CPU Scheduling

Read textbook chapter 5

Scheduling CPU is taken also as scheduling a single core
In a multicore system

Process scheduling is taken as thread scheduling

Read what the “Dispatcher” is and what it does

Main goal of CPU scheduling is “keep CPU busy all the time”

CPU and I/O Bursts

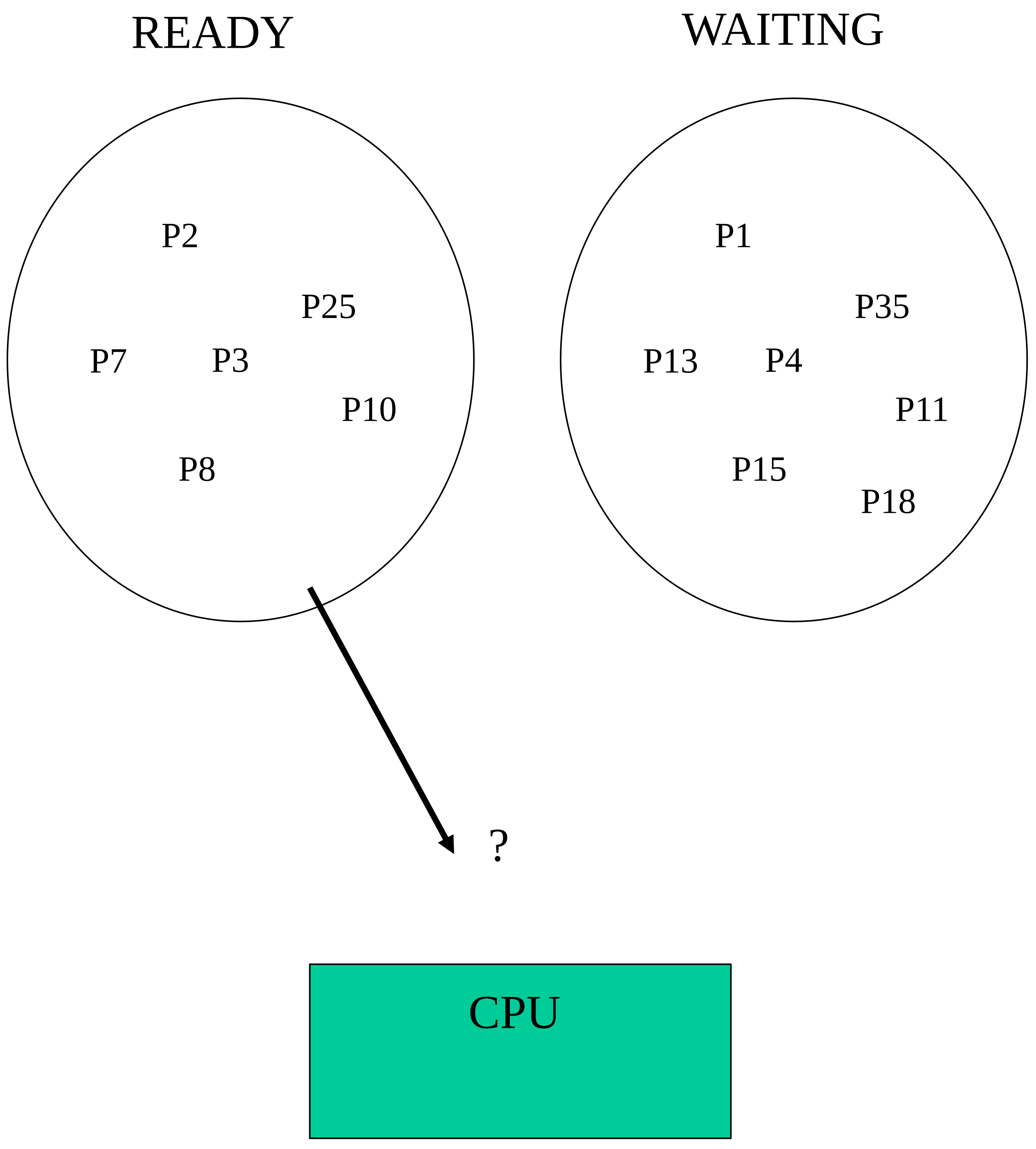
⋮

Initialize structures Do computations Read from file	CPU Burst
WAIT for I/O	I/O Burst
Modify structures Do computations Write from file	CPU Burst
WAIT for I/O	I/O Burst
Initialize structures Do computations Read from file	CPU Burst
WAIT for I/O	I/O Burst

⋮

Typically processes have many short CPU bursts and few long ones

How to select next process to run ?



General Criteria for CPU Scheduling

CPU utilization

(40% light 90% heavy)

Throughput

(# of processes per unit time)

MAXIMIZE

Waiting time

(time spent waiting in the ready queue)

Response time

(for interactive systems)

MINIMIZE

CPU scheduling cannot affect the CPU time nor the I/O time required by a process. However, it affects the amount of time a process spends in the ready queue and the average waiting time for processes in the ready queue.

First Come First Served

FCFS or FIFO

Processes are assigned the CPU in the same order as they arrive at the Ready Queue

Advantage: Easy to implement and fairness

Disadvantage: Low overall system throughput

Example:

Process	Next Burst Time
P1	30
P2	6
P3	3

Average waiting time : (0 + 30 + 36)/3 = 22 msec

Shortest Next CPU Burst First

Non - Preemptive

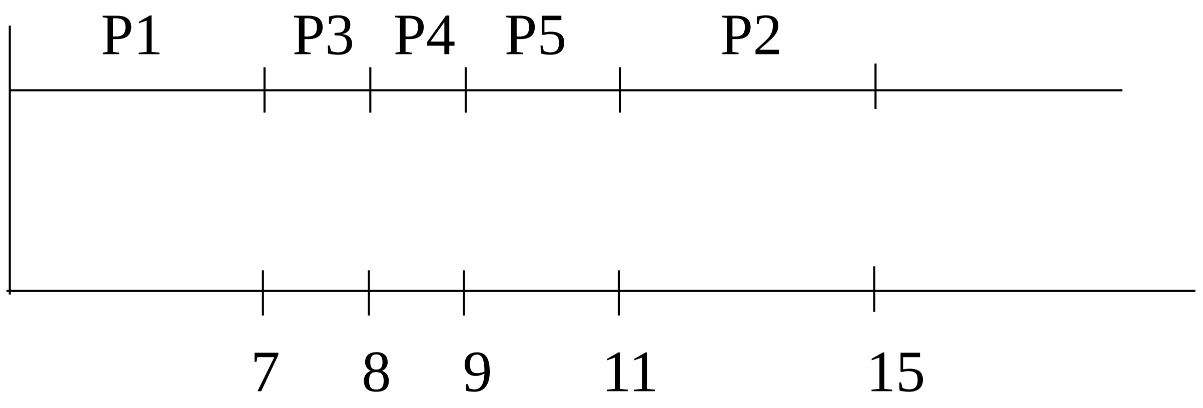
Give the CPU to the process with the smallest next burst time

Advantage: easy to implement

Disadvantage: Non preemptive nature increases average waiting time

Example:

Process	Arrival time	Estimated next run time
P1	0	7
P2	1	4
P3	2	1
P4	3	1
P5	6	2



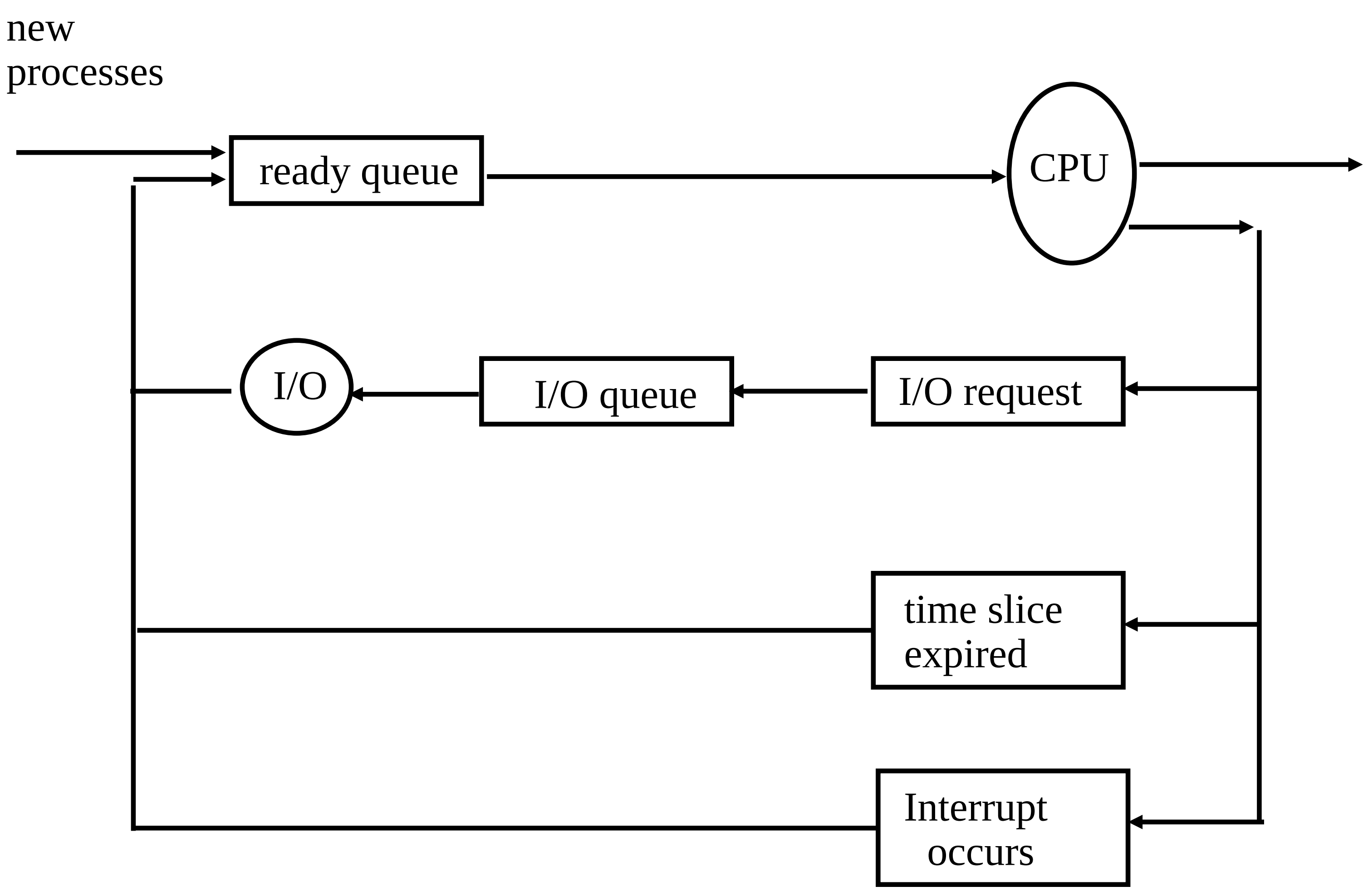
P1 waits:0 P2 waits:10 P3 waits:5 P4 waits:5 P5 waits:3

Average waiting time : (0 + 10 + 5 + 5 + 3)/5 = 4.6

What would the average waiting time been if I had used FCFS ?

Round Robin Scheduling

When an interrupt occurs, the process that had the CPU is placed at the tail of the ready queue.
So, this is a Pre-emptive scheduling scheme

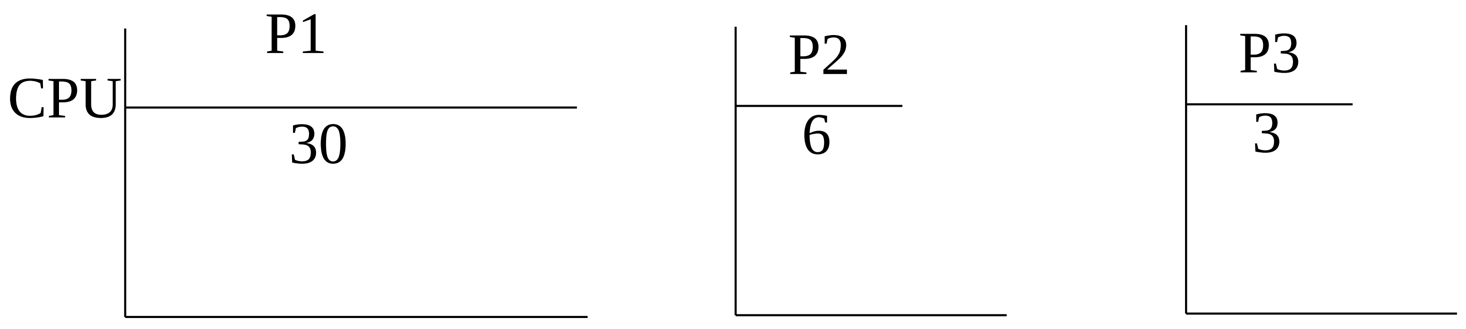


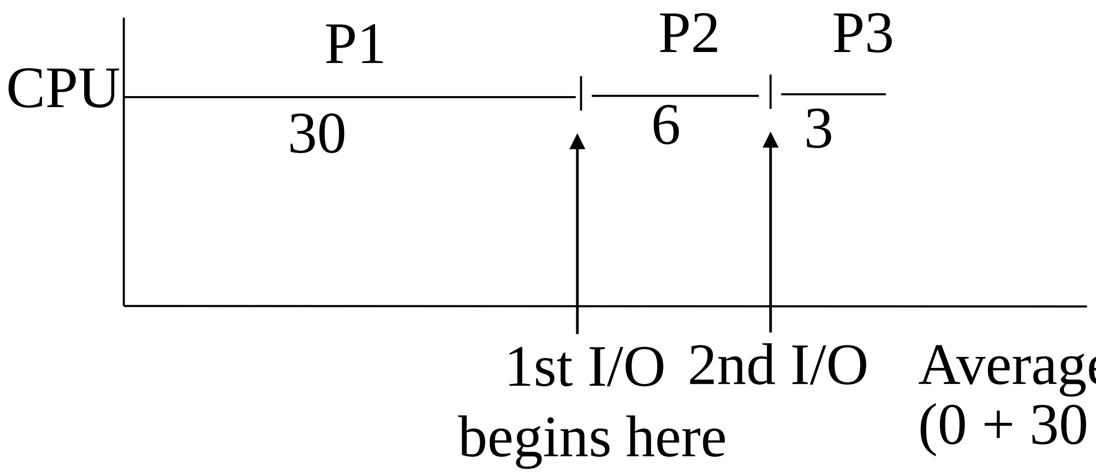
Advantage: Easy to implement for time sharing environments and fairness

Disadvantage: Long average waiting time

Lowering Average Waiting Time

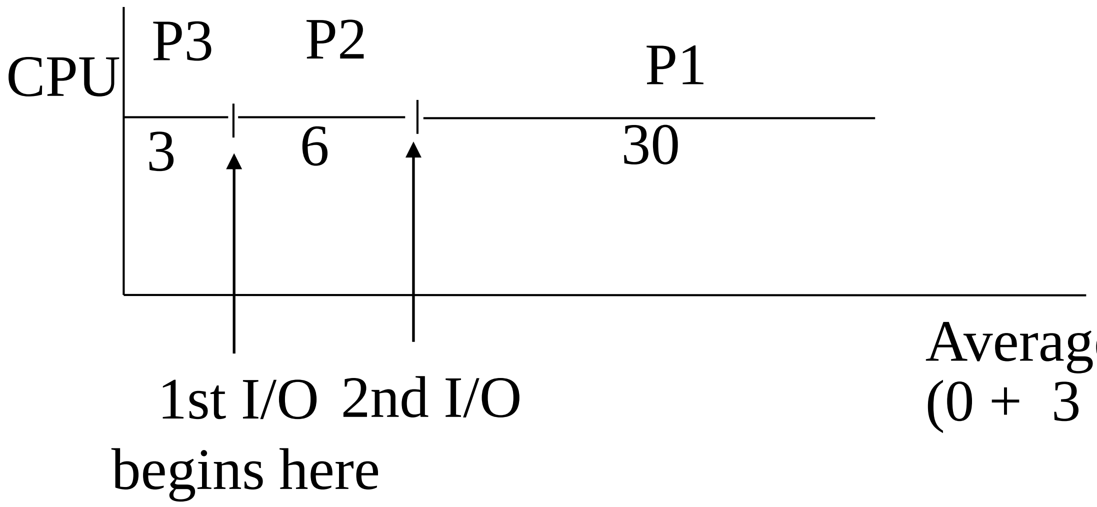
Shortest Next CPU Burst Scheduling
Increases Throughput





FCFS
scheduling

Average waiting time :
(0 + 30 + 36)/3 = 22 msec



Short CPU
Burst First
scheduling

Average waiting time :
(0 + 3 + 9)/3 = 4 msec

The earlier the I/O begins - the more work done by the **system** !!

Shortest Remaining Time First
Preemptive

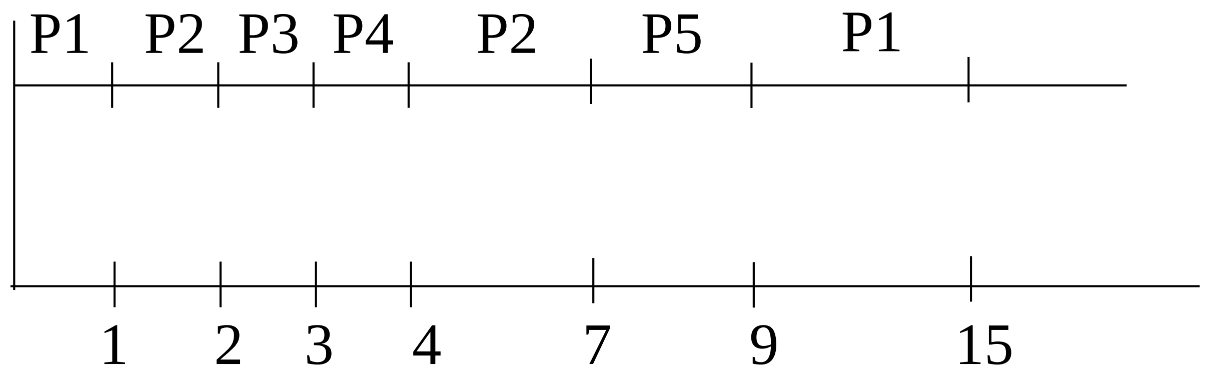
After every interrupt select the process with shortest next burst time

Advantage: Can yield minimum average waiting time

Disadvantage: Increased Overhead

Example:

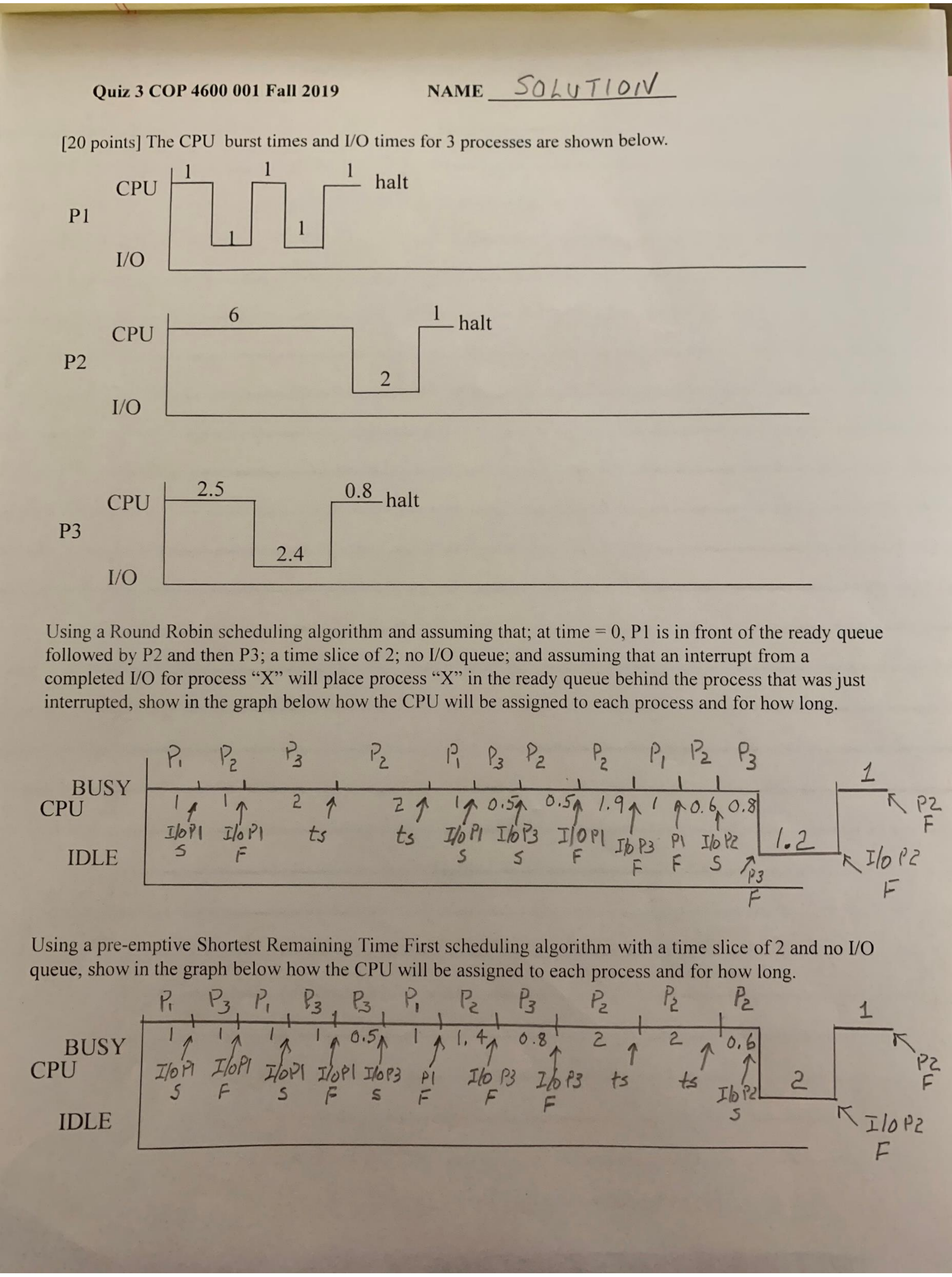
Process	Arrival time	Estimated run time
P1	0	7
P2	1	4
P3	2	1
P4	3	1
P5	6	2



P1 waits:8 P2 waits:2 P3 waits:0 P4 waits:0 P5 waits:1

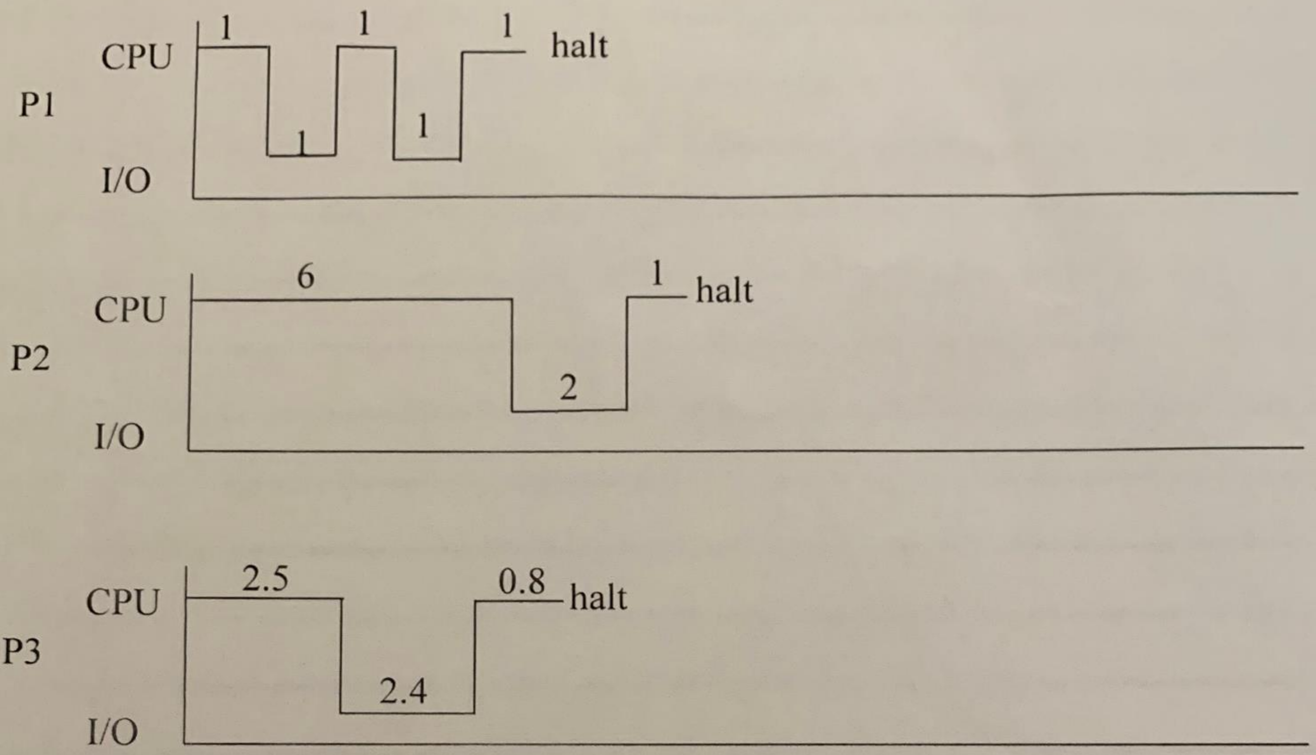
Average waiting time : (8 +2 + 0 + 0 + 1)/5 = 2.2

Work this problem

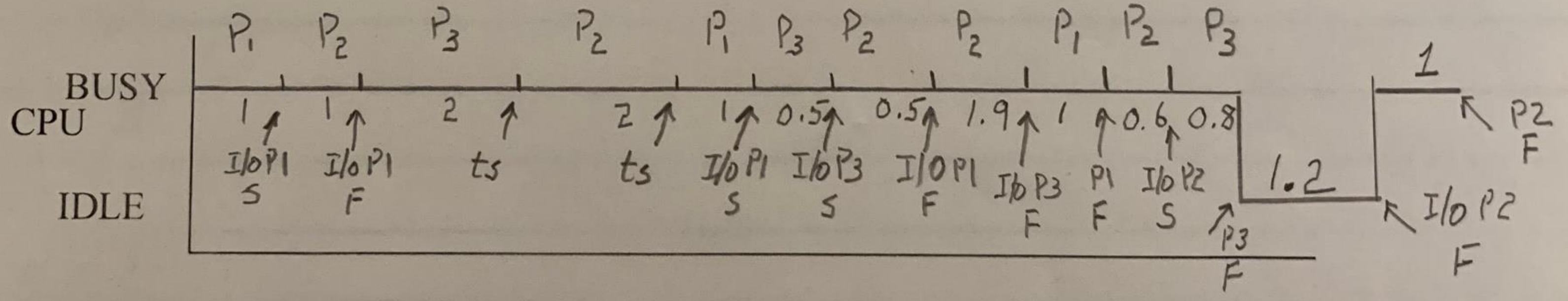


Quiz 3 COP 4600 001 Fall 2019 **NAME** SOLUTION

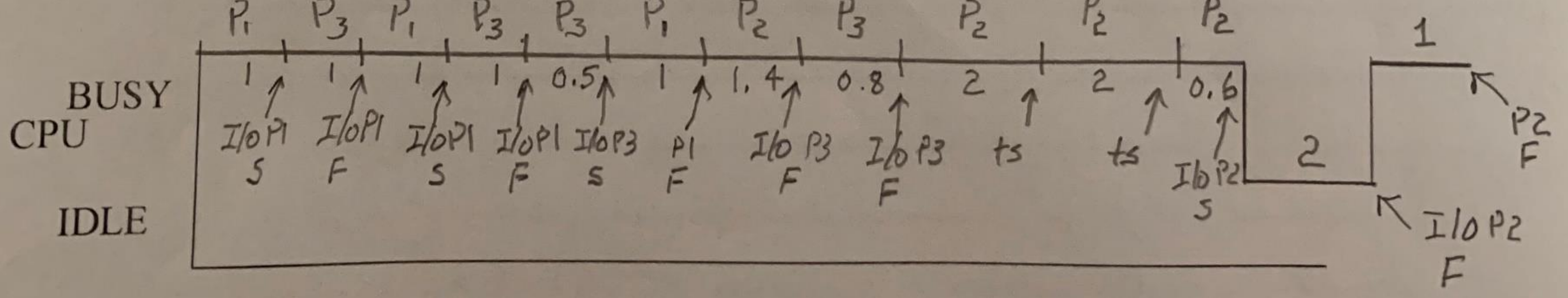
[20 points] The CPU burst times and I/O times for 3 processes are shown below.



Using a Round Robin scheduling algorithm and assuming that; at time = 0, P1 is in front of the ready queue followed by P2 and then P3; a time slice of 2; no I/O queue; and assuming that an interrupt from a completed I/O for process “X” will place process “X” in the ready queue behind the process that was just interrupted, show in the graph below how the CPU will be assigned to each process and for how long.



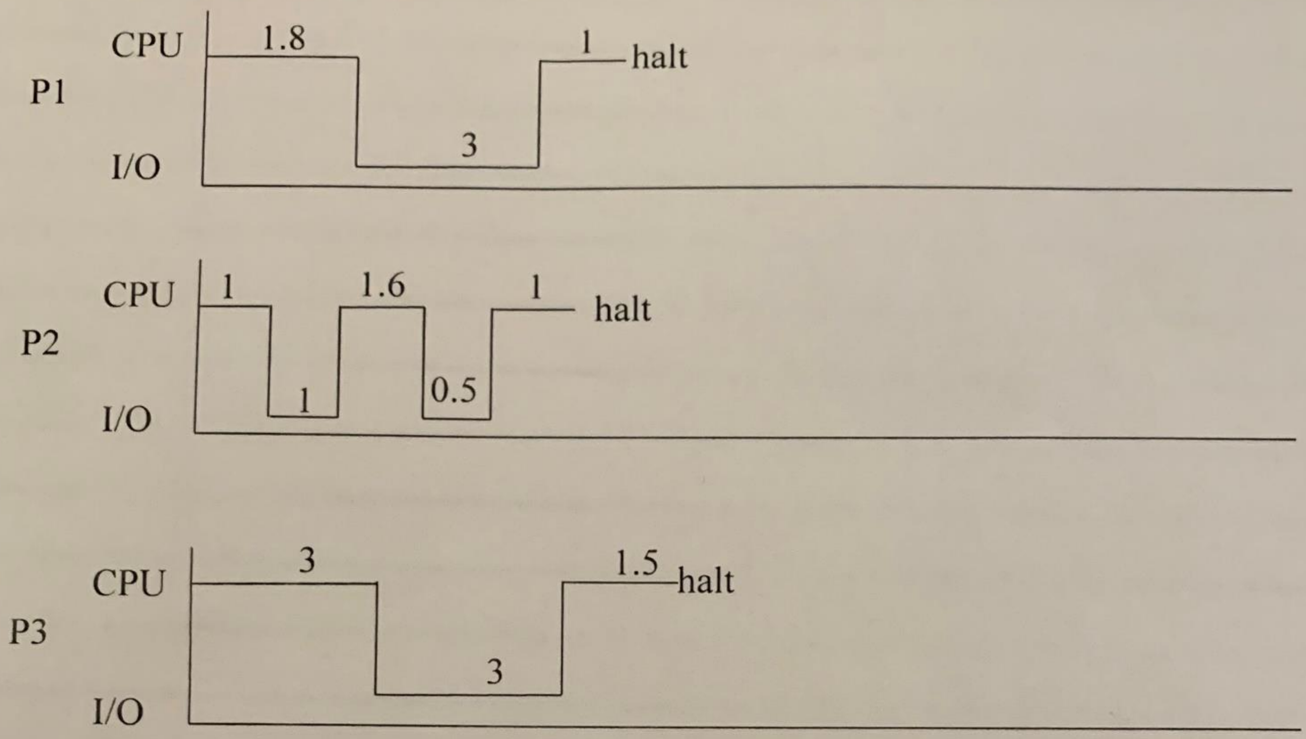
Using a pre-emptive Shortest Remaining Time First scheduling algorithm with a time slice of 2 and no I/O queue, show in the graph below how the CPU will be assigned to each process and for how long.



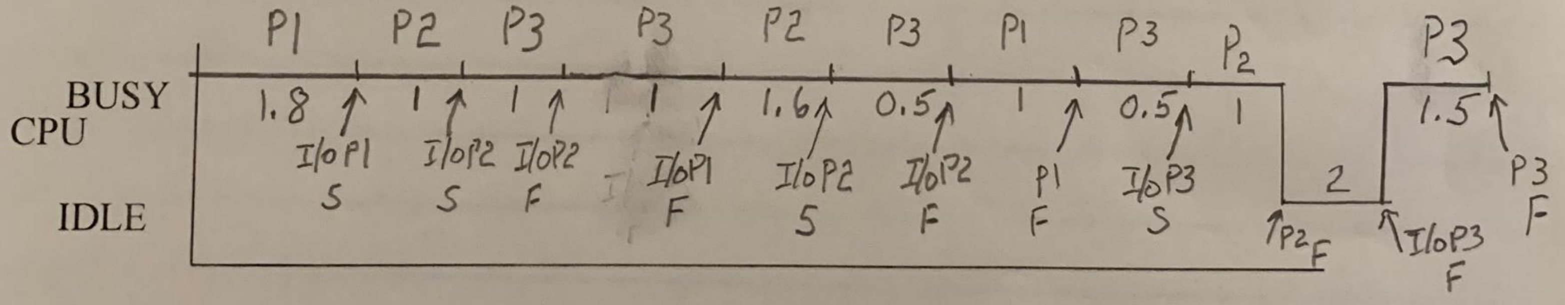
Work this Problem

Quiz 3 COP 4600 002 Fall 2019 **NAME** SOLUTION

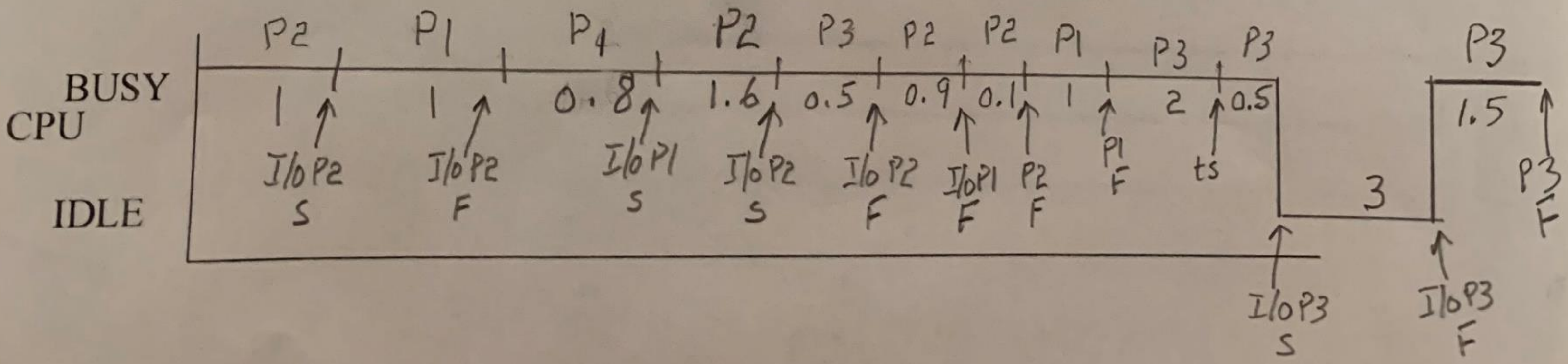
[20 points] The CPU burst times and I/O times for 3 processes are shown below.



Using a Round Robin scheduling algorithm and assuming that; at time = 0, P1 is in front of the ready queue followed by P2 and then P3; a time slice of 2; no I/O queue; and assuming that an interrupt from a completed I/O for process "X" will place process "X" in the ready queue behind the process that was just interrupted, show in the graph below how the CPU will be assigned to each process and for how long.



Using a pre-emptive Shortest Remaining Time First scheduling algorithm with a time slice of 2 and no I/O queue, show in the graph below how the CPU will be assigned to each process and for how long.



Prediction of Next CPU burst

Predict future burst using past bursts

Predicted time = k(last burst time) + (1 - k) last prediction

Start by using some system average

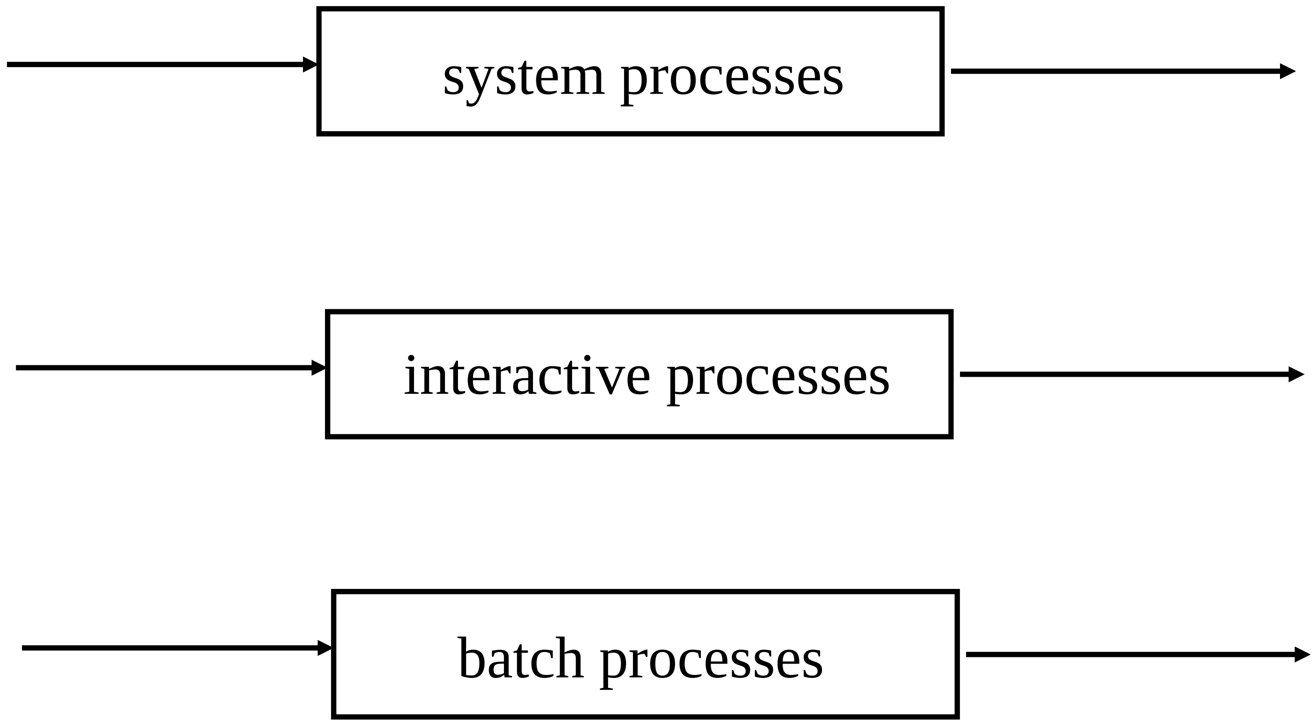
Example (with k = 1/2)

	prediction	actual
0	10	6
1	8	6
2	7	3
3	5	3
4	4	6
5	5	7
6	6	7

Multilevel Queue Scheduling

Processes are assigned to different scheduling queues.

Each queue may have absolute priority over those on levels below. Or time slices could be allocated to each different queue



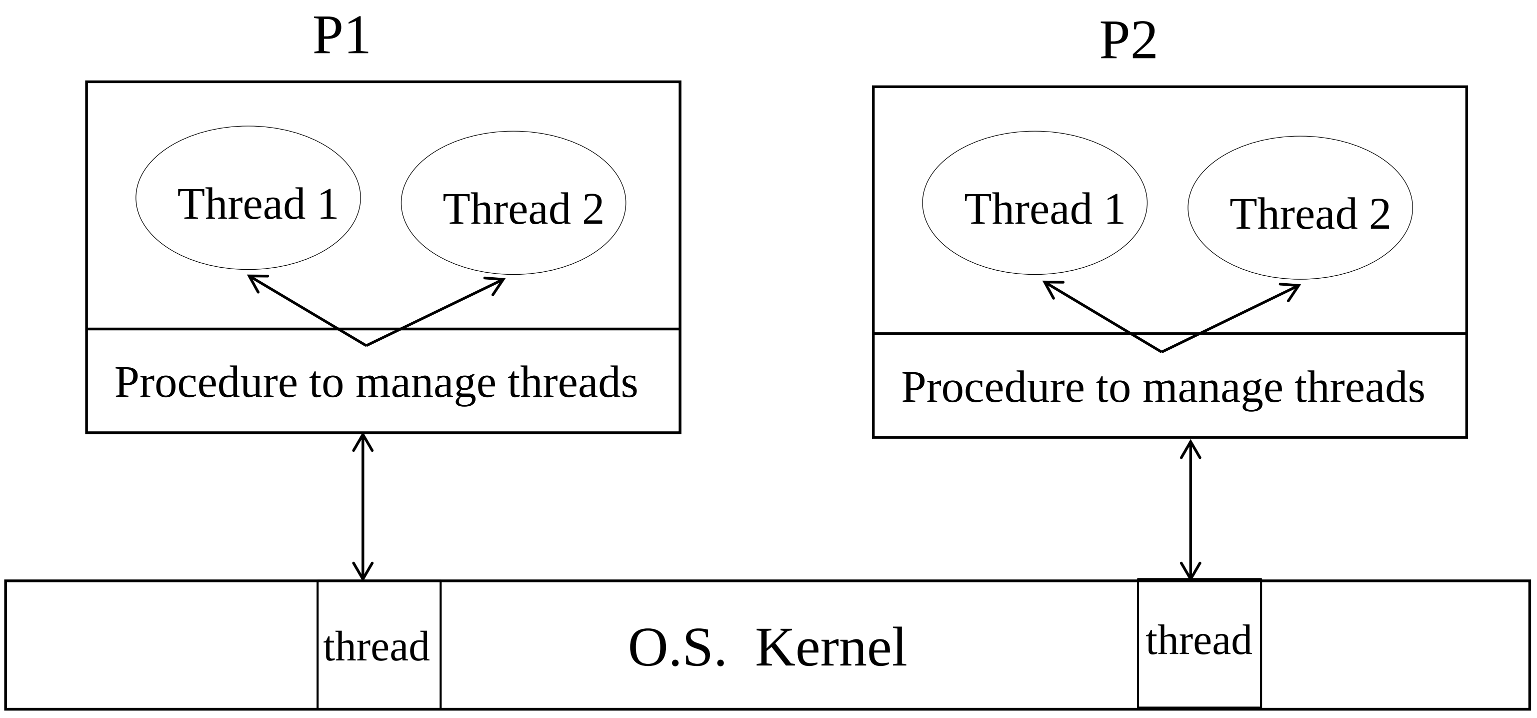
Scheduling Threads

Depends on which model the operating System uses

- Many-to-One Model

- One-to-One Model

Many-to-One Model



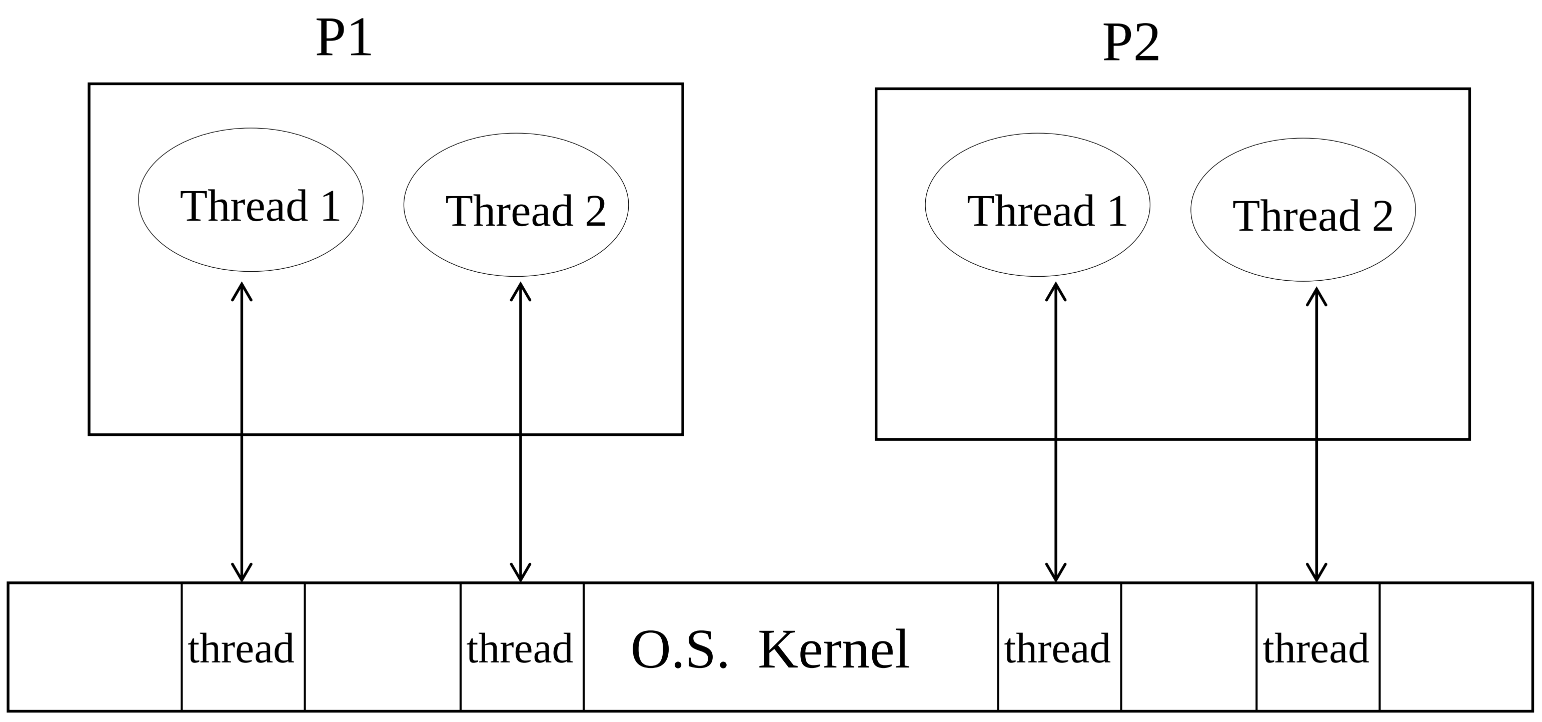
Kernel sees only the processes -- it does not know about switching among threads within a process. If a thread makes a system call, all threads within that process are blocked

Switching among threads of a process is done by calls to library modules and is done very quickly (an order of magnitude faster that kernel-level thread switching). The individual thread must voluntarily give up the CPU

Does not take advantage of multiple processing cores

Available in early versions of Java

One-to-One Model



Kernel sees every thread of every process. If a thread makes a system call, other threads within that process can run

Switching among threads of a process is done via interrupts. Context switching is slower that for user-level threads but faster than process context switching.

Takes advantage of multiprocessors

Disadvantage of allowing the creation of many threads – restrictions placed on total number created

Linux and Windows use this model.

Scheduling Multiprocessors or Multicore CPUs

Simplifying assumption:

processors are identical in functionality so that any process can run in any processor

Asymmetric multiprocessing:

One “master” processor schedules and executes all system calls including I/O. all other processors execute only user code

Only the “master” processor accesses OS data structures, making it simpler to implement

Symmetric multiprocessing (SMP):

each processor has its own scheduler which selects a thread to run from the ready queue.
Each processor may have its own queue or there maybe a common queue. See Fig 5.11 in textbook
Linux,Windows, and MAC OS X use this model.

Processor Affinity

It is most efficient to keep a process executing in the same processor because cached data would need to be repopulated in another processor cache.

Soft Affinity

when an OS attempts to keep a process running on the same processor – but does not guarantee it.

Hard Affinity

when an OS allows a process to specify to run in only one processor

Linux uses soft affinity but it has a system call that can set hard affinity for the process

Issues Connected to Symmetric Multiprocessing

Load Balancing

Keep workload evenly distributed across all processors.

Needed only when each processor has its own ready queue

Push Migration approach to load balancing

one OS task checks load on all processors and
moves (pushes) processes from one ready queue to
another to achieve proper load balance

Pull Migration approach to load balancing

an idle processor pulls a process from the ready queue
of a busy processor.

Linux uses a combination of both approaches.