05. Scheduling Algorithms

9^{th} ed: Ch. 6

10^{th} ed: Ch. 5

Objectives

- To understand how to apply several common scheduling algorithms
  - FCFS, SJF, SRTF
  - Priority
  - Round Robin
  - Multilevel Queues
- To understand use of measurement and prediction for unknown scheduling parameters

Outline

- First-Come First-Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority scheduling
- Round Robin (RR)

Outline

- First-Come First-Served (FCFS)
  - Convoy effect
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority scheduling
- Round Robin (RR)

First-Come First-Served (FCFS)

- Schedule depends purely on the order in which processes arrive
- Simplest possible scheduling algorithm
- Not terribly robust to different **arrival processes**

- E.g., suppose processes with the following burst times arrive in the order P_1, P_2, P_3

Process	Burst Time
P_1	24
P_2	3
P_3	3

First-Come First-Served (FCFS)

- Then the Gantt chart is

- The waiting times are

Process	Burst Time	Waiting Time
P_1	24	0
P_2	3	24
P_3	3	27

- This gives an average per-process waiting time of $\frac{0+24+27}{3} = 17$

The Convoy Effect

- Now suppose the same processes arrive in the order P_2, P_3, P_1
- Then the Gantt chart and waiting times are:

Process	Burst Time	Waiting Time
P_1	24	6
P_2	3	0
P_3	3	3

- Gives an average per-process waiting time of $(6 + 0 + 3)/3 = 3$
- First case is an example of the **Convoy Effect**
  - Short-run processes getting stuck behind long-run processes
  - Consider one CPU-bound and many IO-bound processes

Outline

- First-Come First-Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority scheduling
- Round Robin (RR)

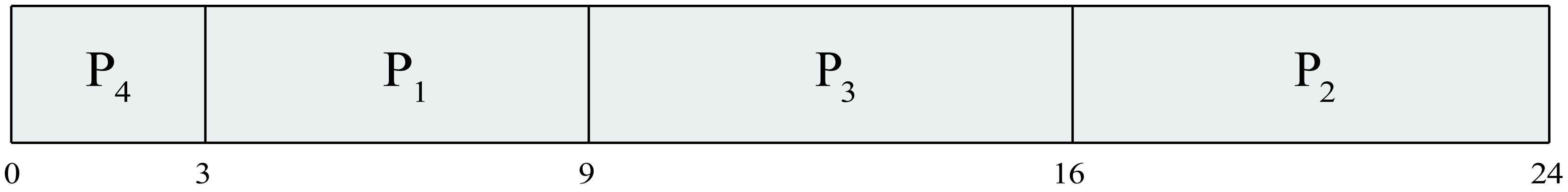
Shortest Job First (SJF)

- Associate length of next CPU burst with each process
- Schedule the process with the shortest next burst
- Optimality: SJF gives the least possible waiting time for a given set of processes
- But how can you know the length of the **next** CPU burst?
  - Ask the user?
  - Ask the developer?
  - Measure and predict?

Shortest Job First (SJF)

- Consider the following arrivals process and resulting Gantt chart:

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



- Gives an average per-process waiting time of $\frac{3+16+9+0}{4} = 7$

Outline

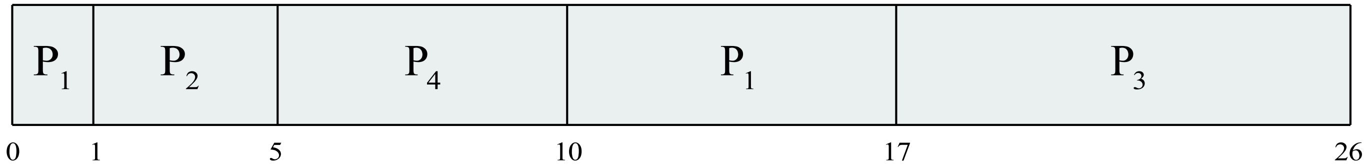
- First-Come First-Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
  - Predicting the future
  - Exponential averaging
- Priority scheduling
- Round Robin (RR)

Shortest Remaining Time First (SRTF)

- Simply a pre-emptive version of SJF
  - Pre-empt current process if a new one arrives with a shorter predicted burst length than the remaining time of the current process
- Distinguish **arrival time** and **burst length**, e.g.,

Process	Arrival Time	Burst Length
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- Gives Gantt chart



- Average waiting time is now $\frac{(10-1)+(1-1)+(17-2)+(5-3)}{4} = \frac{26}{4} = 6\,{}^{1}/_{2}$

Optimality in the future

- If SJF is optimal given a known set of processes (**demand**), then surely SRTF is optimal in the face of new runnable processes arriving?
- No! Why?
- Context switches are not free, so if short burst processes keep arriving the OS will start thrashing the CPU, so no useful work gets done

- More fundamentally,

  how can we know the length of a **future** burst?

Predicting burst lengths

- Assume the next burst will not be too different from the previous
- Then
  - measure burst lengths as processes are scheduled,
  - predict next burst length, and
  - choose the process with the shortest predicted burst length
- E.g., exponential averaging on length of previous bursts
  - Set t_n to be the measured length of the n^{th} CPU burst
  - Define τ_{n+1}, predicted length of $n+1^{\text{th}}$ burst as $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

Examples of exponential averaging

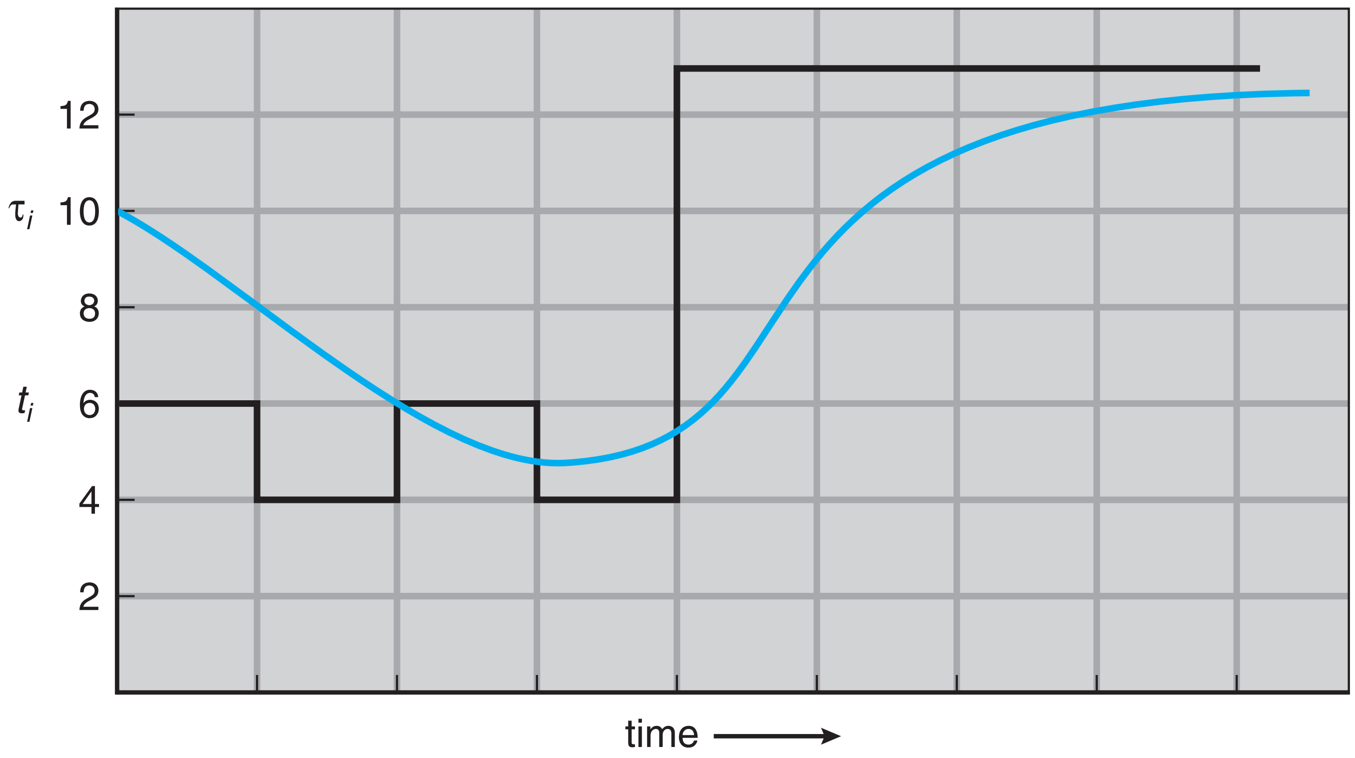
- Expanding this formula gives

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

where τ_0 is some constant

- As both $\alpha, 1-\alpha \leq 1$, each term has less weight than its predecessor
- Choose value of α according to our belief about the system, e.g,
  - If we believe past history irrelevant, choose $\alpha \approx 1$ and then get $\tau_{n+1} \approx t_n$
  - If we believe recent history irrelevant, choose $\alpha \approx 0$ and then get $\tau_{n+1} \approx \tau_n$
- Exponential averaging is often a good predictor if the variance is small
  - NB. Also should consider load, else (counter-intuitively) priorities increase with the load

Examples of exponential averaging



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

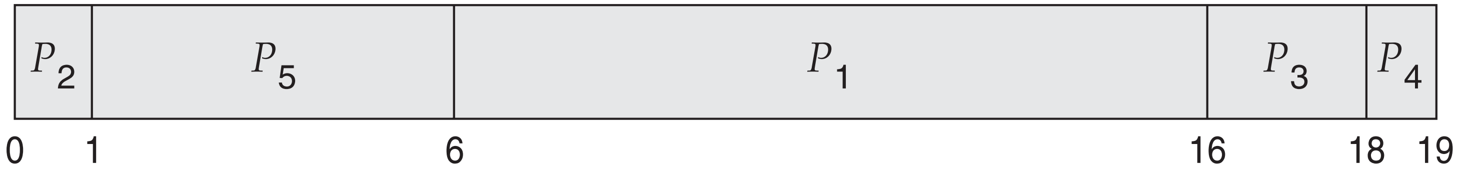
Outline

- First-Come First-Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority scheduling
  - Dynamic priorities
  - Computed priorities
- Round Robin (RR)

Priority scheduling

- Associate integer priority with process, and schedule the highest priority (~ lowest number) process, e.g.,

Process	Priority	Burst Length
P_1	3	10
P_2	1	1
P_3	4	2
P_4	5	1
P_5	2	5



- Average waiting time now

$$\frac{(1+5)+0+(1+5+10)+(1+5+10+2)+1}{5}=\frac{41}{5}=8\,{}^{1}/_{5}$$

- Consider: SJF as priority scheduling using inverse of predicted burst length

Dynamic priority scheduling

- **Starvation** can occur if low priority processes never execute
- Urban legend?
  - When the IBM 7074 at MIT was shut down in 1973, low-priority processes were found that had been submitted in 1967 and had not yet been run...
- This is the biggest problem with static priority systems!
  - A low priority process is not guaranteed to run — ever!
- Solve by making priorities **dynamic**
  - E.g., **aging** increases priority starting from a static base as time passes without process being scheduled

Computed Priority

- E.g., traditional UNIX scheduler
  - Priorities 0–127; user processes ≥ PUSER = 50
  - Round robin within priorities, quantum e.g. 100ms
- Priority of process j at start of interval i is based on
  - *nice* level, a user controllable parameter between -20 and 20, and
  - $load_j$ the sampled average length of the run queue for process j

$$P_j(i) = \mathrm{Base_j} + \frac{\mathrm{CPU}_j(i-1)}{4} + 2\times\mathrm{nice_j}$$

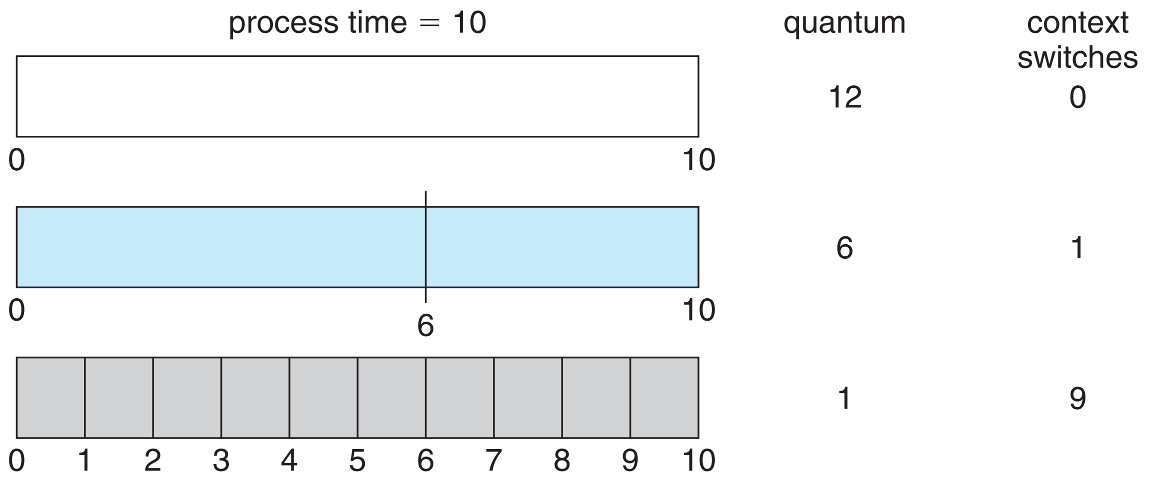
$$\mathrm{CPU}_j(i) = \frac{2\times\mathrm{load_j}}{(2\times\mathrm{load_j})+1}\mathrm{CPU}_j(i-1) + \mathrm{nice_j}$$

Outline

- First-Come First-Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority scheduling
- Round Robin (RR)
  - Multilevel queues
  - Multilevel feedback queues

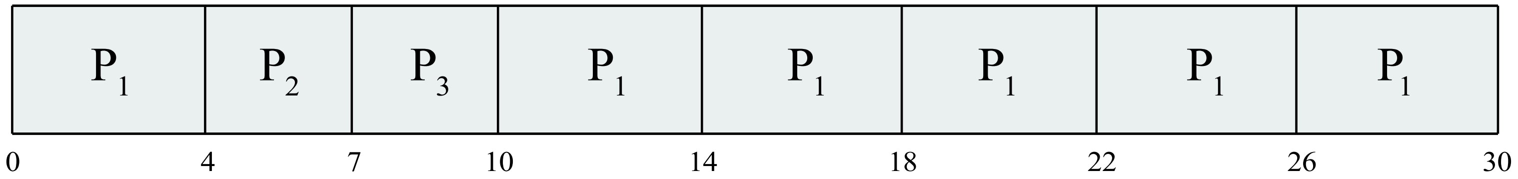
Round Robin

- A pre-emptive scheduling scheme for time-sharing systems
  - Give each process a **quantum** (or time-slice) of CPU time e.g., 10—100 milliseconds
  - Once quantum elapsed, process is pre-empted and appended to the ready queue
  - Timer interrupts every quantum to schedule next process
- Can be tricky to choose q correctly
  - q too large degenerates into a FIFO queue ($\sim$ FCFS)
  - q too small makes the context switch overhead too great
- q usually 10ms to 100ms, while context switch < 10 μsec



Round Robin

- Consider the first example again

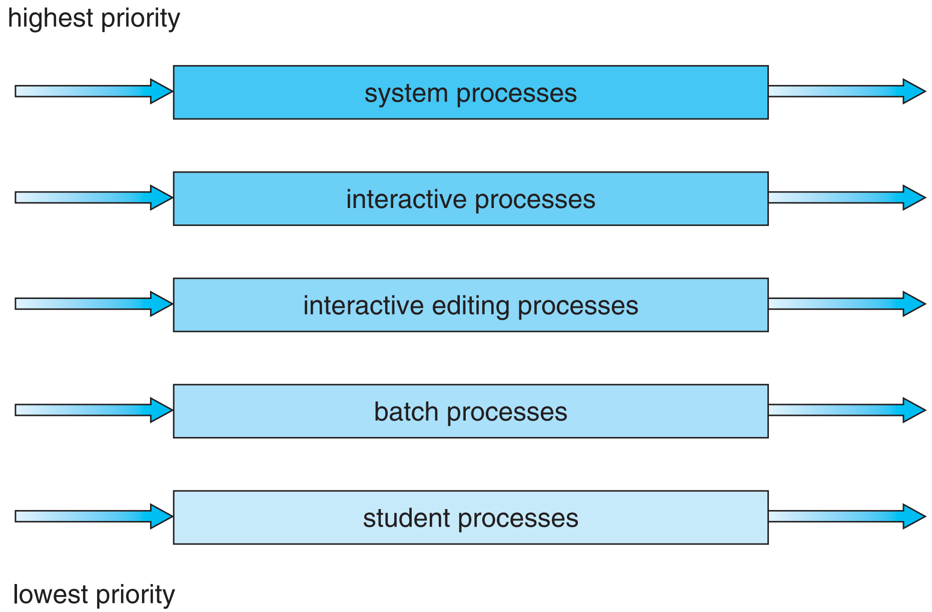


Process	Burst Time
P_1	24
P_2	3
P_3	3

- For quantum q and n processes ready,
  - **Fair**: each process gets $^1/_n$ CPU time in chunks of at most q time units, and
  - **Live**: no process ever waits more than $(n-1)q$ time units
- Typically
  - higher average turnaround time than SRTF, but
  - better average response time

Multilevel Queues

- Partition Ready queue into many queues for different types of process, e.g.,
  - Foreground/interactive processes
  - Background/batch processes
- Each process is permanently assigned a given queue
- Each queue runs its own scheduling algorithm, e.g.,
  - Foreground runs Round Robin
  - Background runs First-Come First-Served

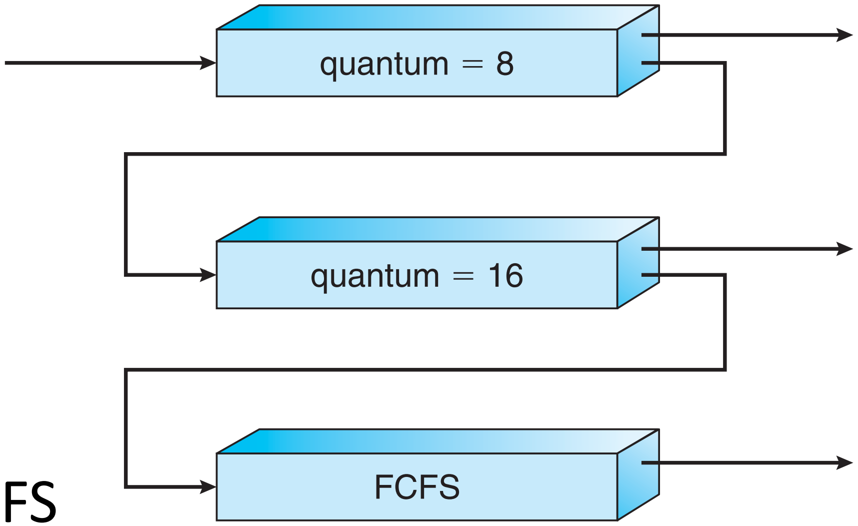


Multilevel Feedback Queues

- Now scheduling must be done between the queues:
  - **Fixed priority**, e.g., serve all from foreground then from background, permits starvation
  - **Time slice**, each queue gets a certain amount of CPU time which it can schedule amongst its processes, e.g., 80% to foreground in RR, 20% to background in FCFS
- A process can move between the various queues
  - Aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Multilevel Feedback Queues

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS



- Scheduling
  - A new job enters queue Q_0 which is served FCFS
  - When it gains CPU, job receives 8 milliseconds
  - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
  - If it still does not complete, it is pre-empted and moved to queue Q_2

Summary

- First-Come First-Served (FCFS)
  - Convoy effect
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
  - Predicting the future
  - Exponential averaging
- Priority scheduling
  - Dynamic priorities
  - Computed priorities
- Round Robin (RR)
  - Multilevel queues
  - Multilevel feedback queues