LUT University

LAND OF THE CURIOUS

JANUARY 17, 2023

OPERATING SYSTEMS AND SYSTEMS PROGRAMMING (CT30A3370) 6 CREDITS

Venkata Marella

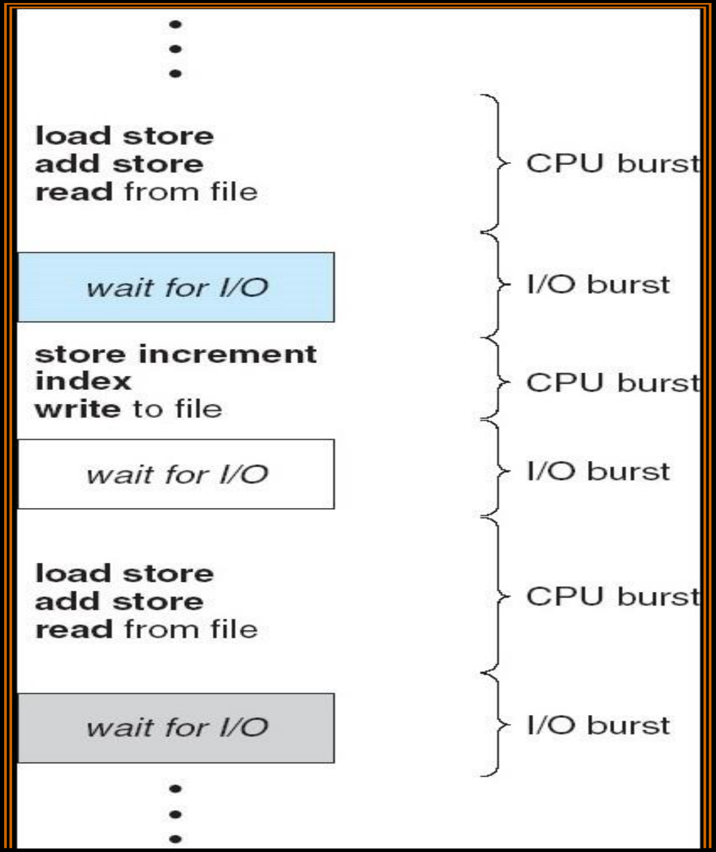
CHAPTER 5: CPU SCHEDULING

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Thread Scheduling
- Operating Systems Examples
- Java Thread Scheduling
- Algorithm Evaluation

BASIC CONCEPTS

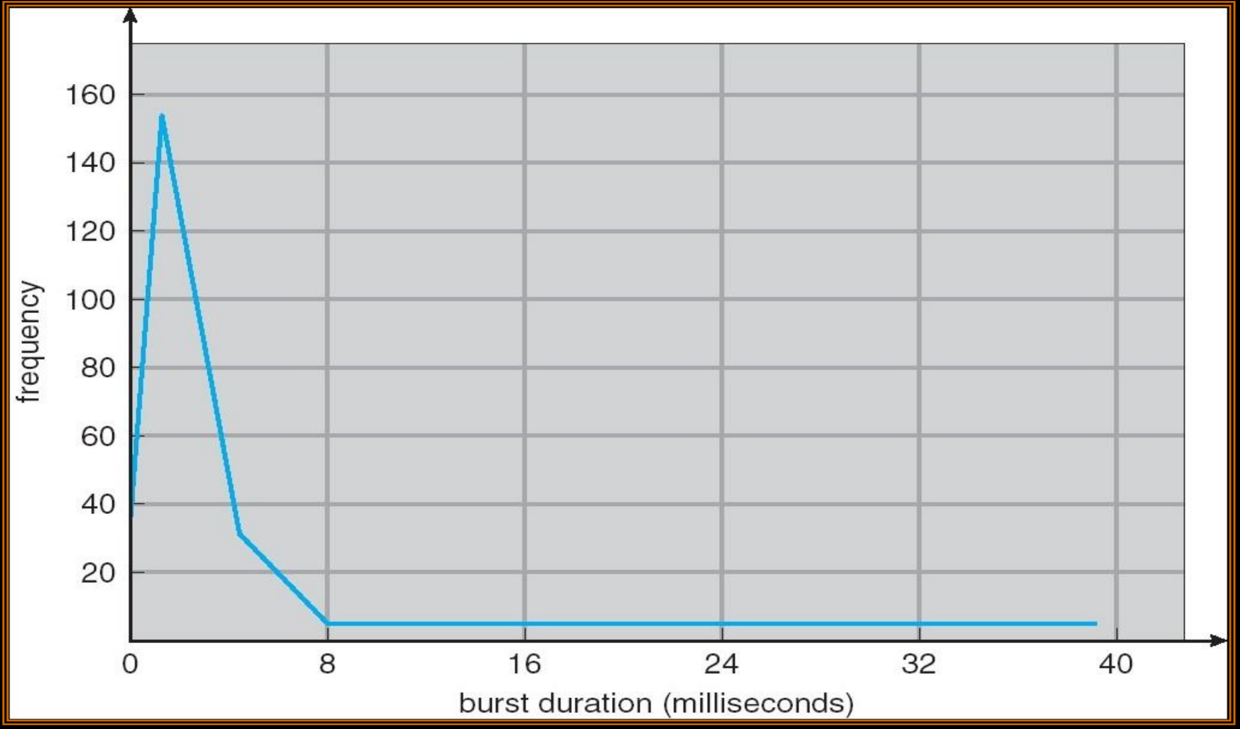
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
- CPU burst distribution

Alternating Sequence of CPU And I/O Bursts



A burst is an instance of breaking

HISTOGRAM OF CPU-BURST TIMES



CPU SCHEDULER

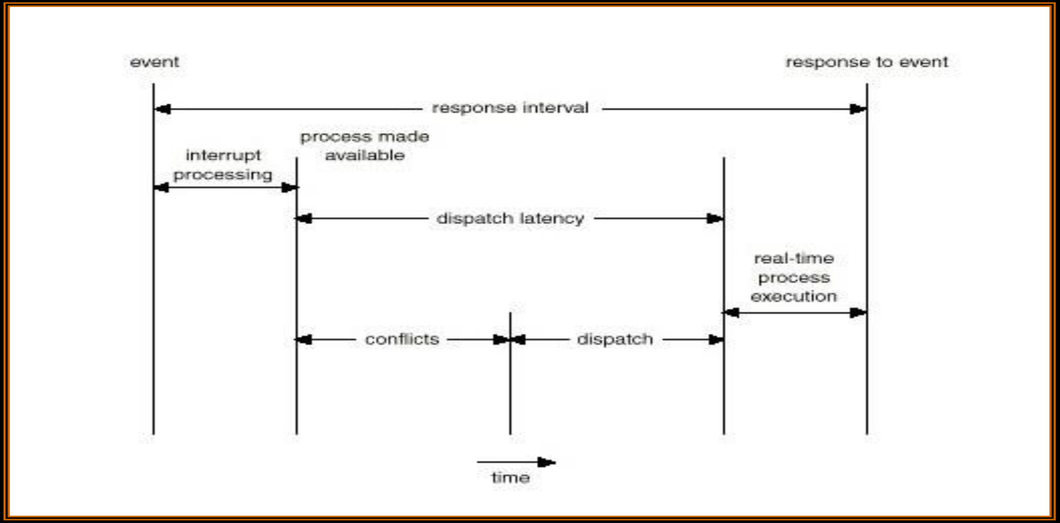
Remember the long-term and short-term schedulers

- Selects from 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
- Scheduling under 1 and 4 is *nonpreemptive*
- All other scheduling is *preemptive*

DISPATCHER

- 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

DISPATCH LATENCY



The conflict phase of dispatch latency contains two parts:

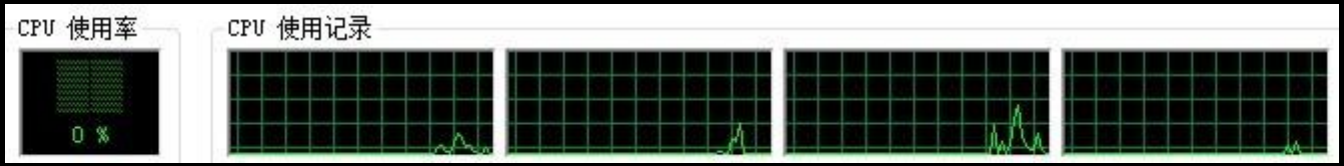
1. Preemption of any process running in the kernel
2. Release resources from old process for the new process

SCHEDULING CRITERIA

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround -- amount of time to execute a particular process
- 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)

OPTIMIZATION CRITERIA

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



FCFS SCHEDULING

- Process Burst Time

Process	Time
P1	24
P2	3
P3	3

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

- Waiting time 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 (I/O-bound processes wait for the CPU-bound one)

SHORTEST-JOB-FIRST (SJF) SCHEDULING

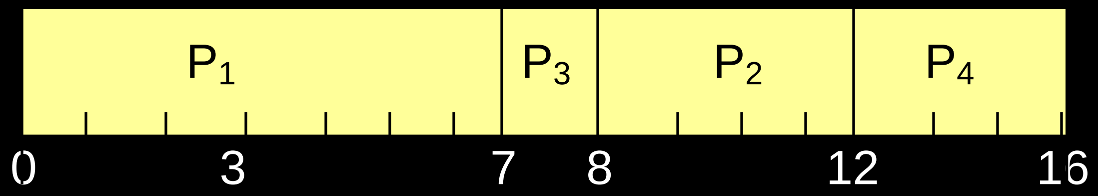
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Why is it optimal?

EXAMPLE OF NON-PREEMPTIVE SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

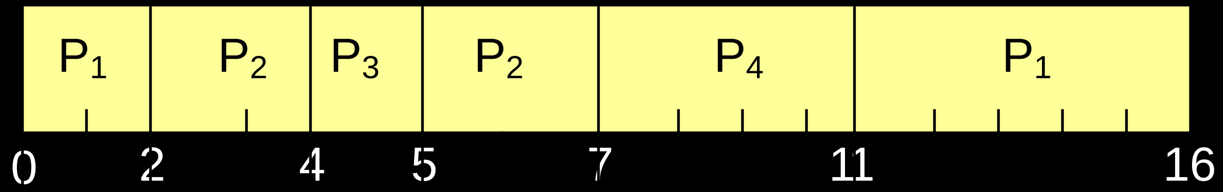


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

EXAMPLE OF PREEMPTIVE SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



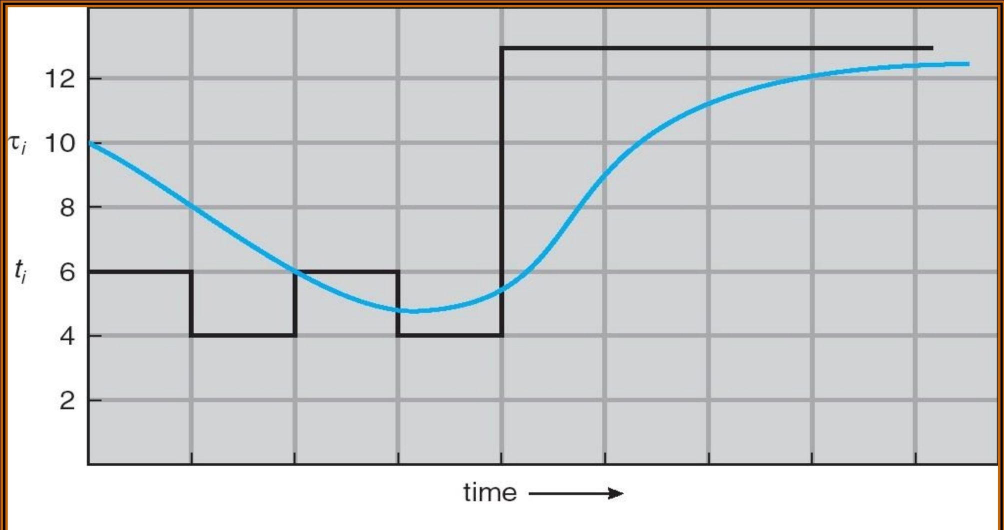
- Average waiting time = (9 + 1 + 0 +2)/4 = 3

DETERMINING LENGTH OF NEXT CPU BURST

- Unfortunately, no way to know the length of the next 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. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define: $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$

PREDICTION OF THE LENGTH OF THE NEXT CPU BURST



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

EXAMPLES OF EXPONENTIAL AVERAGING

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

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

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

PRIORITY SCHEDULING

- 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
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

ROUND ROBIN (RR)

- Each process gets a small unit of CPU time (*time quantum*), 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.
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high
- *Question: What are the waiting times of RR?*

EXAMPLE OF RR WITH TIME QUANTUM = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

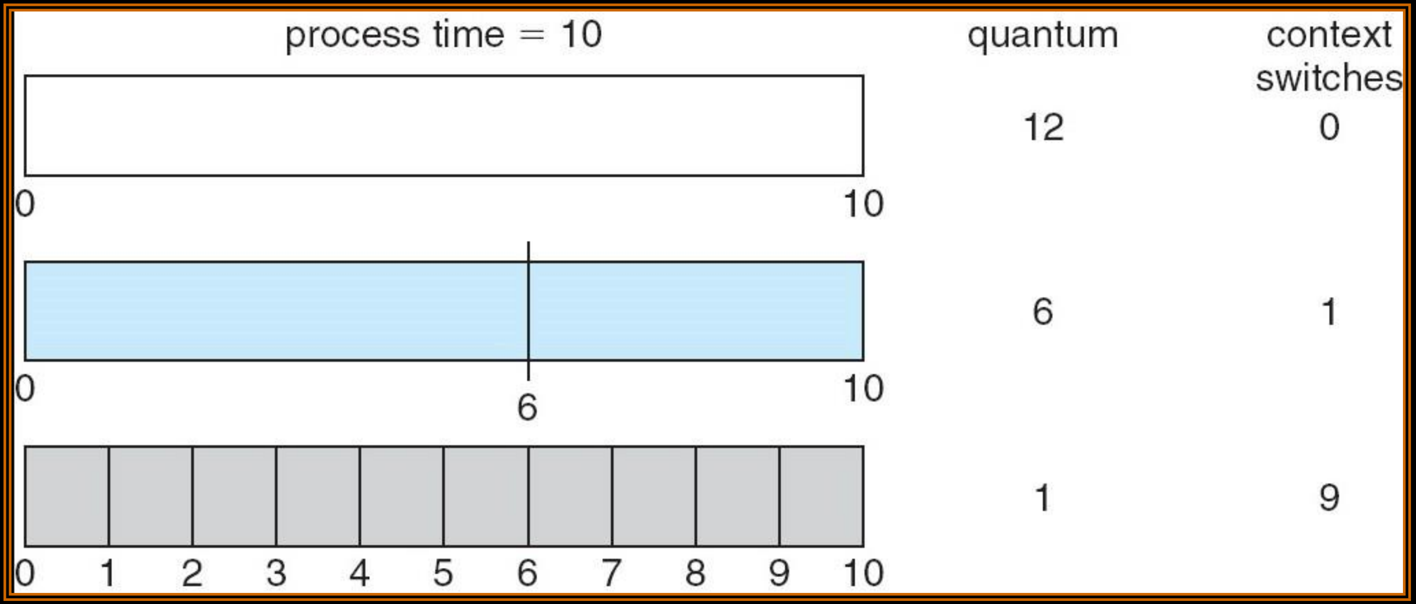
- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

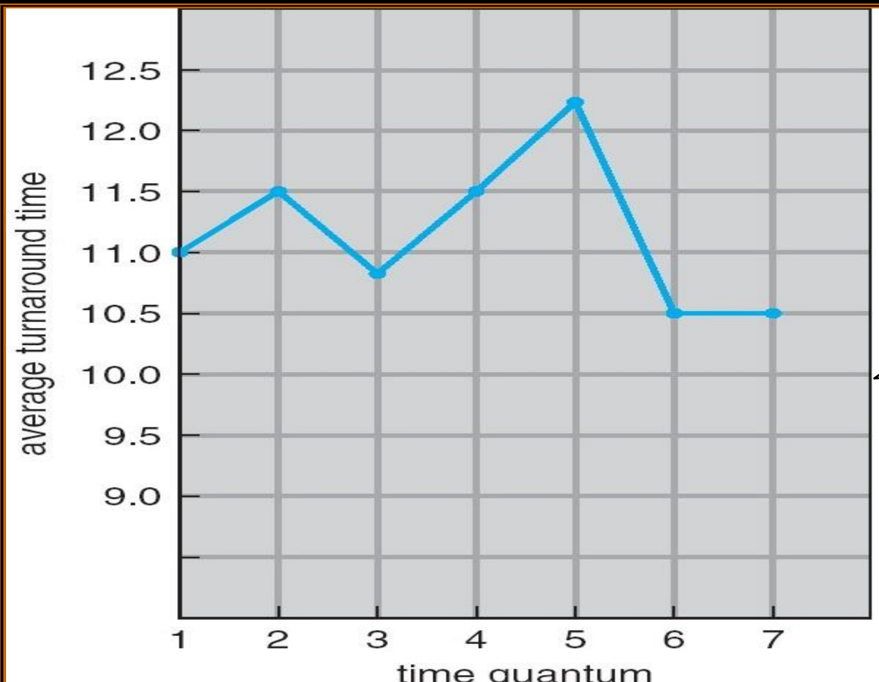
0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average turnaround than SJF, but better *response*

TIME QUANTUM AND CONTEXT SWITCH TIME



TURNAROUND TIME VARIES WITH THE TIME QUANTUM



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Question: what's the turnaround time for SJF?

(1+4+11+18) /4 = 8.5

AN EXAMPLE

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

For the FCFS scheduling algorithm, the **average waiting time** is :
For the SJF scheduling algorithm, the average waiting time is :
For the Round Robin (quantum is 2) scheduling algorithm, the average turnaround time is:

AN EXAMPLE

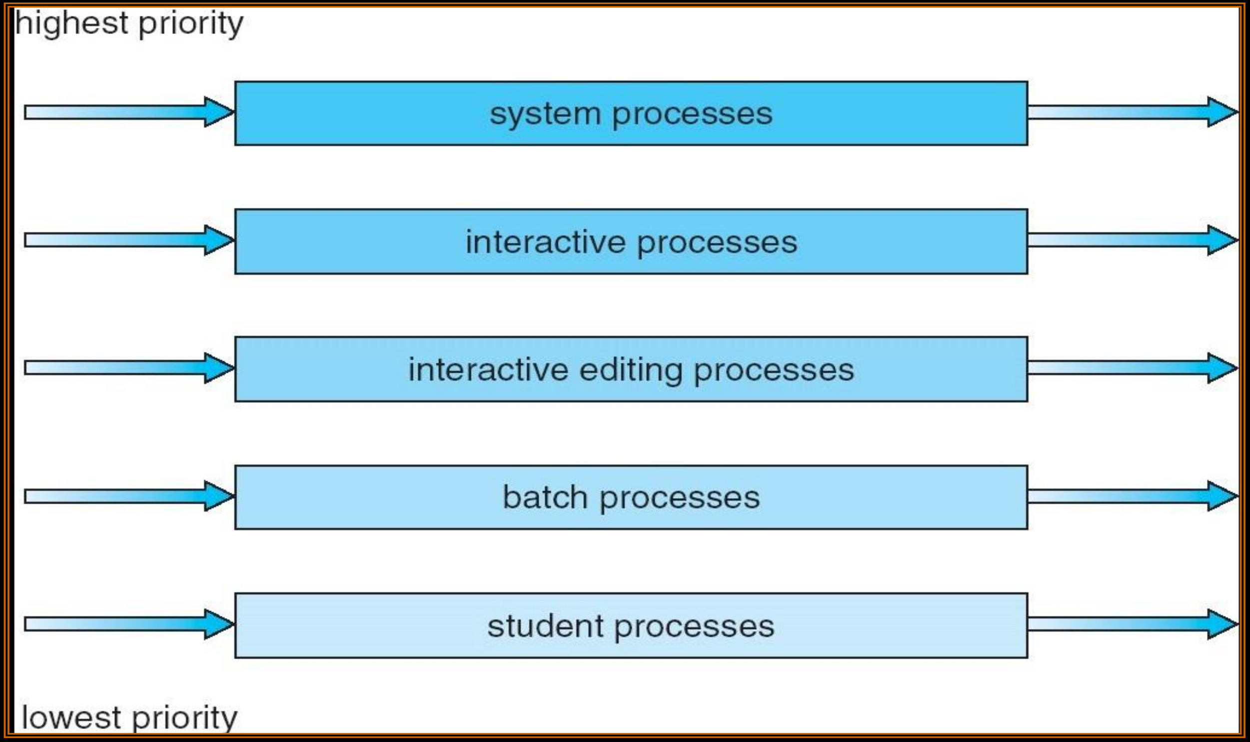
Process	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	5
P4	3	3

For the FCFS scheduling algorithm, the **average waiting time** is : 31/4
For the SJF scheduling algorithm, the average waiting time is : 28/4
For the Round Robin (quantum is 2) scheduling algorithm, the average turnaround time is: 59/4

MULTILEVEL QUEUE

- Ready queue is partitioned into separate queues:
  foreground (interactive)
  background (batch)
- 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; i.e., 80% to foreground in RR
  - 20% to background in FCFS

MULTILEVEL QUEUE SCHEDULING



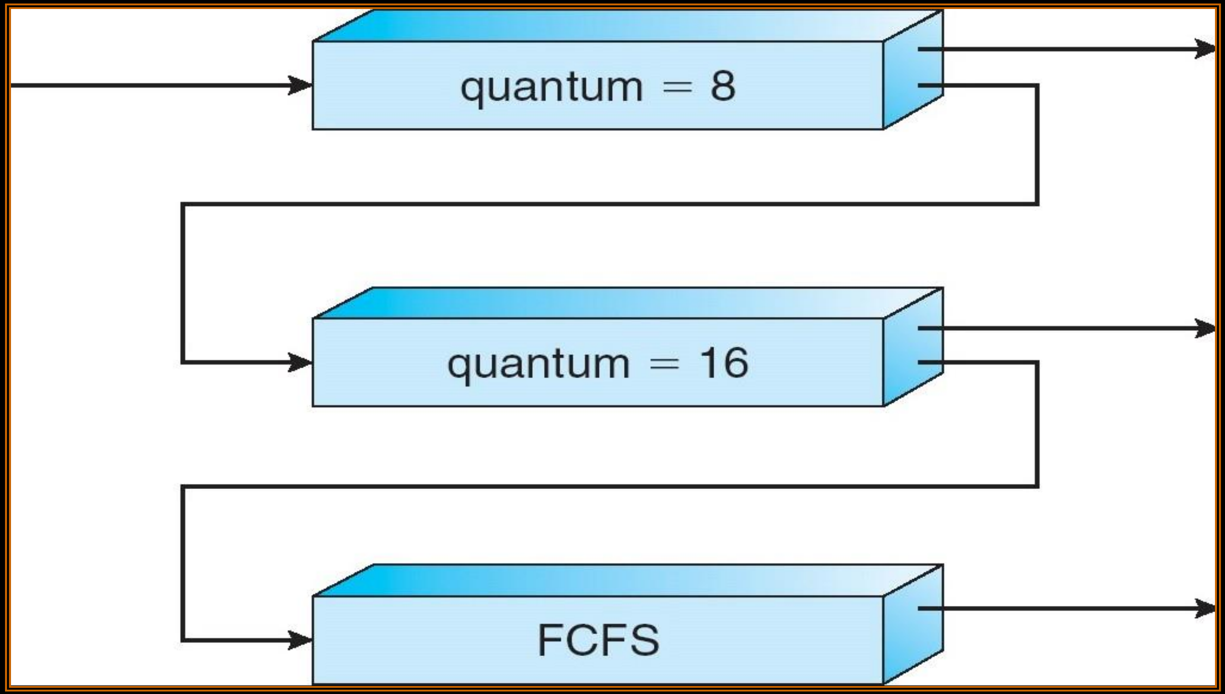
MULTILEVEL FEEDBACK QUEUE

- 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

EXAMPLE OF MULTILEVEL FEEDBACK QUEUE

- 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 preempted and moved to queue Q_2.

MULTILEVEL FEEDBACK QUEUES



MULTIPLE-PROCESSOR SCHEDULING

- CPU scheduling more complex when multiple CPUs are available
- *Homogeneous processors* within a multiprocessor
- *Load balancing*
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing; others execute only user code.
- *Symmetric multiprocessing* (SMP) – each processor is self-scheduling. Multiple processors might access and update a common data structure.

REAL-TIME SCHEDULING

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones
- *Hard* – missing a deadline is a total system failure.
- *Firm* – infrequent deadline misses are tolerable, but may degrade the system's quality of service. The usefulness of a result is zero after its deadline.
- *Soft* – the usefulness of a result degrades after its deadline, thereby degrading the system's quality of service.

THREAD SCHEDULING

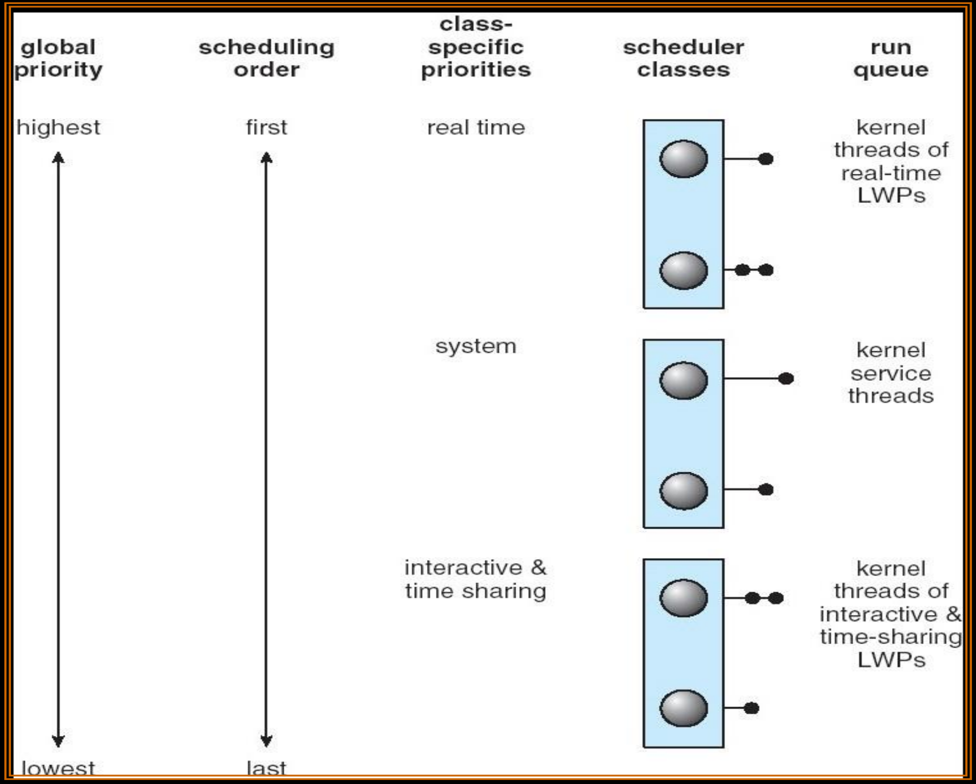
- Local Scheduling / Process Contention Scope– How the threads library decides which thread to put onto an available LWP (not run yet)

- Global Scheduling / System Contention Scope – How the kernel decides which kernel thread to run next

OPERATING SYSTEM EXAMPLES

- Solaris scheduling
- Linux scheduling

SOLARIS 2 SCHEDULING



SOLARIS DISPATCH TABLE

priority	time quantum	time quantum expired	return from sleep
0 low	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	[illegible]
40	40	30	[illegible]
45	40	35	[illegible]
50	40	40	58
55	40	45	58
59 high	20	49	59

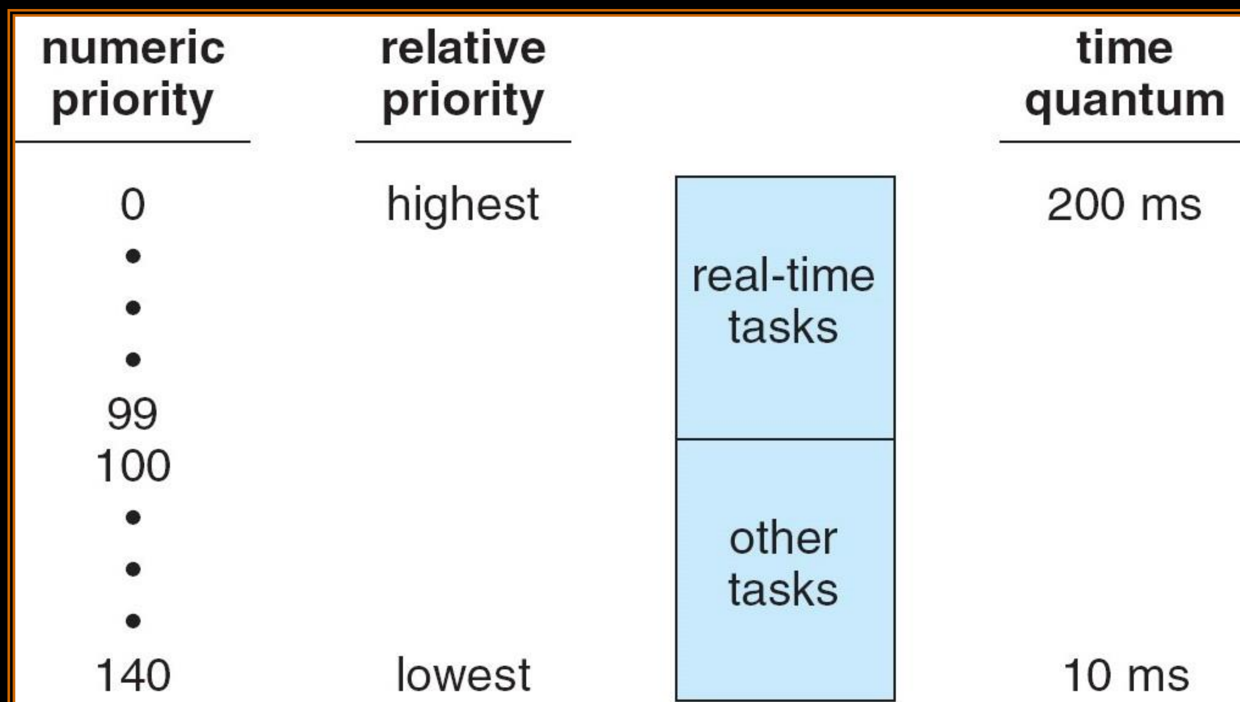
Higher priority for better interactivity

Priority lowered when quantum expired.

LINUX SCHEDULING

- Two algorithms: time-sharing and real-time
- Time-sharing
  - Prioritized credit-based – process with most credits is scheduled next
  - Credit subtracted when timer interrupt occurs
  - When credit = 0, another process chosen
  - When all processes have credit = 0, recrediting occurs
    - Based on factors including priority and history
- Real-time
  - Soft real-time
  - Posix.1b compliant – two classes
    - FCFS and RR
    - Highest priority process always runs first

THE RELATIONSHIP BETWEEN PRIORITIES AND TIME-SLICE LENGTH



ALGORITHM EVALUATION

- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Queueing models

  Little's Law: $n = \lambda \cdot W$

Thank you

LUT University