

CPU Scheduling

Didem Unat
Lecture 5

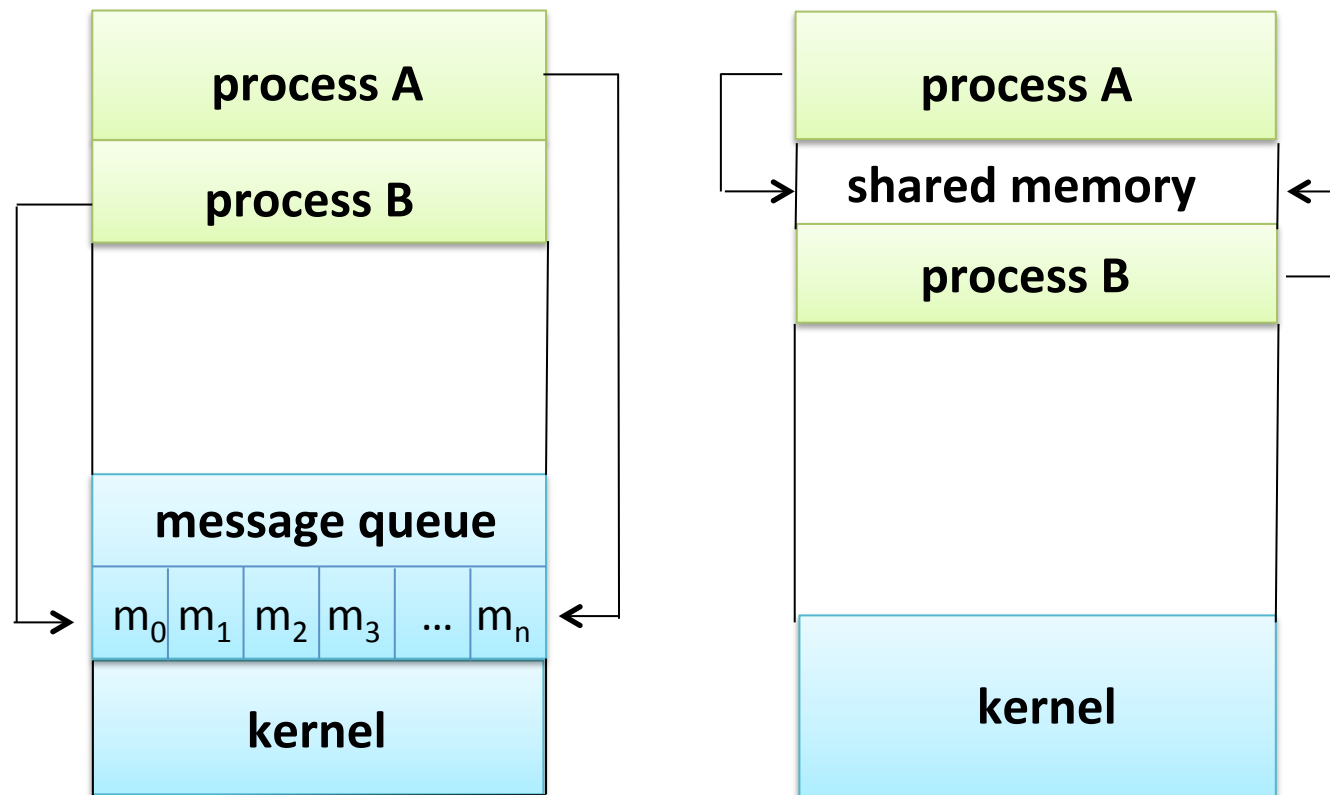
COMP304 - Operating Systems (OS)

Inter-process Communication (IPC)

- *An independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* processes can affect or be affected by the execution of another processes
- Cooperating processes need **inter-process communication**
- Two models of IPC
 - **Shared memory**
 - **Message passing**

Two Models of Communication

Message Passing vs Shared Memory



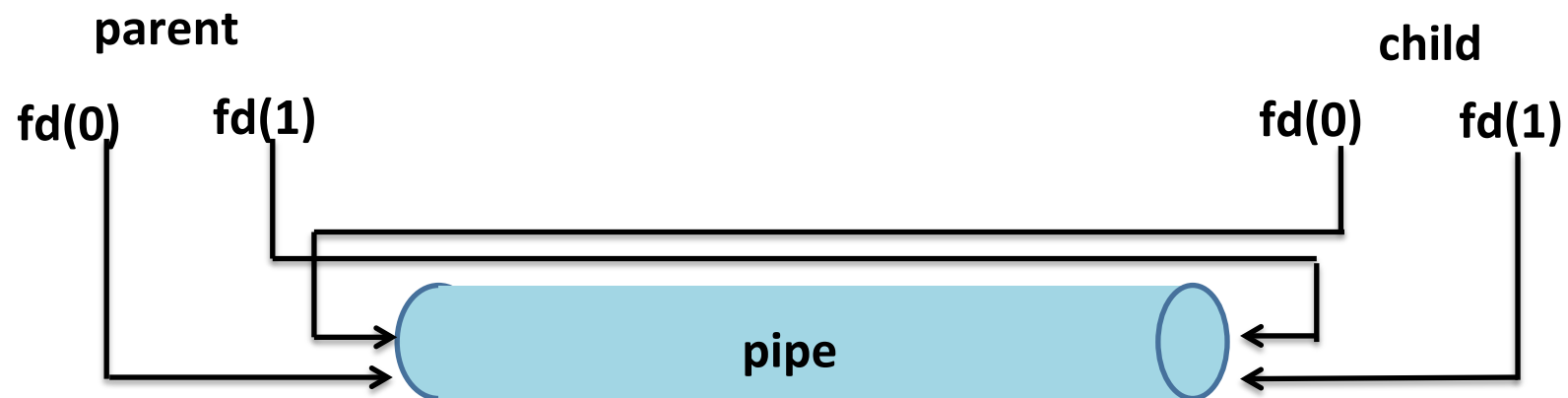
- Message passing requires the message of A to be copied to a buffer and copied to process B's memory – thus it is slower but safer

Pipes

- Acts as a conduit allowing two processes to communicate
 - Ordinary Pipes
 - Named Pipes
- **Issues**
 - Is communication unidirectional or bidirectional?
 - Must there exist a relationship (i.e. *parent-child*) between the communicating processes?
 - Can the pipes be used over a network?

Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore **unidirectional**
- Require parent-child relationship between communicating processes



IPC POSIX Producer

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE 4096;
    /* name of the shared memory object */
    const char *name = "OS";
    /* strings written to shared memory */
    const char *message_0 = "Hello";
    const char *message_1 = "World!";

    /* shared memory file descriptor */
    int shm_fd;
    /* pointer to shared memory object */
    void *ptr;

    /* create the shared memory object */
    shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    /* configure the size of the shared memory object */
    ftruncate(shm_fd, SIZE);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

    /* write to the shared memory object */
    sprintf(ptr,"%s",message_0);
    ptr += strlen(message_0);
    sprintf(ptr,"%s",message_1);
    ptr += strlen(message_1);

    return 0;
}
```

Create a shared memory segment

Memory-mapped file

Writing into the shared memory object

IPC POSIX Consumer

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
    /* the size (in bytes) of shared memory object */
    const int SIZE 4096;
    /* name of the shared memory object */
    const char *name = "OS";
    /* shared memory file descriptor */
    int shm_fd;
    /* pointer to shared memory object */
    void *ptr;

    /* open the shared memory object */
    shm_fd = shm_open(name, O_RDONLY, 0666);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

    /* read from the shared memory object */
    printf("%s", (char *)ptr);

    /* remove the shared memory object */
    shm_unlink(name);

    return 0;
}
```

Create a shared memory segment for readonly

Memory-mapped file for reading

Read from the shared memory object

Scheduling

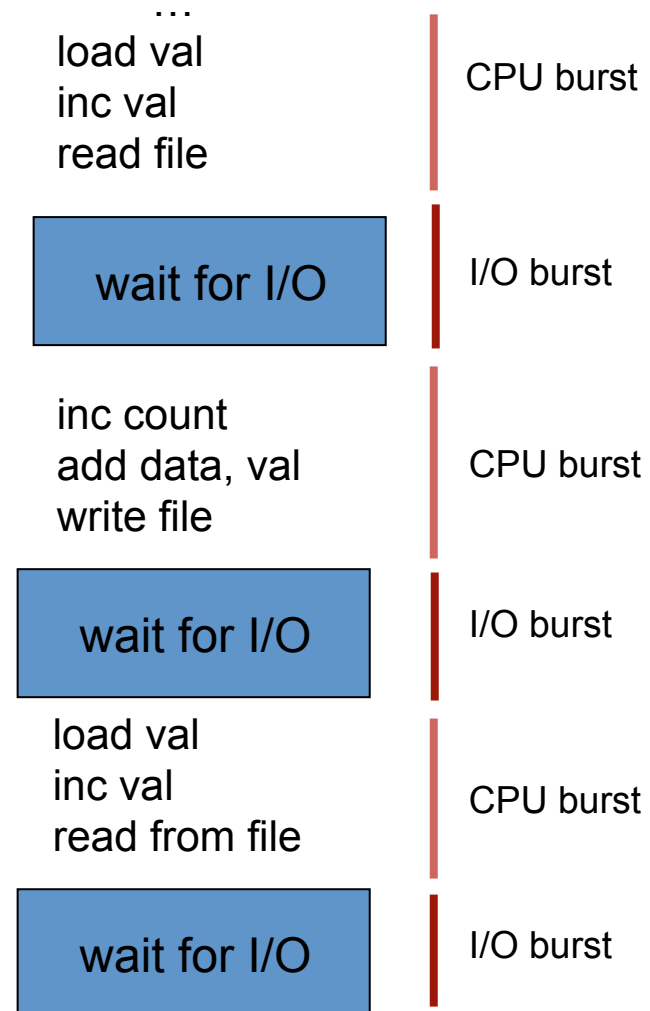
- One of the main tasks of an OS is to schedule processes to execute.
- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.

Schedulers

- Short-term scheduler is invoked very frequently (milliseconds) \Rightarrow (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) \Rightarrow (may be slow).
- The long-term scheduler controls the *degree of multiprogramming*.

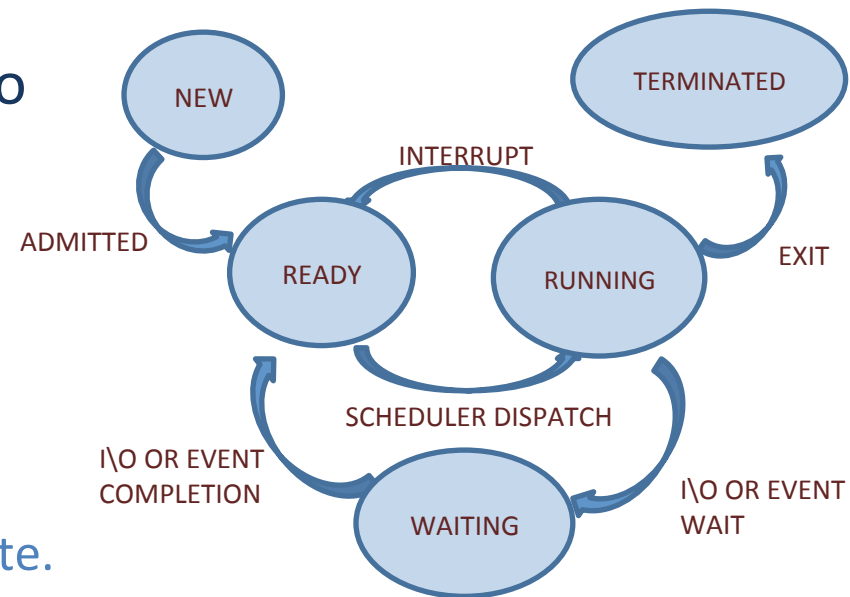
CPU-I/O Burst Cycle

- **CPU-I/O Burst Cycle** – Process execution consists of a *cycle* of CPU execution and I/O wait
- Processes can be described as either:
 - *I/O-bound process* – spends more time doing I/O than computations; many, short CPU bursts.
 - *CPU-bound process* – spends more time doing computations; few, very long CPU bursts.



CPU Scheduler

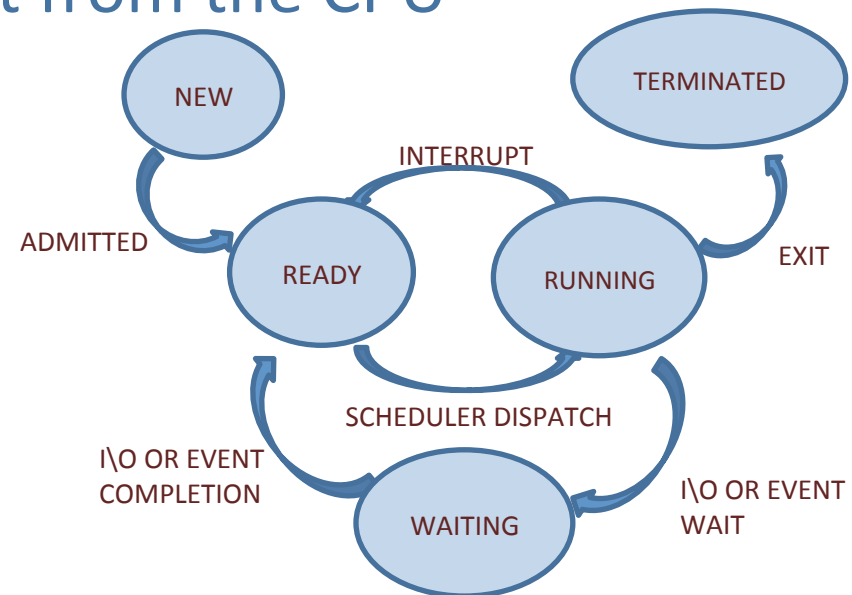
- Selects among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
 1. Switches from running to waiting state.
 2. Switches from running to ready state.
 3. Switches from waiting to ready.
 4. Terminates.



(Non)-preemptive

- Non-preemptive
 - Process voluntarily releases CPU
- Preemptive
 - OS kicks the process out from the CPU

- 1 and 4 non-preemptive
- 2 and 3 are preemptive



Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that completes their execution per time unit
- **Turnaround time** – amount of time to execute a particular process (time between entry and exit)
- **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)

Scheduling

- Let $P = \{p_i \mid 0 \leq i < n\}$ = set of processes
- Let $S(p_i) \in \{\text{running, ready, waiting}\}$
- Let $t(p_i)$ = Time process needs to be in running state (the service time, CPU burst)
- Let $T_{\text{TRnd}}(p_i)$ = Time from p_i first enters ready to last exits system (turnaround time)
- Batch Throughput rate = inverse of avg T_{TRnd}
- Let $R(p_i)$ = Time p_i is in ready state before first transition to running (or response time) (different than “waiting time”)

Optimization Criteria

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

It is important to minimize the variance in response time than minimize the average response time – provides fairness

Dispatcher

- **Dispatcher** module is part of the OS that 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.

Question

_____ is the number of processes that are completed per time unit.

- A) CPU utilization
- B) Response time
- C) Turnaround time
- D) Throughput

Scheduling Algorithms

If you were OS, how would you decide who should run next?

First-Come, First Served (FCFS)

<u>Process</u>	<u>Burst Time</u>
P ₁	24
P ₂	3
P ₃	3

- Suppose that the processes arrive in the order: P₁ , P₂ , P₃
The Gantt Chart for the schedule is:



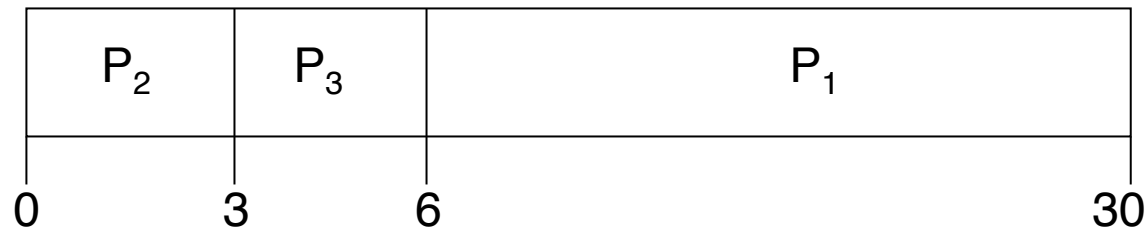
- Waiting times for: P₁ = 0; P₂ = 24; P₃ = 27
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

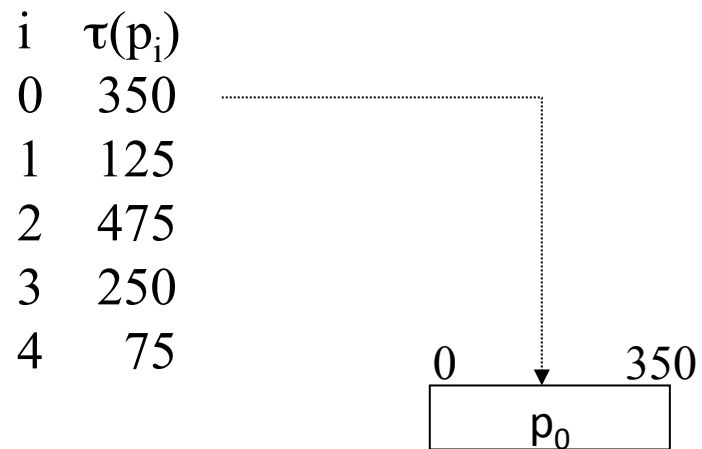
P_2, P_3, P_1

- 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

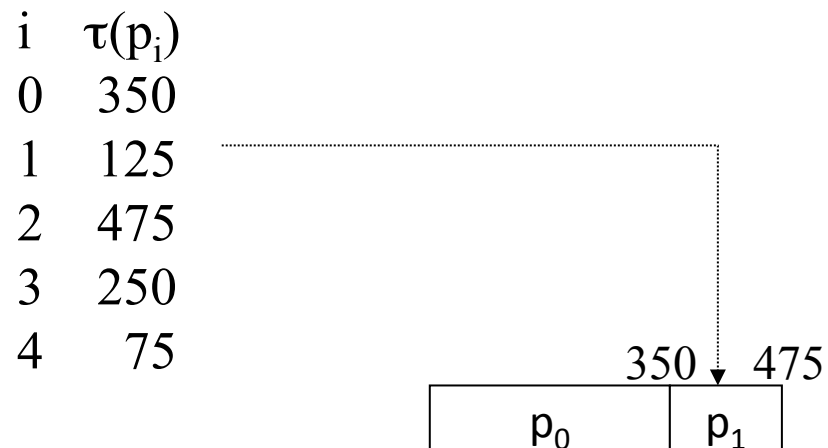
FCFS Scheduling (Cont.)



$$T_{\text{TRnd}}(p_0) = \tau(p_0) = 350$$

$$R(p_0) = 0$$

FCFS Scheduling (Cont.)



$$T_{\text{TRnd}}(p_0) = \tau(p_0) = 350$$

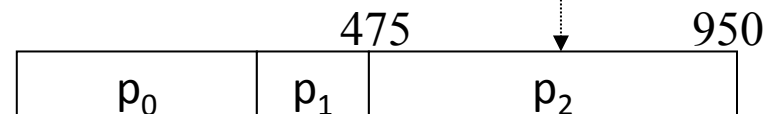
$$T_{\text{TRnd}}(p_1) = (\tau(p_1) + T_{\text{TRnd}}(p_0)) = 125 + 350 = 475$$

$$R(p_0) = 0$$

$$R(p_1) = T_{\text{TRnd}}(p_0) = 350$$

FCFS Scheduling (Cont.)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = \tau(p_0) = 350$$

$$T_{\text{TRnd}}(p_1) = (\tau(p_1) + T_{\text{TRnd}}(p_0)) = 125 + 350 = 475$$

$$T_{\text{TRnd}}(p_2) = (\tau(p_2) + T_{\text{TRnd}}(p_1)) = 475 + 475 = 950$$

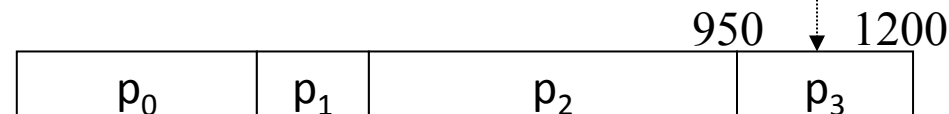
$$R(p_0) = 0$$

$$R(p_1) = T_{\text{TRnd}}(p_0) = 350$$

$$R(p_2) = T_{\text{TRnd}}(p_1) = 475$$

FCFS Scheduling (Cont.)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = \tau(p_0) = 350$$

$$T_{\text{TRnd}}(p_1) = (\tau(p_1) + T_{\text{TRnd}}(p_0)) = 125 + 350 = 475$$

$$T_{\text{TRnd}}(p_2) = (\tau(p_2) + T_{\text{TRnd}}(p_1)) = 475 + 475 = 950$$

$$T_{\text{TRnd}}(p_3) = (\tau(p_3) + T_{\text{TRnd}}(p_2)) = 250 + 950 = 1200$$

$$R(p_0) = 0$$

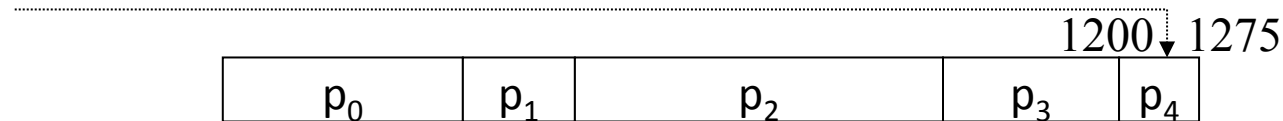
$$R(p_1) = T_{\text{TRnd}}(p_0) = 350$$

$$R(p_2) = T_{\text{TRnd}}(p_1) = 475$$

$$R(p_3) = T_{\text{TRnd}}(p_2) = 950$$

FCFS Scheduling (Cont.)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = \tau(p_0) = 350$$

$$T_{\text{TRnd}}(p_1) = (\tau(p_1) + T_{\text{TRnd}}(p_0)) = 125 + 350 = 475$$

$$T_{\text{TRnd}}(p_2) = (\tau(p_2) + T_{\text{TRnd}}(p_1)) = 475 + 475 = 950$$

$$T_{\text{TRnd}}(p_3) = (\tau(p_3) + T_{\text{TRnd}}(p_2)) = 250 + 950 = 1200$$

$$T_{\text{TRnd}}(p_4) = (\tau(p_4) + T_{\text{TRnd}}(p_3)) = 75 + 1200 = 1275$$

$$R(p_0) = 0$$

$$R(p_1) = T_{\text{TRnd}}(p_0) = 350$$

$$R(p_2) = T_{\text{TRnd}}(p_1) = 475$$

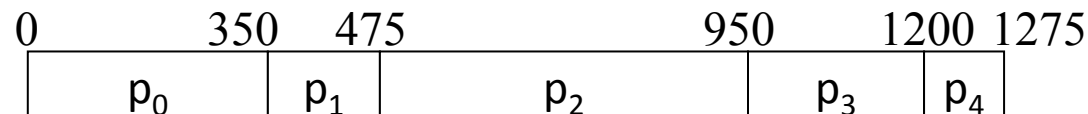
$$R(p_3) = T_{\text{TRnd}}(p_2) = 950$$

$$R(p_4) = T_{\text{TRnd}}(p_3) = 1200$$

FCFS Scheduling- Average Wait Time

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- Easy to implement
- Not a great performer
- Non-preemptive



$$T_{\text{TRnd}}(p_0) = \tau(p_0) = 350$$

$$R(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) = (\tau(p_1) + T_{\text{TRnd}}(p_0)) = 125 + 350 = 475$$

$$R(p_1) = T_{\text{TRnd}}(p_0) = 350$$

$$T_{\text{TRnd}}(p_2) = (\tau(p_2) + T_{\text{TRnd}}(p_1)) = 475 + 475 = 950$$

$$R(p_2) = T_{\text{TRnd}}(p_1) = 475$$

$$T_{\text{TRnd}}(p_3) = (\tau(p_3) + T_{\text{TRnd}}(p_2)) = 250 + 950 = 1200$$

$$R(p_3) = T_{\text{TRnd}}(p_2) = 950$$

$$T_{\text{TRnd}}(p_4) = (\tau(p_4) + T_{\text{TRnd}}(p_3)) = 75 + 1200 = 1275$$

$$R(p_4) = T_{\text{TRnd}}(p_3) = 1200$$

$$\text{Average response (wait) time } R_{\text{avg}} = (0 + 350 + 475 + 950 + 1200) / 5 = 2974 / 5 = 595$$

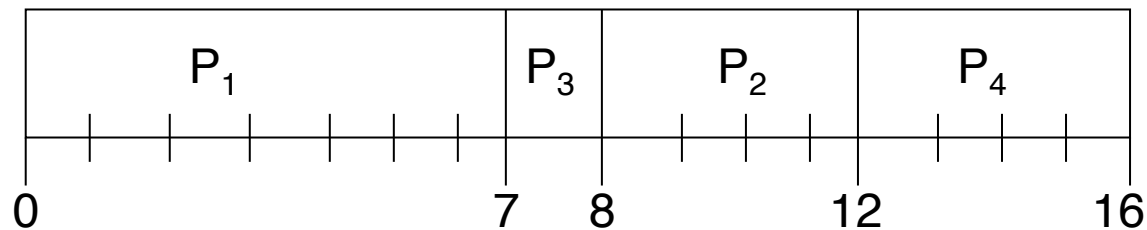
2. Shortest Job First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
 - **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 less than remaining time of current executing process, preempt. This is known as the **Shortest-Remaining-Time-First (SRTF)**.
- SJF is **optimal** – gives minimum average waiting time for a given set of processes.

Non-preemptive (SJF) Scheduling

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P ₁	0.0	7
P ₂	2.0	4
P ₃	4.0	1
P ₄	5.0	4

- SJF (non-preemptive)

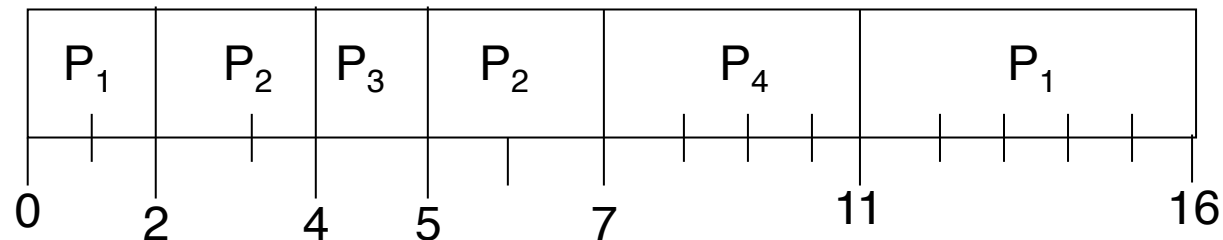


- Average waiting time ?
= $(0 + 6 + 3 + 7)/4 = 4$

Example of Preemptive SJF

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P ₁	0.0	7
P ₂	2.0	4
P ₃	4.0	1
P ₄	5.0	4

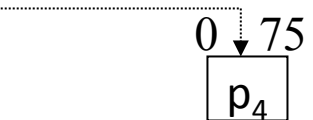
- SJF (preemptive)



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

Example of SJF

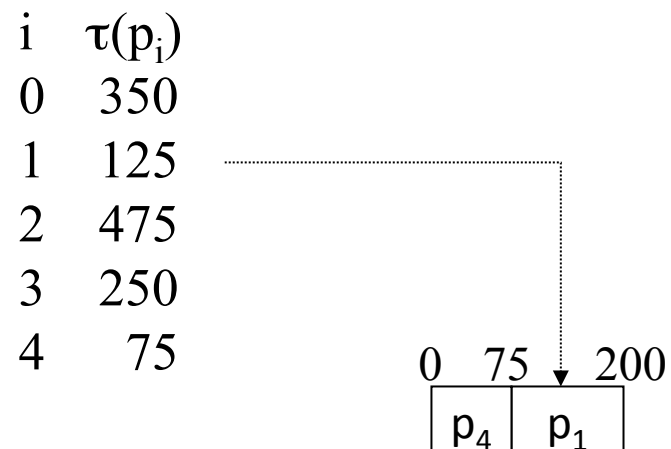
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_4) = \tau(p_4) = 75$$

$$R(p_4) = 0$$

Example of SJF



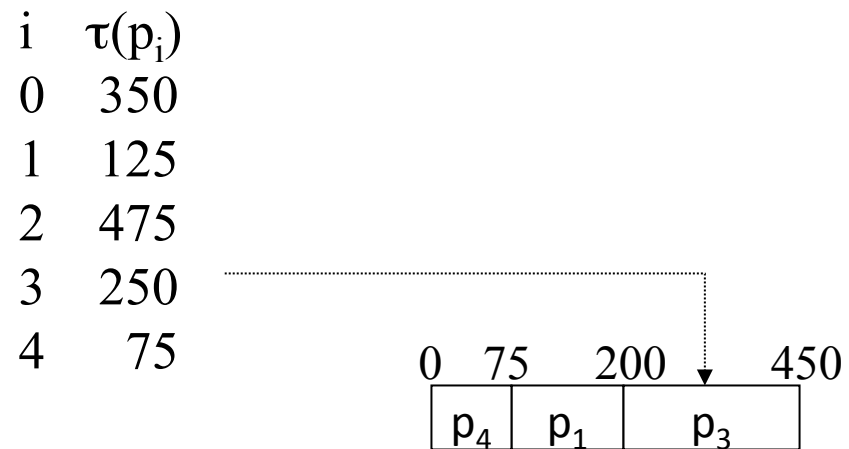
$$T_{\text{TRnd}}(p_1) = \tau(p_1) + \tau(p_4) = 125 + 75 = 200$$

$$R(p_1) = 75$$

$$T_{\text{TRnd}}(p_4) = \tau(p_4) = 75$$

$$R(p_4) = 0$$

Example of SJF



$$T_{\text{TRnd}}(p_1) = \tau(p_1) + \tau(p_4) = 125 + 75 = 200$$

$$R(p_1) = 75$$

$$T_{\text{TRnd}}(p_3) = \tau(p_3) + \tau(p_1) + \tau(p_4) = 250 + 125 + 75 = 450$$

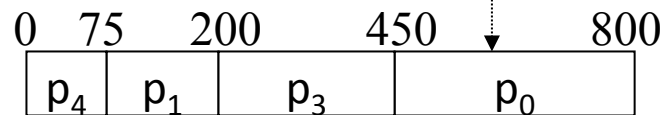
$$R(p_3) = 200$$

$$T_{\text{TRnd}}(p_4) = \tau(p_4) = 75$$

$$R(p_4) = 0$$

Example of SJF

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = \tau(p_0) + \tau(p_3) + \tau(p_1) + \tau(p_4) = 350 + 250 + 125 + 75 = 800$$

$$R(p_0) = 450$$

$$T_{\text{TRnd}}(p_1) = \tau(p_1) + \tau(p_4) = 125 + 75 = 200$$

$$R(p_1) = 75$$

$$T_{\text{TRnd}}(p_3) = \tau(p_3) + \tau(p_1) + \tau(p_4) = 250 + 125 + 75 = 450$$

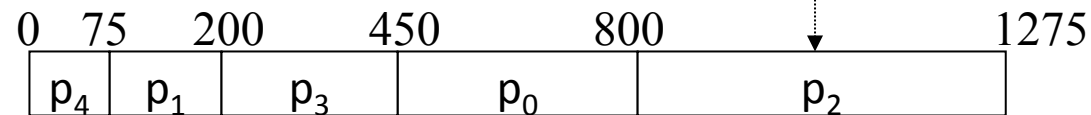
$$R(p_3) = 200$$

$$T_{\text{TRnd}}(p_4) = \tau(p_4) = 75$$

$$R(p_4) = 0$$

Example of SJF

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = \tau(p_0) + \tau(p_3) + \tau(p_1) + \tau(p_4) = 350 + 250 + 125 + 75 = 800$$

$$R(p_0) = 450$$

$$T_{\text{TRnd}}(p_1) = \tau(p_1) + \tau(p_4) = 125 + 75 = 200$$

$$R(p_1) = 75$$

$$T_{\text{TRnd}}(p_2) = \tau(p_2) + \tau(p_0) + \tau(p_3) + \tau(p_1) + \tau(p_4) = 475 + 350 + 250 + 125 + 75 = 1275$$

$$R(p_2) = 800$$

$$T_{\text{TRnd}}(p_3) = \tau(p_3) + \tau(p_1) + \tau(p_4) = 250 + 125 + 75 = 450$$

$$R(p_3) = 200$$

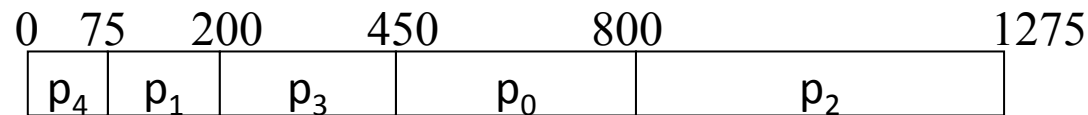
$$T_{\text{TRnd}}(p_4) = \tau(p_4) = 75$$

$$R(p_4) = 0$$

Example of SJF

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- Minimizes waiting time – Why?
- May starve large jobs



$$T_{\text{TRnd}}(p_0) = \tau(p_0) + \tau(p_3) + \tau(p_1) + \tau(p_4) = 350 + 250 + 125 + 75 = 800$$

$$R(p_0) = 450$$

$$T_{\text{TRnd}}(p_1) = \tau(p_1) + \tau(p_4) = 125 + 75 = 200$$

$$R(p_1) = 75$$

$$T_{\text{TRnd}}(p_2) = \tau(p_2) + \tau(p_0) + \tau(p_3) + \tau(p_1) + \tau(p_4) = 475 + 350 + 250 + 125 + 75 = 1275$$

$$R(p_2) = 800$$

$$T_{\text{TRnd}}(p_3) = \tau(p_3) + \tau(p_1) + \tau(p_4) = 250 + 125 + 75 = 450$$

$$R(p_3) = 200$$

$$T_{\text{TRnd}}(p_4) = \tau(p_4) = 75$$

$$R(p_4) = 0$$

$$\text{Average response (wait) time } R_{\text{avg}} = (450 + 75 + 800 + 200 + 0) / 5 = 1525 / 5 = 305$$

Shortest Job First

- The SJF is provably optimal
- It gives the minimum average waiting time for a given set of processes
- Moving a short process before a long one decreases the waiting time of the short process more than it increases the waiting time of the long process
- What is the difficulty of using the shortest job first scheduler?

Determining the Length of the Next CPU Burst

- Can only **estimate the length**.
- Can be done by using the length of previous CPU bursts, using **exponential averaging**.

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 :

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

Examples of Exponential Averaging

- $\alpha = 0$

- $\tau_{n+1} = \tau_n$

- Recent information does not count.

- $\alpha = 1$

- $\tau_{n+1} = t_n$

- Only the actual last CPU burst counts.

- If we expand the formula, we get:

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

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

Question

- The strategy of making processes that are logically runnable to be temporarily suspended is called :
 - a) Non preemptive scheduling
 - b) Preemptive scheduling
 - c) Shortest job first
 - d) First come First served

Answer is b)

Puzzle

- A group of people wants to get through a tunnel.
 - A can make it in 1 minute,
 - B can in 2 minutes,
 - C can in 4 and
 - D can in 5 minutes.
- Unfortunately, not more than two persons can go through the narrow tunnel at one time, moving at the speed of the slower one.
- They have a single torch to show their way in dark
- What is the minimum time for all to make it to the other side of the tunnel?

Reading

- Read Chapter 5
- Read Chapter 4 (Linux Kernel Development)
- Acknowledgments
 - These slides are adapted from
 - Öznur Özkasap (Koç University)
 - Operating System and Concepts (9th edition) Wiley