Unit-II
Scheduling

Unit-II Scheduling

- Scheduling Workload Assumptions, Scheduling Metrics, First In, First Out (FIFO), Shortest Job First (SJF), Shortest Time-to-Completion First (STCF),
- A New Metric: Response Time, Round Robin, Incorporating I/O,
- The Multi-Level Feedback Queue, The Priority Boost, Attempt, Better Accounting, Multiprocessor Scheduling, Synchronization, Cache Affinity,
- Single-Queue Scheduling, Multi-Queue Scheduling, Linux Multiprocessor Schedulers

CPU Scheduling

- A process execution consists of a cycle of CPU execution and I/O execution.
- Normally every process begins with CPU burst that may be followed by I/O burst, then another CPU burst and then I/O burst and so an eventually in the last will end up with CPU burst.
- CPU bound processes: There are those processes which require most of time on CPU.
- I/O bound processes: There are those processes which require most of time on I/O devices.
- The system has to run both the types of processes.

Non-pre-emptive and Pre-emptive Processes

- Two ideas of CPU scheduling: Non-Preemptive and preemptive

- Non-Preemptive
  - When a process completes its execution, then next process can run.
  - When a process leaves CPU voluntarily to perform some I/O operation or to wait for an event.

- Preemptive:
  - If a process enters in the ready state either from new or waiting state and it is a high priority process.
  - If a process switches from running state to ready state because time quantum expired.

Scheduling Policy

- Scheduling Policy: It is the piece of code which decides which process to run next.
- On context switch, which process to run next from set of ready processes?
- On scheduler schedules the CPU requests (bursts) of processes
  - CPU burst=the CPU time used by a process in a continuous stretch
  - If a process comes back after I/O wait, it counts as a fresh CPU burst.

CPU Scheduling Terminology

- Burst Time/Running Time/Execution Time: It is the time required for process to running on CPU.

- Waiting Time: Time spend by a process in ready state waiting for CPU.

- Arrival Time: when process enters in ready state.

- Exit time: when process completes its execution and exit from system.

- Turn Around Time: Total time spend by a process in a system

  =Burst Time + Waiting time

  OR

  Exit time-Arrival Time

- Response Time: Time between a process enters ready queue and get scheduled on the CPU for the first time

Criteria for CPU Scheduling Algorithm

- We can not change process arrival time and process burst time

- We can change process wait time by using different scheduling policy.

  - Average Waiting Time
  - Average Response Time
  - CPU Utilization
  - Throughput: Number of processes executed in per unit time.

What are we trying to optimize?

- Scheduler will maximize the utilization of CPU. Maximize (Utilization= fraction of time CPU is used)

- Minimize average (turnaround time= time from process arrival to completion)
    - Turnaround time: Time Completion – time arrival

- Minimize average (response time= time from process arrival to first scheduling)

- Fairness: all processes must be treated equally

Which Algorithms are used in Linux OS?

Algorithms

- First Come First Serve (Always Non-Preemptive)
- Shortest Job First
  - Non-Preemptive
  - Preemptive
- Round Robin (Always Preemptive)

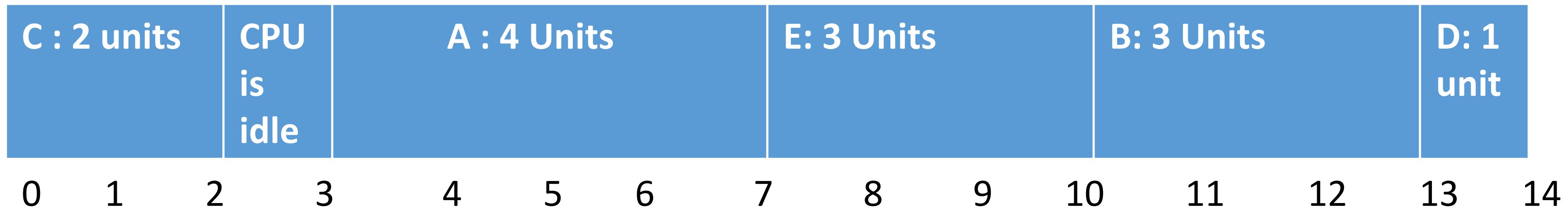
First Come First Serve (FCFS)

- Also called as First In First Out.
- Simple Scheduling Algorithm. It assigns CPU to the process which arrives first.
- Easy to understand and can easily be implemented using queue data structures.
- Non-preemptive algorithm

FCFS: Example

Process ID	Arrival Time	Burst Time
A	3	4
B	5	3
C	0	2
D	5	1
E	4	3

Process ID	Turnaround Time (Exit time - Arrival Time)	Waiting Time=(Turnaround Time-Burst Time)
A	7-3=4	4-4=0
B	13-5=8	8-3=5
C	2-0=2	2-2=0
D	14-5=9	9-1=8
E	10-4=6	6-3=3

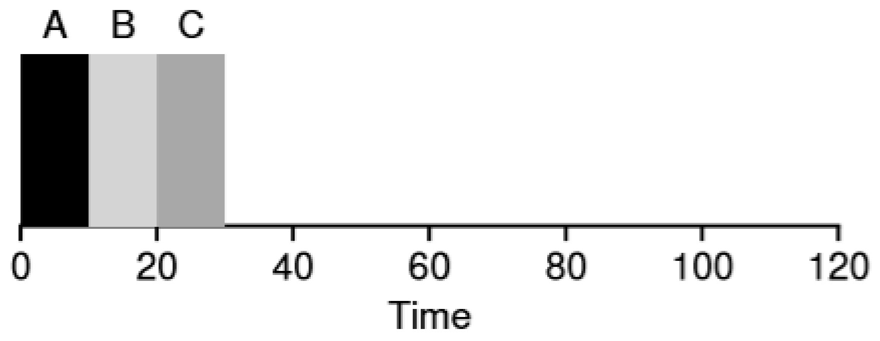


- Some more examples of FCFS…..

FCFS: Example:1

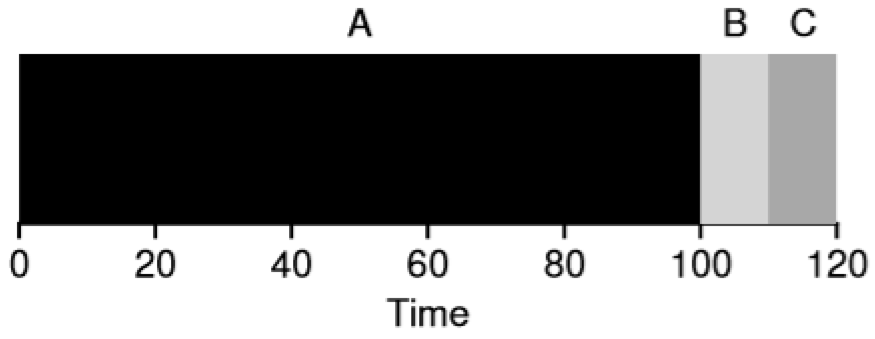
- Example: Three processes arrive at t=0 in the order A,B, C
- A finished at 10, B at 20, and C at 30.
- Thus, the average turn around time for the three jobs is

(10+20+30)/ 3 = 20



FCFS: Example:2

- Example: Three processes arrive at t=0 in the order A,B, C
- Again assume three jobs (A, B, and C), but this time A runs for 100 seconds while B and C run for 10 each
- The average turnaround time for the system is high:110 seconds (100+110+120)/ 3 = 110. This problem is generally referred to as the convoy effect.

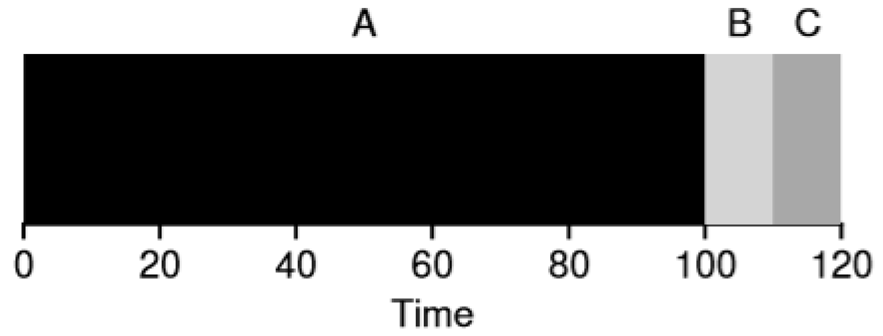


What is Convoy Effect?

- Convoy Effect is a situation where many processes, who need to use a resource for short time are blocked by one process holding that resource for a long time.
- This essentially leads to poor utilization of resources and hence poor performance.

Why FIFO is not that great

- Problem: Convoy effect
- Turnaround times tend to be high



FCFS : Advantages and Disadvantages

- Advantages:
  - Simple and easy to understand
  - Easy to implement
  - Must be used for background processes where execution is not urgent

- Disadvantages:
  - Suffer from convoy effect
  - Higher average waiting time
  - Should not be used for interactive system

Starvation in FCFS

- No, there is no starvation in FCFS.
- Processor is not biased

Shortest Job First (SJF)

- Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

- This is the best approach to minimize waiting time.

- It is of two types:
  - Non Pre-emptive (Shortest Job First)
  - Pre-emptive (Shortest Remaining Time First)

- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.

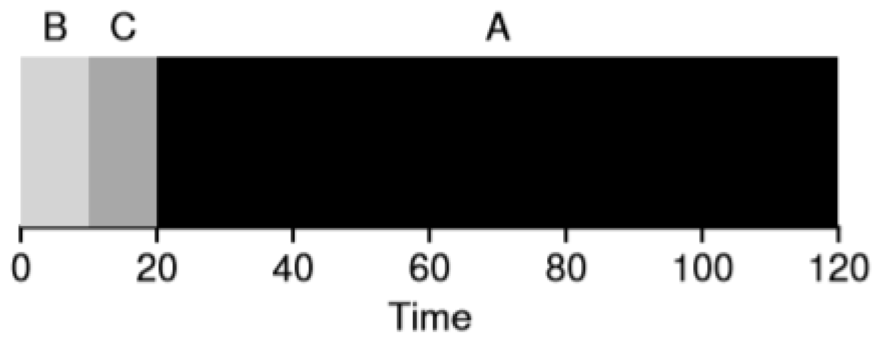
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

Shortest Job First (SJF)

- Shortest Job First (SJF): it runs the shortest job first, then the next shortest, and so on.

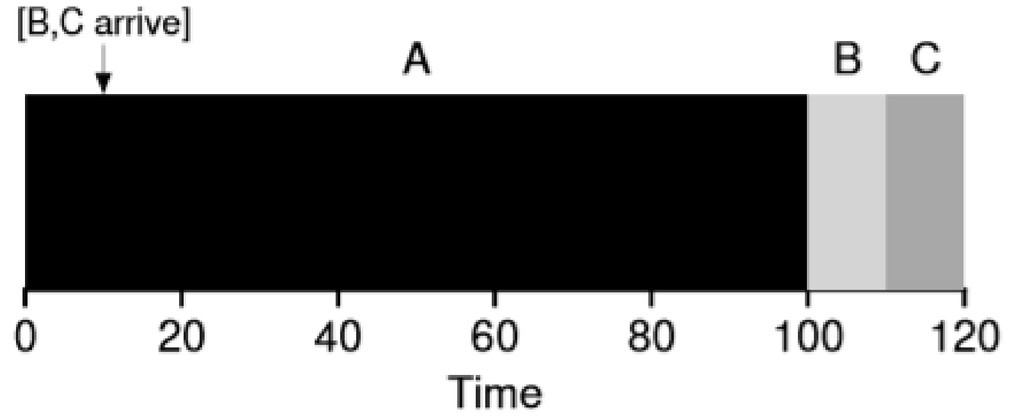
- Again assume three jobs (A, B, and C), A runs for 100 seconds while B and C run for 10 each
- Figure shows the results of running A, B, and C.
- SJF reduces average turnaround from 110 seconds (slide 16) to 50

(10+20+120)/3 = 50, more than a factor of two improvement



SJF with late arrival of B and C

- **This time, assume A arrives at t = 0 and needs to run for 100 seconds whereas B and C arrive at t = 10 and each need to run for 10 seconds**.
- Even though B and C arrived shortly after A, they still are forced to wait until A has completed.
- Thus suffer the same convoy problem.
- Average turnaround time for these three jobs is 103.33 seconds (100+(110−10)+(120−10))/ 3



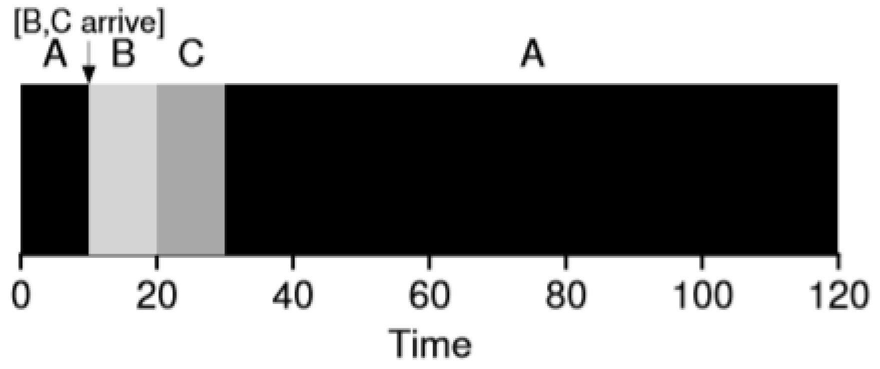
Shortest Time-to-Completion First (STCF)

- It is preemptive SJF.
- The scheduler certainly do something else when B and C arrive.
- Scheduler can preempt job A and decide to run another job, perhaps continuing A later.
- scheduler that add preemption to SJF, known as:
  - Shortest Time-to-Completion First (STCF) or
  - Preemptive Shortest Job First (PSJF)

STCF

- Any time a new job enters the system,

- The STCF scheduler determines which of the remaining jobs (including the new job) has the least time left, and schedules that one.
- **This time, assume A arrives at t = 0 and needs to run for 100 seconds whereas B and C arrive at t = 10 and each need to run for 10 seconds**.
- STCF would preempt A and run B and C to completion;
- only when they are finished would A's remaining time be scheduled.



- The result is a much-improved average turnaround time: 50 seconds $((120-0)+(20-10)+(30-10))/3$

SJF : Example

- Example:

SRTF: Advantages

- Generates minimal average waiting time
- Provide a standard for other algorithm.
- Better average response time compared to FCFS

SRTF: Disadvantages

- Algorithm can not be implemented as there is no way to know the burst time of a process
- Process with longer CPU burst time requirement will into starvation.
- No idea of priority.

Round Robin Scheduling

- This is designed for time sharing system, where it is not necessary to complete one process and then start another.
- To be responsive it divides the time of the CPU among the processes in the ready state.
- It is similar to FCFS scheduling, but pre-emption is added to switch between the processes.
- A time quantum (small unit of time) is defined. CPU can hold the process as per the time quantum defined.
- **Here, ready process is treated as circular queue.**
  - **The CPU scheduler goes around the ready queue,**
  - **Allocates the CPU to each process for a given time interval**
- Example: there are 5 processes,
  - CPU will be allotted to process P1 for given time quantum
  - Then it will be allotted to P2 for given time quantum and so on…
  - When it reaches end of the queue, again it will check is the P1 has completed its execution or not
  - Similarly it will check remaining processes execution

Implementation of RR

- We keep the ready queue as a FIFO queue of processes

- New processes are added to the tail of the ready queue.

- The CPU scheduler takes the first process from the queue (from head of the queue) then sets the timer for one time quantum, and dispatches the process.

    - There may be two things that can happen

Two scenarios:

When the process gets the CPU, the process requires the CPU less than one time quantum

The process itself will release the CPU voluntarily.

The CPU scheduler will then proceed to the next process in the ready queue

The CPU burst of the currently running process is longer than 1 time quantum, the time will go off and call interrupt

The context switch will be executed, and the process will be put on the tail of the ready queue

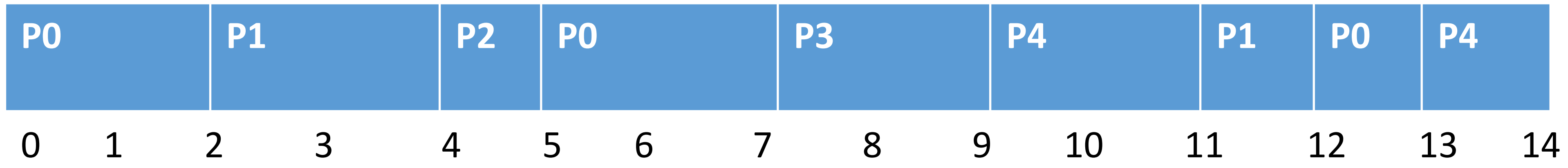
The CPU scheduler will then proceed to the next process in the ready queue

- Examples:

RR: Example

Process ID	Arrival Time	Burst Time
P0	0	5
P1	1	3
P2	2	1
P3	3	2
P4	4	3

Process ID	Turnaround Time (Exit time - Arrival Time)	Waiting Time=(Turnaround Time-Burst Time)
A	13	8
B	11	8
C	3	2
D	6	4
E	7	4



Advantages

- Perform best in terms of average response time
- Works well in case of time sharing system
- In RR all the processes have the equal priority because of fixed time quantum.
- Starvation will never occur because each process in every RR cycle will be schedule for a fixed time slice or time quantum.

Disadvantages

- Performance depends heavily on time quantum
  - Less time quantum:
    - Too many context switches
  - Large time quantum
    - Works like FCFS
  - So, need to keep the balance.

- No priority