SE350 – Project Overview

Second Part

Bo Zhu
bo.zhu@uwaterloo.ca

Winter 2013

Outline

1. Requirements and Assumption
2. Process
3. Scheduling
4. Initialization

1. Basic Requirements & Assumptions

Basic Requirements:

- Multiprogramming (processes)
- Pre-emptive and fixed priority scheduling
- Memory management (i.e., request/release memory blocks)
- Message-based inter-process communication (i.e., send/receive messages)

Assumptions:

- All processes are known (created at start-up; know each other)
- Processes are non-malicious

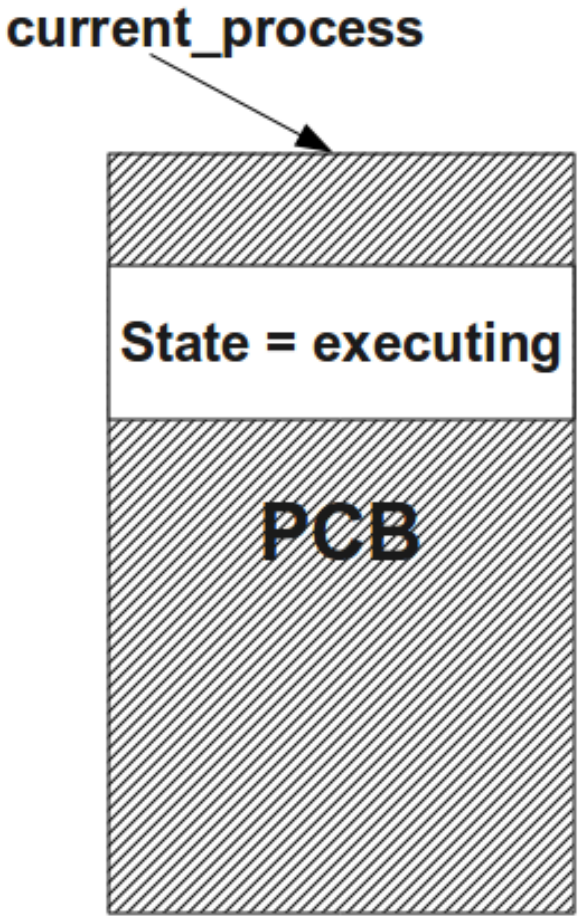
2.1. Process Data Structure

Process control block (PCB):

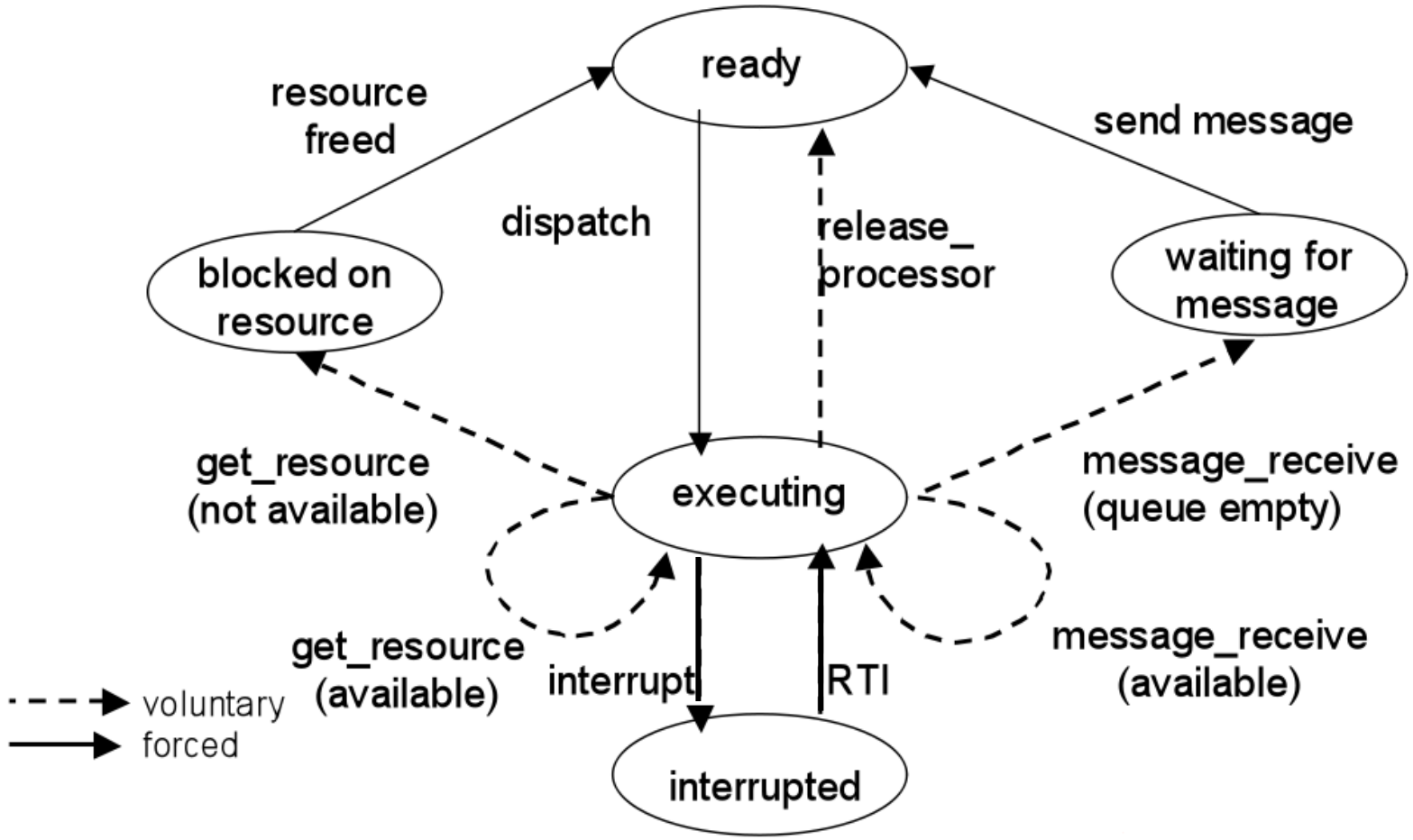
- Needed for each process
- Describing process status and context

current_process variable:

- which always refers to PCB of currently executing process
- OS must know which process is currently executing



2.2. Process States



2.3. Process Switching

Process switching:

1. Select next process to execute using scheduler
2. Invoke context switch to new process

Context switching:

1. Save context of currently executing process
2. Change the process's state back to READY
3. Update `current_process` to new process
4. Set state of new process to RUNNING
5. Restore context of `current_process`
6. Execute `current_process`

3. Scheduling

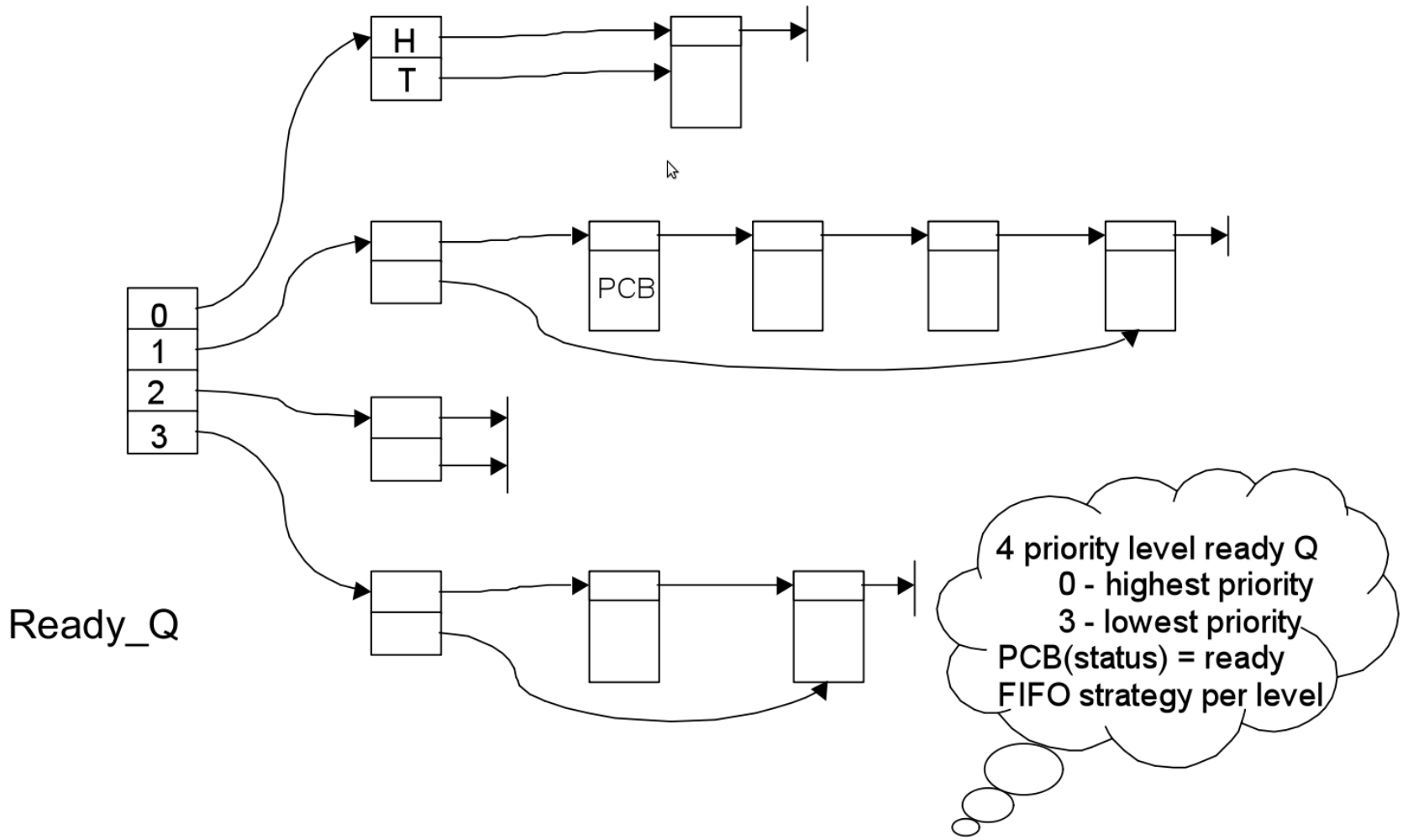
Requirements:

- Fixed, priority-based scheduling
- Each process has assigned priority
  - Highest priority process executes first
  - First-come-first-serve for processes of same priority

Procedure:

1. `process_switch()` invokes scheduler
2. Scheduler selects highest-priority ready process
3. `context_switch(new_proc)` lets the selected process execute

3. Scheduling
Ready Queues for 4 Priorities



3. Scheduling: Null Process

- CPU must execute something
- What to do when ready queues are empty?
  - Possible solution: NULL process
  - NULL has lowest priority and is always ready to run
- Basic example

```
void null_process() {
    while(1) {
        release_processor();
    }
}
```

3. Scheduling: release_processor()

`release_processor()`

1. Set `current_process` to state READY
2. `rpq_enqueue(current_process)`
   put current_process in ready queues
3. `process_switch()`
   invokes scheduler and context-switches to the new process

4. Initialization

- What operations need to be carried out at start-up?
- Initialize all hardware, incl.
  - Serial port(s) and timer(s)
  - Memory mapping (memory allocation for mem blocks and stacks…)
  - Interrupts (hardware and software: vector table & traps)
- Create all kernel data structures
  - PCBs (status=ready), queues…
  - Place PCBs into respective queues

4. Initialization: Initialization Table

How does RTX know which processes to create?

Pre-defined Initialization Table:

- Array of records
- Each record contains spec of its process and additional data structures