Operating Systems

Lecture 4

Processes

Summary

- This lecture shows processes and their managment:
  - Process.
    - Process memory organization
    - Process states and transition diagram
    - Process creation and termination
  - Process Control Block (PCB)
    - Context switching
  - Threads in Linux and in Java
  - Process scheduling
    - Long-term scheduler.
    - Short-term scheduler.
      - FIFO, SJF, SRTF, RR

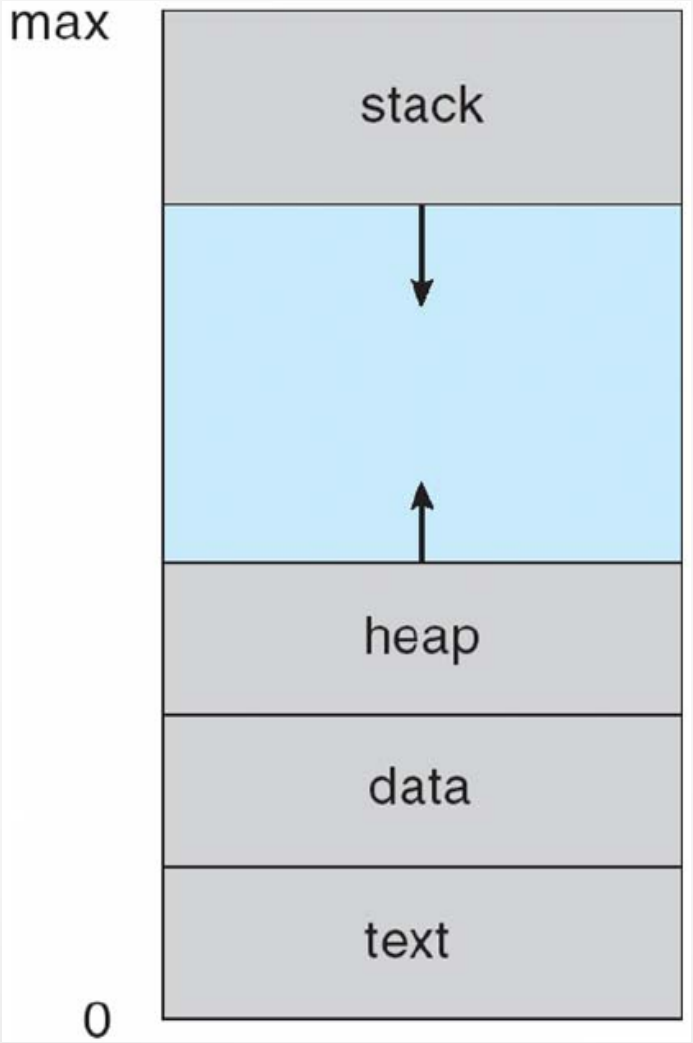
Summary

- This lecture shows processes and their managment:
  - Process.
    - Process memory organization
    - Process states and transition diagram
    - Process creation and termination
  - Process Control Block (PCB)
    - Context switching
  - Threads in Linux and in Java
  - Process scheduling
    - Long-term scheduler
    - Short-term scheduler
      - FIFO, SJF, SRTF, RR

Process

- **PROGRAM**: a piece of code stored on a dysk. Program is a passive entity.
- **PROCESS**: a program, which is running. Processes are active and there can be several processes at a time, which are created from the same program code.
- A **PROGRAM** becomes a **PROCESS** when it is loaded into memory.

Memory organization

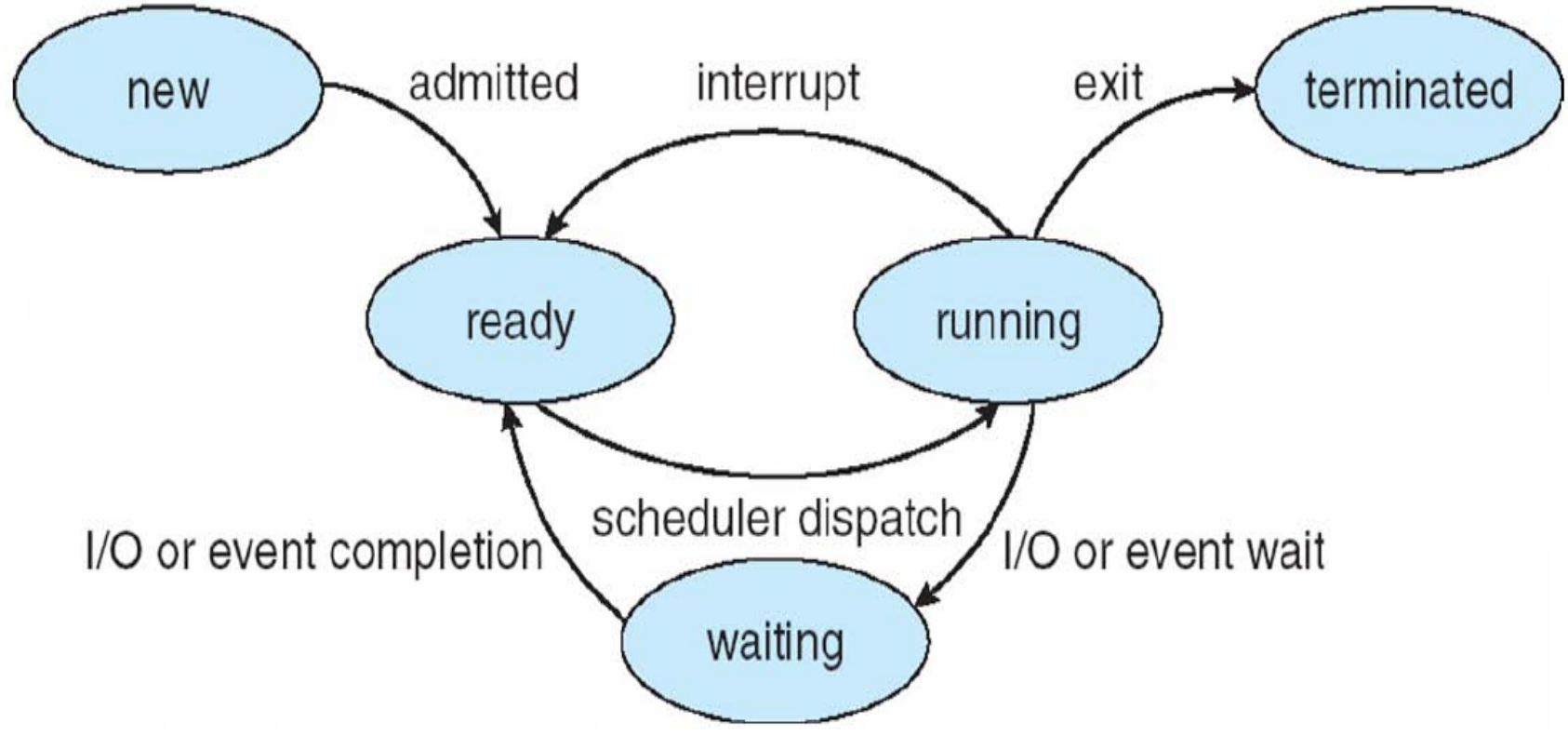


Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Process states

- A process can be in one of the following states:
  - **new**: the process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Process transition diagram

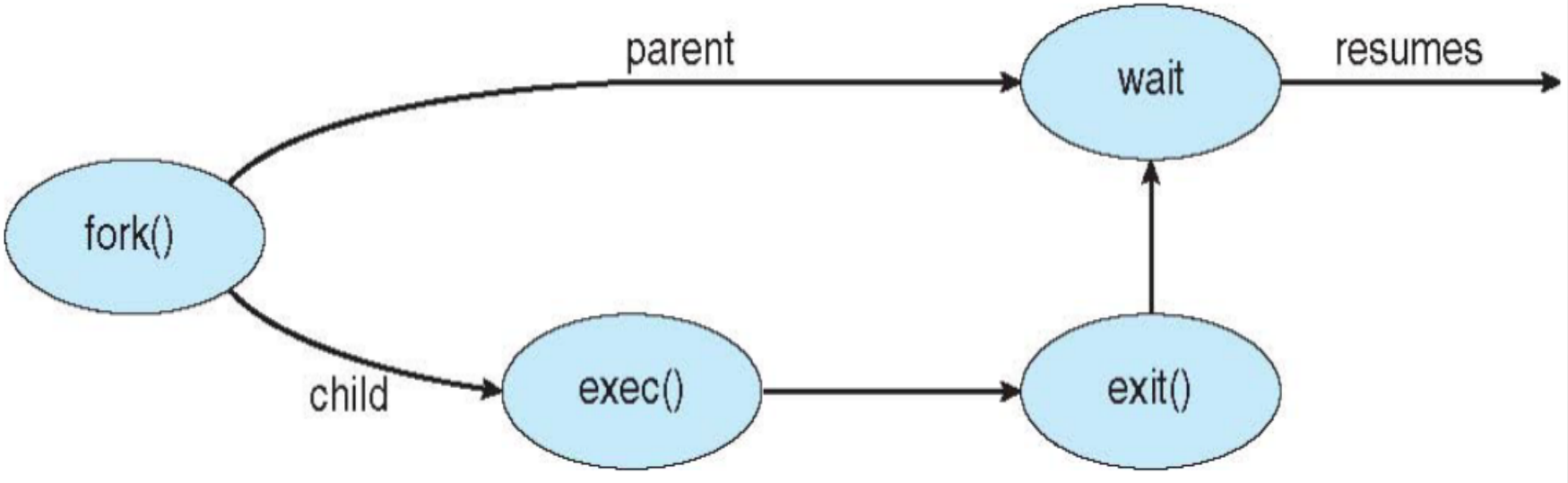


Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Process creation

- Parent process create children processes, which, in turn create other processes, forming a **tree of processes**
- Generally, process is identified and managed by a **process identifier** (pid)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process creation in Unix



Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Terminating a process

- After a process finishes its execution, it notifies OS about this fact (exit() function).
- OS notifies the parent process (if it's waiting) and frees resources posessed by the finished process.
- If a parent process is killed or terminated, its child processes are „adopted” by a process with PID=1 (called *init*).
  - No-one is allowed to kill the *init* process.

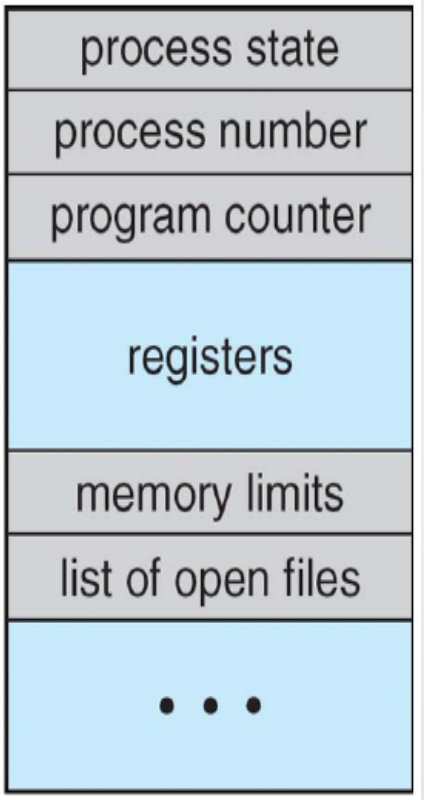
Summary

- This lecture shows processes and their managment:
  - Process.
    - Process memory organization
    - Process states and transition diagram
    - Process creation and termination
  - Process Control Block (PCB)
    - Context switching
  - Threads in Linux and in Java
  - Process scheduling
    - Long-term scheduler
    - Short-term scheduler
      - FIFO, SJF, SRTF, RR

Process Control Block

Information associated with each process (also called task control block)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information – priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

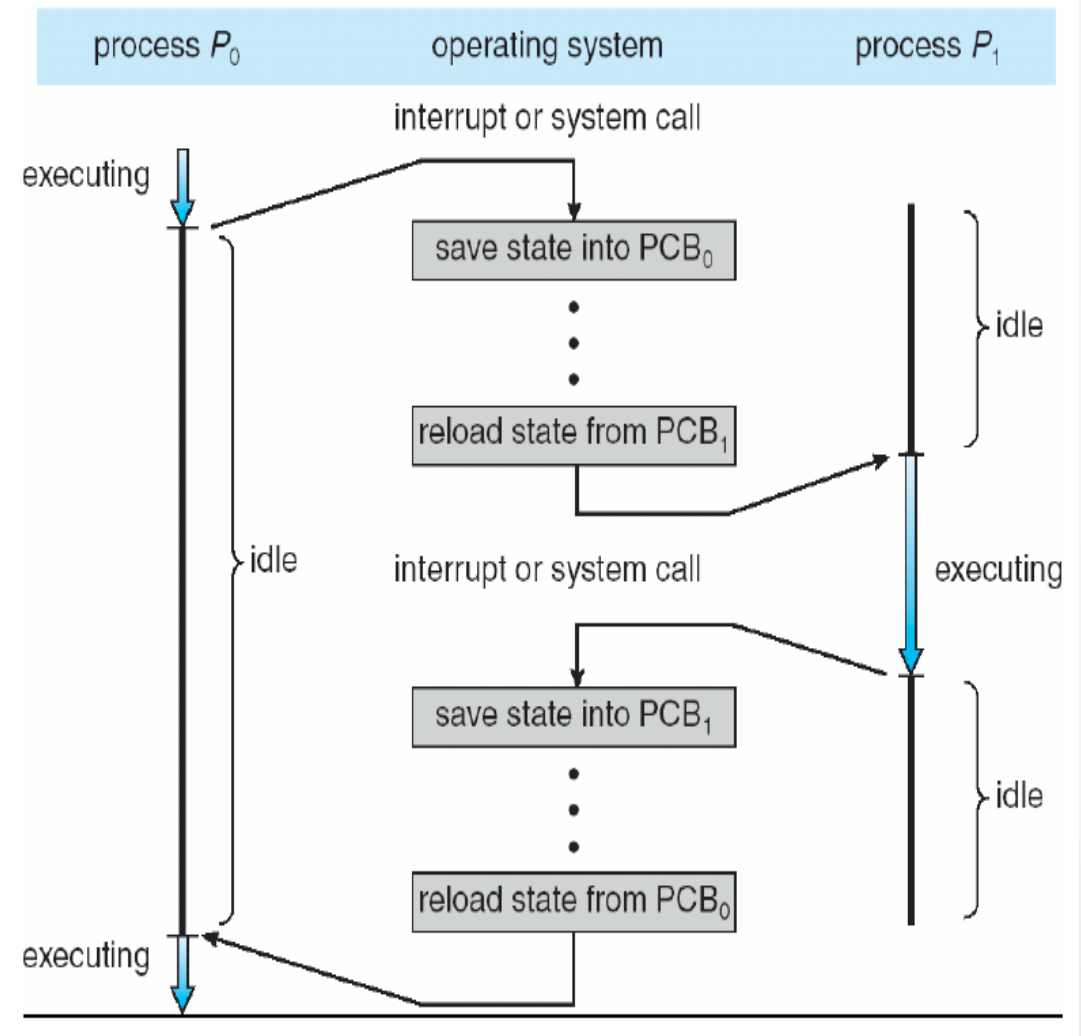


Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Context switching

- When CPU switches to another process, the system must save the state of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

Switching between processes



Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Summary

- This lecture shows processes and their managment:
  - Process.
    - Process memory organization
    - Process states and transition diagram
    - Process creation and termination
  - Process Control Block (PCB)
    - Context switching
  - Threads in Linux and in Java
  - Process scheduling
    - Long-term scheduler
    - Short-term scheduler
      - FIFO, SJF, SRTF, RR

Threads

- Threads represent parallel paths in process execution.
- Threads are sometimes called „liteweight processes” - they occupy less space, they share some resources and their creation is much easier.
- Thread scheduling may be done just like scheduling of other processes.

Threads

- Each thread must have its own registers and program counter – they may execute totally different code.
- They also must have separate stacks for execution.
- Threads share the same address space. It means, that they can exchange data in a very fast and easy way (although still synchronization is needed).

Threads

- In Java, programs are executed with a virtual machine, working under certain operating system.
  - Threads in Java are usually implemented using system-native threads (one system thread per one Java thread).

Summary

- This lecture shows processes and their managment:
  - Process.
    - Process memory organization
    - Process states and transition diagram
    - Process creation and termination
  - Process Control Block (PCB)
    - Context switching
  - Threads in Linux and in Java
  - Process scheduling
    - Long-term scheduler.
    - Short-term scheduler.
      - FIFO, SJF, SRTF, RR

Scheduling

- Scheduling subsystem decides, which processes should be assigned to a CPU at a time.
- Interruption of a currently executed process in order to execute another process is called **preemption**. It requires a **context switch**.
- The number of currently executed processes in a system is limited by the number of computing cores available (we'll assume only one such core, for simplification). Other processes must wait.

Scheduling criteria

- Processor utilisation
- Process throughput
- Waiting time
- Turnaround time
- Response time

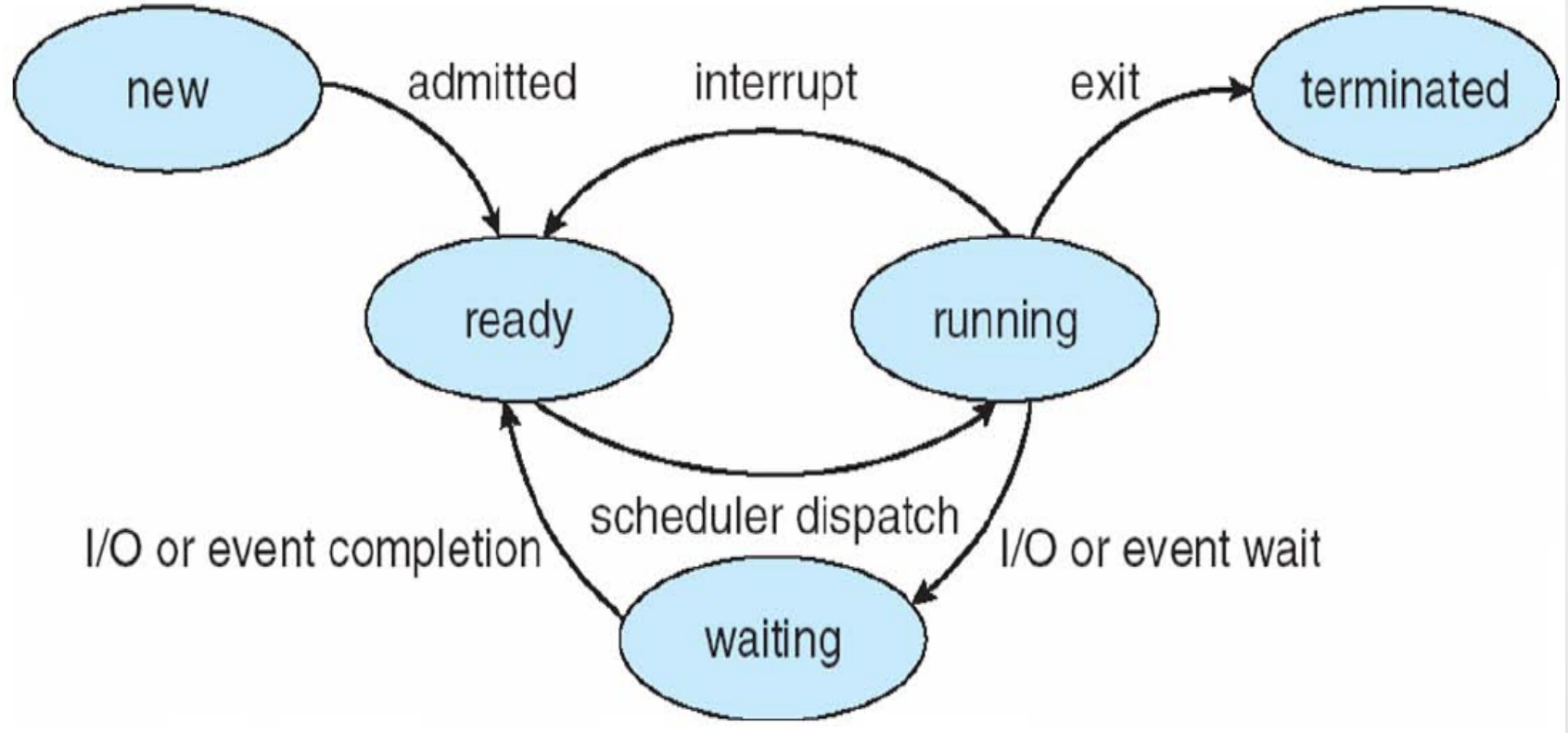
Long- and short-term scheduling

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the degree of multiprogramming
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
  - Sometimes the only scheduler in a system

Scheduling

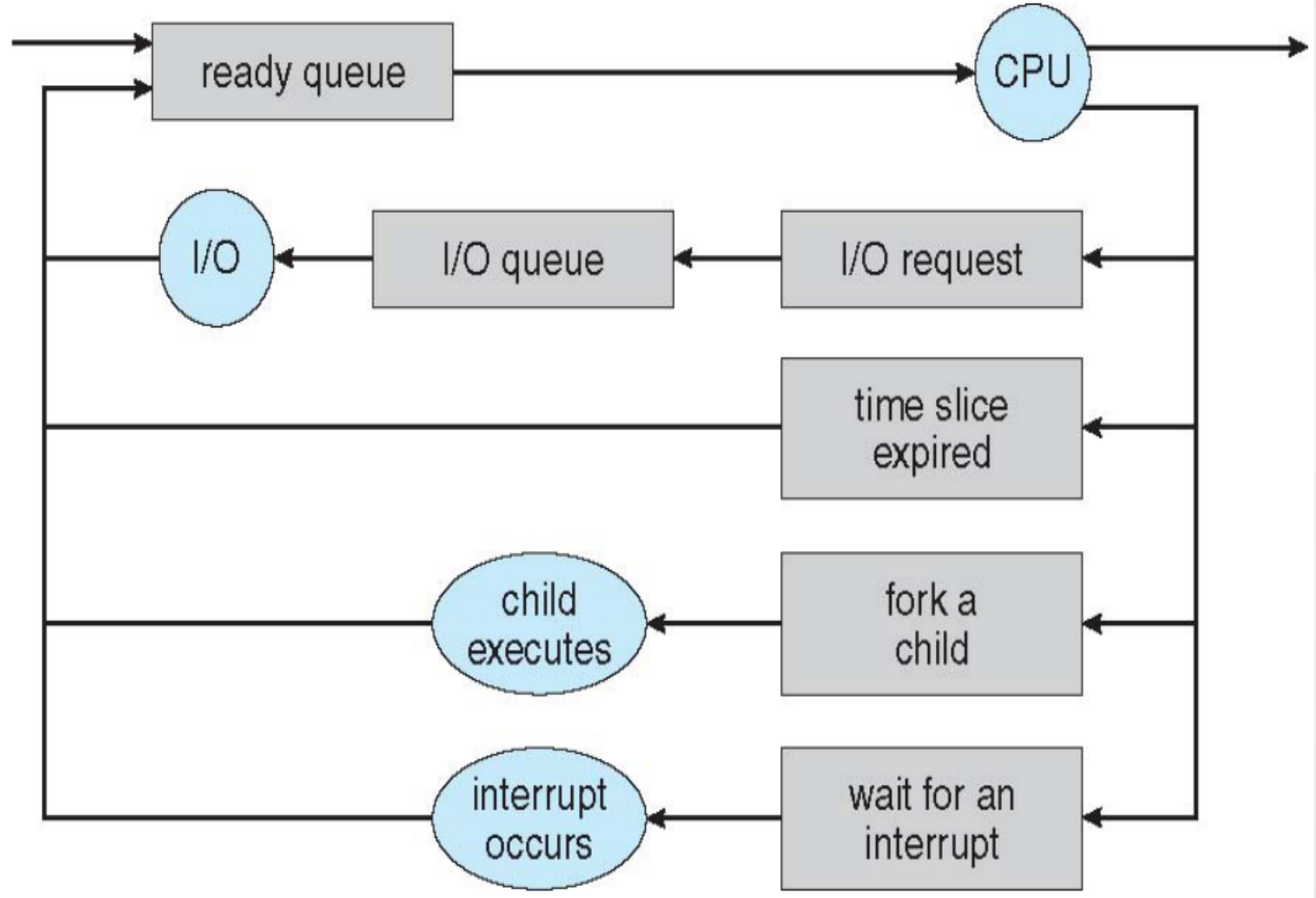
- Decisions taken by scheduler are executed by **dispatcher**, which does the following:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to continue execution of that program.
- Processes which are not active, wait in queues.

Process transition diagram



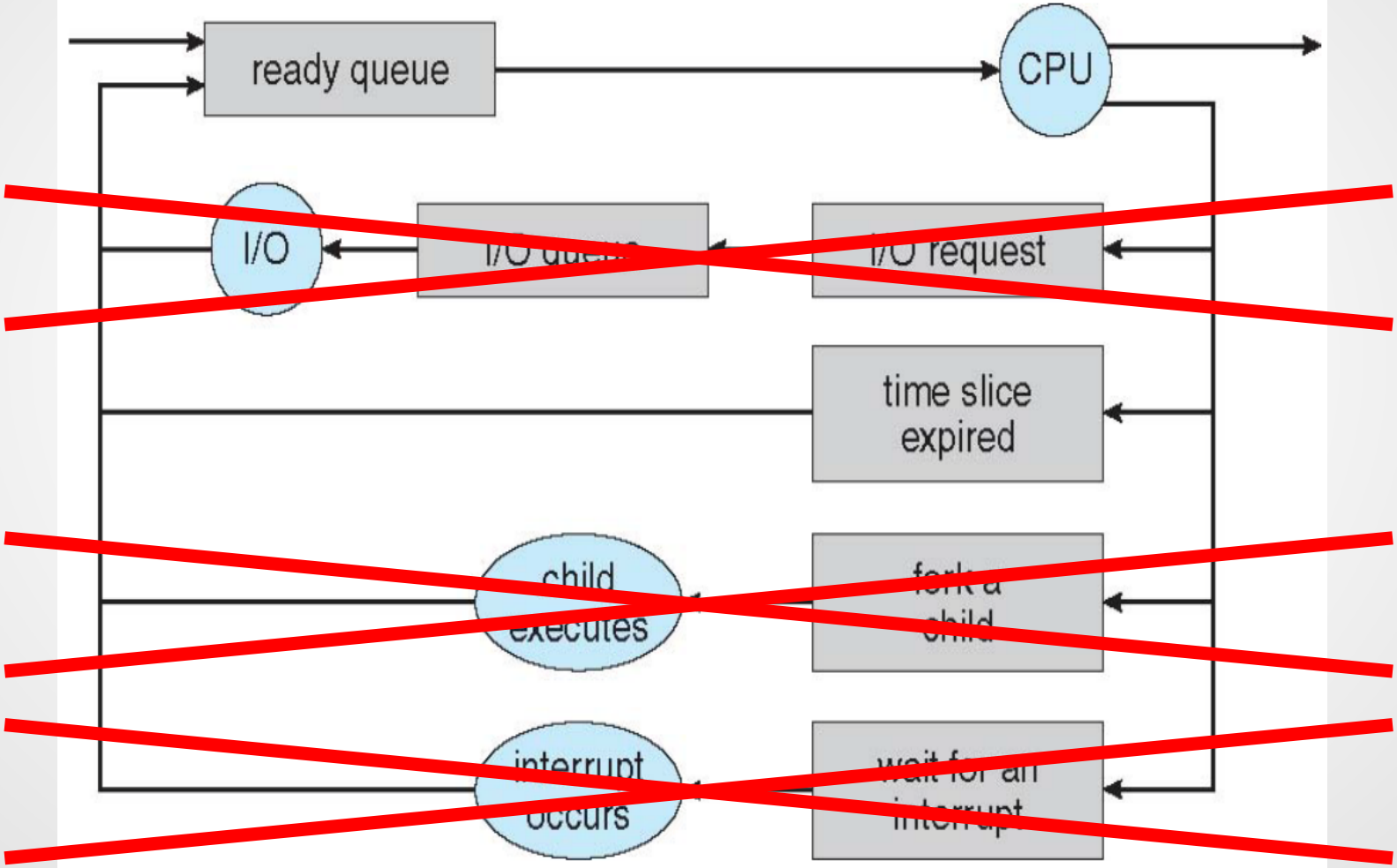
Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Queuing diagram



Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

Assumed simplification



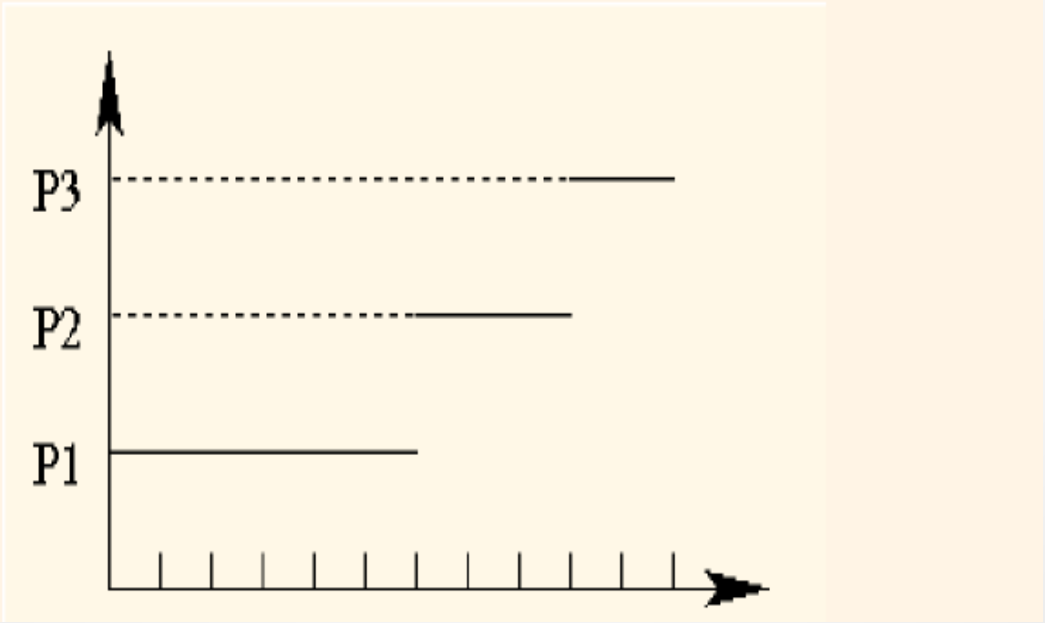
Source: Operating System Concepts – 9th Edition
Silberschatz, Galvin and Gagne ©2013

FCFS (FIFO) policy

- FCFS (*First Come – First Served*) policy assumes, that processes are executed in exactly the same order, in which they were placed in the ready queue.
- Processes are executed without interruption (preemption).

FCFS (FIFO) policy

6	3	2
P1	*P2*	*P3*

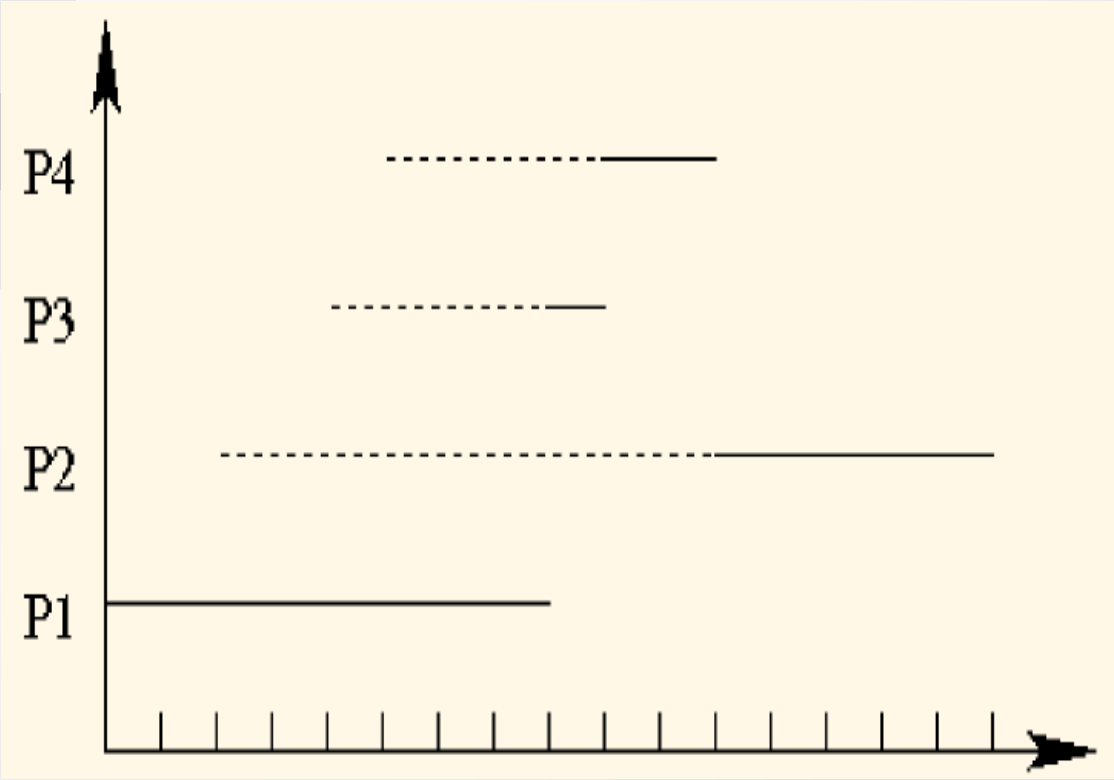


SJF policy

- SJF (*Shortest Job First*) is a modification of FCFS queuing algorithm, in which processes are inserted in the queue according to their predicted lenght (execution time).
- When a scheduling decision is taken, always the shortest ready process is selected for execution.
- Processes are executed without preemption.

SJF policy

Process	Start time	Expected Execution time
P1	0	8
P2	2	5
P3	4	1
P4	5	2

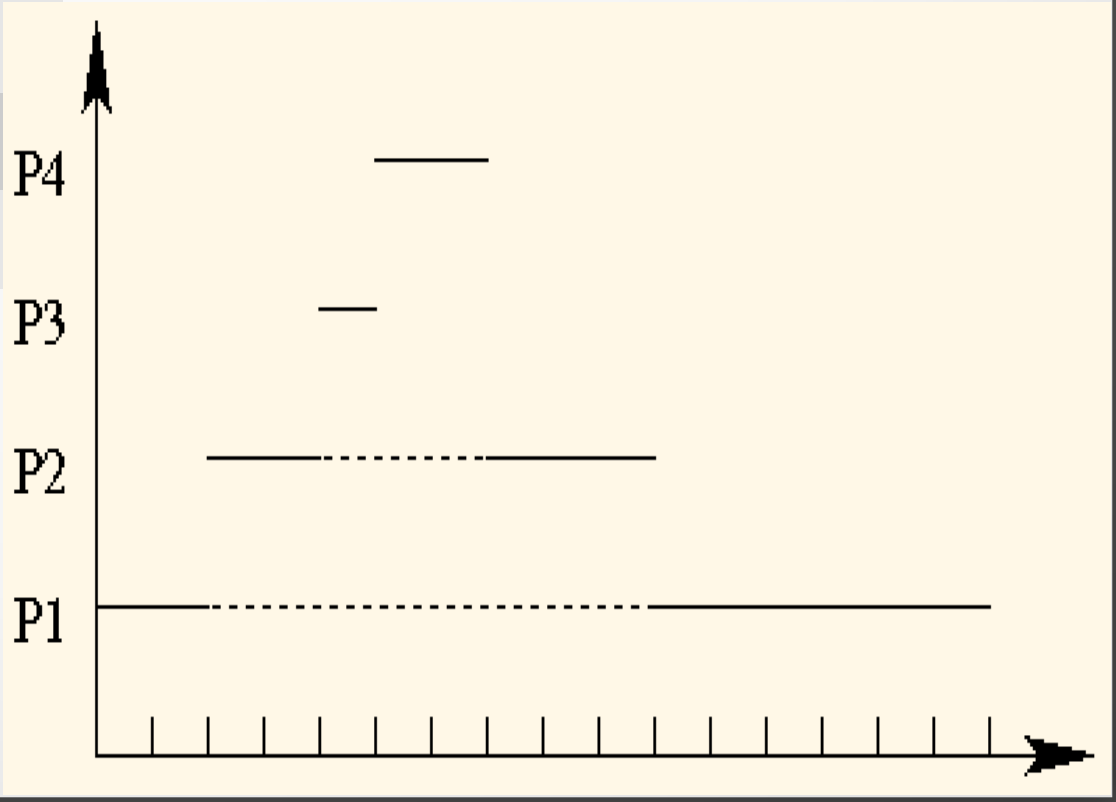


SRTF policy

- SRTF (*Shortest Remainig Time First*) policy extends the SJF policy with preemption.
- In SRTF, the scheduling decision is taken every time a new process is created in the system.
  - In such situation, the scheduler checks the time remaining for the currently executed process and predicted length of the new process. If the new process is shorter, the „old” one is switched with the „new” one.
  - Whenever process execution is finished, the shortest process from the ready ones is selected.

SRTF policy

Process	Start time	Expected Execution time
P1	0	8
P2	2	5
P3	4	1
P4	5	2

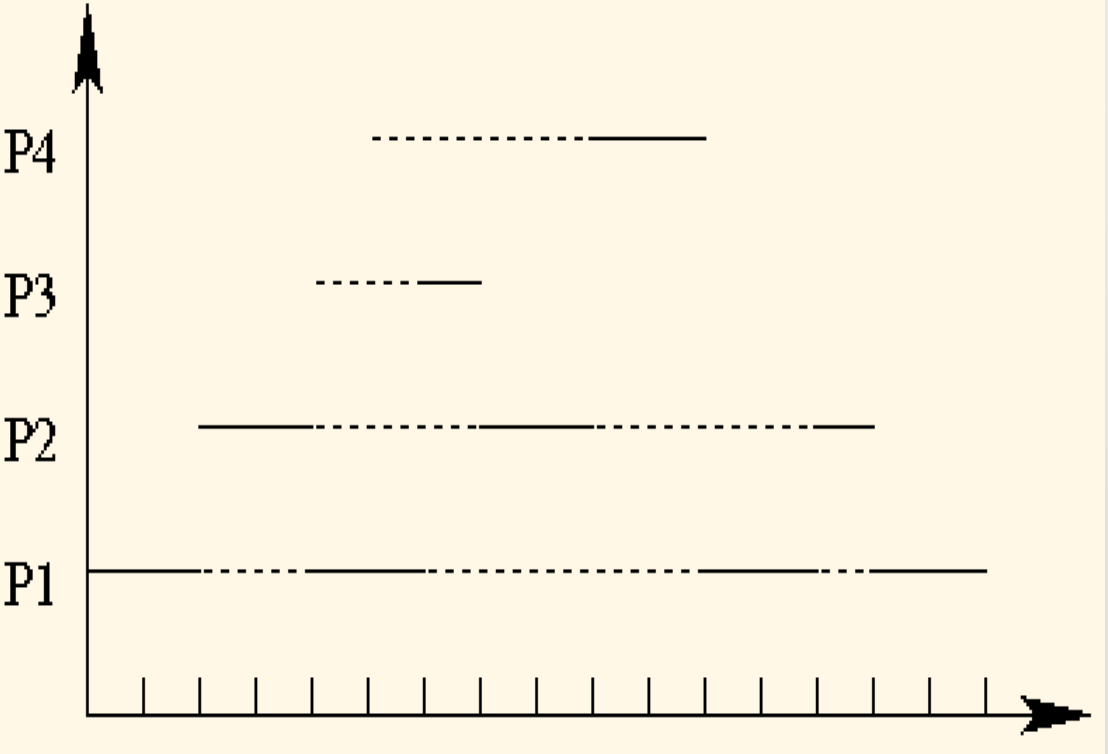


Round Robin policy

- RR (Round Robin) policy consists in division of time into **quantums** and their assignment to processes in a circular manner.
  - A process is assigned to a processor for not longer than a quantum of time. After this period elapses, the executed process is preempted and put back into the queue. Then another process is chosen for execution.

RR scheduling (quantum=2)

Process	Start time	Expected Execution time
P1	0	8
P2	2	5
P3	4	1
P4	5	2



Priorities

- Priorities may be included to change (tune) a standard behavior of scheduling policies.
  - Example: in RR strategy with priorities, processes with higher priority, when preempted, are put not at the end of the queue, but in its middle, according to its priority.
  - Simple strategy with priorities may lead to **process starving**.
    - Modifications including dynamic priorities or separate scheduling for forground and background processes nay be introduced.
- Different scheduling strategies may be needed for I/O-intensive (interactive) and other processes.

Thank You