Process Scheduling

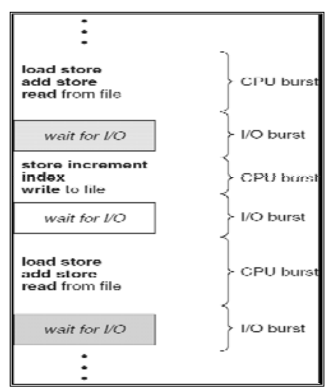
Contents

- Basic concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Linux Scheduling
- Algorithm Evaluation
  - Deterministic Modeling
  - Queueing Models

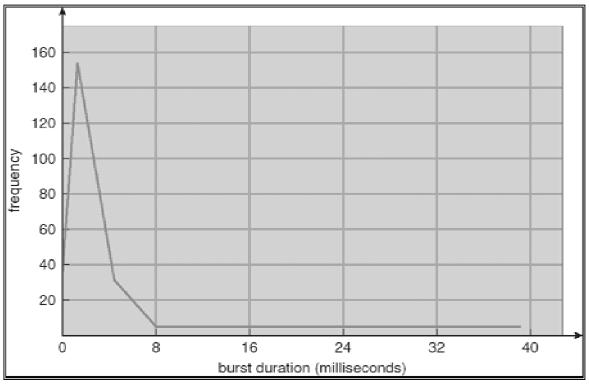
Basic Concepts

- One of the main aims of an OS is to maximize the utilization of the CPU by switching it between processes
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle: Process execution consists of a *cycle* of CPU execution (CPU Burst) and I/O wait (I/O Burst)

CPU and I/O Burst Cycle



CPU Burst Duration (ms)



Histogram of CPU-burst duration

What is a Process?

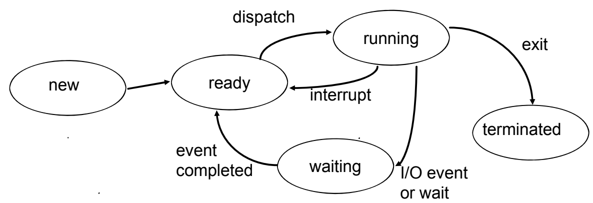


Diagram of process state

Scheduling Opportunities

- When the running process yields the CPU:
  - the process enters a wait state
  - the process terminates
- When an interrupt occurs:
  - process switches from running state to ready state
  - process switches from wait state to ready state
- Preemptive vs. non-preemptive scheduling

CPU Scheduler

- 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 state to waiting state
  2. Switches from running state to ready state
  3. Switches from waiting state to ready state
  4. Terminates
- Scheduling under 1 and 4 is *nonpreemptive; otherwise it* is *preemptive*
- *Under nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to waiting state*

Dispatcher

- Dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- The dispatcher should as fast as possible, Since it is invoked during every process switch
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running – overhead

Scheduling Criteria & Optimization

- CPU utilization – keep the CPU as busy as possible
  - Maximize CPU utilization
- Throughput – # of processes that complete their execution per time unit
  - Maximize throughput
- Turnaround time – amount of time to execute a particular process
  - Minimize turnaround time
- Waiting time – amount of time a process has been waiting in the ready queue
  - Minimize waiting time
- Response time – time from the submission of a request until the first response is produced (response time, is the time it takes to start responding, not the time it takes to output the response)
  - Minimize response time

What is Important in a Scheduling Algorithm?

- Minimize Response Time
- Maximize Throughput
- Fairness
  - Share CPU among users in some equitable way
  - Not just minimizing average response time

Scheduling Algorithms

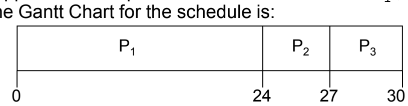
1. First-Come, First-Served (FCFS)
2. Shortest Job First (SJF)
3. Priority Scheduling
4. Round Robin (RR)
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

First-Come, First-Served (FCFS) Scheduling

- The process that requests the CPU first is allocated the CPU first
- Consider 3 processes arrive at time 0, in the given order, with length of CPU burst given in msecs:

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
  The Gantt Chart for the schedule is:



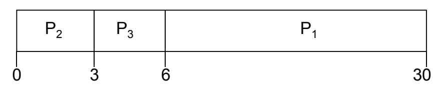
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

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

FCFS Pros and Cons

- Simple
- May not give the best average waiting time.
- Potentially bad for short jobs!
- Performance is highly dependent on the order in which jobs arrive
- *Convoy effect*
  - Short processes get stuck behind long process
- FCFS is nonpreemptive

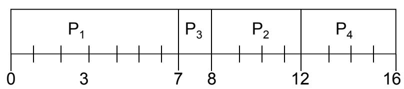
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie
- 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 know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

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)

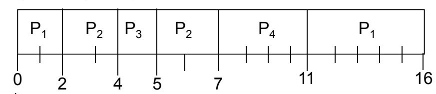


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

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)

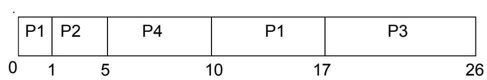


- Average waiting time = (9 + 1 + 0 +2)/4 = 3

Example of Preventive SJF

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

- Gantt Chart:



- Average waiting time:

  ((10-1) + (1-1) + (17-2) + (5-3))/4
  = 6.5 ms

  start time arrival time

- Non-preemptive SJF gives 7.75 ms

SJF Pros and Cons

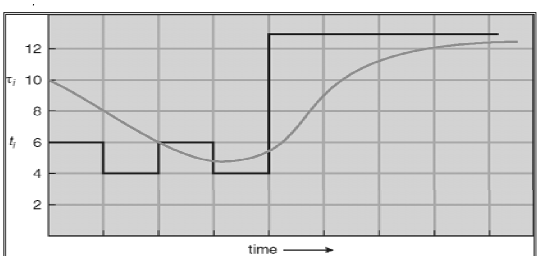
- Provably optimal
  - gives the minimum average waiting time
- Favors short jobs over long
- Starvation
  - If there are many small jobs, large jobs never executed
- *Problem*: it is usually impossible to know the next CPU burst duration for a process
  - solution: guess (predict)

Predicting the next CPU burst time

- The next CPU burst is generally predicted as an exponential average of the measured lengths of previous CPU bursts
- We define the exponential average with following formula:

  $\Gamma_{n+1} = \alpha\, t_n + (1-\alpha)\, \Gamma_n$
- Where,
  - t_n be the actual value of the n^{th} CPU burst
  - Γ_{n+1} be the predicted value for the next ($n+1^{th}$) CPU burst
  - Γ_n be the previous prediction (based on the sequence of old t_i times)
  - α ($0 \le \alpha \le 1$) controls the relative weight of recent and past history; usually $\alpha = 1/2$

Predicting 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 …

Above Fig shows an exponential average with $\alpha = \frac{1}{2}$ and $\Gamma_0 = 10$

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 $\alpha = 1/2$
  - Recent history and past history are equally weighted
- 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

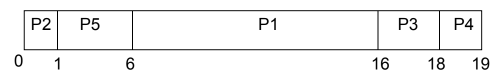
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest number = highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem: Starvation or indefinite blocking – low priority processes may never execute
- Solution: Aging – increase the priority of the process that wait for long time

Example of Priority Scheduling

- Consider 5 processes arrive at time 0, in the given order, with length of CPU burst given in msecs:
-

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

- Gantt Chart:



- Average waiting time: 8.2 ms

Priority Scheduling Pros and cons

- Internal/external priorities.
- Preemptive or non-preemptive.
- How to avoid starvation?
  - aging

Round Robin (RR)

- Preemptive version of FCFS
- Each process gets a small unit of CPU time (time quantum)
  - Usually 10-100 ms
- After quantum expires, the process is preempted and added to the end of the ready queue
- Suppose n processes in ready queue and time quantum is q ms:
  - Each process gets 1/n of the CPU time in chunks of at most q ms
  - What is the maximum wait time for each process?
    - No process waits more than $(n-1)q$ time units.
- Performance
  - q large $\Rightarrow$ RR is same as FCFS
  - q small (1ms) $\Rightarrow$ q must be large with respect to context switch time, otherwise overhead is too high (spending most of your time context switching!)

Example of RR with Time Quantum = 4

Consider 3 processes arrive at time 0, in the given order, with length of CPU burst given in msecs:

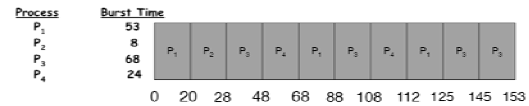
Process	Burst Time
P_1	24
P_2	3
P_3	3

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

- Waiting Time:
  - P1: (10-4) = 6
  - P2: (4-0) = 4
  - P3: (7-0) = 7
- Completion Time (turnaround time):
  - P1: 30
  - P2: 7
  - P3: 10
- Average Waiting Time: (6 + 4 + 7)/3= 5.67
- Average Completion Time: (30+7+10)/3=15.67

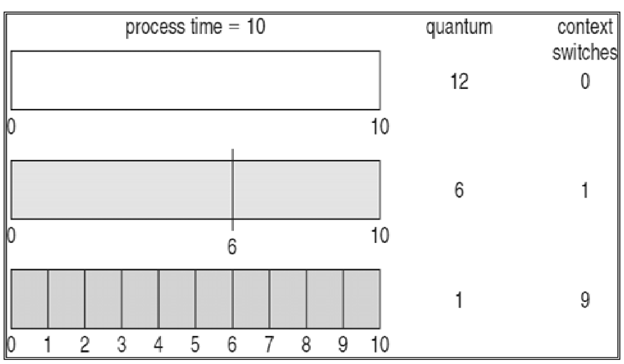
Example of RR with Time Quantum = 20



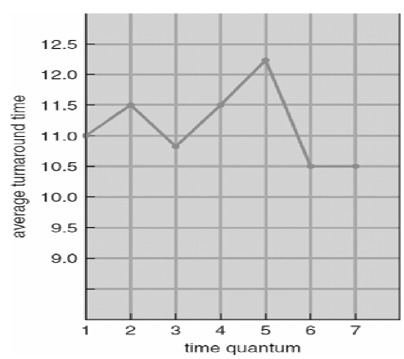
A process can finish before the time quantum expires, and release the CPU.

- Waiting Time:
  - P1: (68-20)+(112-88) = 72
  - P2: (20-0) = 20
  - P3: (28-0)+(88-48)+(125-108) = 85
  - P4: (48-0)+(108-68) = 88
- Completion Time:
  - P1: 125
  - P2: 28
  - P3: 153
  - P4: 112
- Average Waiting Time: (72+20+85+88)/4 = 66.25
- Average Completion Time: (125+28+153+112)/4 = 104.5

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

RR Pros and Cons

- Better for short jobs
- Average waiting time can be quite long.
- Context switching is an important overhead when the time quantum is small.

continued

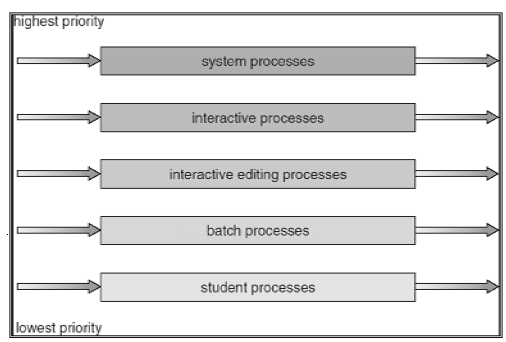
Multilevel Queue (MQ)

- MQ scheduling is used when processes can be classified into groups
  - For example, **foreground** (interactive) processes and **background** (batch) processes
- A MQ scheduling algorithm partitions the ready queue into several separate queues:
  - foreground (interactive)
  - background (batch)

Multilevel Queue (MQ)

- Each process assigned to one queue based on its memory size, process priority, or process type.
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Danger of starvation.
  - Time slice – each queue gets a certain portion 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

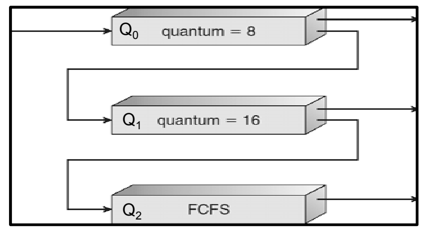


Multilevel Feedback Queue (MFQ)

- In MQ scheduling algorithm processes do not move from one queue to the other
  - MQ has low scheduling overhead, but it is inflexible
- In contrast, the MFQ scheduling algorithm, a process can move between the queues;
- It prevents starvation
- MFQ 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

- Consider multilevel feedback queue scheduler with three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A process queue in Q_0 *is given a time quantum of* 8 milliseconds. If it does not finish in 8 milliseconds, the job is moved to the tail of queue Q_1.
  - When Q_0 is empty, the process at the head of Q_1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and moved to queue Q_2 .



Techniques for Algorithm Evaluation

- Deterministic Modeling (analytic evaluation)
- Queueing Models

Deterministic Modeling

- Deterministic Modeling - takes a particular predetermined workload and defines the performance of each algorithm for that workload
  - FCFS, SJF, and RR (quantum = 10 ms)

Example of Deterministic Modeling

- Consider 5 processes arrive at time 0, in the given order, with length of CPU burst given in msecs:

Process	Burst Time
P1	10
P2	29
P3	3
P4	7
P5	12

Deterministic Modeling: Using FCFS scheduling

- FCFS:

P1	P2	P3	P4	P5	
0	10	39	42	49	61

- Average waiting time:

  (0 + 10 + 39 + 42 + 49)/5
  = 28 ms

Deterministic Modeling: Using nonpreemptive SJF scheduling

- Non-preemptive SJF:

P3	P4	P1	P5	P2	
0	3	10	20	32	61

- Average waiting time:

  (10 + 32 + 0 + 3 + 20)/5
  = 13 ms

Deterministic Modeling: Using RR scheduling

- RR: (Quantum 10 msecs)

P1	P2	P3	P4	P5	P2	P5	P2	
0	10	20	23	30	40	50	52	61

- Average waiting time:

  (0 + 32 + 20 + 23 + 40)/5
  = 23 ms

Results

- For this mix of processes and CPU burst durations:
  - SJF 13 ms
  - RR 23 ms
  - FCFS 28 ms
- SJF policy will always result in the minimum waiting time

Deterministic Modeling Features

- Simple and fast to calculate.
- Requires exact numbers.
- Limited generality, but with enough cases it may reveal some trends.

Queueing Models

- Queueing network analysis
  - The computer system is described as a network of servers
  - Each server has a queue of waiting processes
  - The CPU is a server with its ready queue, so is the I/O system with its device queues
  - Knowing the arrival rates and service rates, we can compute utilization, average queue length, average wait time, etc.

Queueing Models

- Let **n** be the average length of the queue (excluding the process being serviced), let **W** be the average waiting time in the queue, and let **λ** be the average arrival rate for the new processes in the queue (such as three processes per second)
- We expect that during the time W that a process waits, λ x W new processes will arrive in the queue
- If the system is in a steady state, then the number of processes leaving the queue must be equal to the number of processes that arrive
  - Little's formula **(*n* = λ x W)**
  - Formula is valid for any scheduling algorithm or arrival distribution
  - It can be used to compute one of the variables given the other two
  - Example
    - 7 processes arrive on the average of every second
    - There are normally 14 processes in the queue
    - Therefore, the average waiting time per process is 2 seconds

Queueing Model Limitations

- Queueing models are often only approximations of real systems
  - The classes of algorithms and distributions that can be handled are fairly limited
  - The mathematics of complicated algorithms and distributions can be difficult to work with
  - Arrival and service distributions are often defined in mathematically tractable –but unrealistic-ways

Thread Scheduling

- Distinction between user-level and kernel-level threads
- On system implementing the many-to-one and many-to-many models, the thread library schedules user-level threads to run on LWP
  - A scheme known as **process-contention scope (PCS)** since scheduling competition is within the process
- To decide which kernel thread to schedule onto CPU, the kernel uses **system-contention scope (SCS)** – competition among all threads in system

Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
  - PTHREAD SCOPE PROCESS schedules threads using PCS scheduling
  - PTHREAD SCOPE SYSTEM schedules threads using SCS scheduling.

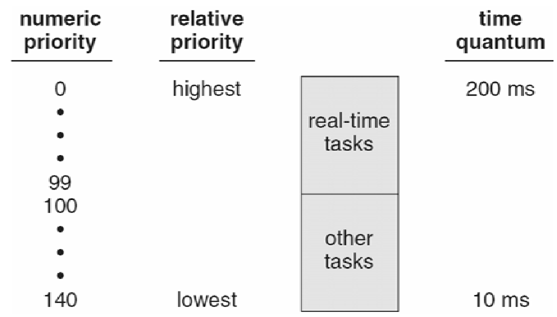
Operating System Example

- Linux scheduling

Linux Scheduling

- The Linux scheduler is a preemptive, priority-based algorithm with two separate priority ranges: time-sharing and real-time
- **Real-time** range from 0 to 99 and **time-sharing (nice) value ranging** from 100 to 140
- The scheduling algorithm that runs in constant time *O*(1)

Priorities and Time-slice length



List of Tasks Indexed According to Priorities