ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Real-Time Systems for Automation M

8. CPU Scheduling

Notice

The course material includes slides downloaded from:

http://codex.cs.yale.edu/avi/os-book/

(slides by Silberschatz, Galvin, and Gagne, associated with Operating System Concepts, 9th Edition, Wiley, 2013)

and

http://retis.sssup.it/~giorgio/rts-MECS.html

(slides by Buttazzo, associated with Hard Real-Time Computing Systems, 3rd Edition, Springer, 2011)

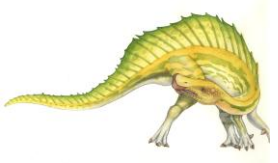
which have been edited to suit the needs of this course.

The slides are authorized for personal use only.

Any other use, redistribution, and any for profit sale of the slides (in any form) requires the consent of the copyright owners.

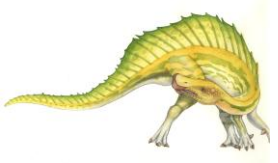
Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Policies
- Thread Scheduling
- Multiple-Processor Scheduling
- Operating Systems Examples
- Algorithm Evaluation



Objectives

- Describe various CPU scheduling algorithms
- Assess CPU scheduling algorithms based on scheduling criteria
- Explain the issues related to multiprocessor and multicore scheduling
- Describe the scheduling algorithms used in the Windows, Linux, and Solaris operating systems
- Apply modeling and simulations to evaluate CPU scheduling algorithms

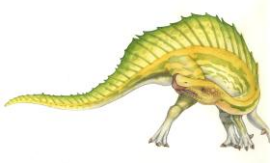


Basic Concepts

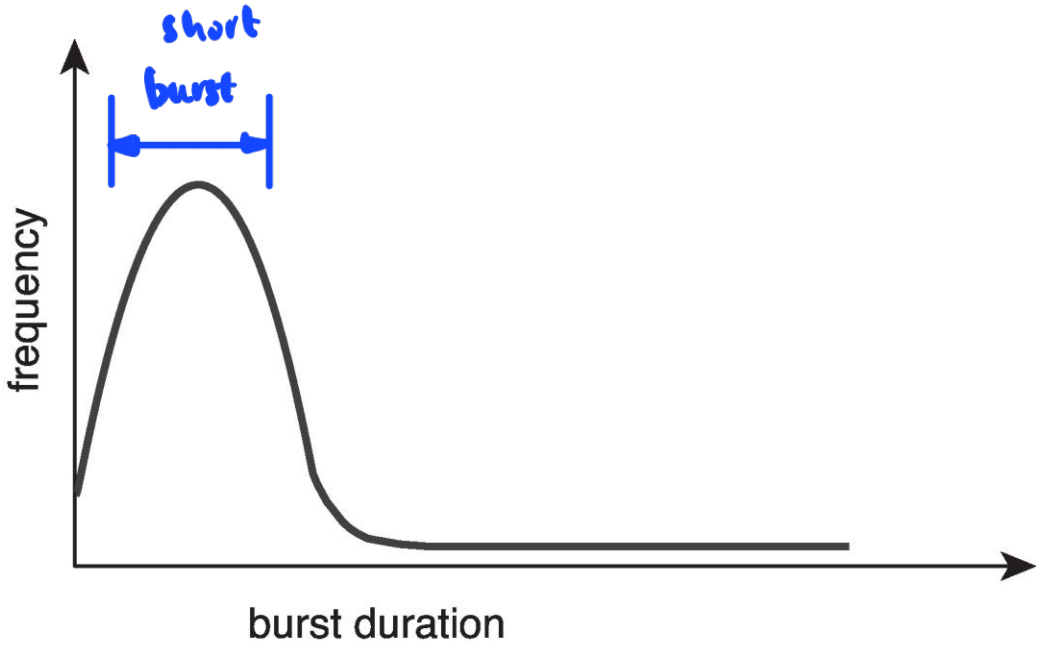
processes can be active at the same time

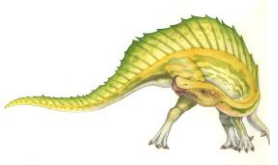
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern

multitasking → giving concurrency to make an "illusion"



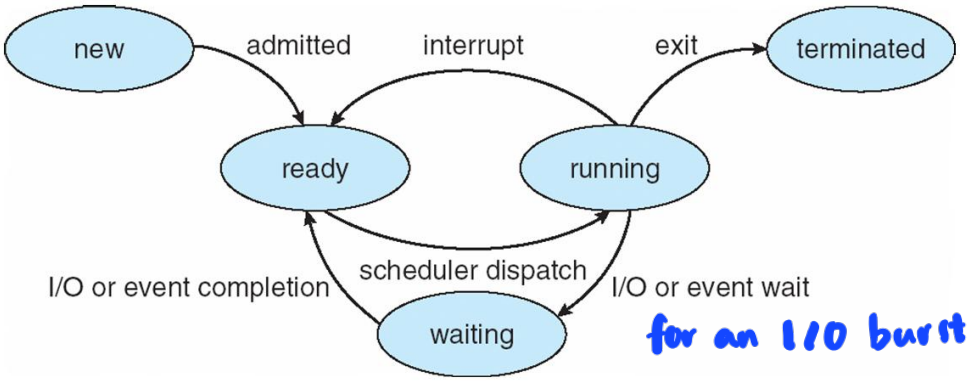
Histogram of CPU-burst Times

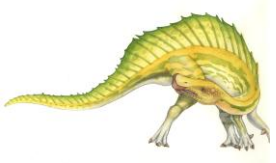




CPU Scheduler

- The **CPU scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. is created and transit from *new* to *ready* state
  2. Switches from running to waiting state (one queue ➔ just dispatching)
  3. Switches from running to ready state
  4. Switches from waiting to ready
  5. Terminates (one queue ➔ just dispatching)

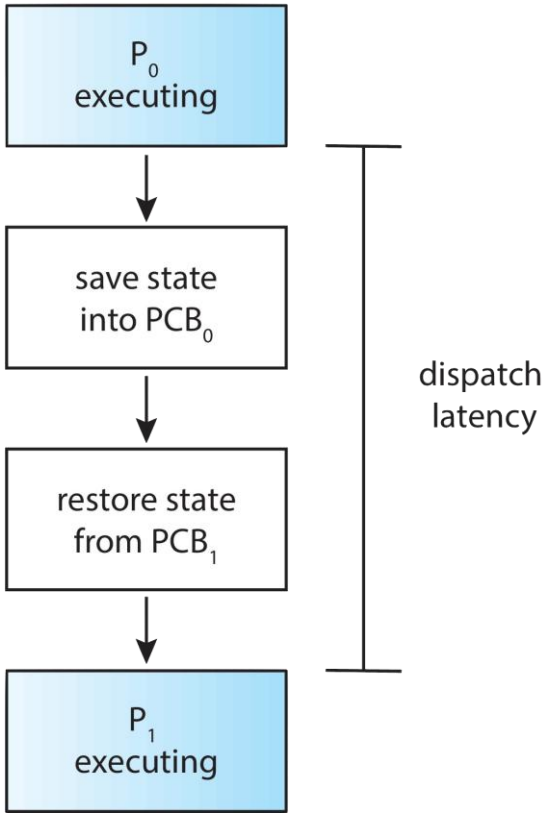


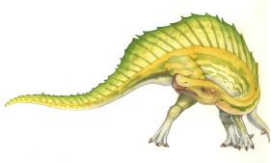


Dispatcher

loads / unloads the process

- Dispatcher module 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





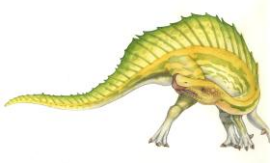
Terminology: preemption

being able to empty a CPU

- Scheduling algorithms can be classified into two categories:
  - **nonpreemptive**: the CPU is allocated to the running process until it either terminates or voluntarily suspends (because of an I/O request)
  - **preemptive**: the running process can be preempted, i.e., the OS can take it out of the CPU and dispatch another process
- Timesharing systems need a preemptive CPU scheduler (multitasking)
- from the viewpoint of the process: with preemptive scheduling the access to the CPU can be interrupted unexpectedly
- With preemptive scheduling, we have additional issues:
  - Consider access to shared data
  - Consider preemption while in kernel mode

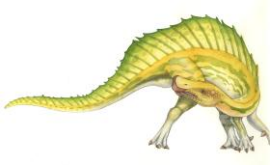
Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible maximize
- **Throughput** – # of processes that complete their execution per time unit maximize
- **Turnaround time** – amount of time from submission to completion for each process minimize it!
- **Waiting time** – amount of time a process has been waiting in the ready queue minimize
- **Response time** – amount of time it takes from when a request was submitted until the *first* response is produced (to have a good user experience, for time-sharing environment,) minimize



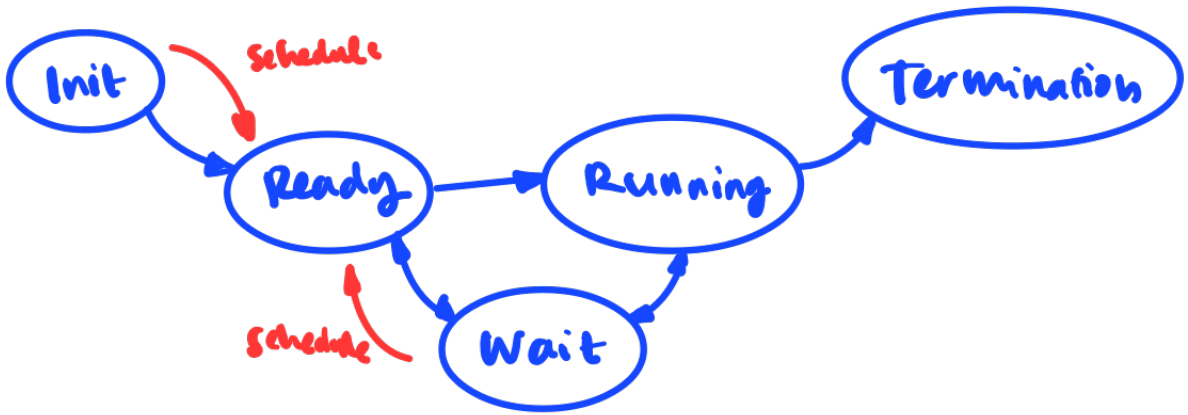
Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time (variance)



Quizzes

- The waiting time of a process P:
  a) is the time spent by P while in the waiting state
  b) is the time spent in the ready queue by P in a given scheduling
  c) is the time spent in the waiting state by P in the average scheduling
  d) is the number of times all other processes have to wait for P
- Without preemption, scheduling decisions are taken only when a process terminates or gets blocked

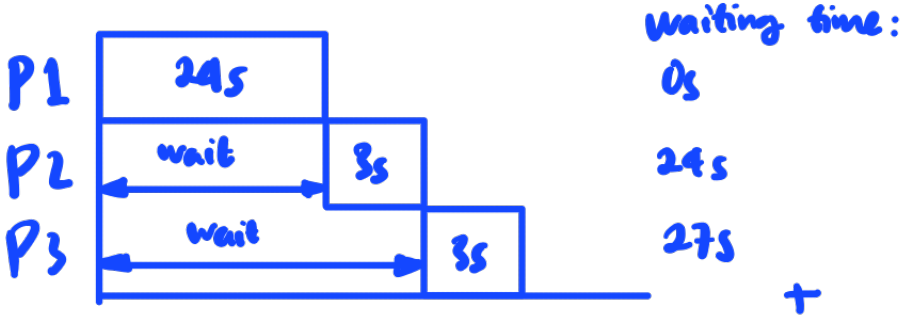


FCFS Scheduling – Example 1

- First Come, First Served
- Nonpreemptive

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- What is the Gantt Chart for the schedule?



- Waiting time?
- Average waiting time?

FCFS Scheduling – Example 1. Solution

- First Come, First Served
- Nonpreemptive

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- Gantt Chart for the schedule:

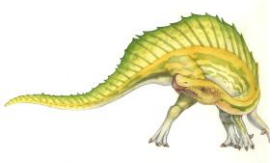
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling – Example 2

Suppose that the processes arrive in the order:

- P_2 , P_3 , P_1
- What is the Gantt chart for the schedule?

- Waiting time?
- Average waiting time?

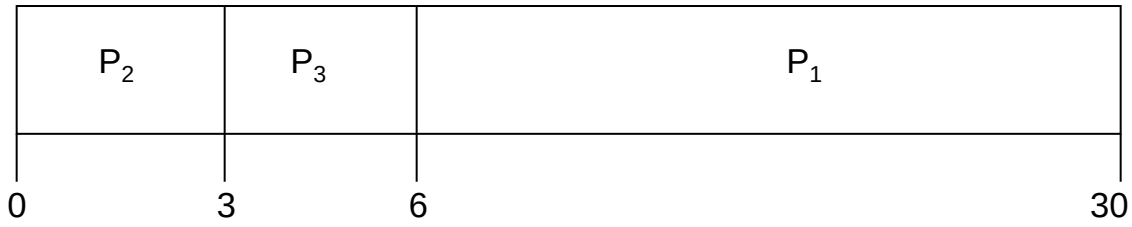


FCFS Scheduling – Example 2. Solution

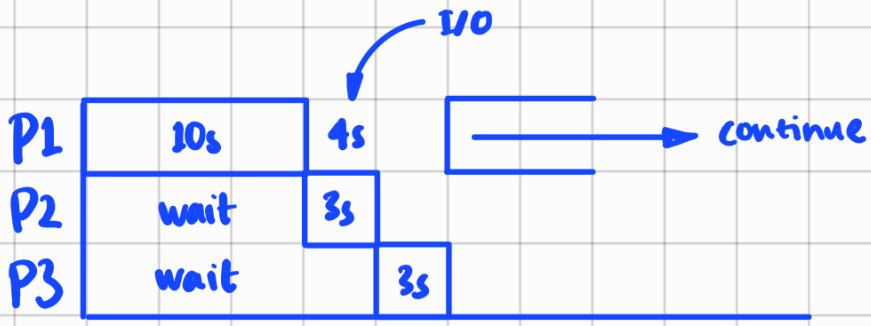
Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- Gantt chart for the schedule:



- 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
  - Consider one CPU-bound and many I/O-bound processes



execute the first on ready list!

Shortest-Job-First (SJF) Scheduling

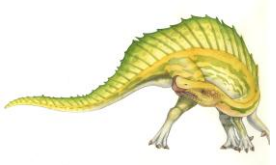
shortest goes first

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

minimizes the process in waiting queue

- 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
    - In batch systems (long-term scheduling)
      - if Good estimate: fast response (incentive)
      - if Time-limit-exceeded, error + resubmission (penalty)

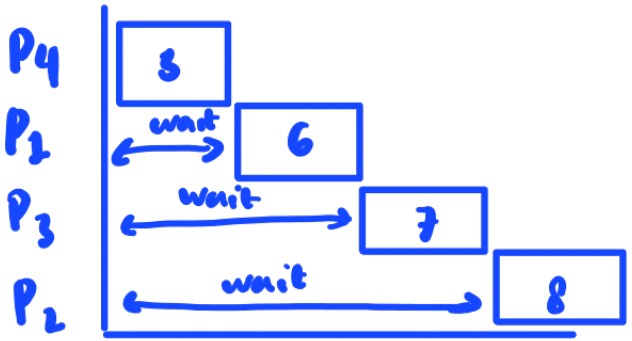
- SJF can be either nonpreemptive or preemptive
- Preemptive version called **shortest-remaining-time-first**



SJF Scheduling – Example

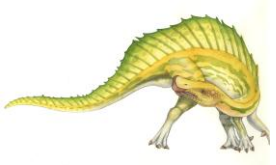
Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



- Average waiting time?

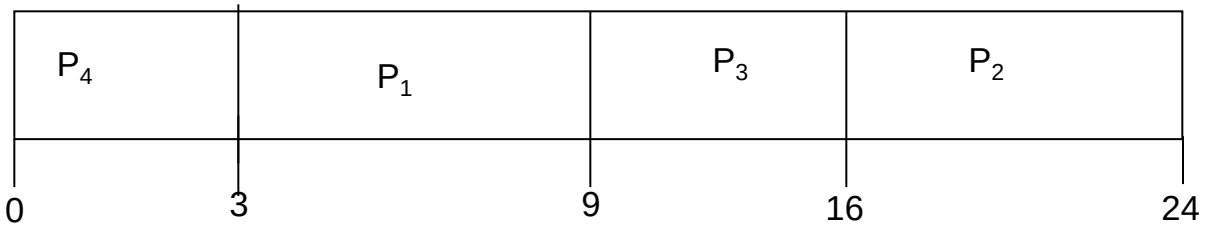
$$\frac{0+3+9+16}{4} = 7$$



SJF Scheduling – Solution

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



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

Example of Shortest-remaining-time-first

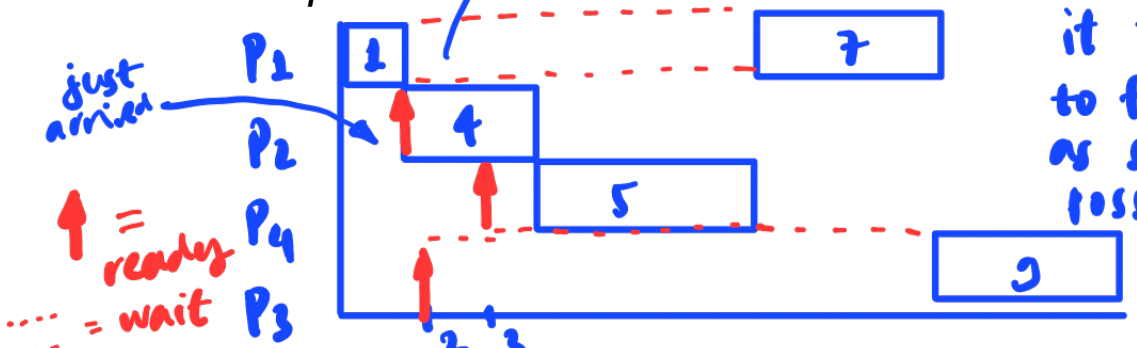
- Now we add the concepts of varying arrival times and preemption to the analysis

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

every time new process arrives. OS does the scheduling

Next CPU burst based on previous experience

- *Preemptive* SJF Gantt Chart



- Average waiting time?

P_1 waits for 9

P_2 waits for 0 bcs it starts running right away

P_3 waits $17 - 2 = 15$

P_4 waits for $5 - 3 = 2$

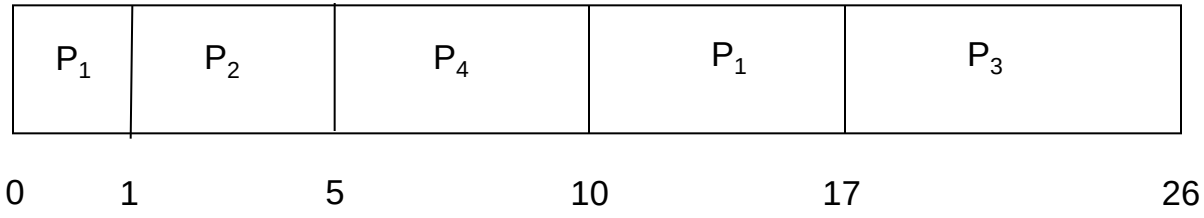
pre-emptive version of SJF

SRTF Scheduling – Solution

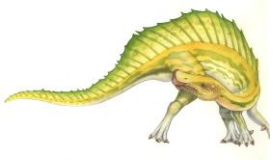
- Now we add the concepts of varying arrival times and preemption to the analysis

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

- *Preemptive* SJF Gantt Chart



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



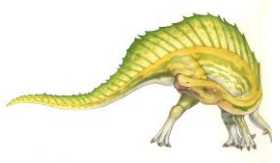
Priority Scheduling

OS assigns a priority on the process

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive

Lower priority number, the higher the priority is.

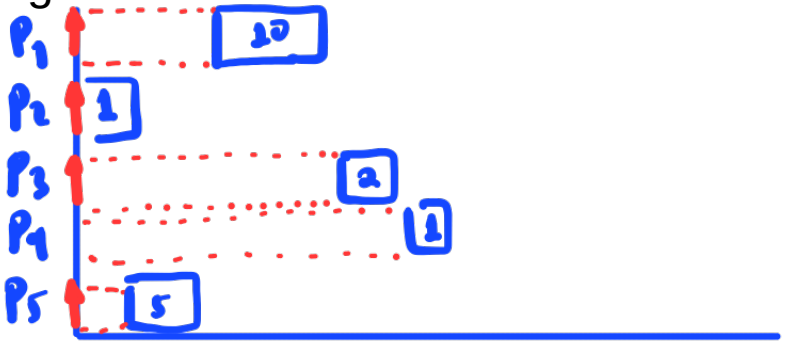
- FCFS is priority scheduling where priority is the arrival time
- SJF is priority scheduling where priority is inverse of next CPU burst time
- Priorities can be defined internally or externally (wrt the OS)
  - Time limits, memory requirements, number of open files, politics
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process



Priority Scheduling - Example

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

- Priority scheduling Gantt Chart

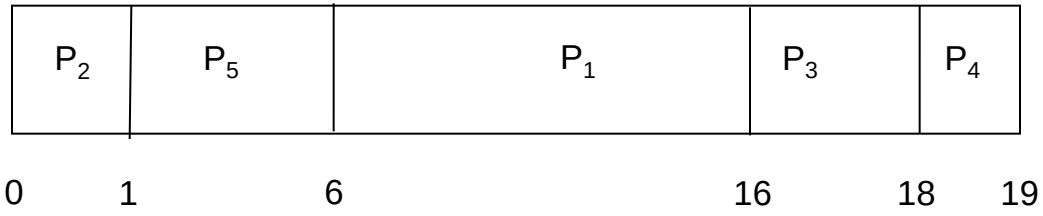


- Average waiting time?

Priority Scheduling - Solution

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

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

Round Robin (RR)

small unit CPU time

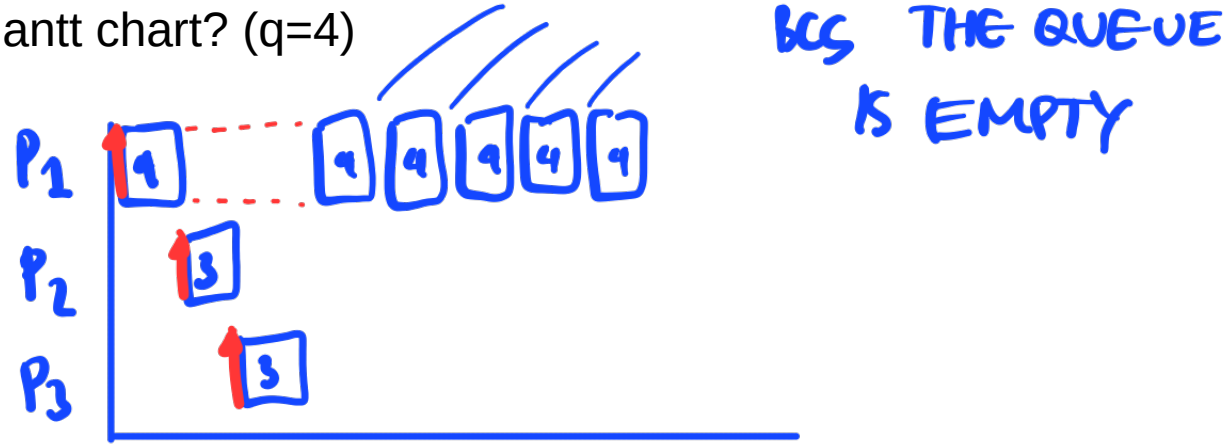
- Each process gets a small unit of CPU time (**time quantum** *q*), usually 10-100 milliseconds. After this time has elapsed, the process is **preempted** and **added to the end of the ready queue**.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than (*n*-1)*q* time units.
- Timer interrupts every quantum to schedule next process
- Perfect for min response time. Worst for throughput and waiting time
- Performance
  - *q* large ⇒ FCFS
  - *q* small ⇒ too many context-switches. *q* must be large wrt context switch, otherwise overhead is too high

timer will start every process start

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

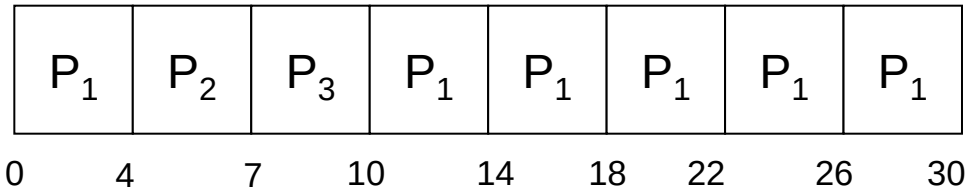
- Gantt chart? (q=4)



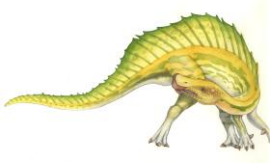
Example of RR - Solution

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 μsec



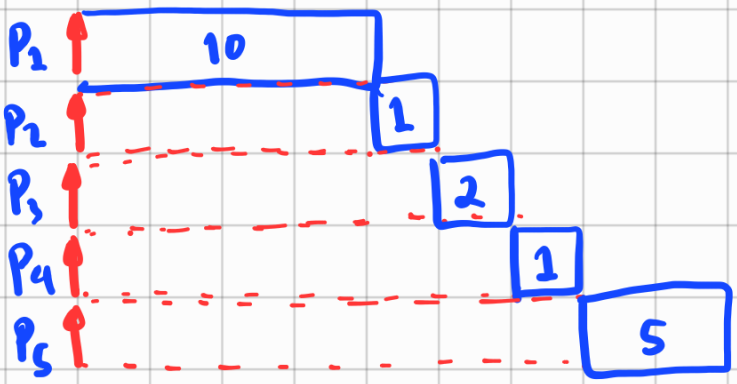
Exercise (sample exam question)

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

arrived on the same time

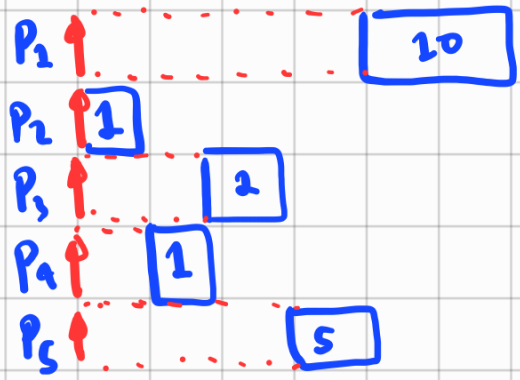
- Arrived in the top-to-bottom order.
  - Illustrate FCFS, SJF, nonpreemptive priority, RR (q=1) using Gantt
    - Turnaround times? → time from arrive to be processed
    - Waiting times?
    - Which algorithm results in the minimum average waiting time?

FCFS



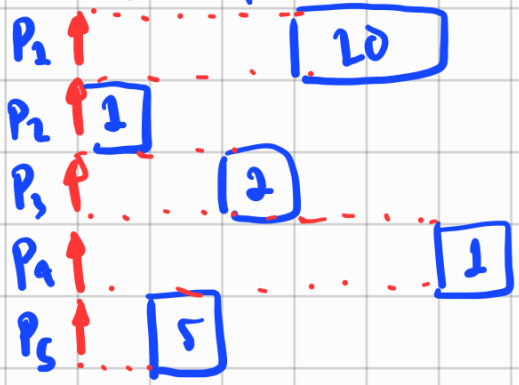
Turnaround	Waiting
10 - 0	0
11 - 0	10
13 - 0	11
14 - 0	13
19 - 0	14

SJF



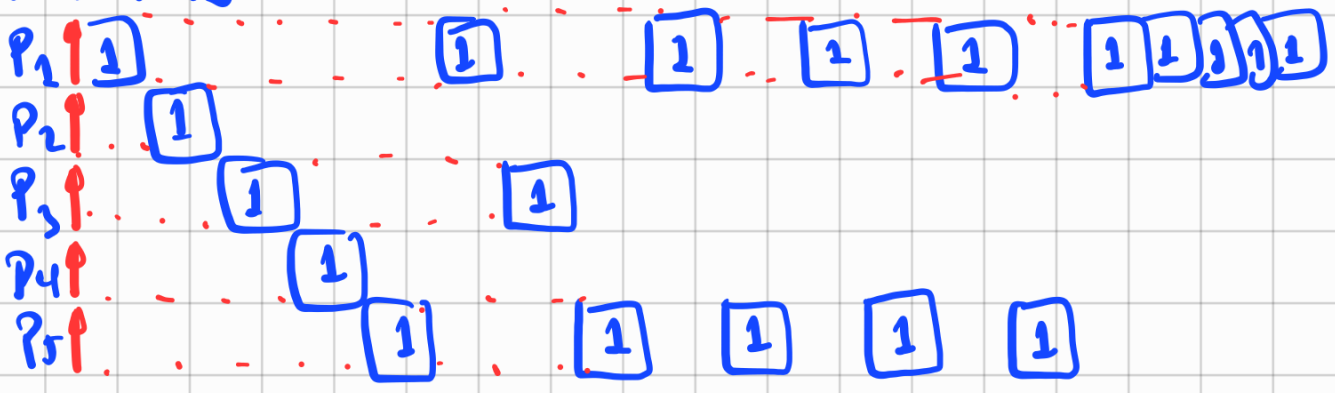
Turnaround	Waiting
19 - 0	9
1 - 0	0
4 - 0	2
2 - 0	1
9 - 0	4

Non-preemptive priority



Turnaround	Waiting
18	8
1	0
8	6
19	18
6	1

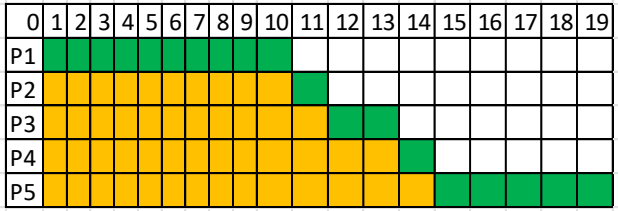
RR FCFS



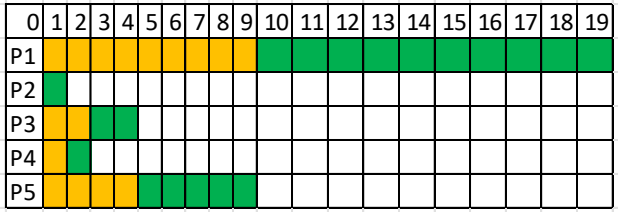
T	W
19	
2	
7	

Exercise

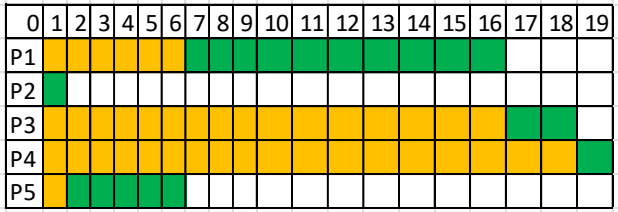
	B	P
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2



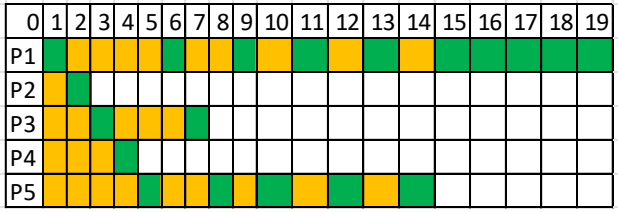
		P1	P2	P3	P4	P5	Avg
FCFS	Turnaround	10	11	13	14	19	13.4
	Waiting	0	10	11	13	14	9.6



		P1	P2	P3	P4	P5	Avg
SJF	Turnaround	19	1	4	2	9	7
	Waiting	9	0	2	1	4	3.2



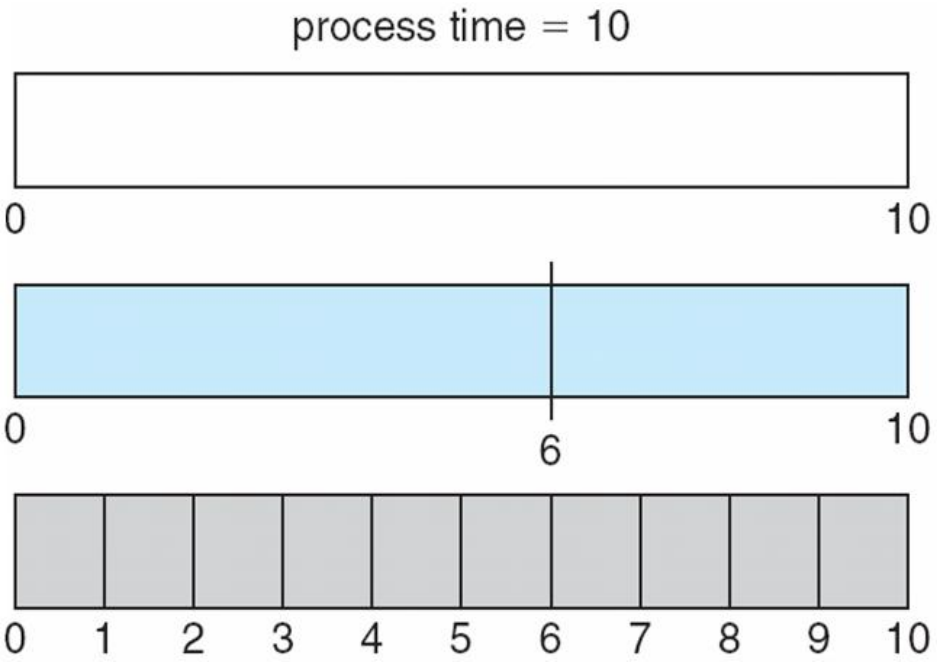
		P1	P2	P3	P4	P5	Avg
Priority	Turnaround	16	1	18	19	6	12
	Waiting	6	0	16	18	1	8.2



		P1	P2	P3	P4	P5	Avg
RR	Turnaround	19	2	7	4	14	9.2
	Waiting	9	1	5	3	9	5.4

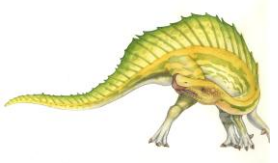
Time Quantum and Context Switch Time

how to determine the quantum?



quantum	context switches
12	0
6	1
1	9

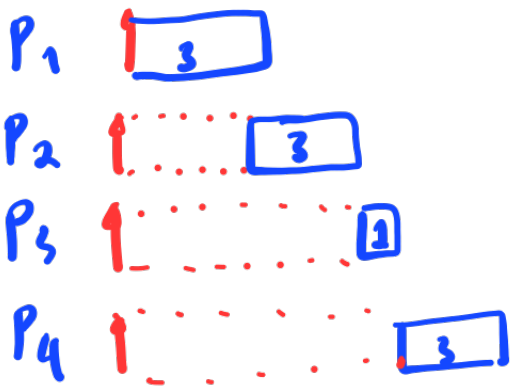
- it seems the longer is the quantum, the better for turnaround time, but…



RR Waiting Time & Quantum

- What average waiting time if RR q = 3? q = 4? q = 2?

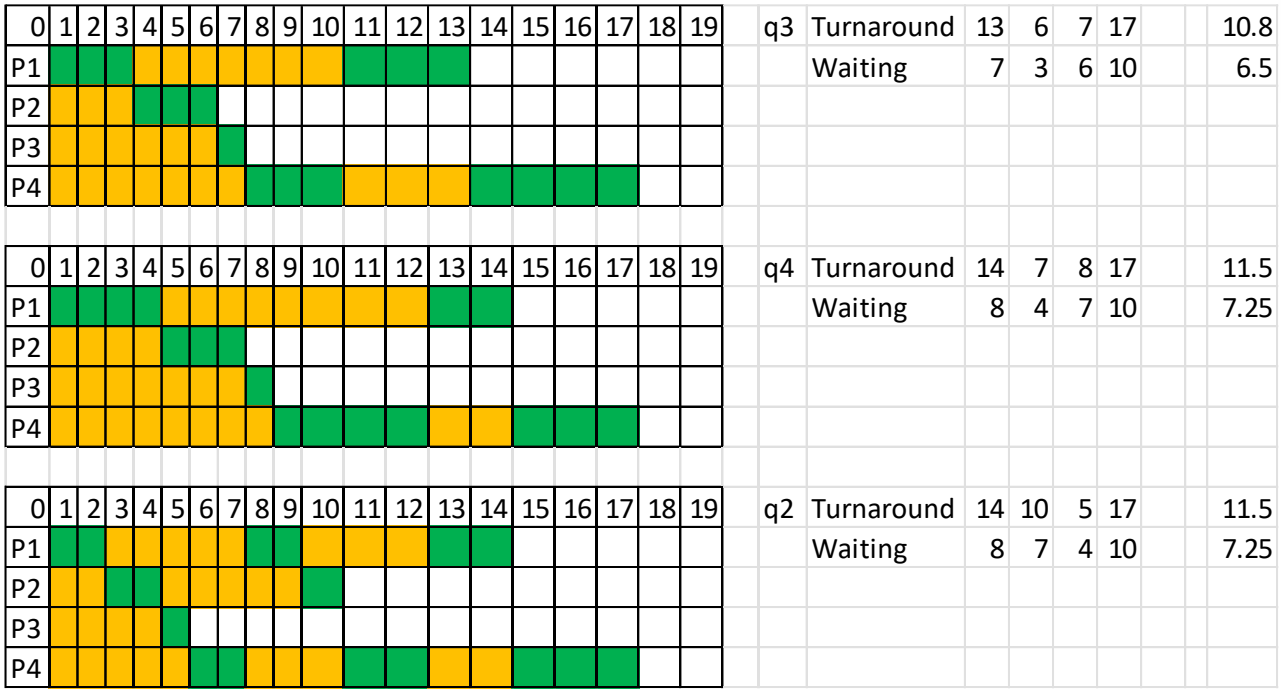
process	time
P_1	6
P_2	3
P_3	1
P_4	7



RR Waiting Time & Quantum

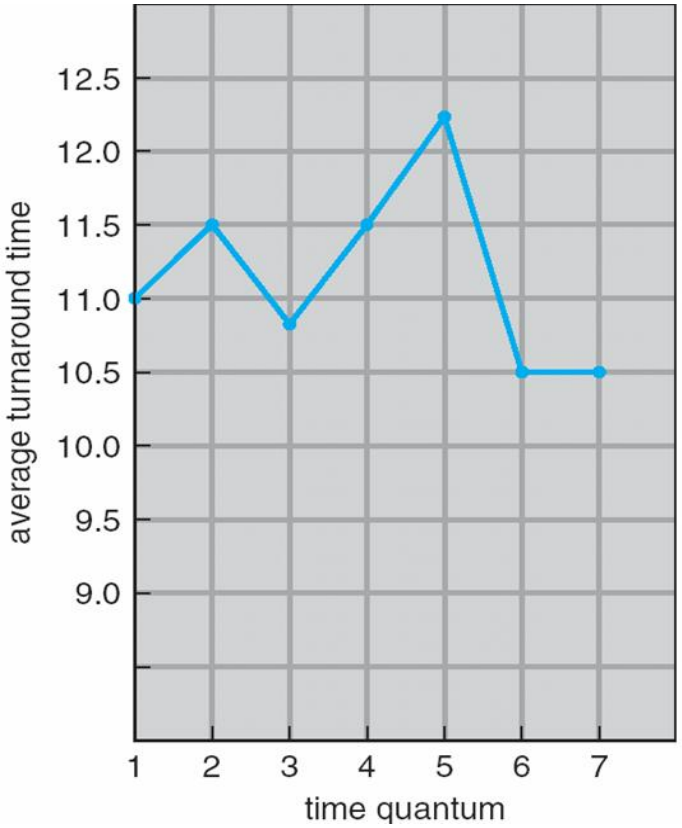
- What average waiting time if RR q = 3? q = 4? q = 2?

process	time
P_1	6
P_2	3
P_3	1
P_4	7



q = 3 is the best!

Turnaround Time Varies With The Time Quantum



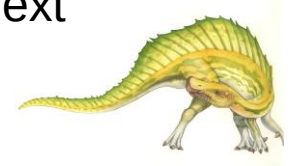
process	time
P_1	6
P_2	3
P_3	1
P_4	7

← non-linear relation

80% of CPU bursts should be shorter than q in general

we can't really know how long the quantum should be

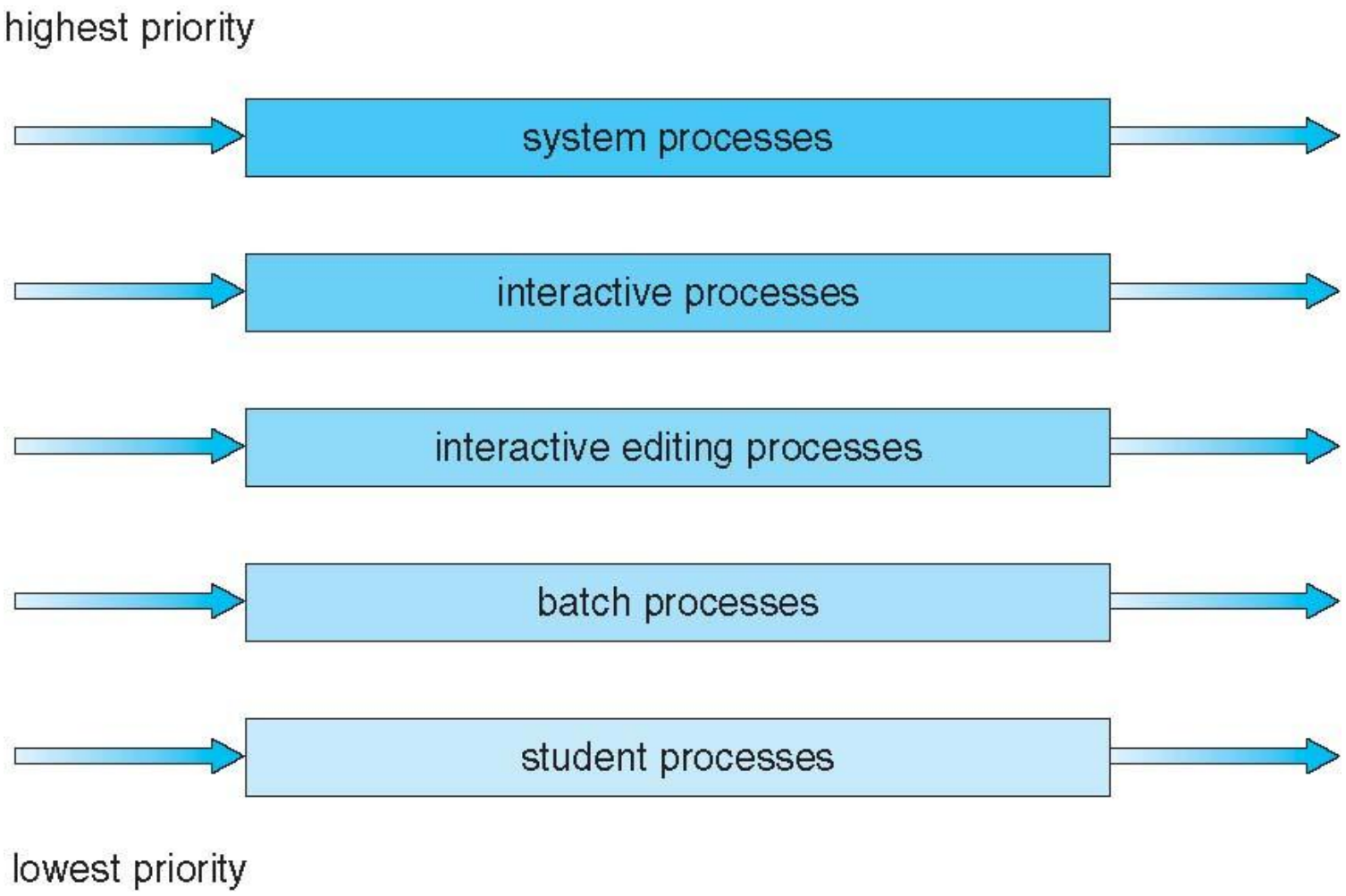
- Increasing q does not necessarily improve average turnaround
- In general: improvement whenever most processes finish their next CPU burst in 1q



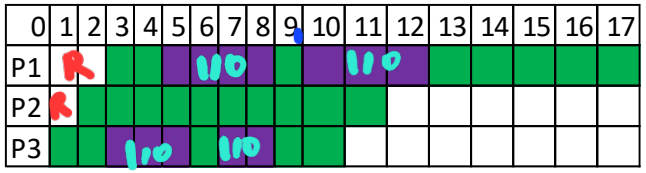
Multilevel Queue

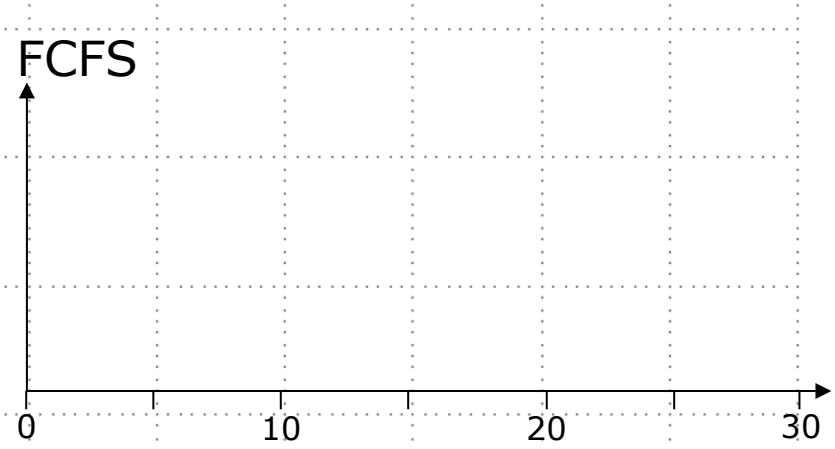
- Ready queue is partitioned into separate queues, eg:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm, for example:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of 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.

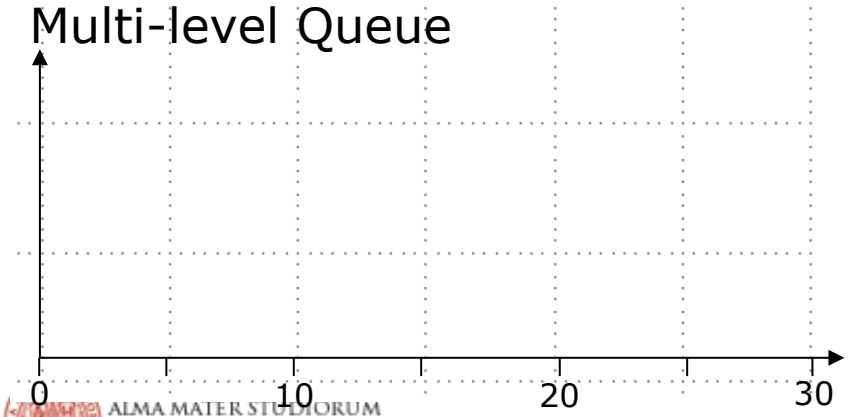
Multilevel Queue Scheduling



Exam exercise (July 11, 2017)







E3) Consider a scheduler and a task set $\{P_1, P_2, P_3\}$, where:

- P_1 is released after 2 units of time and has a CPU burst of 2 units, followed by an I/O burst of 4 units, a second CPU burst of 1 unit, a second I/O burst of 3 units, and a final CPU burst of 5 units;
- P_2 is released after just 1 unit of time, and has a CPU burst of 10 units;
- P_3 is released at the beginning of time and starts with a CPU burst of 2 units, followed by an I/O burst of 3 units, a second CPU burst of 1 unit, a second I/O burst of 2 units, and a final CPU burst of 2 units.

Assume that these are the only tasks in the system, that each task uses dedicated I/O devices (so I/O bursts can progress in parallel without interference), and that there is only 1 CPU.

E3.1) Show a **First-Come-First-Served** (FCFS) schedule of the task set $\{P_1, P_2, P_3\}$.

E3.2) Compare it with a **Multi-Level Queue** schedule of the same task set $\{P_1, P_2, P_3\}$ where two queues are defined: a *foreground* queue with *high priority*, where P_1 belongs, and a *background* queue with *low priority*, where P_2 and P_3 belong. The *background* queue is managed FCFS.

Be sure to motivate your answer. Base your comparison on this task set's **waiting times** and efficient use of computer resources.

Exam exercise (July 11, 2017)

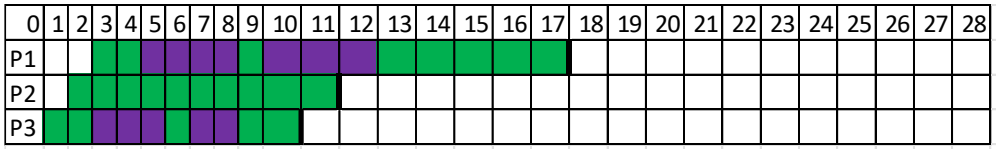
E3) Consider a scheduler and a task set $\{P_1, P_2, P_3\}$, where:

- P_1 is released after 2 units of time and has a CPU burst of 2 units, followed by an I/O burst of 4 units, a second CPU burst of 1 unit, a second I/O burst of 3 units, and a final CPU burst of 5 units;
- P_2 is released after just 1 unit of time, and has a CPU burst of 10 units;
- P_3 is released at the beginning of time and starts with a CPU burst of 2 units, followed by an I/O burst of 3 units, a second CPU burst of 1 unit, a second I/O burst of 2 units, and a final CPU burst of 2 units.

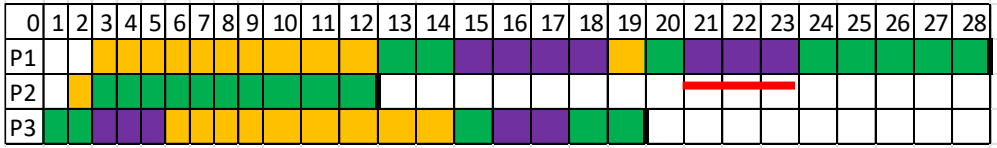
Assume that these are the only tasks in the system, that each task uses dedicated I/O devices (so I/O bursts can progress in parallel without interference), and that there is only 1 CPU.

E3.1) Show a **First-Come-First-Served** (FCFS) schedule of the task set $\{P_1, P_2, P_3\}$.

E3.2) Compare it with a **Multi-Level Queue** schedule of the same task set $\{P_1, P_2, P_3\}$ where two queues are defined: a *foreground* queue with *high priority*, where P_1 belongs, and a *background* queue with *low priority*, where P_2 and P_3 belong. The *background* queue is managed FCFS.

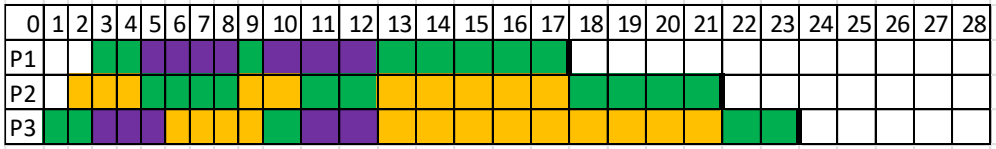


		P1	P2	P3	AVG
FCFS	Turnaround	26	11	19	18.7
	Waiting	11	1	9	7.0



		P1 (FQ)	P2 (BQ)	P3 (BQ)	AVG
MLQ	Turnaround	15	19	25	19.7
	Waiting	0	9	15	8.0

preempted low-priority keep position in the queue (typical)



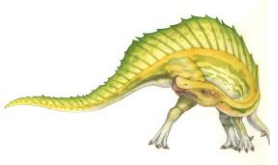
MLQ	Turnaround	15	20	23	19.3
	Waiting	0	10	16	8.7

preempted low-priority do not keep position in the queue (just as example)

Multilevel Feedback Queue

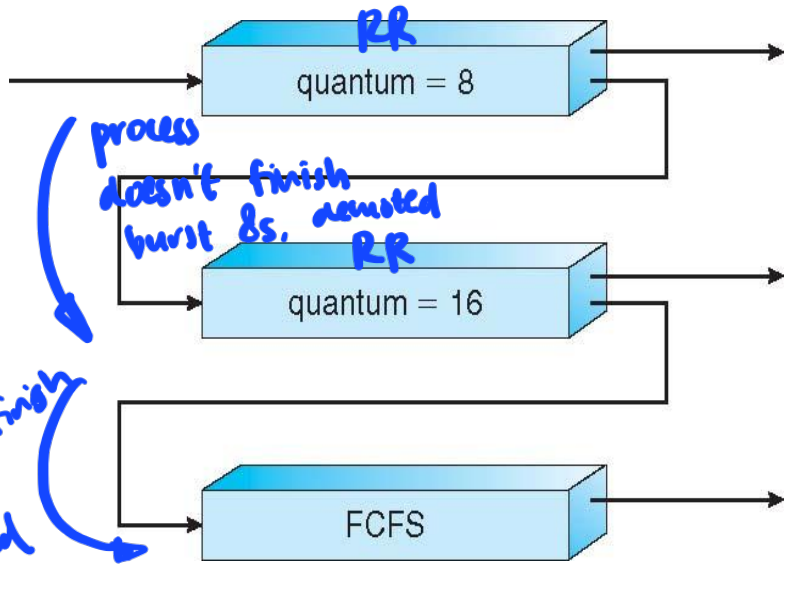
process can be moved

- 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
  1. which queue the process joins upon creation (starting queue)
  2. schedule from which queue (priority-based)
  3. scheduling algorithms for each queue
  4. which queue the process joins when re-enters after quantum expiration/waiting state
     - determine when to promote/demote a process ?



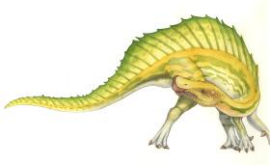
Example of Multilevel Feedback Queue

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

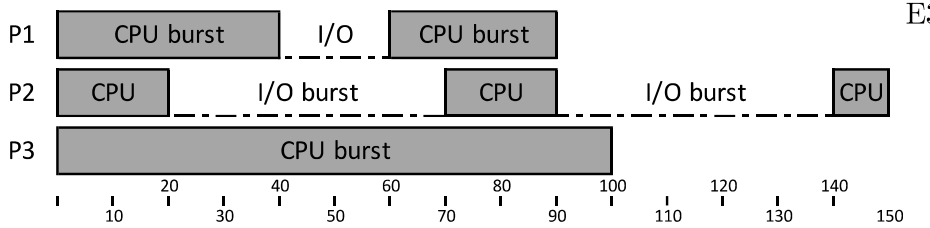


- Scheduling
  - A new job enters queue Q_0 which is served RR
    - When it gains CPU, job receives 8 msec
    - If it does not finish in 8 msec, job is moved to queue Q_1 (demotion)
  - At Q_1 job is again served and receives +16 msec
    - If it still does not complete, it is preempted and moved to queue Q_2

➔always demotion, never promotion



Exercise (June 12, 2019)

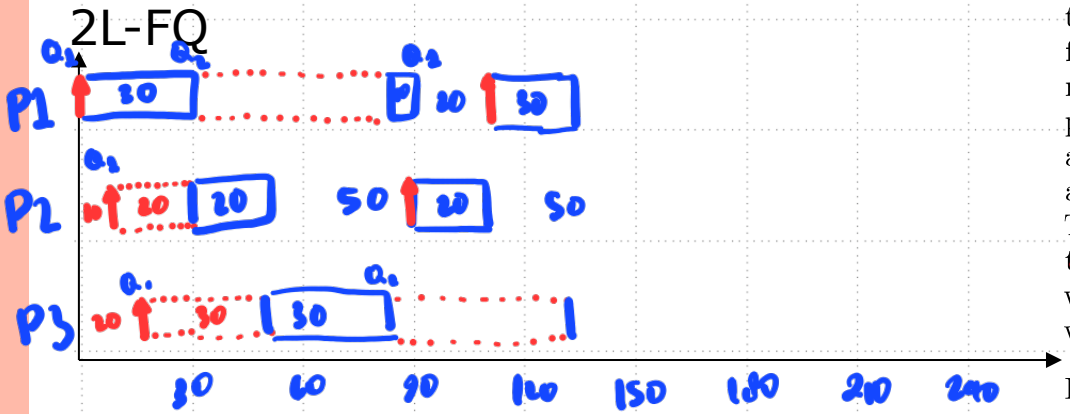


E3) Consider a **preemptive** scheduler and a task set $\{P_1,P_2,P_3\}$ as illustrated in Figure 2, where:

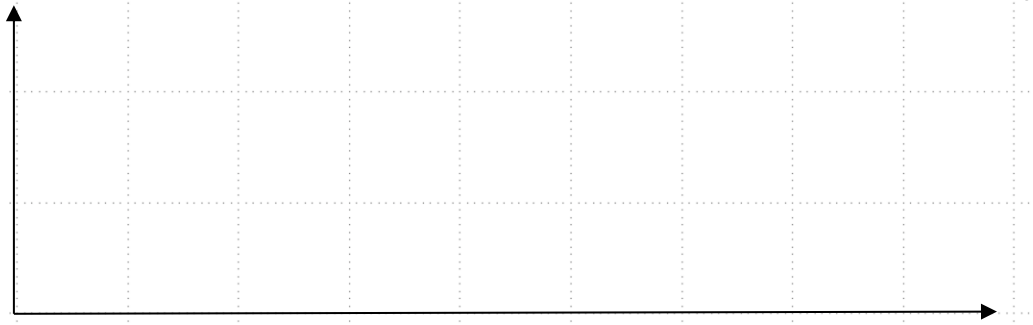
- P_1 is released at time 0 and has two CPU bursts of 40ms and 30ms, separated by a 20ms I/O burst;
- P_2 is released at 10ms and has two 20ms CPU bursts each followed by a 50ms I/O burst, and a last 10ms CPU burst;
- P_3 is released at 20ms and has a single 100ms CPU burst.

Assume that these are the only tasks in the system, and that there is only 1 CPU. The scheduler uses a **multi-level feedback queue**, composed of a high-priority queue (Q1) managed in round-robin with quantum = 30ms, and a low-priority queue (Q2) managed FCFS. New tasks join Q1 and are dispatched as soon as the CPU is available. If they use up all the time slice (30ms) they are demoted and placed in Q2. Tasks that issue an I/O request ~~before the end of their allotted time slice~~ will be placed in Q1 and will resume execution with high priority. Tasks that are interrupted while running with low priority will keep their low priority when resumed.

2L-FQ

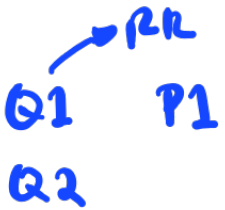


SRTF

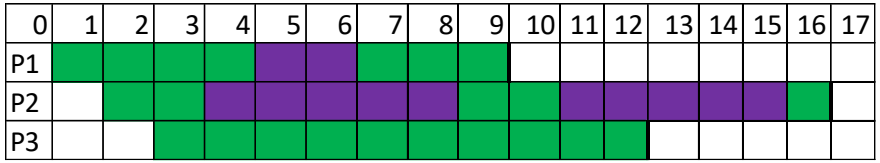


E3.1) Show the schedule using a Gantt chart, indicating the intervals in which tasks are waiting in Q1 or Q2.

E3.2) Compare its performance with that of a **shortest-remaning-time-first** scheduling of the same task set $\{P_1,P_2,P_3\}$.



Exercise (June 12, 2019)



Preemption	Multi-level feedback queue
Q1: RR q3	Use of all q = demotion
Q2: FCFS	New in Q1
	I/O request = promotion

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
P1	Q1	Q1	Q1	Q2	Q2	Q2	Q2	Q2	Q2	Q1	Q1	Q1	Q1	Q1	Q1								
P2		Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1	Q1					
P3			Q1	Q1	Q1	Q1	Q1	Q1	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	Q2	

MLFQ

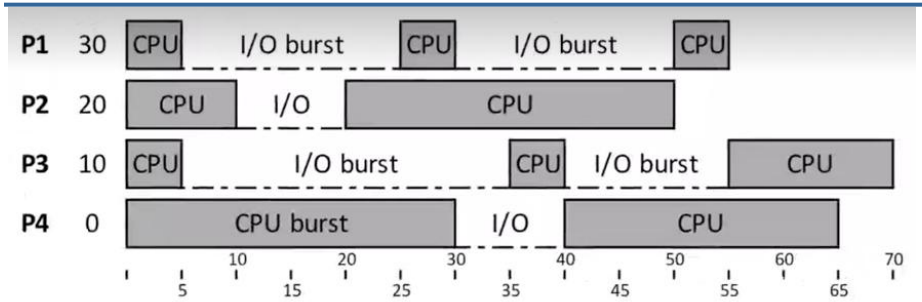
	P1	P2	P3	AVG
Turnaround	15	17	20	17
Waiting	6	2	10	6

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
P1																							
P2																							
P3																							

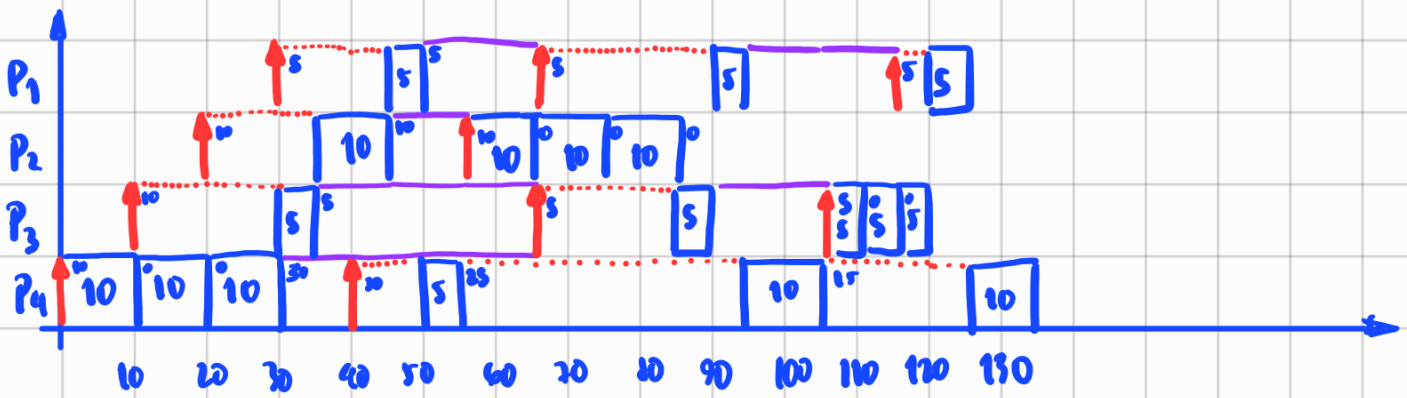
SRTF

	P1	P2	P3	AVG
Turnaround	13	15	20	16
Waiting	4	0	10	5

Possible alternative for EX.2



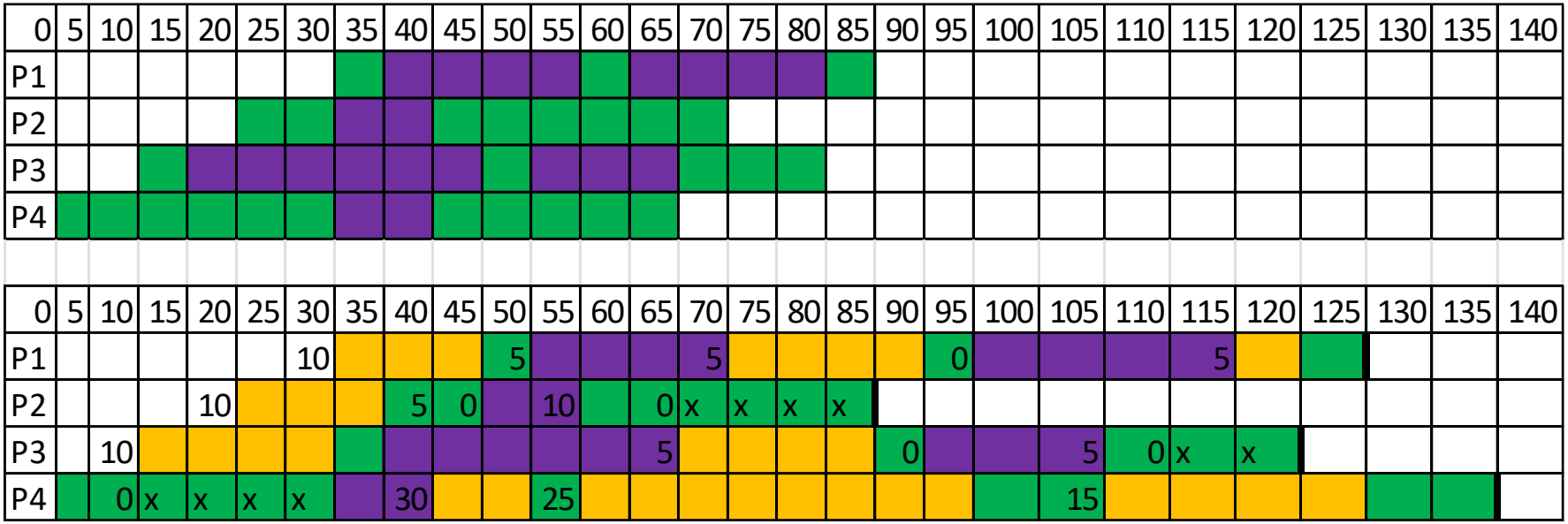
E1.2) Compare with a **Shortest Remaining Time First** scheduling of the same tasks, where the remaining time is **estimated** to be the same as the previous CPU burst of the same process, or 10 ms for each process's first CPU burst.


P1
P2
P3
P4
10
20
30
40
50
60
70
80
90
100
110
120
130

Possible alternative for EX.2

	P1	P2	P3	P4		AVG
Turnaround	95	65	110	135		101.3
Waiting	40	15	40	70		41.3

E1.2) Compare with a **Shortest Remaining Time First** scheduling of the same tasks, where the remaining time is **estimated** to be the same as the previous CPU burst of the same process, or 10 ms for each process's first CPU burst.

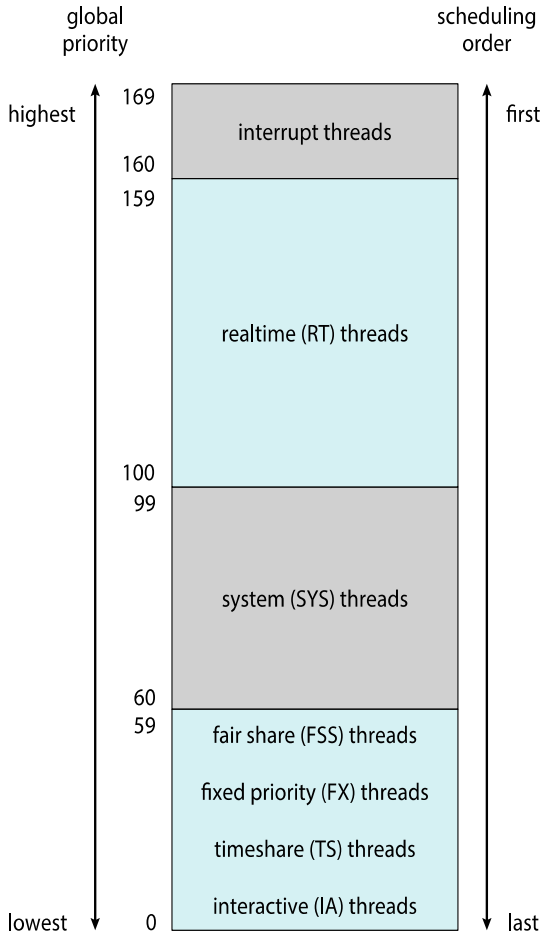


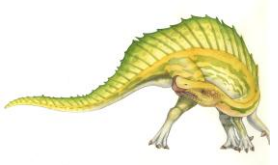
Quizzes

- SJF is a kind of priority scheduling algorithm Yes
- Multilevel feedback queues and FCFS are incompatible False
- FCFS is a kind of priority scheduling algorithm Yes, which one comes first in ready q.
- Multilevel feedback queues cannot implement aging false
- SJF may cause a process' starvation Yes
- Multilevel queue scheduling may cause starvation Yes

Example of OS Scheduling - Solaris

- Scheduler converts class-specific priorities into a per-thread global priority
  - Thread with highest priority runs next
  - Runs until it either
    1. blocks, or
       - May lead to promotion
    2. uses time slice, or
       - May lead to demotion
    3. is preempted by higher-priority thread
  - Multiple threads at same priority selected via RR





Solaris Dispatch Table

low

high

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

Exercise (October 9, 2020)

E1) Consider a task set $\{A,B,C\}$ as illustrated in **Figure 2**, with the following characteristics:

- A: initial priority = 30; release time = 20ms;
  CPU burst = 10ms;
  I/O burst = 20ms;
  CPU burst = 65ms;
- B: initial priority = 20; release time = 10ms;
  CPU burst = 20ms;
  I/O burst = 25ms;
  CPU burst = 10ms;
  I/O burst = 15ms;
  CPU burst = 15ms;
- C: initial priority = 10; release time = 0;
  CPU bursts = 25ms;
  I/O burst = 15ms;
  CPU burst = 10ms;

Assume that these are the only tasks in the system, and that there is only 1 CPU. The scheduler is a priority-based multi-level feedback queue that uses the **dispatch table** in **Figure 1**. Notice that here **higher numbers correspond to higher priorities**. Tasks that are preempted will maintain the same priority until they are rescheduled. Once they are rescheduled they have the unused portion of their quantum. Tasks that make an I/O request will be placed in the queue at the priority indicated in column "return from sleep" at the end of the I/O burst.

E1.1) Show the schedule using a Gantt chart.

E1.2) Compare with a **shortest-remaining-time-first** scheduling of the same tasks.

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
10	160	0	51
20	120	10	52
30	80	20	53
40	40	30	55
41	40	31	56
42	40	32	56
43	40	33	56
50	40	40	58
51	40	41	58
52	40	42	58
53	40	43	58
58	40	48	58

Figure 1: Dispatch table for Exercise E1.

Exercise (October 9, 2020) - Solution

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
10	160	0	51
20	120	10	52
30	80	20	53
40	40	30	55
41	40	31	56
42	40	32	56
43	40	33	56
50	40	40	58
51	40	41	58
52	40	42	58
53	40	43	58
58	40	48	58

MLFQ

	A	B	C	AVG
Turnaround	120	120	170	136.7
Waiting	25	35	120	60.0

Total IDLE: 30

SRTF

	A	B	C	AVG
Turnaround	135	120	50	101.7
Waiting	40	35	0	25.0

Total IDLE: 0