

CYB510 Operating Systems

Prepared Prof. Lamiaa Fattouh

Lecture 3

جامعة الجميع الذكية

تسعى لتكون الكيان الأول الذي يتبنى مبدأ المجتمع التعليمي
التعاوني ويقدم العلم مقابل العمل لتنمية وبناء المجتمعات.

CYB510 Thread and CPU Scheduling



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

- Overview
- Multithreading Models
- Threading Issues



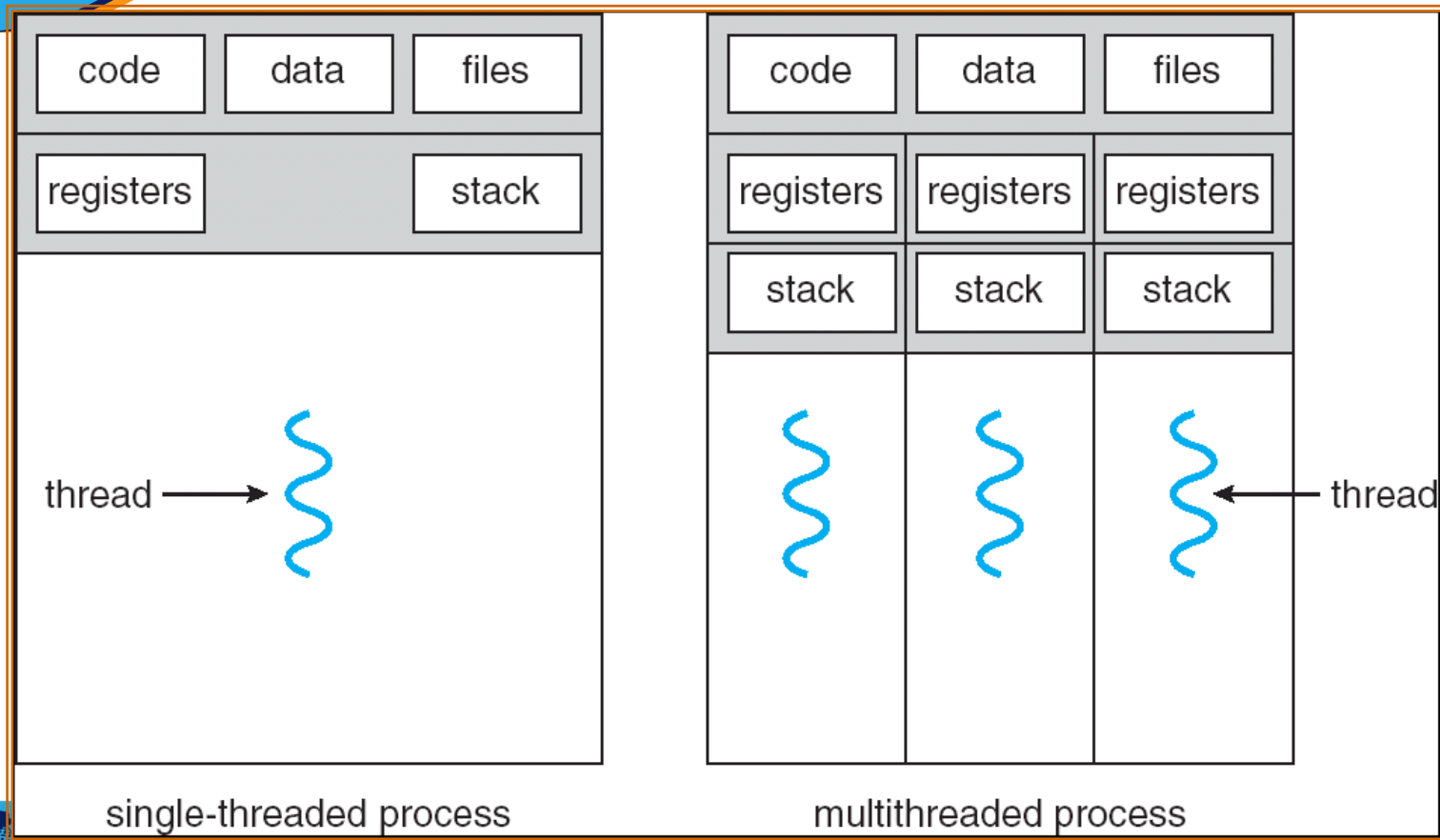
Thread comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating system resources, such as open files and signals.

If the process has multiple threads of control, it can do more than one task at a time.



Chapter 4-5

Single and Multithreaded Processes



Heavy weight process

**Thread called a Light
weight process LWP**

- A application typically is implemented as a separate process with several threads of control.

Examples:

1. A web browser might have one thread display images or text while another thread retrieves data from the network.
2. A word processor may have a thread for displaying graphics, another thread for reading keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.
3. A web server accepts client requests for web pages, images, sound, and so forth. The server would create a separate thread that would listen for client requests; when a request was made, rather than creating another process, it would create another thread to service the request.



- **Responsiveness:** multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. For instance, a multithreaded web browser could still allow user interaction in one thread while an image is being loaded in another thread.
- **Resource Sharing:** by default, threads share the memory and the resources of the process to which they belongs. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space.



- **Economy:** allocating memory and resource for process creation is costly. In Solaris 2, creating a process is about 30 times slower than is creating a thread, and context switching is about five times slower.
- **Utilization of Multiprocessor Architectures:** the benefits of multithreading can be greatly increased in a multiprocessor architecture, where each thread may be running in parallel on a different processor. In a singleprocessor architecture, only one thread is running at a time.



- Thread management done by user-level threads library. The library provides support for thread creation, scheduling and management with no support from the kernel.
- Three primary thread libraries:
 - POSIX Pthreads
 - Win32 threads
 - Java threads



- Supported by the Kernel
- Examples
 - Windows XP/2000
 - Solaris
 - Linux
 - Tru64 UNIX
 - Mac OS X



- Many-to-One
- One-to-One
- Many-to-Many

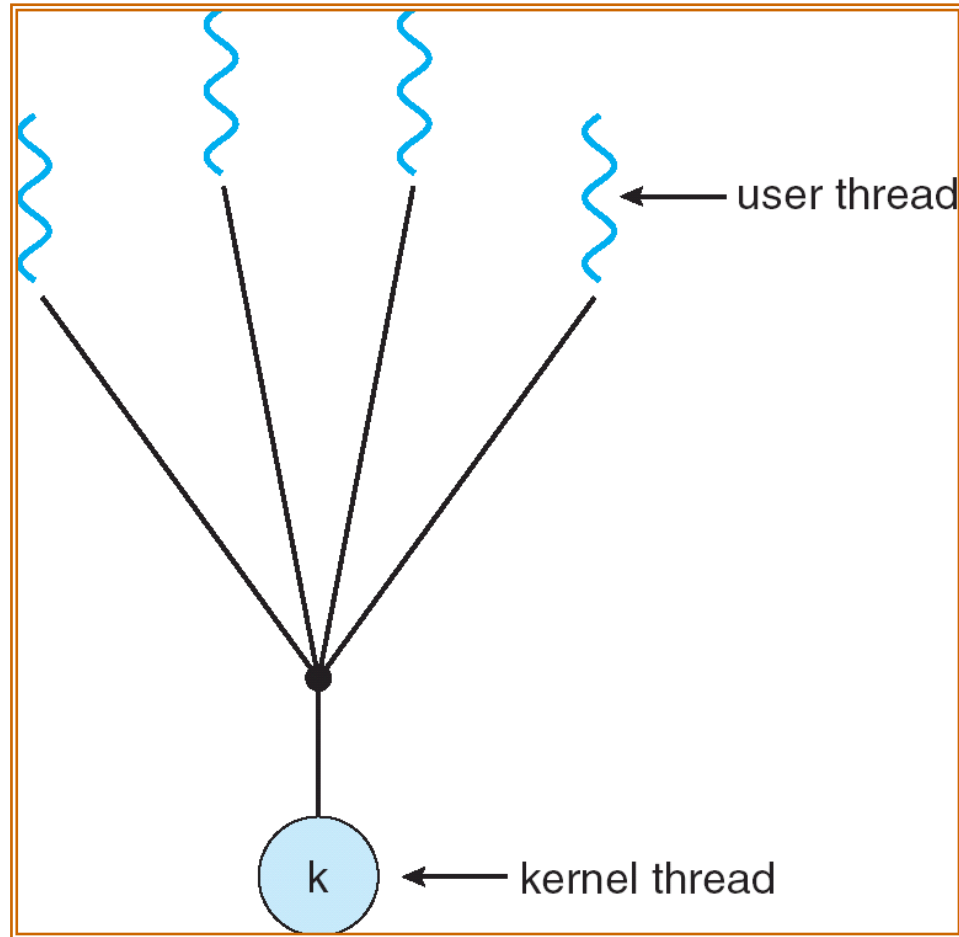


- Many user-level threads mapped to single kernel thread. Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call. Also, because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.
- Examples:
 - Solaris Green Threads
 - GNU Portable Threads



Chapter 4-5

Many-to-One Model

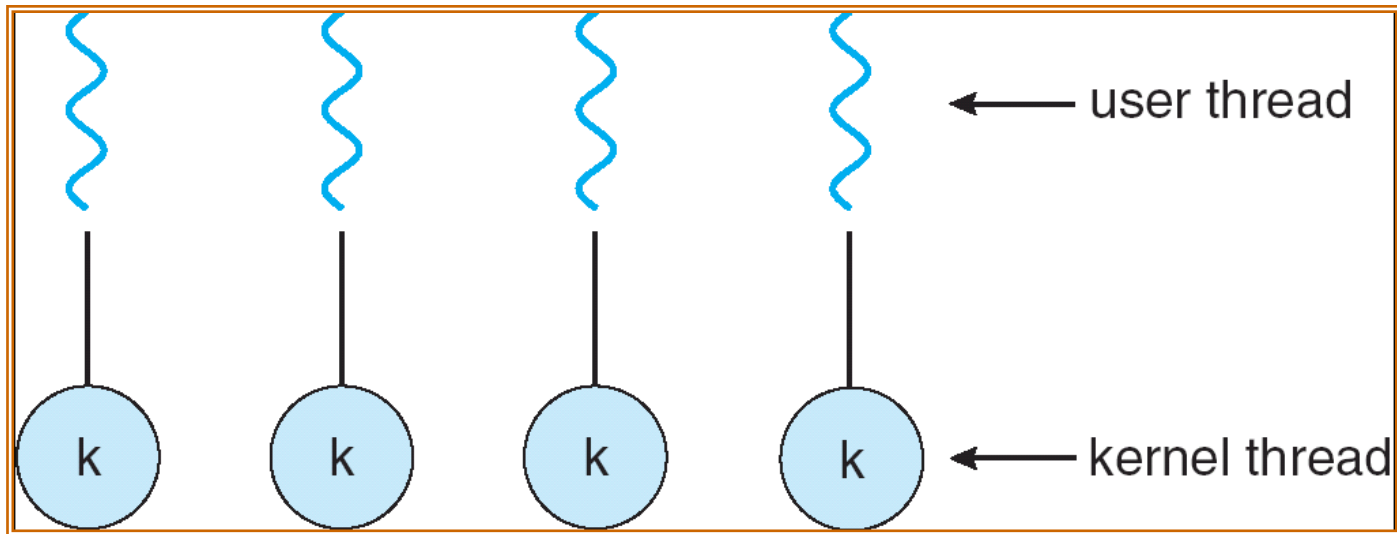


- Each user-level thread maps to kernel thread.
- Examples
 - Windows NT/XP/2000
 - Linux
 - Solaris 9 and later



Chapter 4-5

One-to-one Model

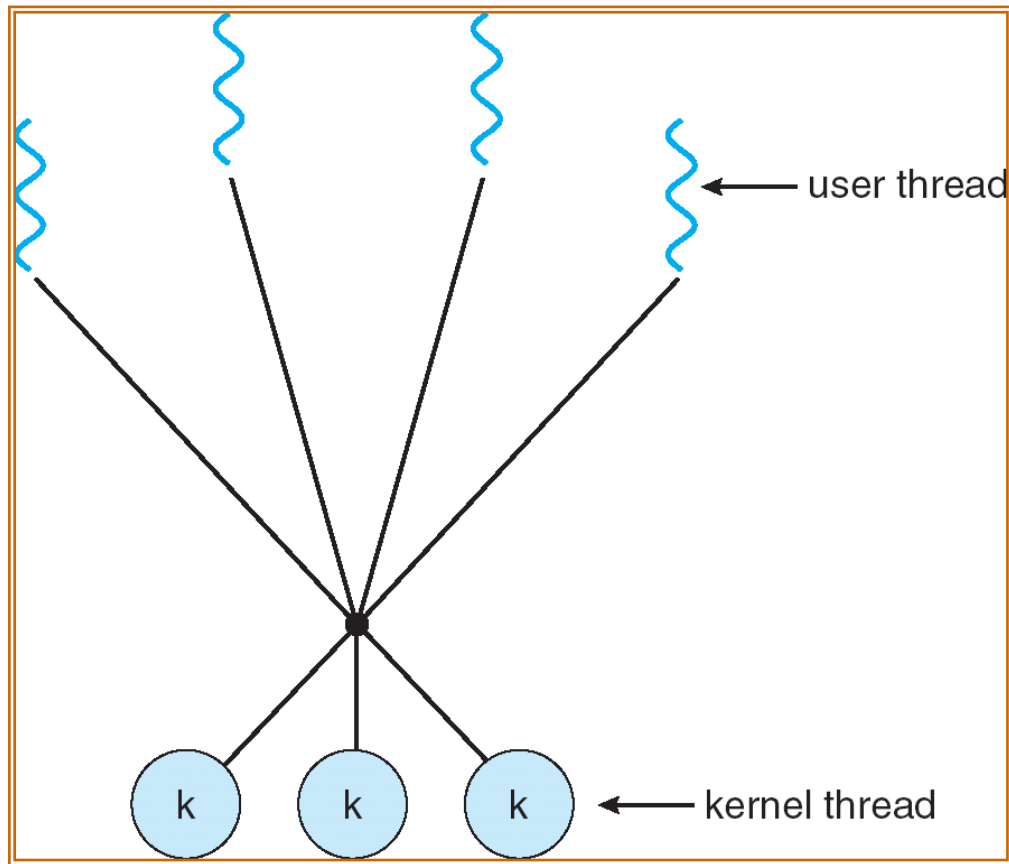


- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows NT/2000 with the *ThreadFiber* package



Chapter 4-5

Many-to-Many Model



- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
 - IRIX
 - HP-UX
 - Tru64 UNIX
 - Solaris 8 and earlier



- To introduce CPU scheduling, which is the basis for multi programmed operating systems
- To describe various CPU-scheduling algorithms (There is no one best solution. **No one is perfect!**)
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system



- Single Process → one process at a time
- Maximum CPU utilization obtained with multiprogramming
- CPU idle :waiting time is wasted

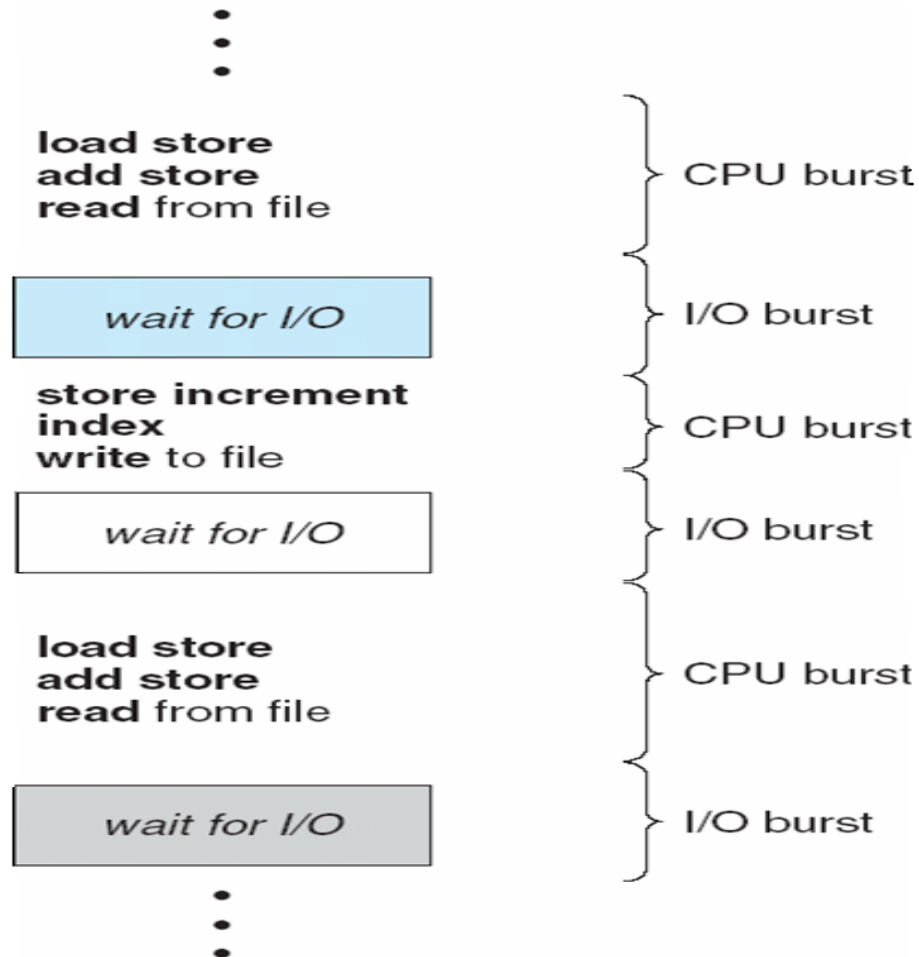


- **Cycle** – Process execution consists of a *cycle* of CPU execution and I/O wait
- **CPU burst**: a time interval when a process uses CPU only.
- **I/O burst**: a time interval when a process uses I/O devices only.

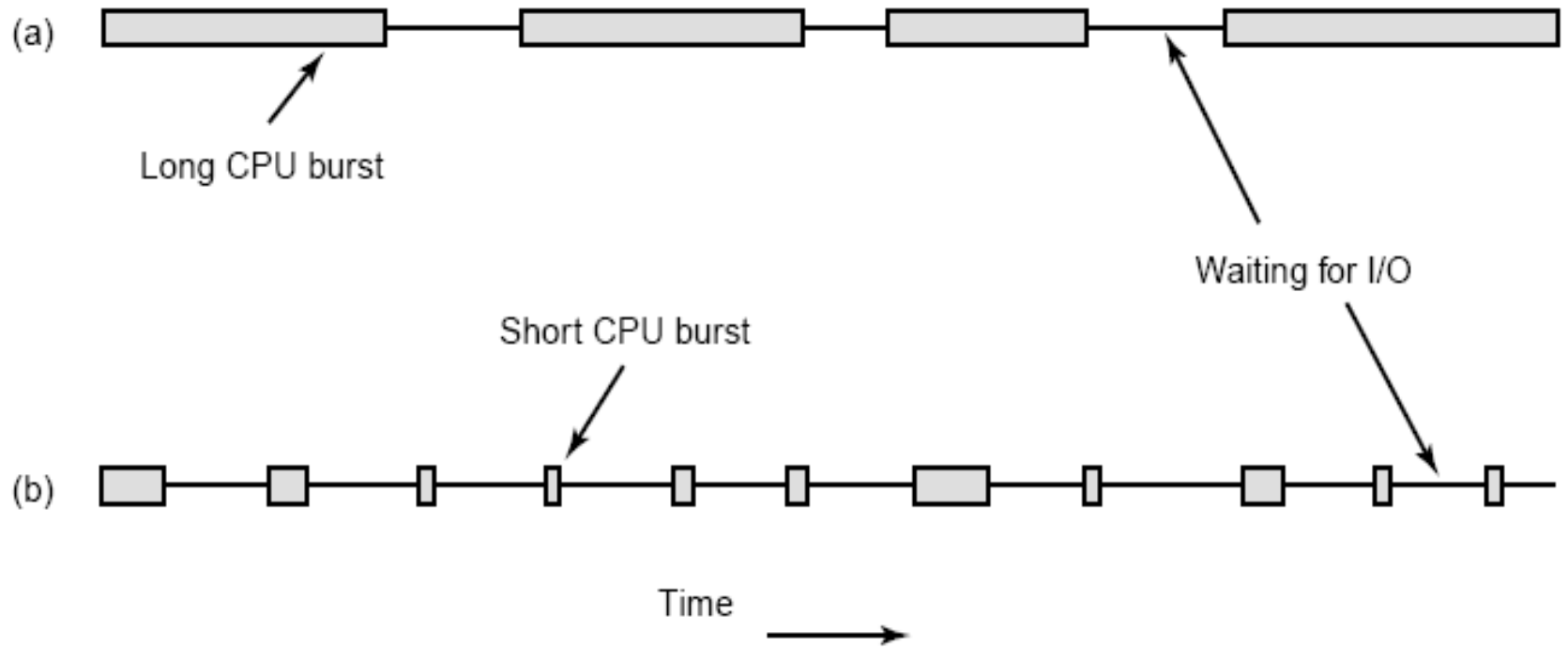


Chapter 4-5

Alternating Sequence of CPU And I/O Bursts



Chapter 4-5



- CPU idle , operating system selects one process from ready Queue to be executed
- Many mechanisms: FIFO, priority ...
- The Record in the queue are generally process control block PCBs of the processes



- Long term scheduling
- Short term scheduling: CPU scheduler
- Medium term scheduling



- 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 in scheduling scheme is **nonpreemptive** or **cooperative**;
- All other scheduling is **preemptive**



- **Scheduler** (Algorithm + Data Structure) decides *which process to run next*. Remember that decision always take time and resources.
- **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
- At each clock interrupt, the OS gets to run and decide whether the currently running process should be allowed to continue or whether it has to choose another process and give it the CPU.



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



- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time



- **CPU utilization and response time:** CPU **utilization is increased** if the overheads associated with **context switching is minimized**. The context switching overheads could be lowered by performing context switches infrequently. This could however result in increasing the response time for processes.
- **Average turnaround time and maximum waiting time:** Average turnaround time is **minimized** by executing the shortest tasks first. Such a scheduling policy could however starve long-running tasks and thereby increase their waiting time.
- **I/O device utilization and CPU utilization:** CPU utilization is **maximized** by **running long-running CPU-bound tasks** without performing context switches. I/O device utilization is **maximized** by **scheduling I/O-bound jobs** as soon as they become ready to run, thereby incurring the overheads of context switches.



- First-Come, First-Served Scheduling
- Shortest-Job-First Scheduling
- Priority Scheduling
- Round-Robin Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling



- By far the ***simplest*** CPU-scheduling algorithm is the **first-come, first-served (FCFS) scheduling algorithm**.
- With this scheme, the process that requests the CPU first is allocated the CPU first.
- The implementation of the FCFS policy is easily managed with a FIFO queue



FIRST-COME, FIRST SERVED:

- (FCFS) same as FIFO
- Simple, fair, but poor performance. Average queueing time may be long.
- What are the average queueing and residence times for this scenario?
- How do average queueing and residence times depend on ordering of these processes in the queue?



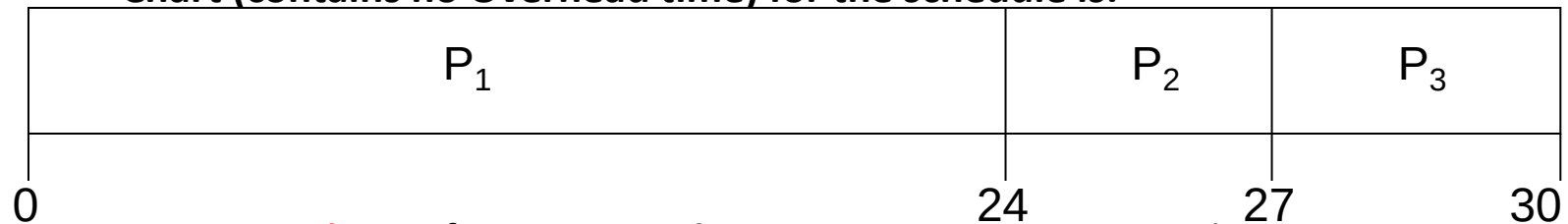
Chapter 4-5

First-Come, First-Served (FCFS) Scheduling

- One of the simplest scheduling policies
 - **non-preemptive**. Once process is scheduled to run (get CPU), it keeps it until terminated or request I/O

<u>Process</u>	<u>Burst Time</u>
P_1	24
P_2	3
P_3	3

- Suppose that processes arrive in the order: P_1, P_2, P_3 all at time 0.
A Gantt Chart shows which process executed by the CPU at each time. **The Gantt Chart (contains no Overhead time) for the schedule is:**



- **Turnaround time** for P_1, P_2 , and P_3 are 24, 27, 30 respectively
- **Waiting time** = turnaround time - burst time
 - Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
 - Average waiting time: $(0 + 24 + 27)/3 = 17$



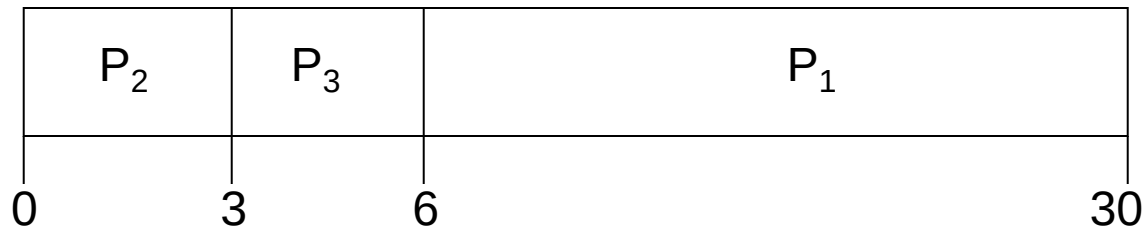
Chapter 4-5

FCFS Scheduling (Cont)

Suppose that the processes arrive in the order

$$P_2, P_3, P_1$$

- The Gantt chart for the schedule is:



- **Turnaround time** for P_1, P_2 , and P_3 are 30, 3, 6 respectively
- **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. Lower CPU and device utilization than might be possible if the shorter processes were allowed to get CPU first.
- FCFS is not suitable for time-sharing systems, where processes get a share of CPU at **regular intervals**-> **suitable for batch systems**



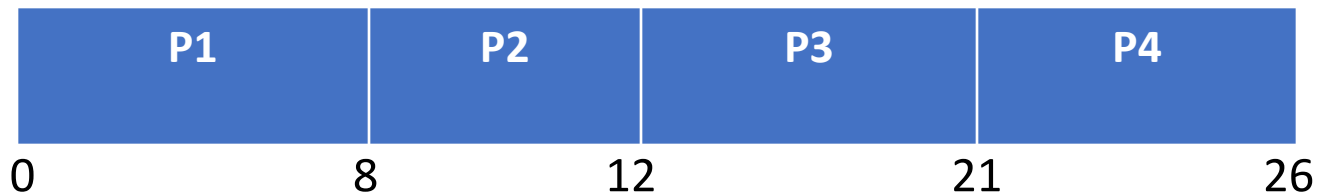
Chapter 4-5

FCFS Scheduling (Cont)

- Example without ignoring arrival time.

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Gantt Chart



- Average waiting time = $[(0) + (8-1) + (12-2) + (21-3)] / 4 = 8.75$ ms
- Average Turn Around time = $[(8-0) + (12-1) + (21-2) + (26-3)] / 4 = 15.25$ ms
- Throughput = $4 \text{ jobs} / 26 \text{ ms} = 0.15385 \text{ jobs/ms}$



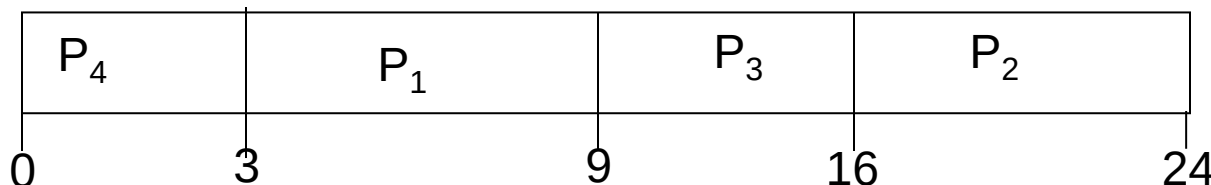
Chapter 4-5

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**. If several processes have the same CPU next burst time, use FCFS to select one

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	6
P_2	0.0	8
P_3	0.0	7
P_4	0.0	3

**Non-
preemptive
Version**



□ Non-Preemptive SJF scheduling Gantt chart (ignore arrival time)

□ **Turnaround time** for P_4, P_1, P_3 and P_2 are 3, 9, 16, 24 respectively

□ Average waiting time = $(0 + 3 + 9 + 16) / 4 = 7$ (ignore arrival time)

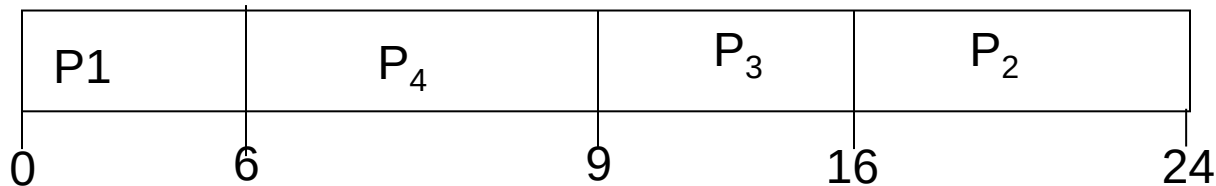
□ Same problem with FCFS = 10.25 msec

Chapter 4-5

Non-preemptive Version SJF

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

- Non-Preemptive SJF scheduling Gantt chart



- **Turnaround time** for P_1, P_2, P_3 and P_4 are 6, 24, 16, 6 respectively
- Average waiting time = $(0 + (16-2) + (9-4) + (6-5)) / 4 = 5$



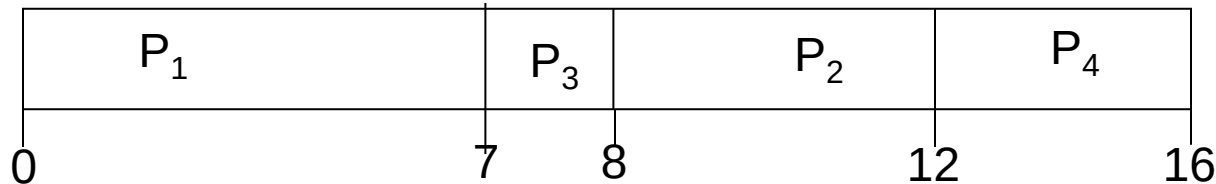
Chapter 4-5

Non-Preemptive Shortest-Job-First (SJF) Scheduling without Ignoring Arrival time

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

**Non-
preemptive
Version**

- Non-Preemptive SJF scheduling Gantt chart



- Turnaround time** for P_1, P_3, P_2 and P_4 are

(7-0), (8-4), (12-2), (16-5) respectively

, 4, 10, 11

- Average waiting time** = $(0 + (7-4) + (8-2) + (12-5)) / 4 = 4$



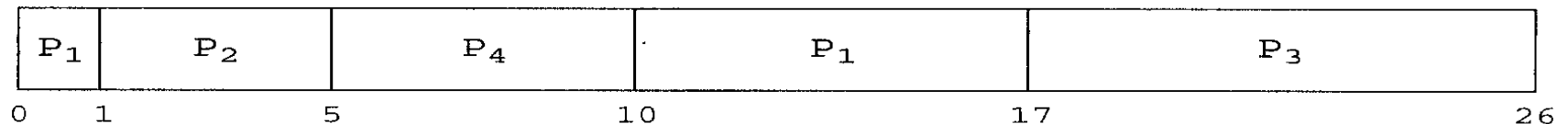
Chapter 4-5

Preemptive SJF Scheduler

- **Preemptive SJF is optimal** – gives minimum average waiting time for a given set of processes because it moves short process before a long one. **It can't be used for short-term scheduling**
- **The difficulty is knowing the length of the next CPU request.** It is used frequently in long-term scheduling in batch systems because user can estimate running time needed

**Preemptive
Version**

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



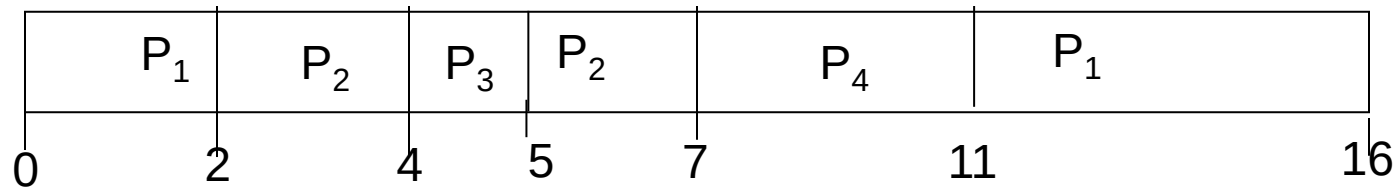
- Average waiting time = $[(10-1)+(1-1)+(17-2)+(5-3)]/4 = 6.5$ ms
- Average Turn Around time = $[(17-0)+(5-1)+(26-2)+(10-3)]/4 = 11.5$ ms
- Throughput = $4 \text{ jobs} / 26 \text{ ms} = 0.15385 \text{ jobs/ms}$
- Non-Preemptive *scheduling* waiting time = 7.75 ms

Chapter 4-5

Shortest-Job-First (SJF) Scheduling without Ignoring Arrival time

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

**preemptive
Version**



□ **Turnaround time** for P_1, P_2, P_3 and P_4 are

$(16-0), (7-2), (5-4), (11-5)$ respectively

16 , 5 , 1 , 6



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

Average waiting time = $((11-2) + (5-4) + (4-4) + (7-5)) / 4$

$$= (9 + 1 + 0 + 2) / 4 = 3$$

Chapter 4-5

Exercise 1

Assume you have the following jobs to execute with one processor, with the jobs arriving in the order listed here

i	$T(p_i)$
0	80
1	20
2	10
3	20
4	50

Suppose a system uses FCFS and SJF scheduling .

1. Create a Gantt chart illustrating the execution of these processes?
2. What is the average wait time for the processes?
3. What is the turnaround time for process p3?



Chapter 4-5

Exercise 2

Assume you have the following jobs to execute with one processor, with the jobs arriving in the order listed here

i	arrival Time	T(pi)
0	0	80
1	3	20
2	5	10
3	7	20
4	10	50

Suppose a system uses FCFS and SJF (preemptive and nonpreemptive) scheduling.

1. Create a Gantt chart illustrating the execution of these processes?
2. What is the average wait time for the processes?
3. What is the turnaround time for process p3?



End lecture 3
Thank You

