

Threads & Introduction to CPU Scheduling

Mohammad S. Hasan

Staffordshire University, UK

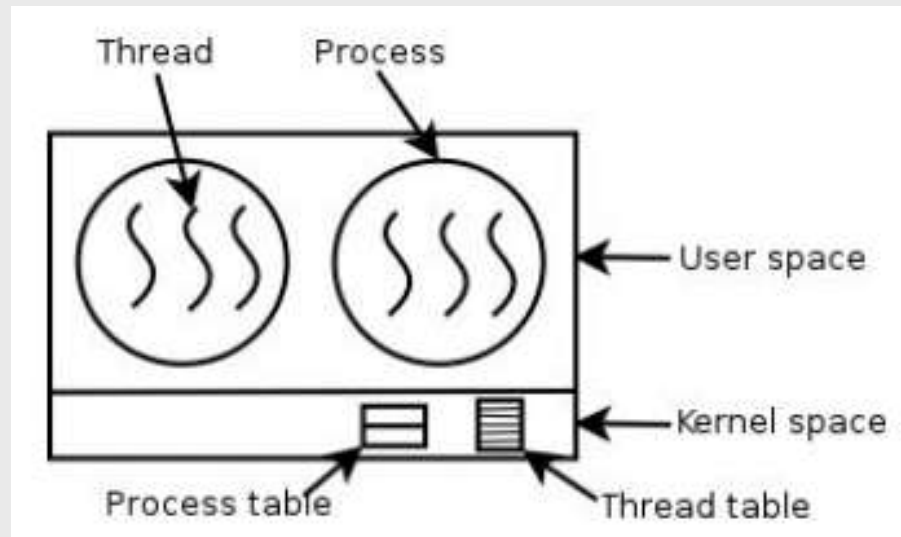
Lecture Outline

- ◆ Definition of a thread and types
- ◆ CPU -I/O burst cycle
- ◆ CPU scheduler
 - preemptive & non-preemptive
- ◆ Dispatcher
- ◆ Scheduling Criteria
- ◆ First Come First Served (FCFS) Scheduling
- ◆ Shortest-Job-First (SJF) Scheduling

Thread - Definition

- ◆ **Smallest unit of processing task for OS**
- ◆ **One or more threads per process**
- ◆ **All the threads can share the resources of the parent process**
- ◆ **Analogy**
 - **Components of a robot / human body**

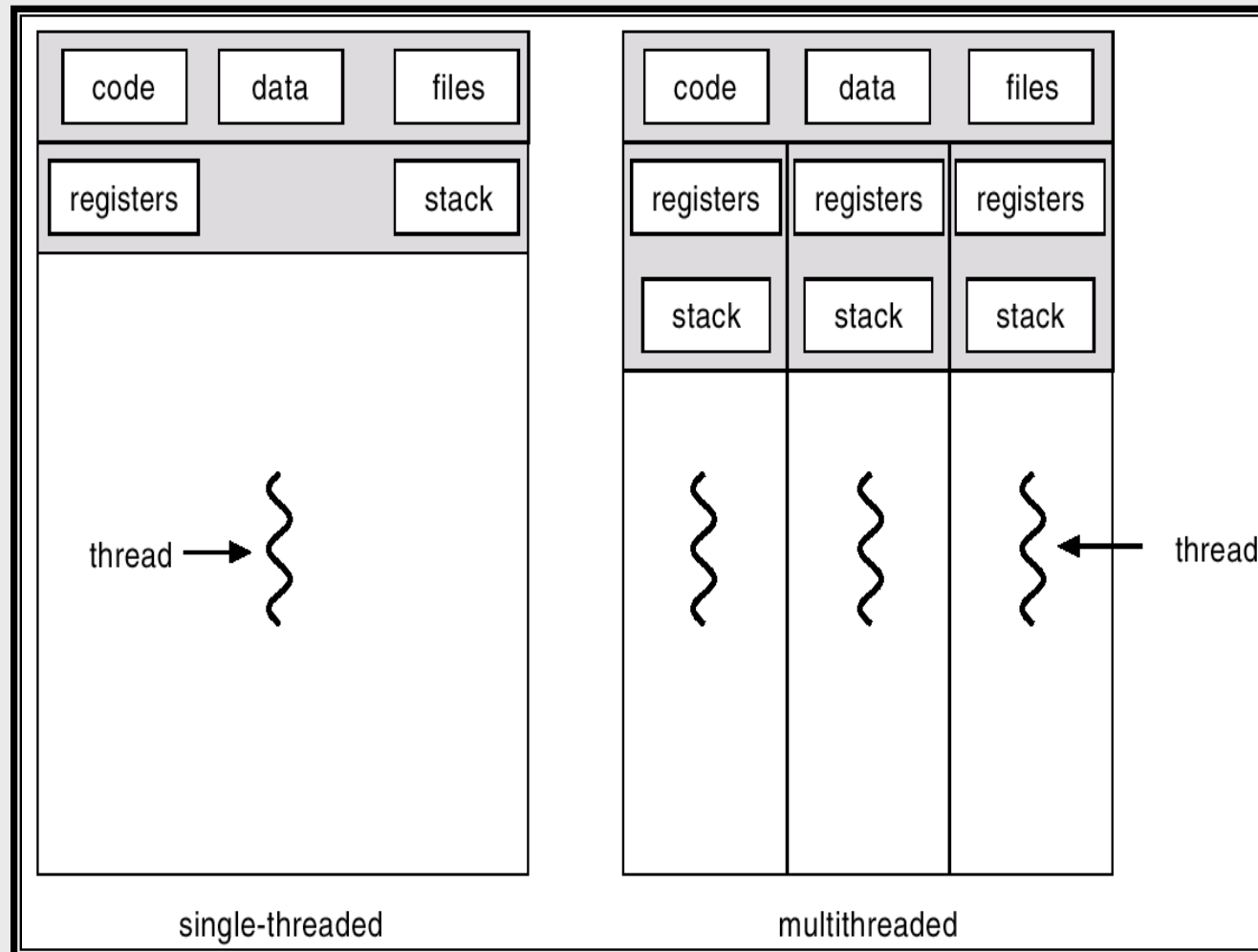
Thread Example



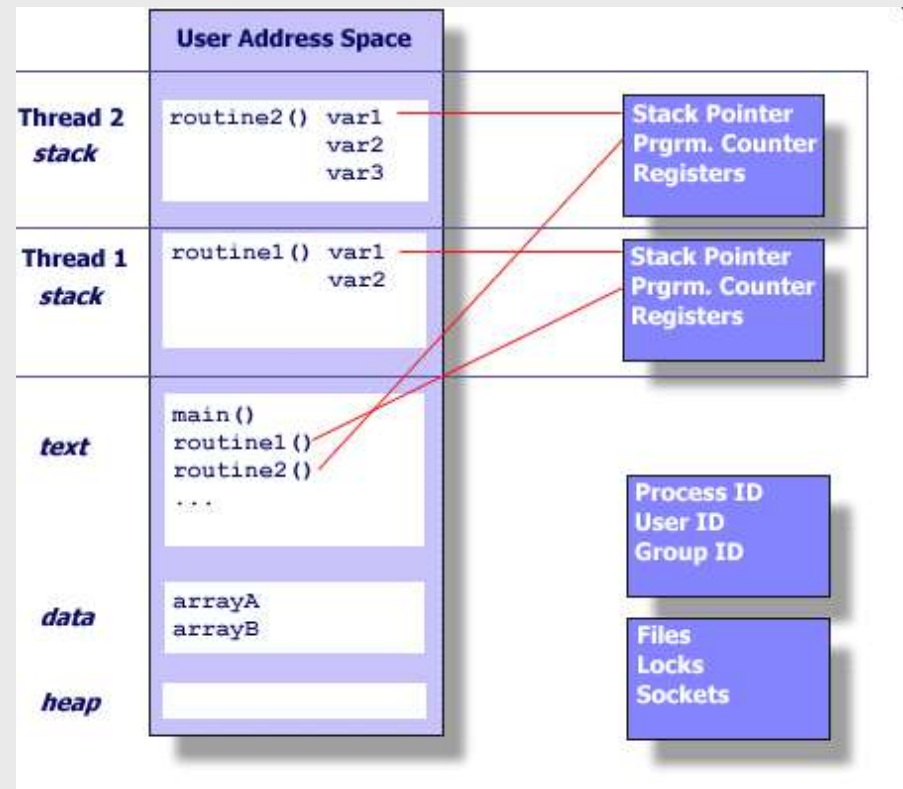
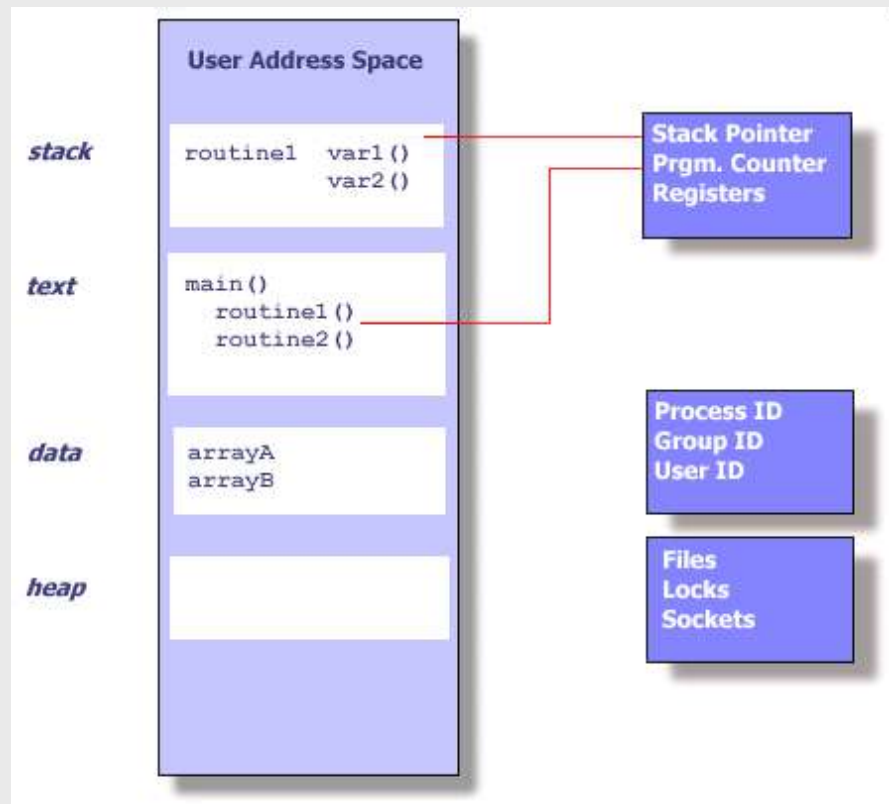
➤ Example

- A server process serving multiple clients
- Process: Microsoft Word
 - Thread – auto saving
 - Thread – spell checking

Single and Multithreaded Process



Single Threaded and Multi Threaded Process



Process and Thread (cont.)

❖ Process

- Independent entity
- Each process has its own address space
- Processes need to use inter-process Communication (IPC) mechanism to communicate
- Switching from one process to another is slower
- Higher overhead and longer delay for process creation

❖ Thread

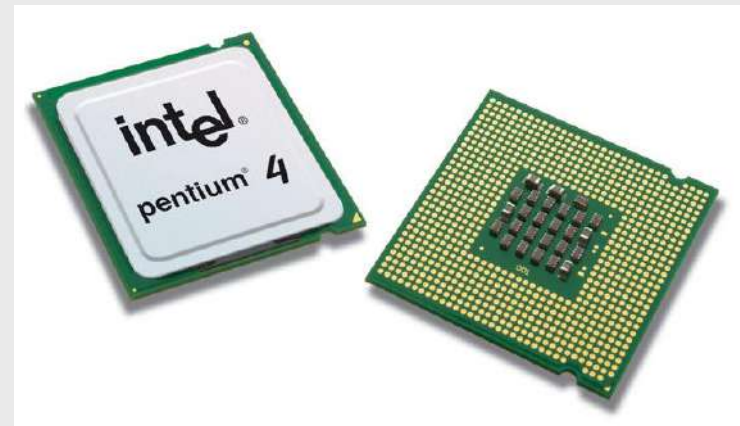
- Thread exists inside a process
- Threads share the address space of their parent process
- Threads within a process can directly communicate
- Switching from one Thread to another within the same process is much faster
- A new thread creation within an existing process is much faster and needs less overhead for the OS



Intel Processor and Threading

Intel's Hyper-Threading Technology or HT Technology

- The OS treats each processor core as separate processor
- Execution of two or more threads simultaneously
- Load sharing



Processor core and Threads

- ◆ **Single core processor**
 - **Time division multiplexing to execute the threads**
- ◆ **Dual and Quad core processors**
 - **Run two and four threads at a time**



Thread Types

◆ Kernel Threads

- Supported by the Kernel

- Examples

- Windows
- Solaris
- Tru64 UNIX
- Linux

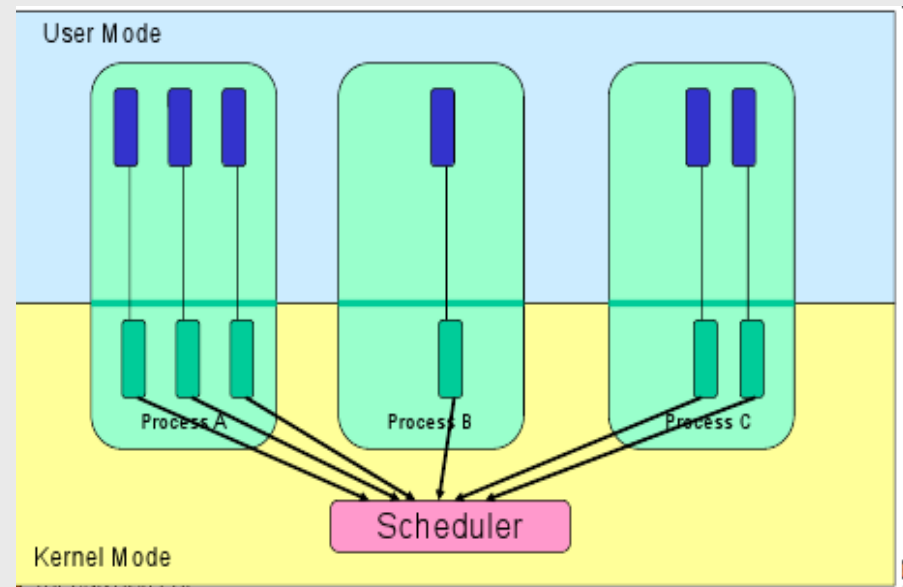
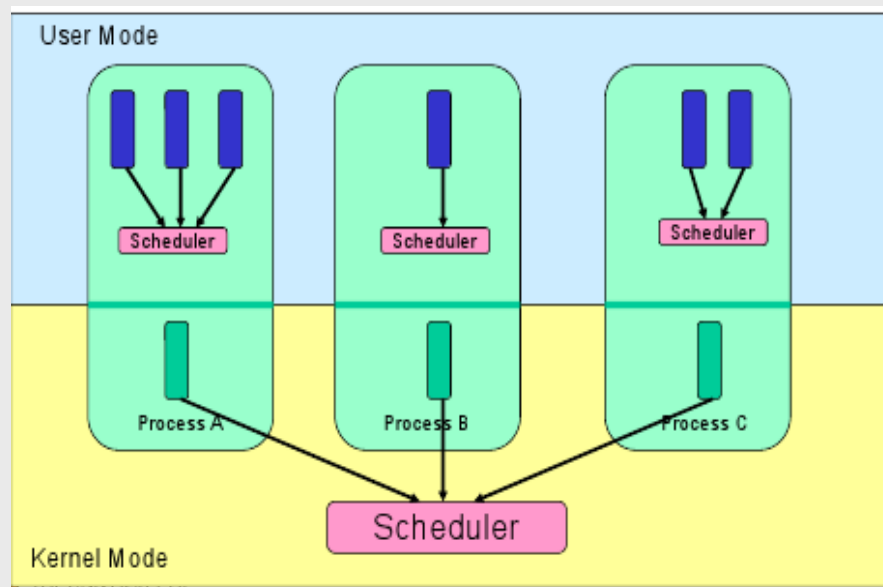
◆ User Thread

- Thread management done by user-level threads library

- Examples

- POSIX Pthreads
- Mach C-threads
- Solaris threads

Thread Types: One-To-Many and One-To-One



Windows Thread Model

➤ Implements the one-to-one mapping

➤ Each thread contains

- a thread id
- register set
- separate user and kernel stacks
- private data storage area

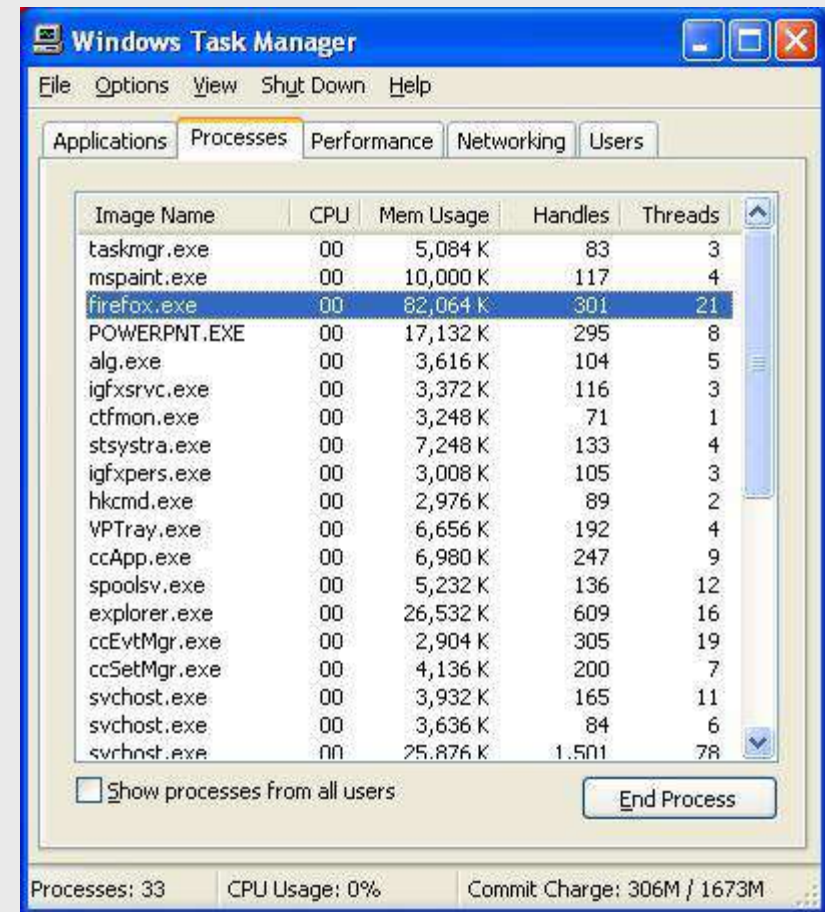


Image Name	CPU	Mem Usage	Handles	Threads
taskmgr.exe	00	5,084 K	83	3
mspaint.exe	00	10,000 K	117	4
firefox.exe	00	82,064 K	301	21
POWERPNT.EXE	00	17,132 K	295	8
alg.exe	00	3,616 K	104	5
igfxsvc.exe	00	3,372 K	116	3
ctfmon.exe	00	3,248 K	71	1
stsysstra.exe	00	7,248 K	133	4
igfxpers.exe	00	3,008 K	105	3
hkcmd.exe	00	2,976 K	89	2
VPTray.exe	00	6,656 K	192	4
ccApp.exe	00	6,980 K	247	9
spoolsv.exe	00	5,232 K	136	12
explorer.exe	00	26,532 K	609	16
ccEvtMgr.exe	00	2,904 K	305	19
ccSetMgr.exe	00	4,136 K	200	7
svchost.exe	00	3,932 K	165	11
svchost.exe	00	3,636 K	84	6
svrhost.exe	00	25,876 K	1,501	78

Processes: 33 CPU Usage: 0% Commit Charge: 306M / 1673M

CPU-I/O Burst Cycle

CPU-I/O Burst Cycle

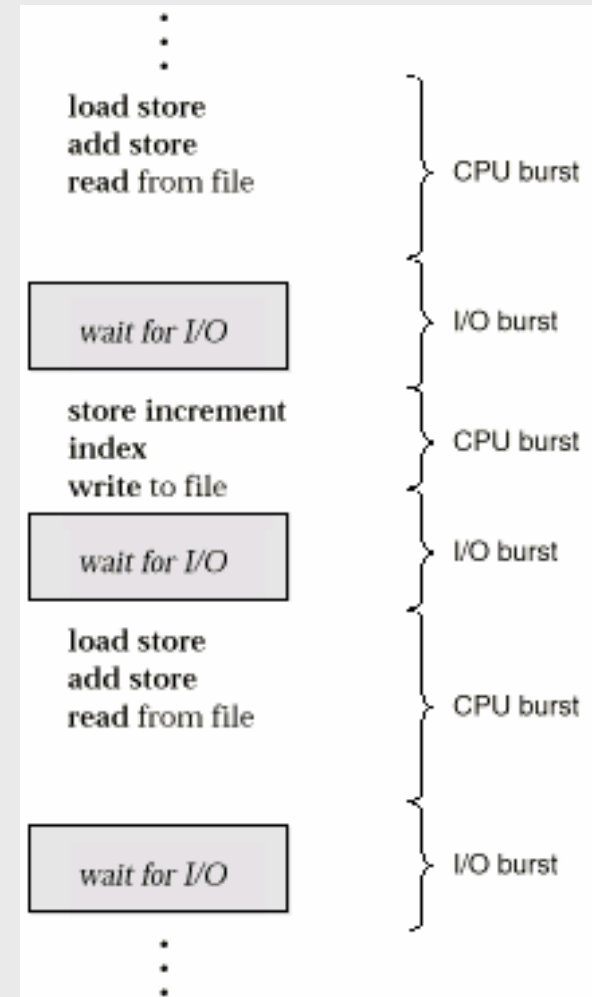
- Process execution consists of a cycle of CPU execution and I/O wait.

CPU burst

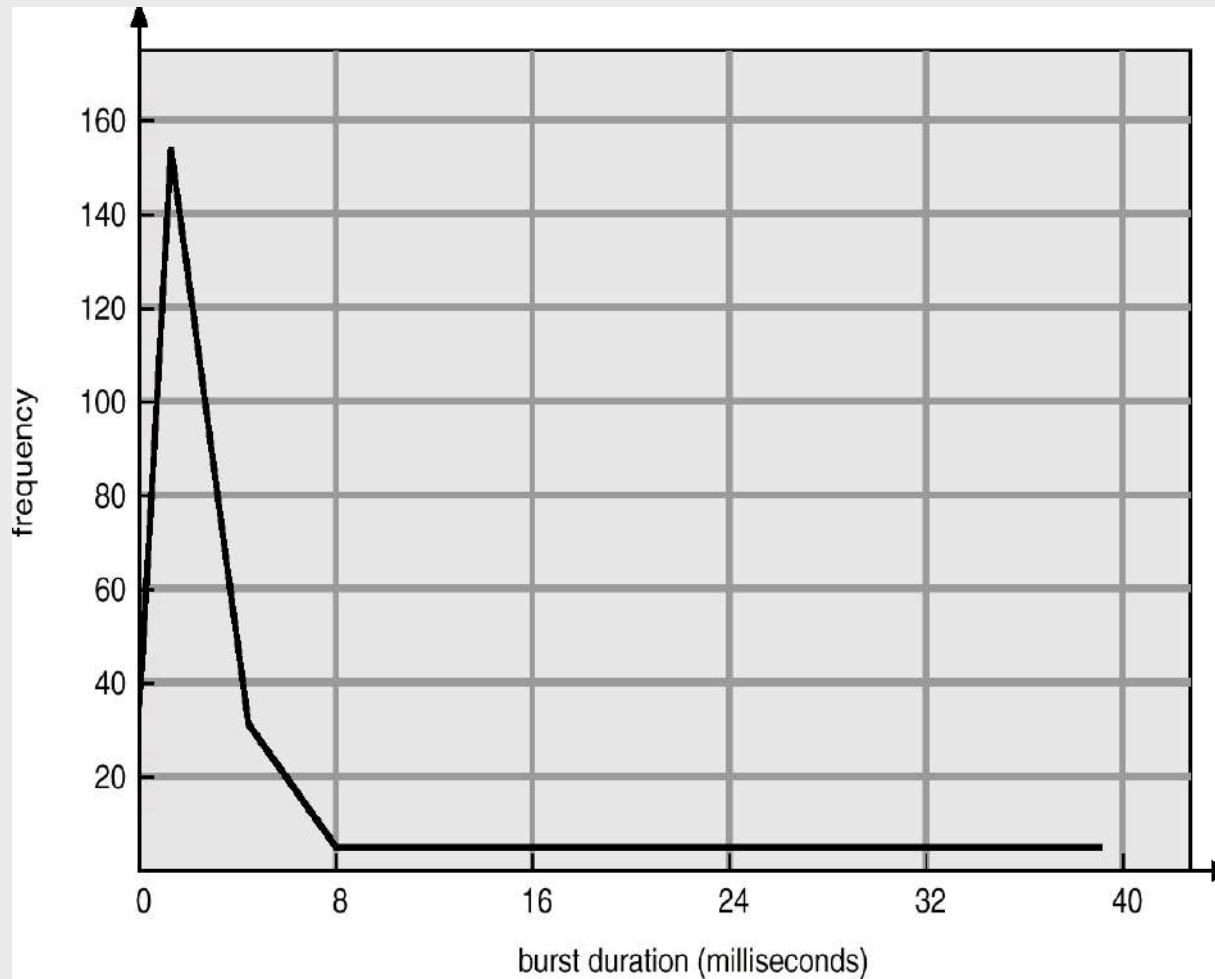
- length of time process needs to use CPU before it next makes a system call (normally request for I/O).

I/O burst

- length of time process spends waiting for I/O to complete.

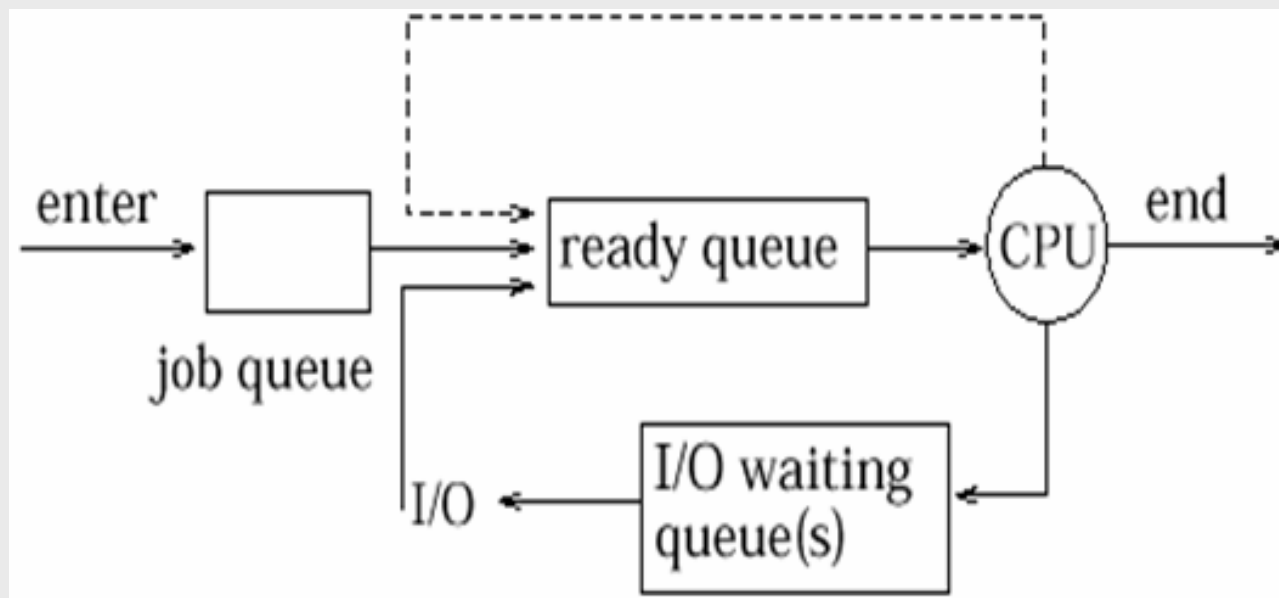


Typical CPU burst distribution



CPU Scheduler

- ◆ Allocates CPU to one in the ready queue



CPU Scheduler (cont.)

- ◆ **CPU scheduling occurs when a process**
 - **Case 1. Switches from running to waiting state (e.g. when I/O request)**
 - **Case 2. Terminates**
 - **Case 3. Switches from waiting to ready (e.g. on I/O completion)**
 - **Case 4. Switches from running to ready state (e.g. Timer interrupt)**

CPU Scheduler (cont.)

◆ Non-preemptive scheduling

- if scheduling occurs only when case 1 or 2 happens
- process keeps CPU until it voluntarily releases it (process termination or request for I/O)

◆ Preemptive

- If scheduling occurs when case 3 or 4
- CPU can be taken away from process by OS (external I/O interrupt or timer interrupt)

The Dispatcher

- ◆ Gives control of the CPU to the process selected by the short-term scheduler
 - switching context
 - switching to user mode
 - jumping to the proper location to restart the process (i.e. last action is to set program counter)
- ◆ Dispatch latency
 - time it takes for the dispatcher to switch between processes and start new one running

Scheduling Criteria

❖ CPU utilisation

● CPU usage

■ to maximise

❖ Throughput

● number of completed processes per time unit

■ to maximise

❖ Turnaround time

● amount of time to execute a particular process

■ to minimise

Scheduling Criteria (cont.)

◀ **Waiting time**

- **amount of time a process has been waiting in the ready queue**

- **to minimise**

◀ **Response time**

- **amount of time it takes from when a job was submitted until it initiates its first response (output)**

- **to minimise**

First Come First Served (FCFS) Scheduling

◆ Order of arrival of process in ready queue

◆ Example

● Process	Burst Time
● P1	24
● P2	3
● P3	3

◆ Suppose that the order of arrival

● P1 , P2 , P3.

FCFS Scheduling – Example 1

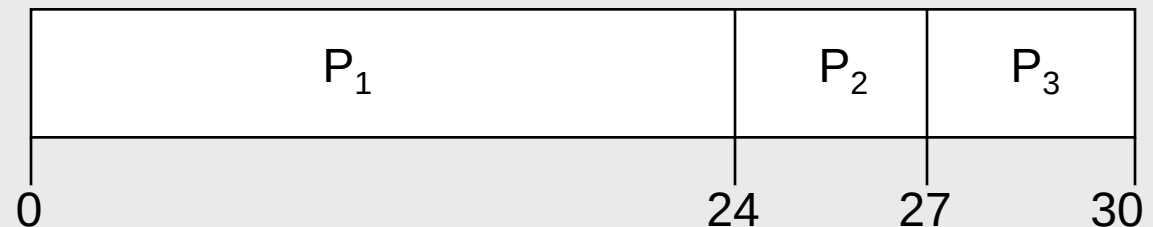
◆ The Gantt Chart for the schedule

◆ Waiting time

● $P1 = 0; P2 = 24; P3 = 27$

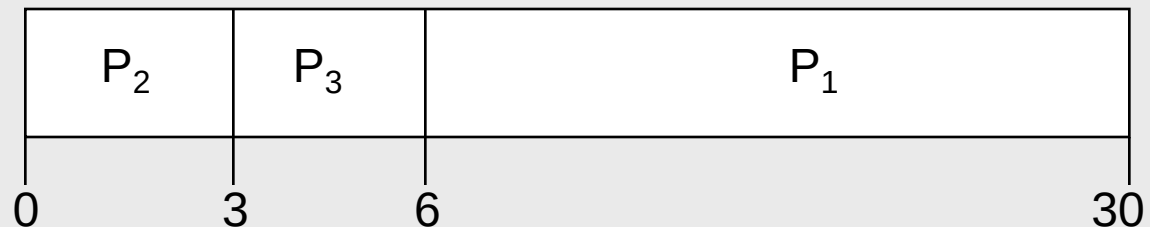
◆ Average waiting time

● $(0 + 24 + 27)/3 = 17$



FCFS Scheduling – Example 2

- ◆ Suppose that order of arrival
 - P2 , P3 , P1
- ◆ The Gantt chart for the schedule
- ◆ Waiting time
 - P1 = 6; P2 = 0; P3 = 3
- ◆ Average waiting time
 - $(6 + 0 + 3)/3 = 3$



FCFS Problems

- ◆ **Waiting time**
 - usually not minimal and large variance in times
- ◆ **Convoy effect**
 - short process may have a long wait before being scheduled onto CPU due to long process being ahead of them

Shortest-Job-First (SJF) Scheduling

- ◆ **Schedules the process with the next shortest CPU burst**
- ◆ **Two schemes**
 - **non-preemptive** – 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 $<$ remaining time of current executing process, preempt.
 - **Also known as Shortest-Remaining-Time-First (SRTF).**
- ◆ **SJF is optimal**
 - **minimum average waiting time for a given set of processes.**

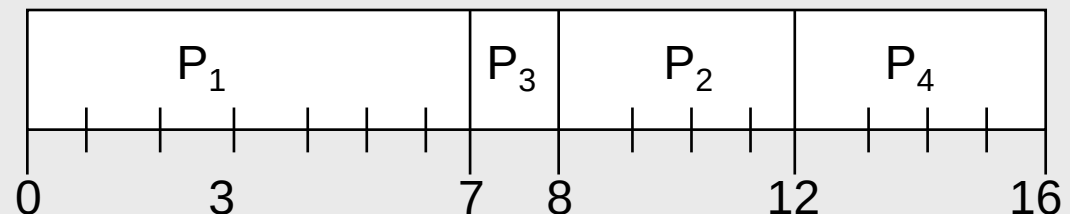
Non-Preemptive SJF Example

Processes: P1, P2, P3 and P4

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

Average waiting time

$$(0 + 6 + 3 + 7)/4 = 4$$



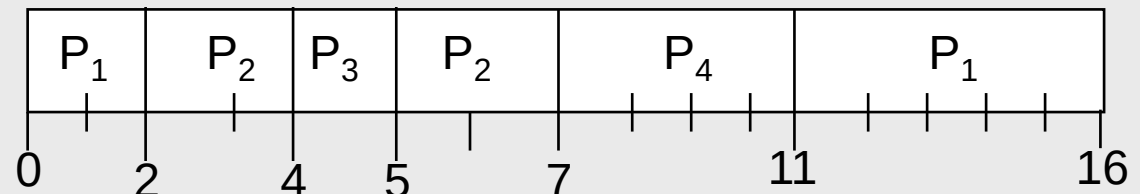
Preemptive SJF Example

Processes: P1, P2, P3 and P4

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

Average waiting time

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



SJF Problem

- ◆ **Determining Length of Next CPU Burst**
 - **Can only estimate the length.**
 - **Using the length of previous CPU bursts**
 - **exponential averaging (decaying average)**

Exponential Averaging (decaying average)

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n.$$

Exponential Averaging (decaying average)

Define the following terms

- t_n = Actual length of n^{th} CPU burst
- τ_{n+1} = Predicted value for the next CPU burst
- α = weight value where $0 \leq \alpha \leq 1$

The next CPU burst can be predicted by the following formula

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n.$$

Exponential Averaging (cont.)

◆ Exponential Averaging formula

- $\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$

◆ If $\alpha = 0$ then $\tau_{n+1} = \tau_n$

- last CPU burst does not count - only longer term history

◆ If $\alpha = 1$ then $\tau_{n+1} = t_n$

- Only the actual last CPU burst counts.

Exponential Averaging (cont.)

◆ By expanding the formula

$$\begin{aligned}\tau_{n+1} = & \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \dots \\ & + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

◆ $\alpha \leq 1$ and $(1 - \alpha) \leq 1$

- each successive term has less weight than its predecessor.

References

◆ **Operating System Concepts - Chapter 4 & 5**