Lecture 3: Process and threads

Contents

- What is process
- Context Switch
- Processes hierarchy
- Process creation and termination
- Threads
- Threads implementation
- Scheduling

What is a process?

Textbooks use the terms *job* and *process* almost interchangeably

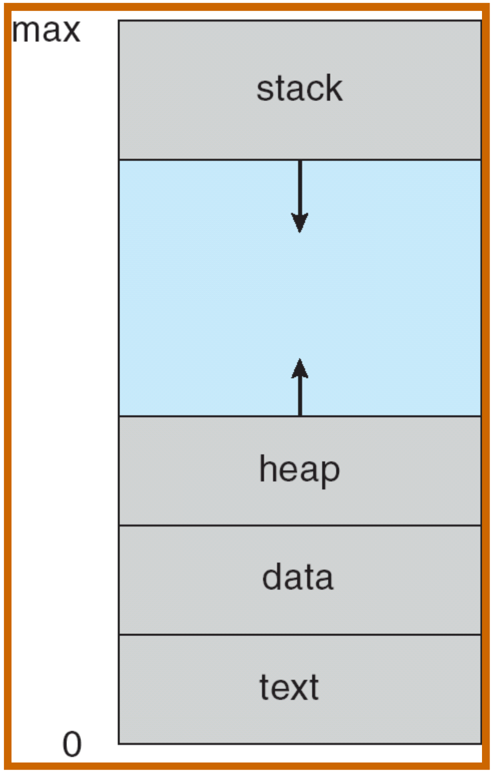
Process – a program in execution; process execution must progress in sequential fashion

A process includes:

- program counter
- stack
- data section.

Information associated with each process:

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information (“process environment”)

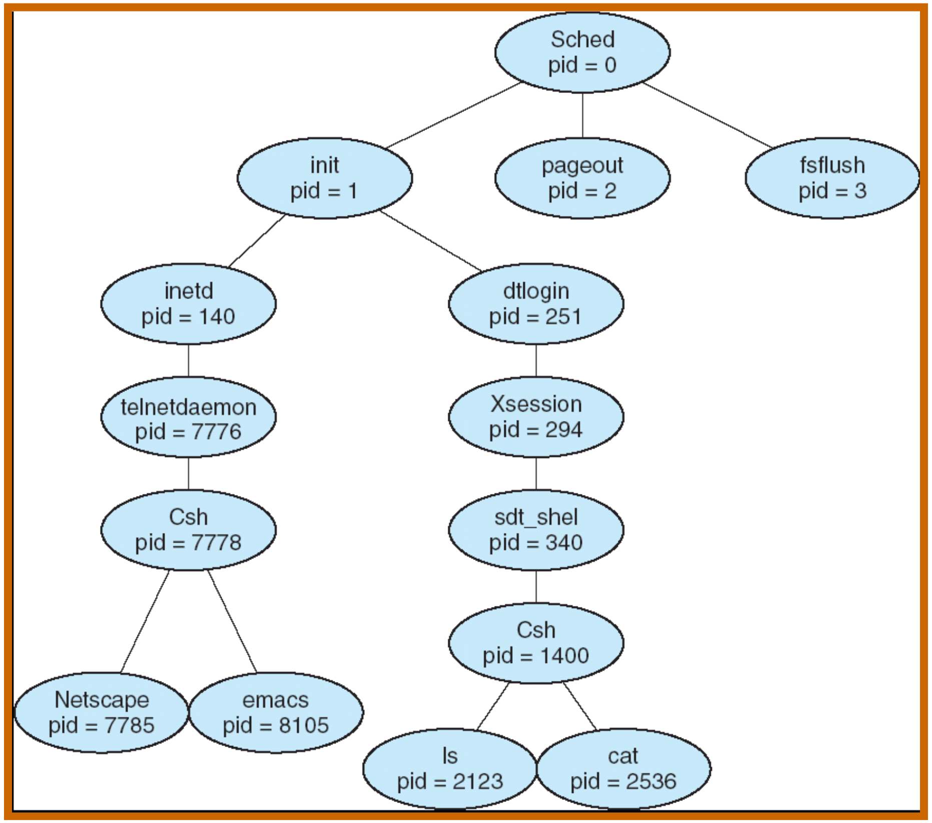


C Program Forking Separate Process

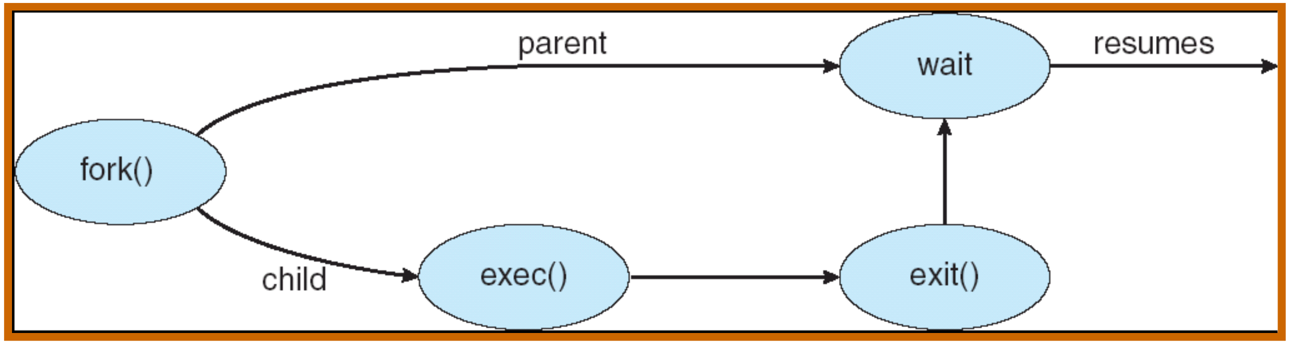
```
int main()
{
   Pid_t  pid;
   /* fork another process */
   pid = fork();
   if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
   }
   else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
   }
   else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
   }
}
```

Process Creation Illustrated

Tree of processes



POSIX parent process waiting for its child to finish



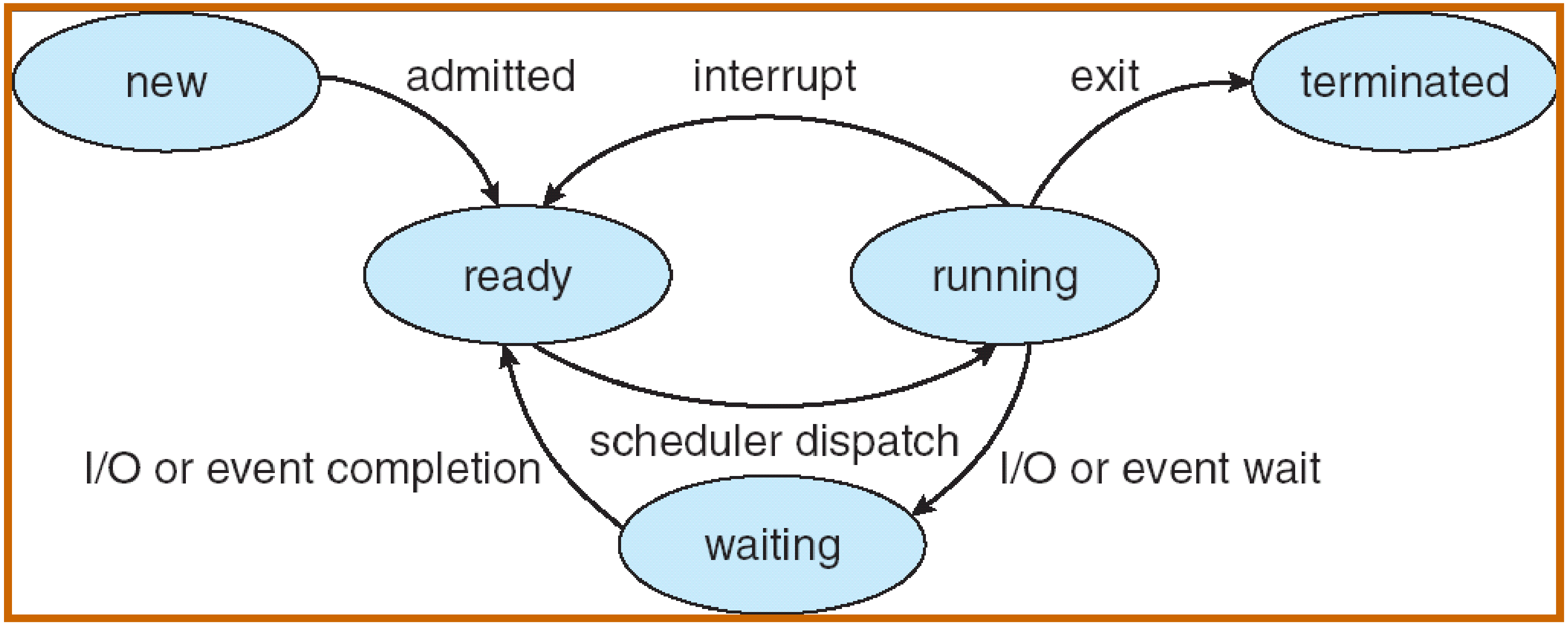
Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system

- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow children to continue if the parent terminates – the problem of '*zombie*'
    - All children terminated - *cascading termination*

Process State

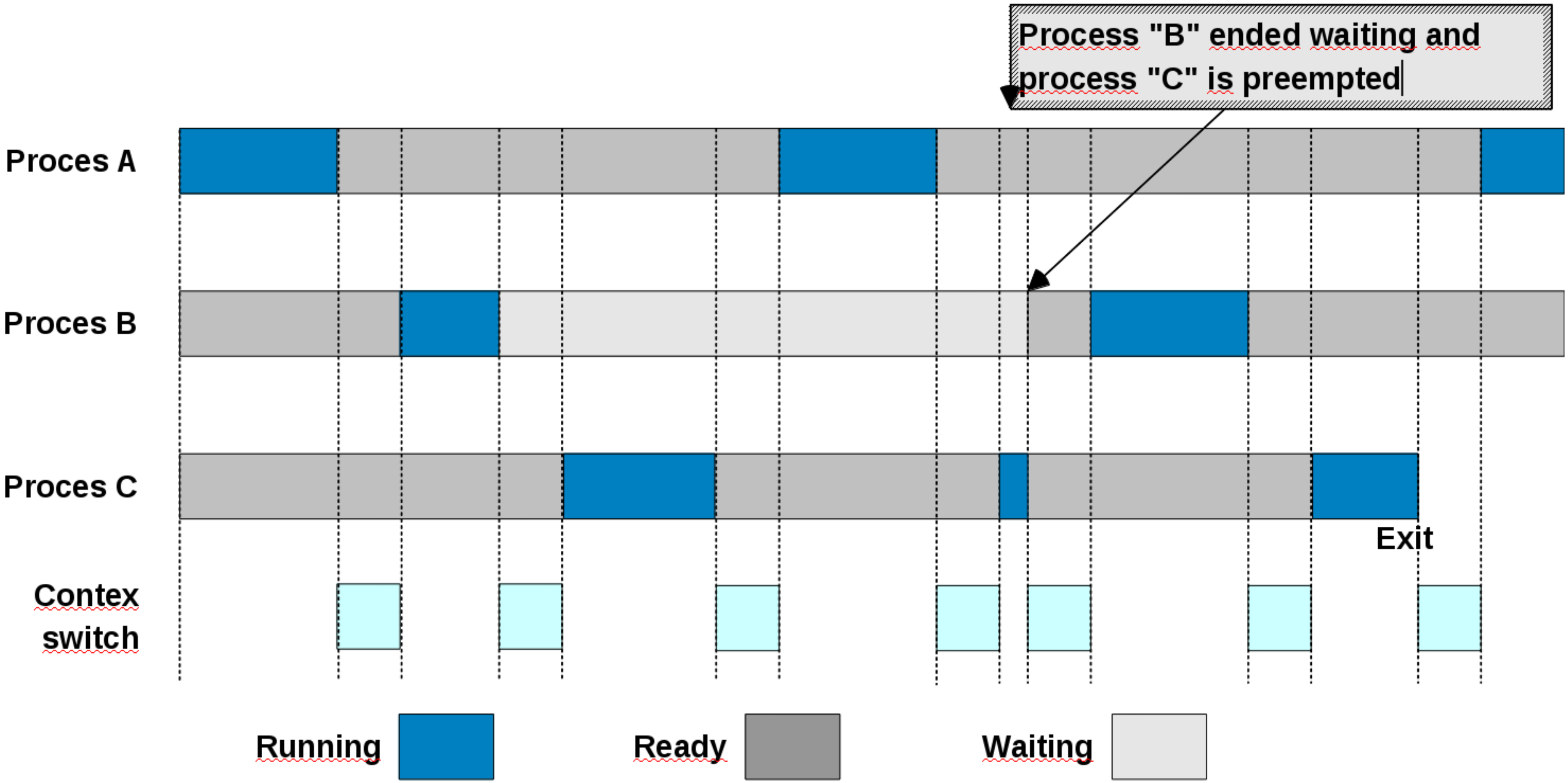
- As a process executes, it changes its ***state***
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a CPU
  - **terminated**: The process has finished execution



Context Switch

- 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
- Context-switch time is *overhead*; the system does no do useful work while switching
- Time dependent on hardware support
  - Hardware designers try to support routine context-switch actions like saving/restoring all CPU registers by one pair of machine instructions

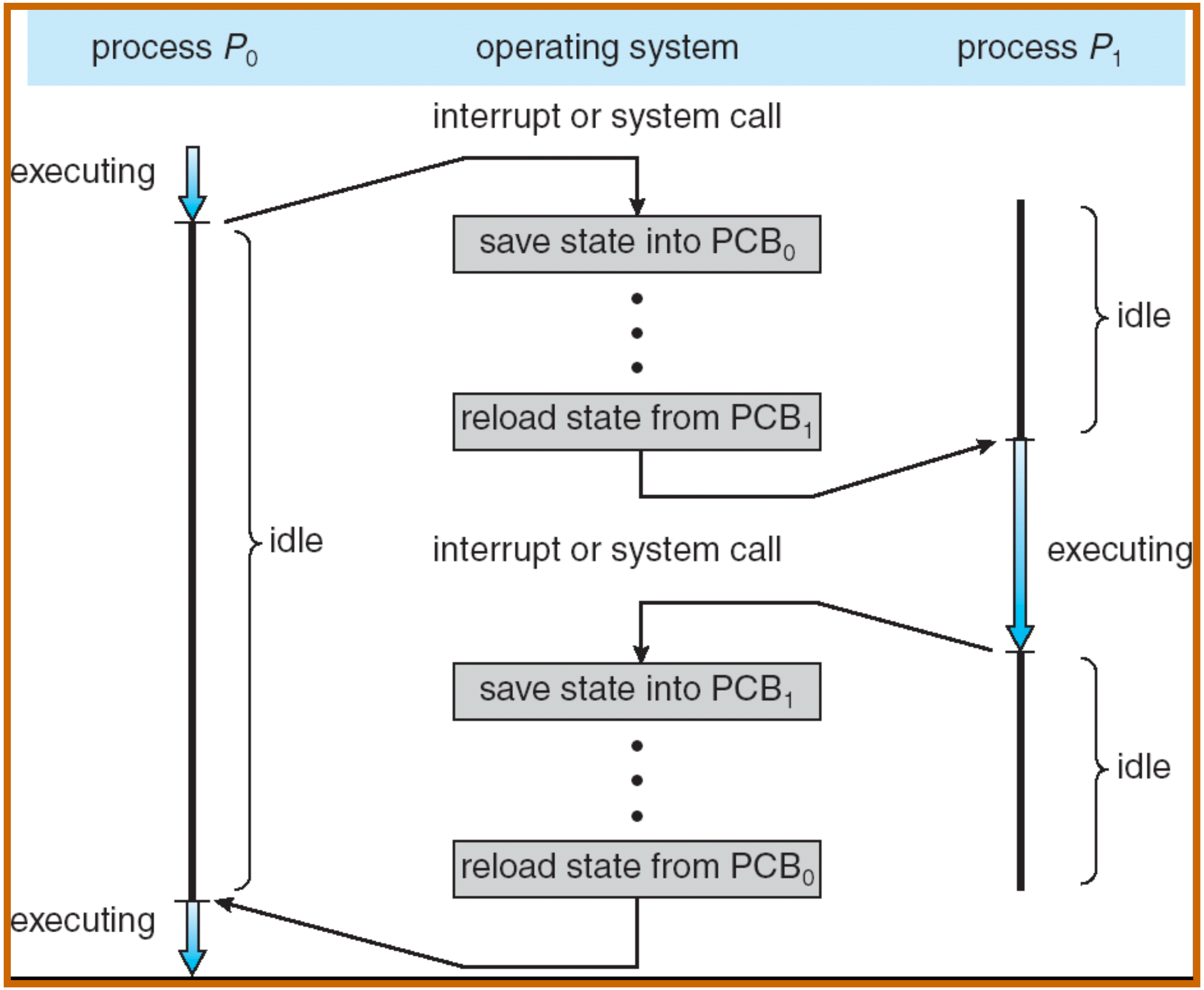
Context switch



Duration of context switch should be sorh

- system overheads

CPU Switch From Process to Process



Context switch is similar to handling an interrupt

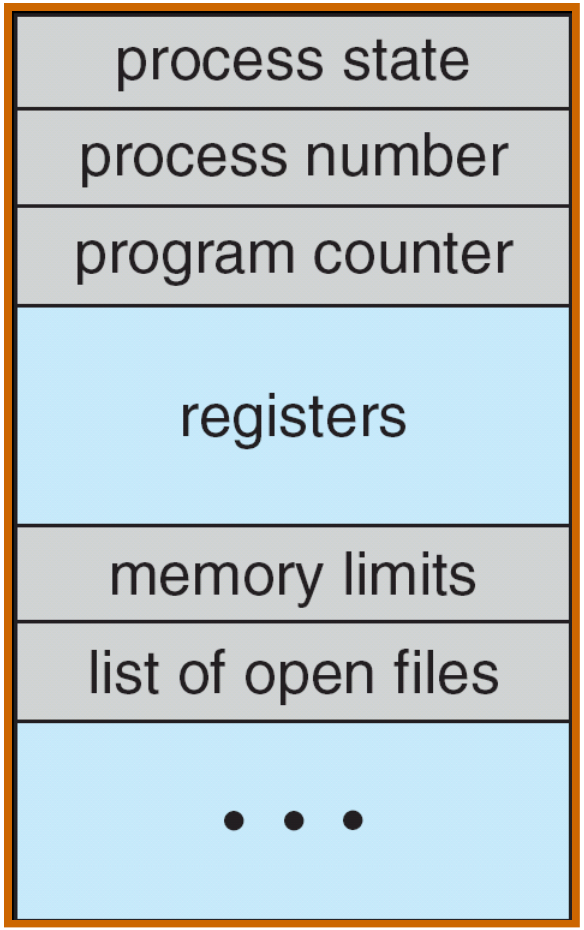
Context switch steps:
1. Save current process to PCB
2. Decide which process to run
3. Reload of new process from PCB

Context switch should be fast, because it is overhead.

Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information (“process environment”)



process state

process number

program counter

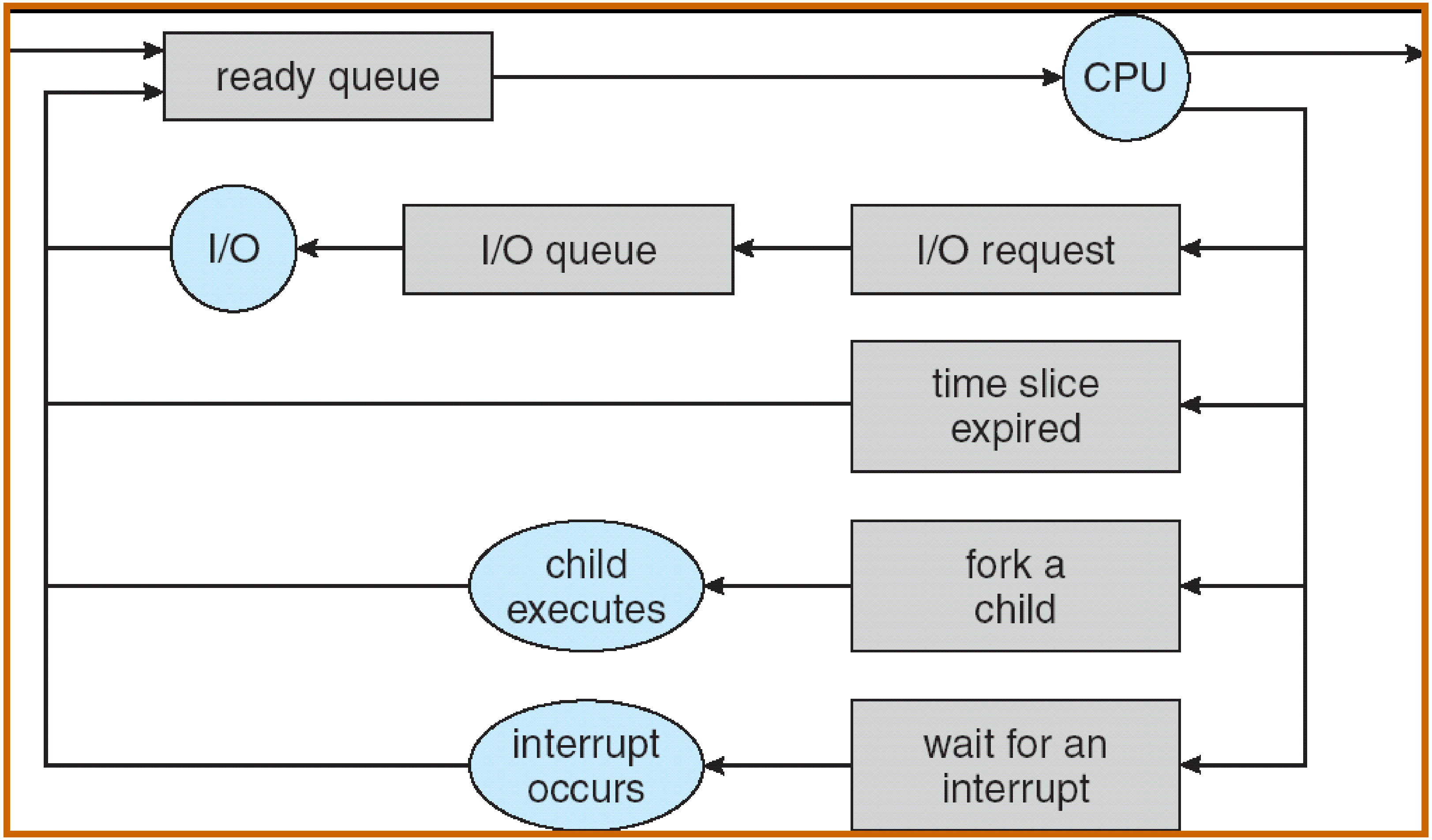
registers

memory limits

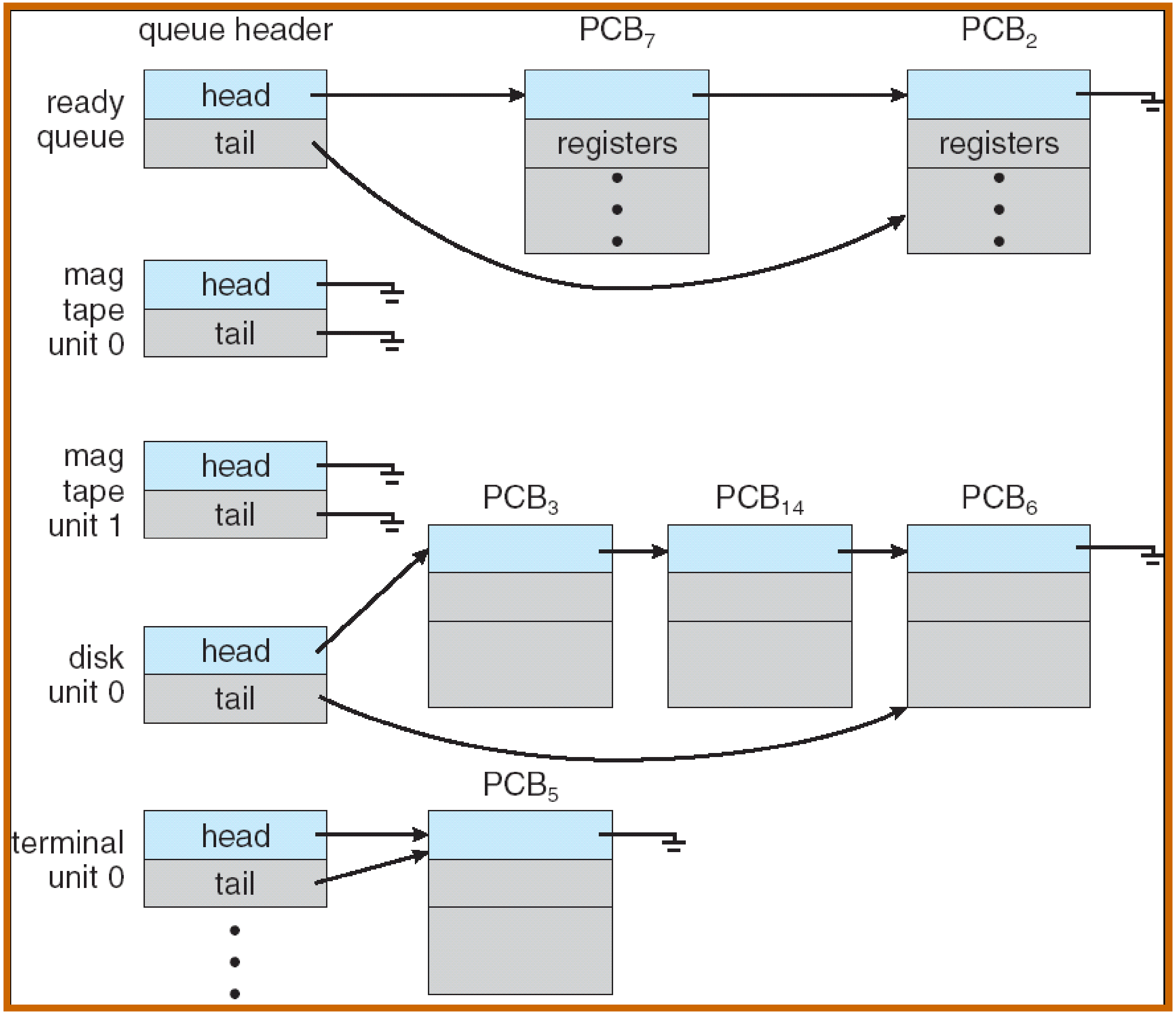
list of open files

• • •

Simplified Model of Process Scheduling



Ready Queue and Various I/O Device Queues



Single and Multithreaded Processes

- Benefits of Multi-threading
  - Responsiveness
  - Easy Resource Sharing
  - Economy
  - Utilization of Multi-processor Architectures

Threads

- Advantages
  - Create thread is faster than to create process
  - Context switch is faster for threads
  - Better design of parallel applications with threads
- Examples
  - File server in LAN
    - Many demands from different users
    - For each demand new thread
  - Symmetric Multiprocessor (SMP)
    - Different thread can use different cores
  - Output displayed in parallel with data computation
  - Parallel algorithm on multi-CPU systems
- More transparent algorithms with threads

Data sharing with threads

- Processes and threads
  - process: unit that contains resources (memory, open files, user rights)
  - thread: unit for scheduling
  - One process can have more threads

Program code:	process
Local and working wariables:	thread
Global data:	process
System resources:	process
Stack:	thread
Memory management:	process
PC – program counter:	thread
CPU register:	thread
Scheduling state:	thread
User identification and rights:	process

User threads - Many-to-One Model

- Thread management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads
- Only old operating systems without thread support

One-to-one Model

- Supported by the Kernel
- Better scheduling – one waiting thread cannot block other threads from the same process
- Examples: Windows XP/2000, Solaris, Linux, Tru64 UNIX, Mac OS X

Threads in JavaAPI

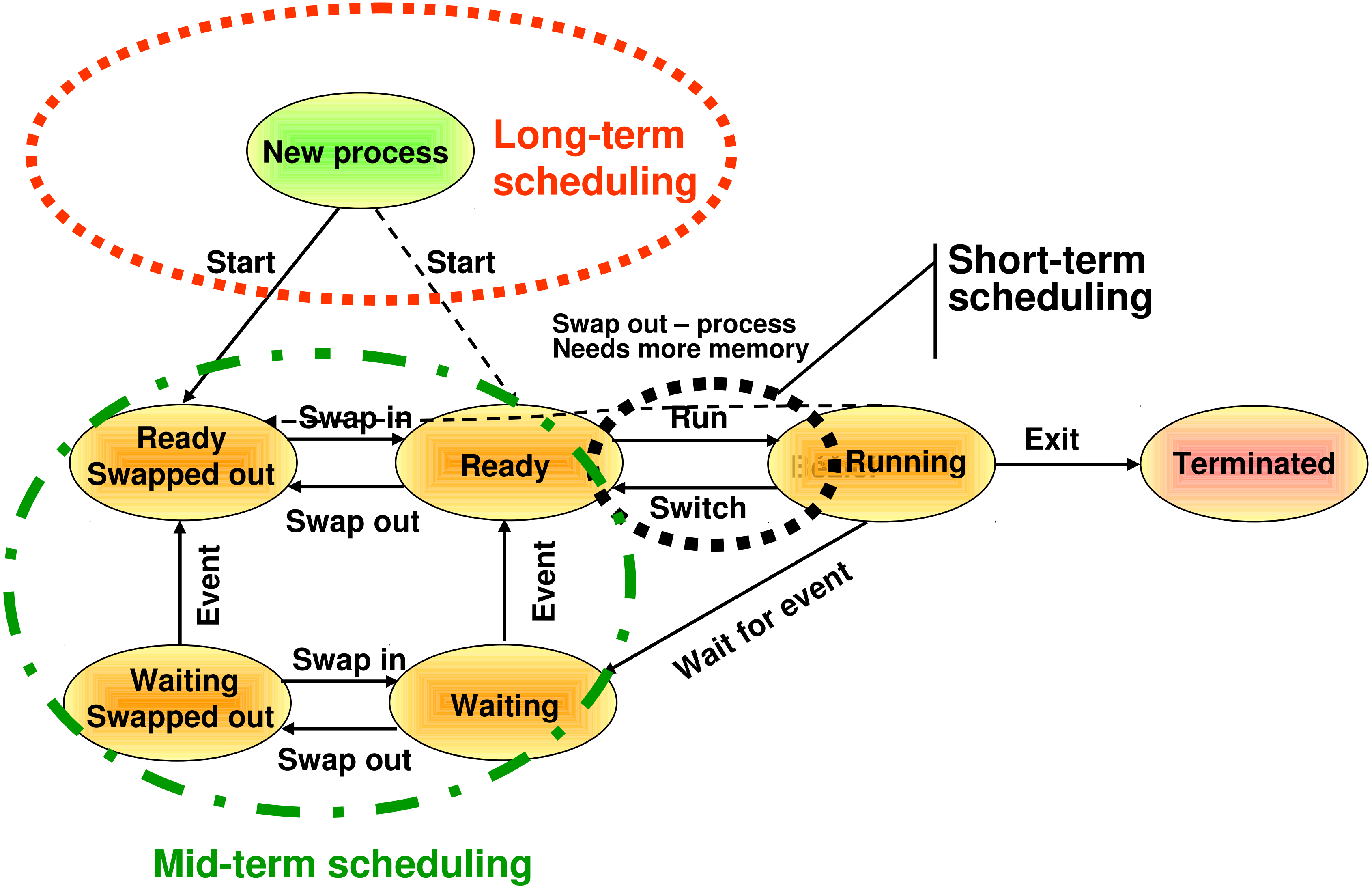
```
class CounterThread extends
Thread {
    public void run() {
        for(int i = 0; i < 10; i+
+) {

System.out.println(i);
        }
    }
}


Thread counterThread = new
CounterThread();

counterThread.start();
```

```
class Counter implements Runnable
{
    public void run() {
        for(int i = 0; i < 10; i+
+) {

System.out.println(i);
        }
    }
}


Runnable counter = new Counter();
Thread counterThread = new
Thread(counter);
counterThread.start();
```

Schedulers

- **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*
- **Mid-term scheduler** (or tactic scheduler) – selects which process swap out to free memory or swap in if the memory is free
  - Partially belongs to memory manager
- **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)

Process states with swapping



CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is *nonpreemptive*
- 2 and 3 scheduling are *preemptive*

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running – overhead

Scheduling Criteria & Optimization

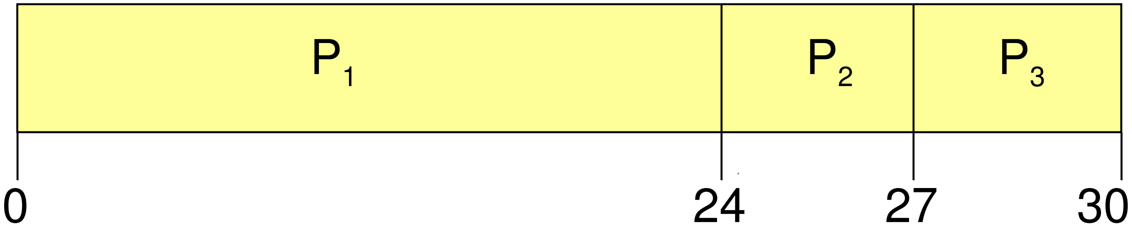
- CPU utilization – keep the CPU as busy as possible
  - Maximize CPU utilization
- Throughput – # of processes that complete their execution per time unit
  - Maximize throughput
- Turnaround time – amount of time to execute a particular process
  - Minimize turnaround time
- Waiting time – amount of time a process has been waiting in the ready queue
  - Minimize waiting time
- Response time – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing and interactive environment)
  - Minimize response time

First-Come, First-Served (FCFS) Scheduling

- Most simple nonpreemptive scheduling.

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



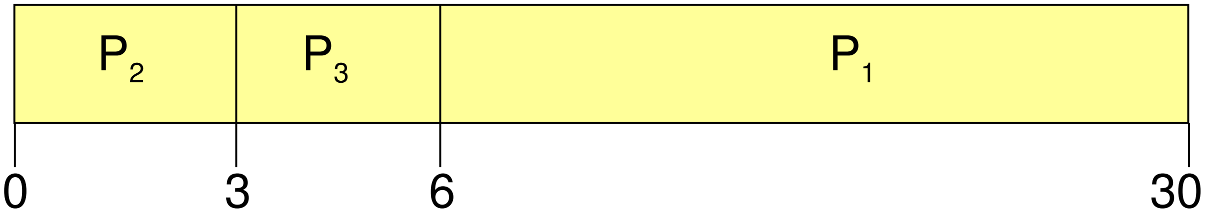
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- *Convoy effect* short process behind long process

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time (SRT)
- SJF is optimal – gives minimum average waiting time for a given set of processes

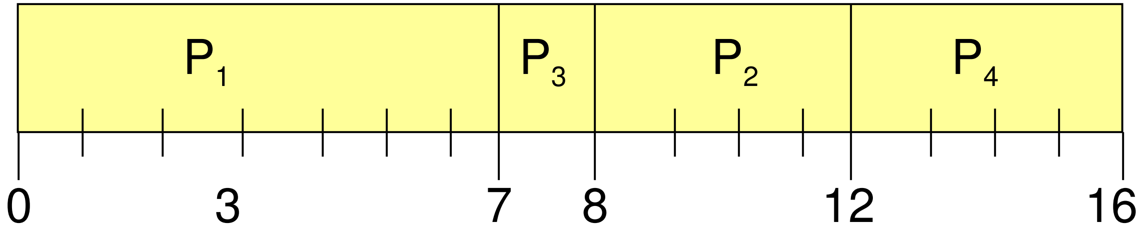
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time (SRT)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

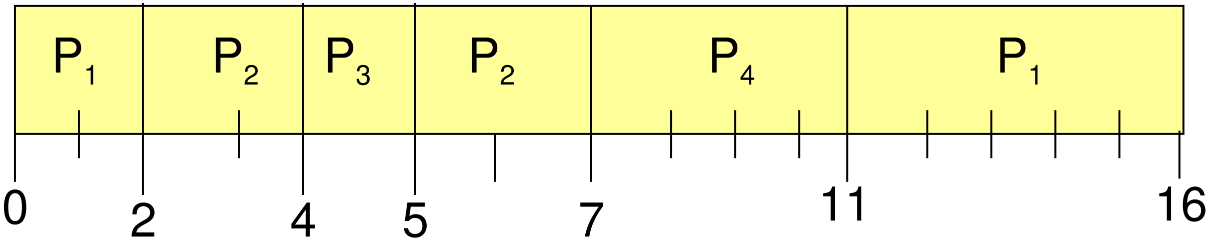


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

End of Lecture 4