CS343: Operating System

Process Management and Scheduling

Lect10 : 18th Aug 2023

Dr. A. Sahu

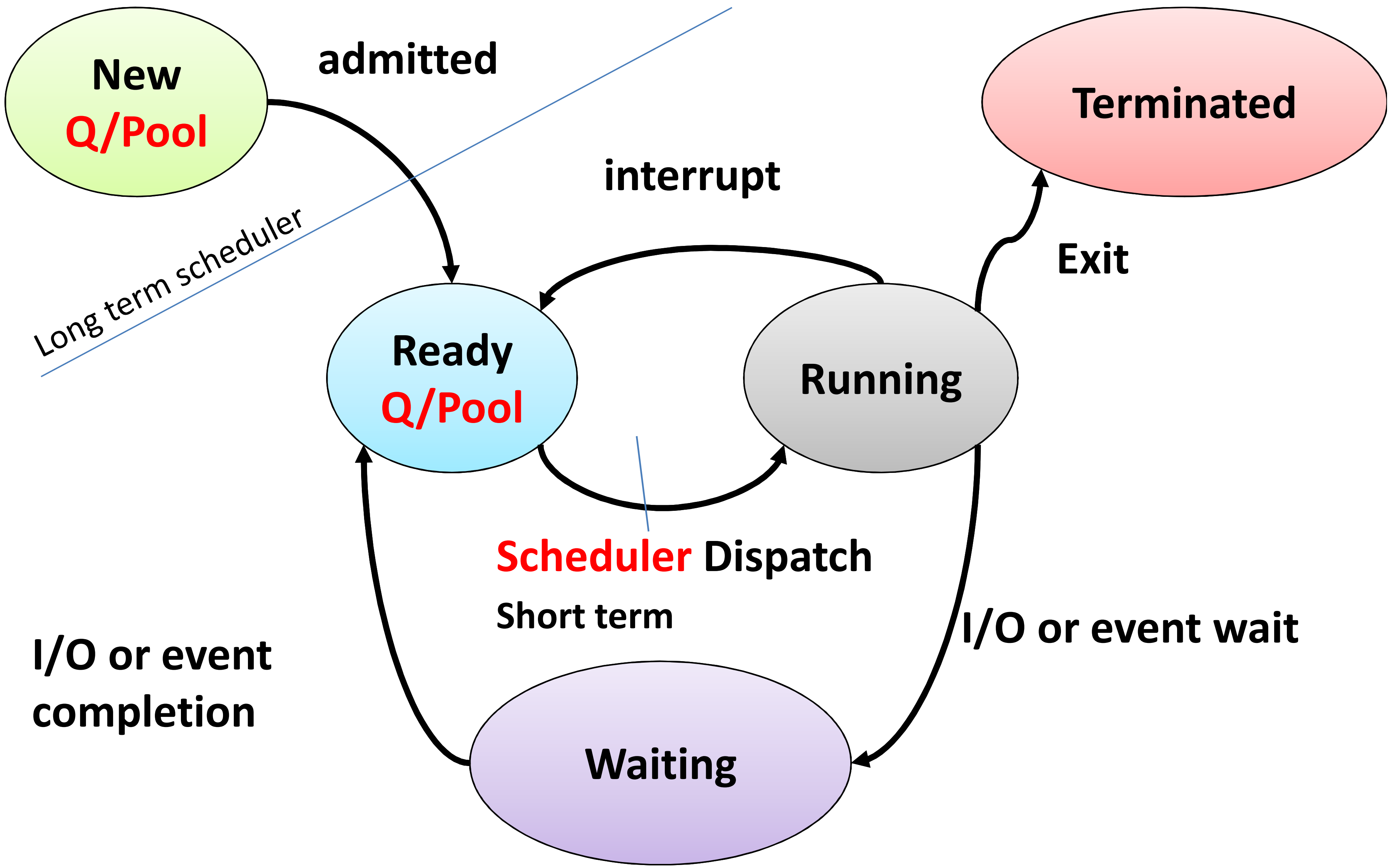
Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Memory Layout of C Program
- Process Concepts
- Process States, Process Control Block
- Context Switch
- IPC (Inter Process Communication)
- Threads ()
- Scheduling: **Theoretical Analysis**

Process State: State Diagram

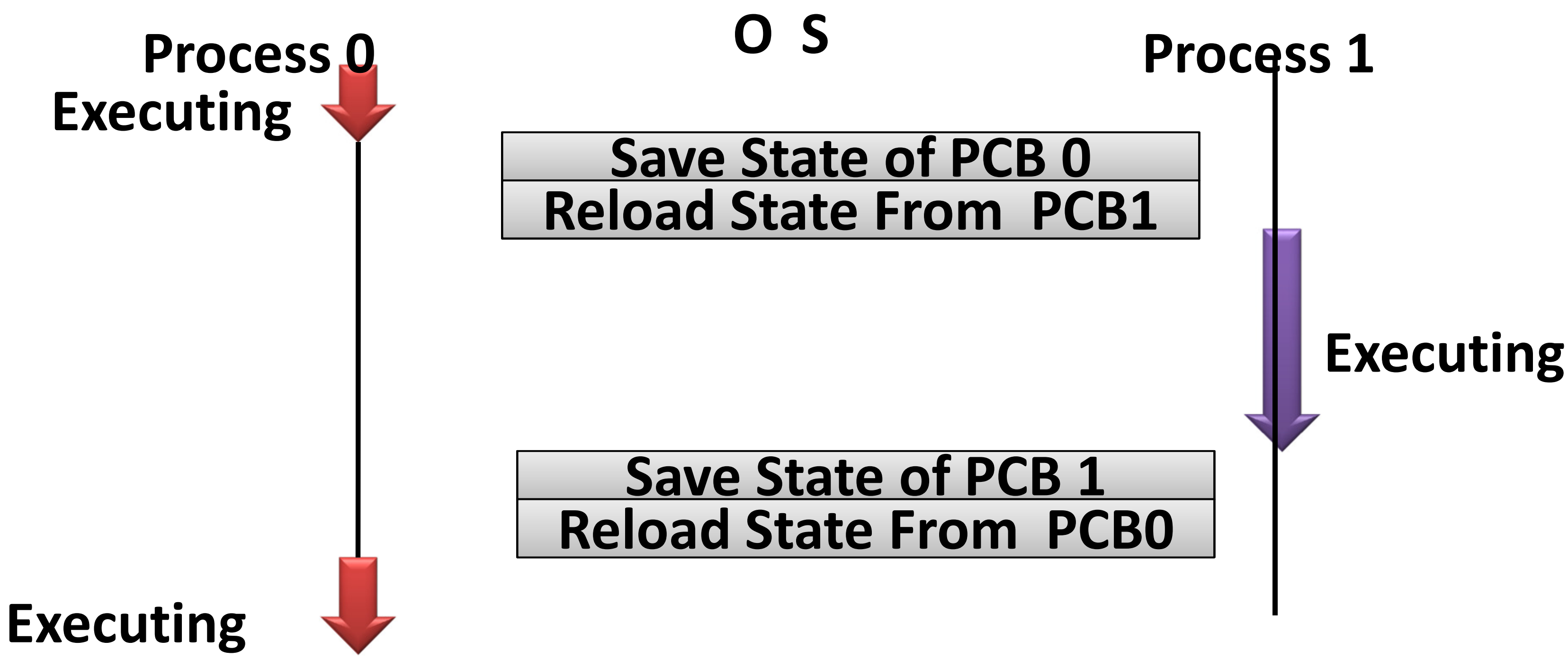


Process Control Block (PCB)

Process State
Process Number
Program Counter
Registers
Memory Limits
List of Open Files
……

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 via a **context switch**
- **Context** of a process represented in the PCB



Context Switch

- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Many time Context switch code written Manually.
Compiler Generated code may not be efficient

Operations on Processes

- One should know how to create/delete process
- System must provide mechanisms for:
  - process creation
  - process termination

Process Creation

- **Parent** process create **children** processes
  – which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)

ps : report a snapshot of current process

```
[asahu@asahu ~]$ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:02 systemd
    2 ?        00:00:01 kthreadd
    3 ?        00:00:22 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:00 kworker/u:0H
    8 ?        00:00:11 migration/0
    9 ?        00:00:07 watchdog/0
   10 ?        00:00:07 migration/1
   12 ?        00:00:00 kworker/1:0H
   13 ?        00:00:20 ksoftirqd/1
   14 ?        00:00:17 watchdog/1
   15 ?        00:00:12 migration/2
   17 ?        00:00:00 kworker/2:0H
   18 ?        00:00:22 ksoftirqd/2
   19 ?        00:00:12 watchdog/2
```

pstree

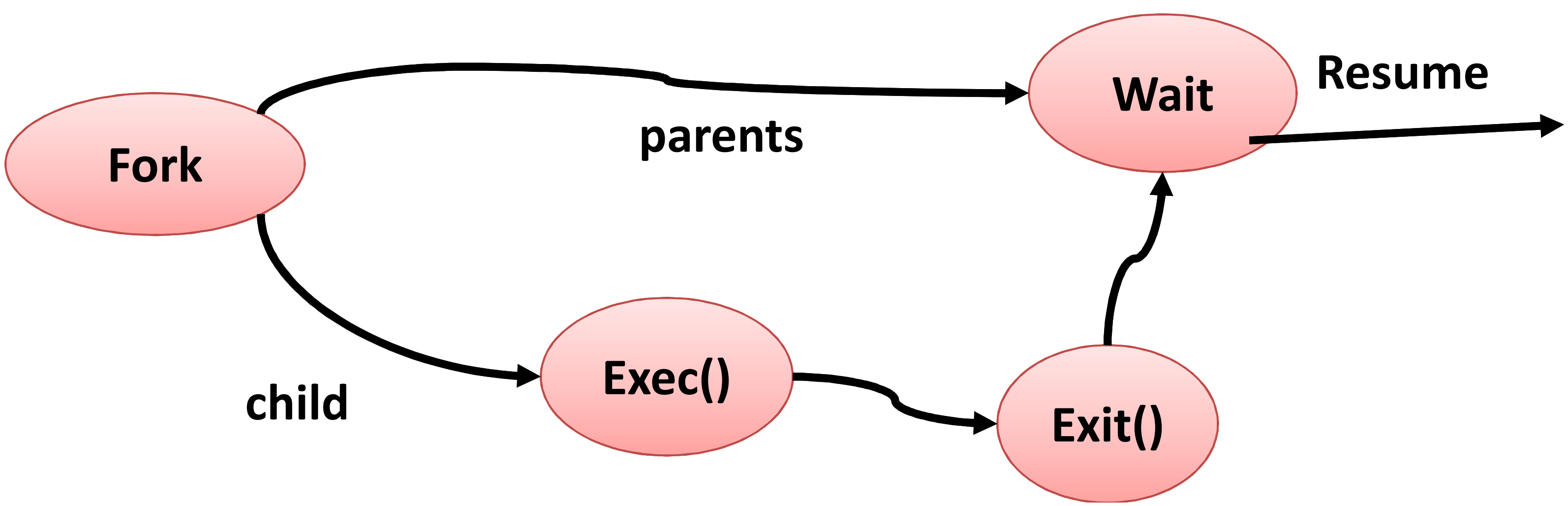
```
[asahu@asahu ~]$ pstree
systemd─┬─NetworkManager───{NetworkManager}
        ├─VBoxSVC─┬─VirtualBox───18*[{Virtual
        │         └─11*[{VBoxSVC}]
        ├─VBoxXPCOMIPCD
        ├─abrt-dump-oops
        ├─abrtd
        ├─accounts-daemon───{accounts-daemo}
        ├─acpid
        ├─at-spi-bus-laun───2*[{at-spi-bus-la
        ├─atd
        ├─auditd─┬─audispd─┬─sedispatch
        │        │         └─{audispd}
        │        └─{auditd}
        ├─avahi-daemon───avahi-daemon
        ├─colord───{colord}
        ├─console-kit-dae───63*[{console-kit-
```

Process Creation

- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - `fork()` system call creates new process
  - `exec()` system call used after a `fork()` to replace the process' memory space with a new program



C Program Forking Separate Process

```
int main (){   //code uploaded to course website
        pid_t pid;
        pid=fork();
        if(pid<0){
                printf("fork failed");
                return 1;
        }
        else if (pid==0){ execlp("/bin/ls","ls",NULL);}
        else {
                wait(NULL);
                printf("Child complete\n\n");
        }
        return 0;
}
```

How to terminate a Process

- ☺ ☺ ☺ ☺
  - **Click the Cross symbol located on right side of the application window**
- In linux command mode
  - **Send termination signal to process**

$kill <pid>

Process Termination

- Process executes last statement and then asks the OS to delete it using the **`exit()`** system call.
  - Returns status data from child to parent (via **`wait()`**)
  - Process' resources are deallocated by operating system

Process Termination

- Parent may terminate the execution of children processes using the **`abort()`** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Process Scheduling

Scheduling Evaluation Metrics

- CPU utilization
  - –Percentage of time the CPU is not idle
- Throughput
  - –Completed processes per time unit
- Turnaround time
  - –Submission to completion

Scheduling Evaluation Metrics

- Waiting time
  - time spent on the ready queue
- Response time
  - response latency
  - “response time” most important for interactive jobs (I/O bound)
- Predictability
  - variance in any of these measures

The right evaluation metric depends on the context

Scheduling Algorithm Optimization Criteria

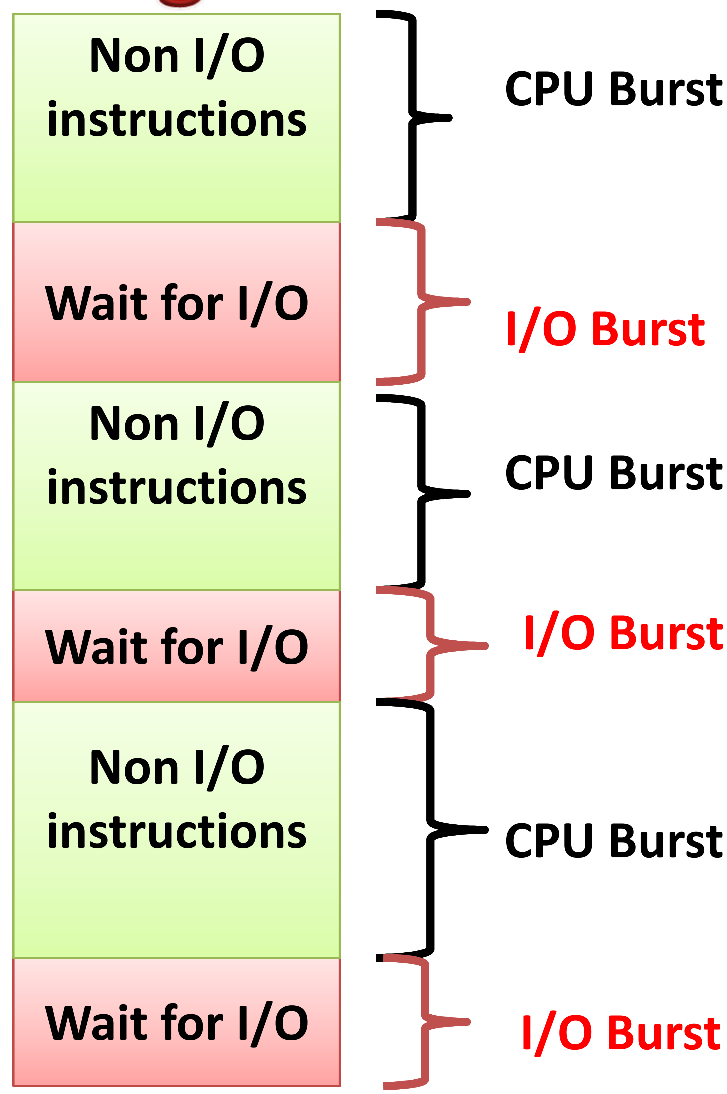
- Maximize
  - CPU utilization
  - Throughput
- Minimize
  - Turnaround time
  - Waiting time
  - Response time

“The perfect CPU scheduler”

- Minimize latency
  - Response or job completion time
- Maximize throughput
  - Maximize jobs / time.
- Maximize utilization: keep I/O devices busy.
  - Recurring theme with OS scheduling
- Fairness: everyone makes progress, no one starves

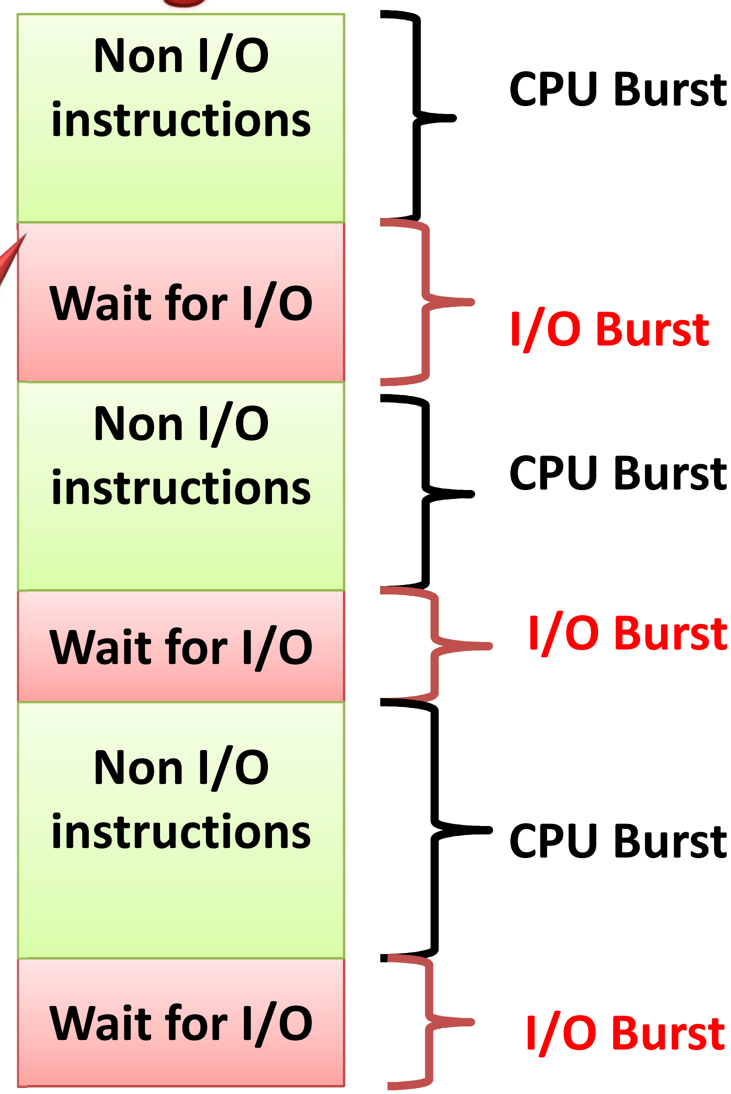
Max CPU Util. obtained with Multiprogramming

- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**



Max CPU Util. obtained with Multiprogramming

- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**



<u>To Maximize CPU Utilization</u>
When process request form I/O
CPU switches to other process

Assumption

- Task/Process/Job used interchangbly
- Let all tasks in memory
- Let all tasks don’t do any IPC
- All task are independent
- All tasks don’t require any other resource other then CPU

Five Popular Scheduling

1. First Come First Serve
2. Shortest Job First
   - Shortest Remaining Time First
   - SJF-I
3. Round-Robin Scheduler
4. Priority Scheduler
   - Priority-I
5. Multi-Level Priority Queue
   - Feed Back Priority Queue

Flow Time of a Job : Turn Around Time

- $F_i = C_i - A_i$
  - C_i= completion time, A_i=Arrival time
  - Flow time depended on both C_i and A_i, If A_i=0, depends on C_i
  - In uniprocessor: F_i=W_i the waiting time + CC_i Compute time

Flow Time of a Job : Turn Around Time

- $F_i = C_i - A_i$
  - C_i= completion time, A_i=Arrival time
  - Flow time depended on both C_i and A_i, If A_i=0, depends on C_i
  - In uniprocessor: F_i=W_i the waiting time + CC_i Compute time
- Average Flow time = $F_i' = \sum F_i$
  - In uniprocessor : Minimize Agv Flow Time = Mimimize Avg Waiting time
  - Min(F_i') =Min($\sum W_i$) as CPU is completely busy for all the time
- Minimize Weighted Flow time: Priority Version

FCFS Scheduling

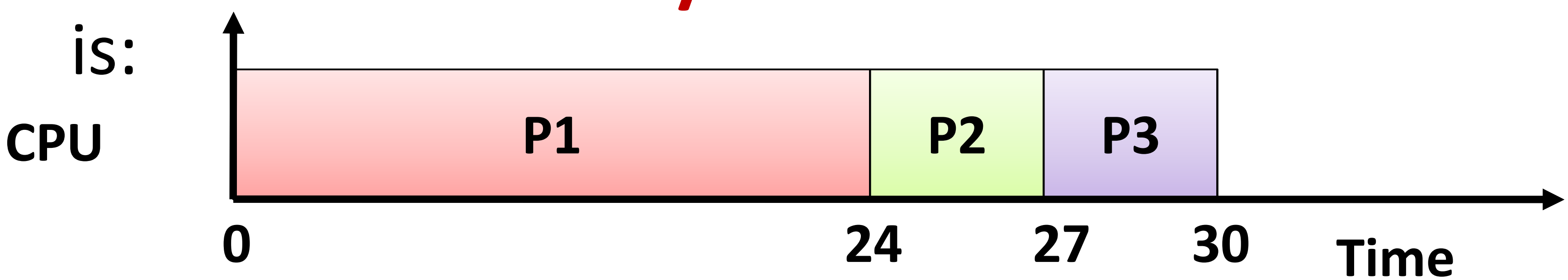
Process	Burst Time/ Execution Time
P1	24
P2	3
P3	3

- Assume: processes arrive in the order: P_1 , P_2 , P_3

FCFS Scheduling

Process	Burst Time/ Execution Time
P1	24
P2	3
P3	3

- Assume: processes arrive in the order: P_1 , P_2 , P_3
  The **Gantt's Chart/Gantt Chart** for schedule is:



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

FCFS

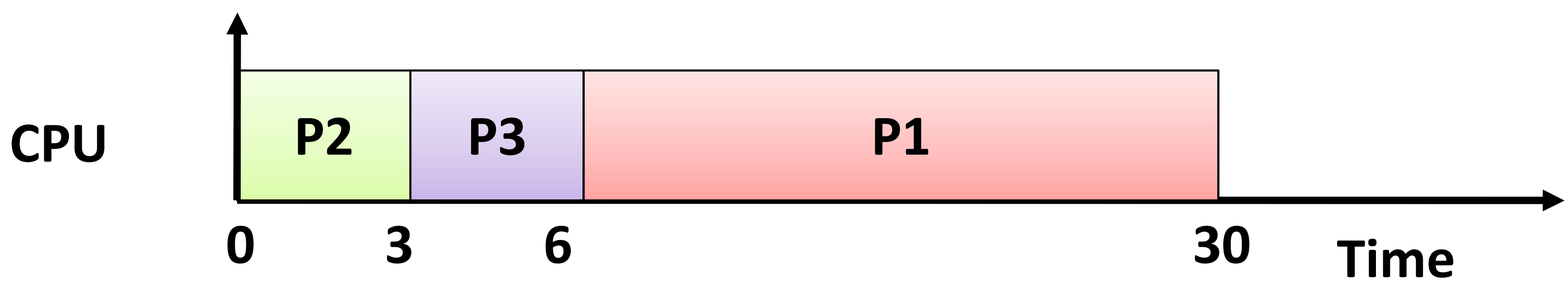
Convoy effect

short process behind long process

FCFS Scheduling

Process	Burst Time/Execution Time
P1	24
P2	3
P3	3

- Assume: 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: $(6 + 0 + 3)/3 = 3$
- AWT much better than previous case : $3 < 17$

Problem: Cook of Restaurant

- You work as a short-order cook
  - Customers come in and specify which dish they want
  - Each dish takes a different amount of time to prepare
- Your goal:
  - Minimize average time the customers wait for their food
- What strategy would you use ?
  - Note: most restaurants use FCFS.

Shortest-Job-First (SJF) Scheduling

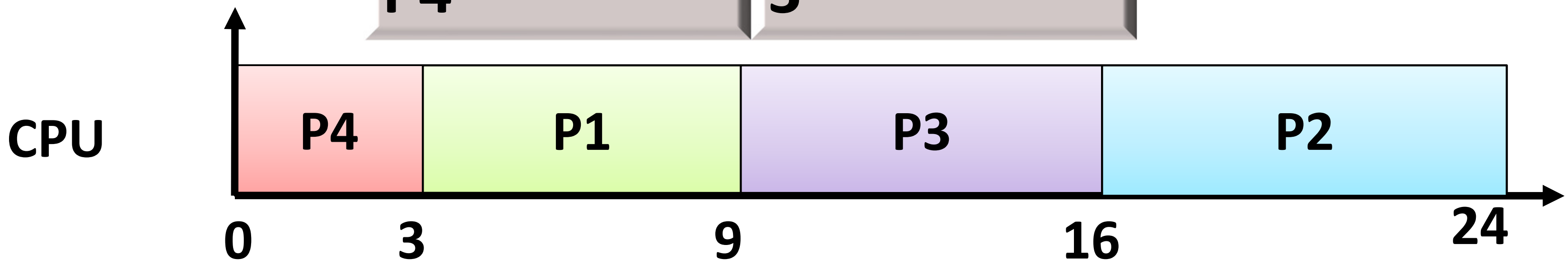
- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time

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
- **SJF is optimal** – gives minimum average waiting time for a given set of processes
  - **The difficulty is knowing the length of the next CPU request**
  - Could ask the user

Example of SJF

Process	Burst Time
P1	6
P2	8
P3	7
P4	3



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

SJF-Preemptive: Shortest-remaining-time-first

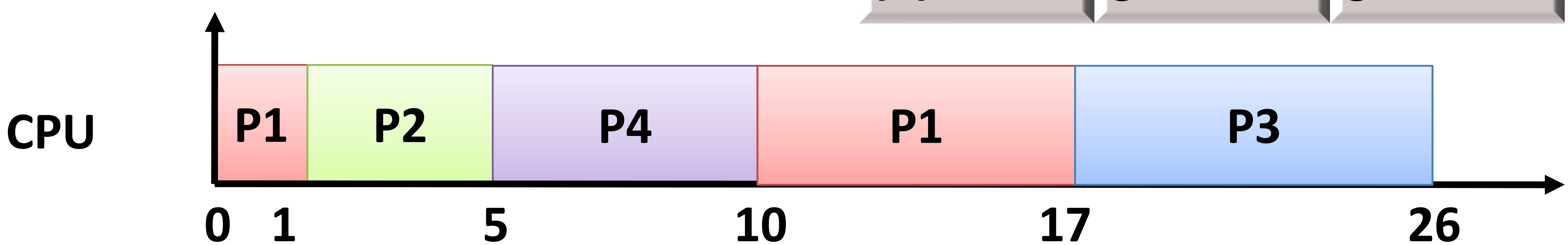
- Now we add the concepts of varying arrival times and preemption to the analysis
- **Dynamic Decision @Runtime**

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

SJF-Preemptive: Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis
- **Dynamic Decision @Runtime**

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5



- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

Thanks