting initial values of priority, and slice
- Queuing in the corresponding queue.

That must be done in createProcess syscall



Process

system_lib

createProcessPrio(Priority)

- R0 <- CREATE_PROC_SYSCALL_CODE
- R1 <- Priority
- Trap
- return R0

createProcessPrio

- Create process()
- BCP.priority = priority
- BCP.slice = SLICE
- enqueue(ready.priority)
- R0 <- BCP.id

checkPlan

```
SI pA!=null && pA.prio=low
  pA.slice=pA.slice-1
  SI pA.slice==0
     pN = scheduler_prio();
     pA <-> pN
```

scheduler_prio() { … }

Exercise
solution

1. Starting approach
   1. Operating system structure
   2. Analysis of modifications
2. Answer the questions
3. Review the answers

Exercise
solution

Basing on the proposed approach, answer the questions

Data structures:

- PCB:
  - Priority
  - Slice
- Implement two ready state queues instead of one:
  - Low priority processes
  - High priority processes

Exercise

solution

Functions:

scheduler_prio()

```
If non_empty(read_state_queue_high_priority)
    Proc=GetFirstProcess(read_state_queue_high_priority)
    Remove(read_state_queue_high_priority,Proc)
Else // empty queue
    Proc=GetFirstProcess(read_state_queue_low_priority)
    Remove(read_state_queue_low_priority,Proc)
Return Proc
```

Exercise

solution

clock_interruption_handler()

- Ticks = Ticks + 1;
- Insert_Software_Interruption(checkPlan)
- Software_Interruption();

Exercise

solution

```
checkPlan()
  • Si ( (current == null) || (current.priority == high))
      • return
  • current.slice = current. slice - 1
  • Si (current.slice == 0)
      • current.slice = TICKS_PER_SLICE  // == 100 milliseconds
      • current.state = ready
      • enqueue(ready_state_queue_low_priority, current)
      • Proc=scheduler_prio()
      • Proc.state = execution
      • current=Proc
      • swapContext(current.context_t, Proc.context_t)
```

Exercise
solution

Functions on user space:

int createProcessPrio(priority):

- R0 = CREATE_PROC_PRIO_SYSCALL_CODE
- R1 = priority
- Trap
- Return R0

Exercise
solution

System call create_process(priority)

- Create process ()

- PCB.priority = *priority*
- PCB.slice = SLICE // makes sense only if priority==low
- PCB.state = READY

- If (*priority* == high)
    - Enqueue process in ready state queue (high priority)
- Else
    - Enqueue process in ready state queue (low priority)

- R0 = PCB.id

Exercise

solution

1. Starting approach
    1. Operating system structure
    2. Analysis of modifications
2. Answer the questions
3. Review the answers

Exercises 2

Process scheduling

ARCOS

Operating Systems Design

Degree in Computer Engineering

University Carlos III of Madrid