CPU scheduling

Pertemuan 9 dan 10

Kompetensi Khusus

- Mahasiswa mampu menggunakan algoritma penjadwalan pada prosesor untuk megeksekusi sebuah thread (C3, A2)(C1)

Materi

1. Basic Concepts
2. Scheduling Criteria
3. Scheduling Algorithms
4. Thread Scheduling
5. Multi-Processor Scheduling
6. Real-Time CPU Scheduling
7. Algorithm Evaluation

1. Basic Concepts

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

1.1 Alternating Sequence of CPU And I/O Bursts

Process execution begins with a CPU burst.
That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution

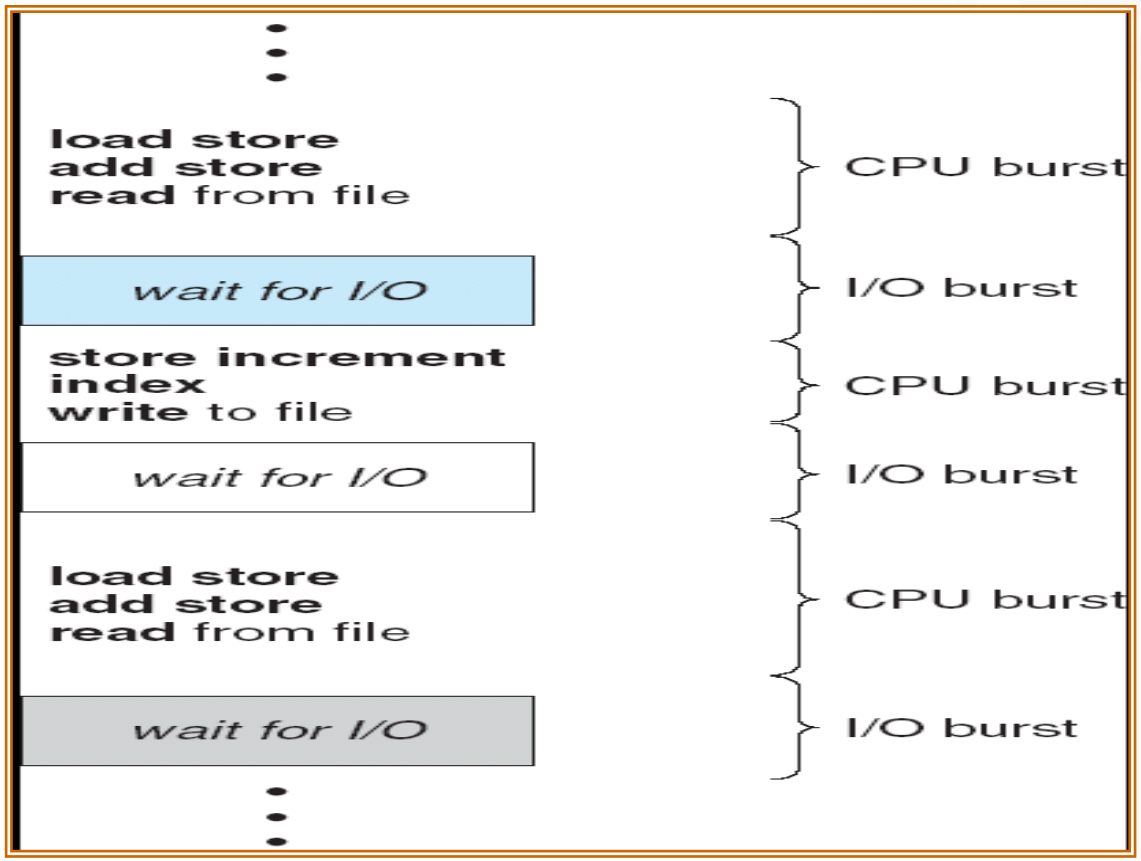


Fig. Alternating sequence of CPU and I/O bursts

1.1.1 Histogram of CPU-burst Times

The durations of CPU bursts have been measured extensively. Although they vary greatly from process to process and from computer to computer, they tend to have a frequency curve similar

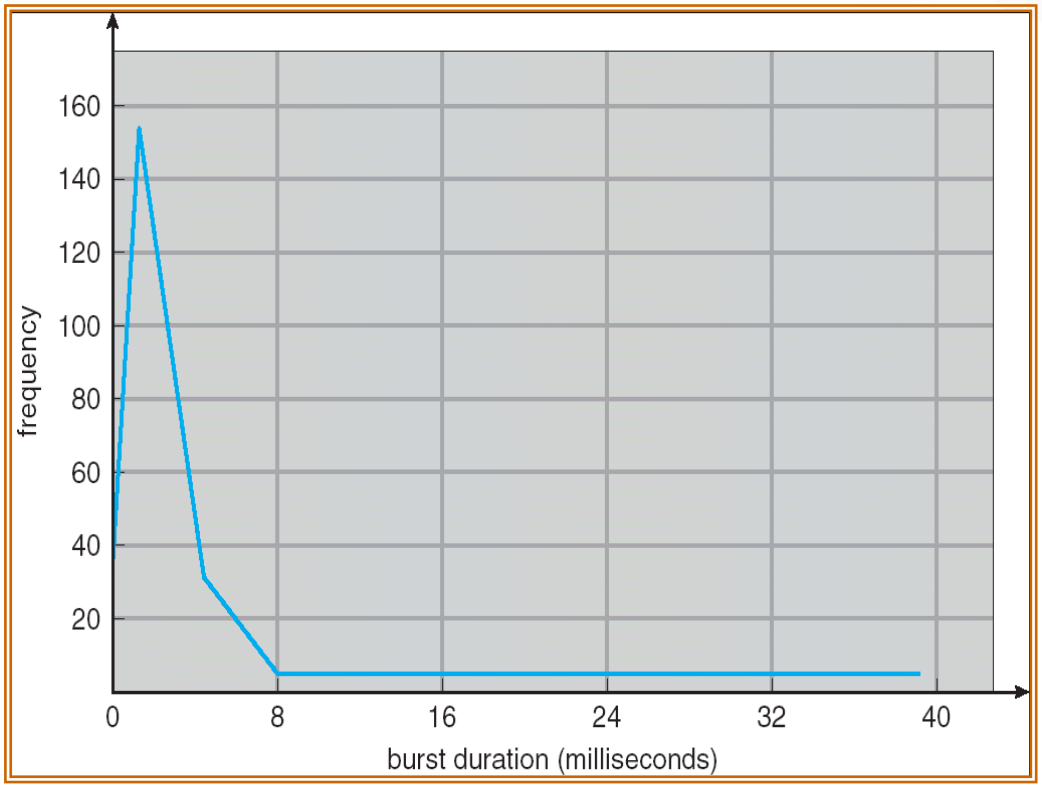


Fig. Histogram of CPU-burst durations.

1.2 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 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

1.3 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

2. Scheduling Criteria

2.1 Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – 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)

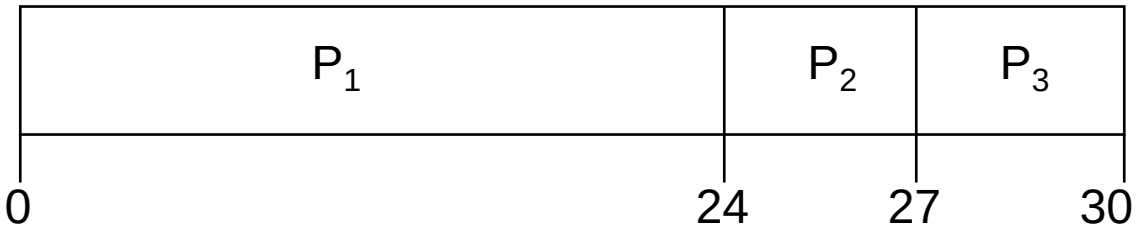
2.2 Optimization Criteria

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

2.3 First-Come, First-Served (FCFS) Scheduling

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:



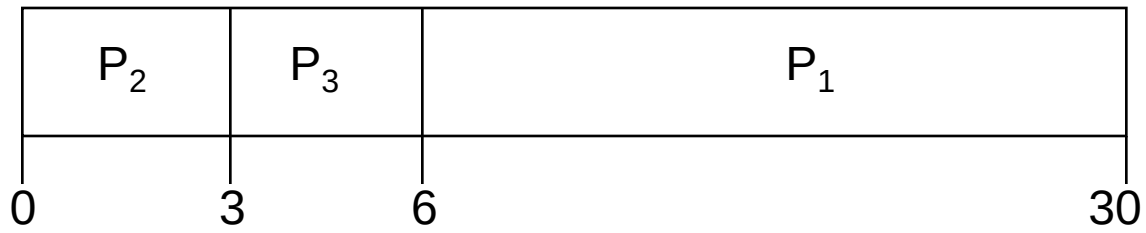
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

2.3 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

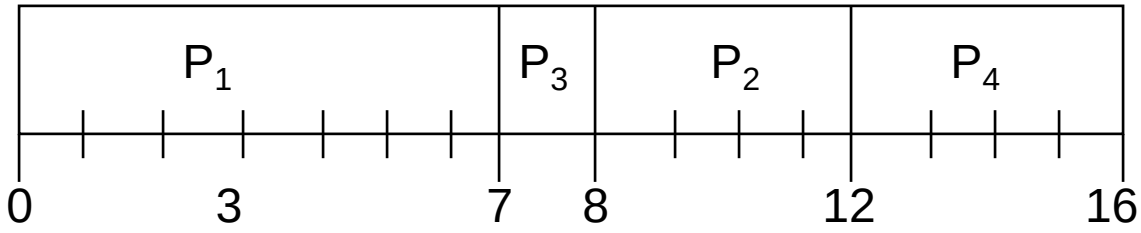
2.4 Shortest-Job-First (SJF) Scheduling

- 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 know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

2.4.1 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)

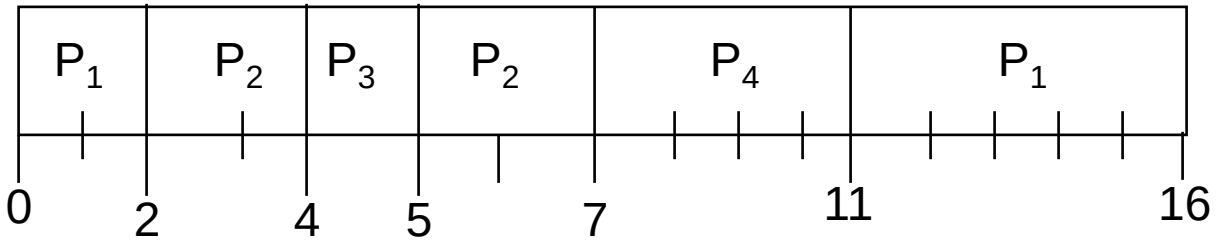


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

2.4.2 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

2.5 Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

2.5.1 Prediction of the Length of the Next CPU Burst

Formula defines an exponential average with $\alpha = 1/2$ and $t_0 = 10$.

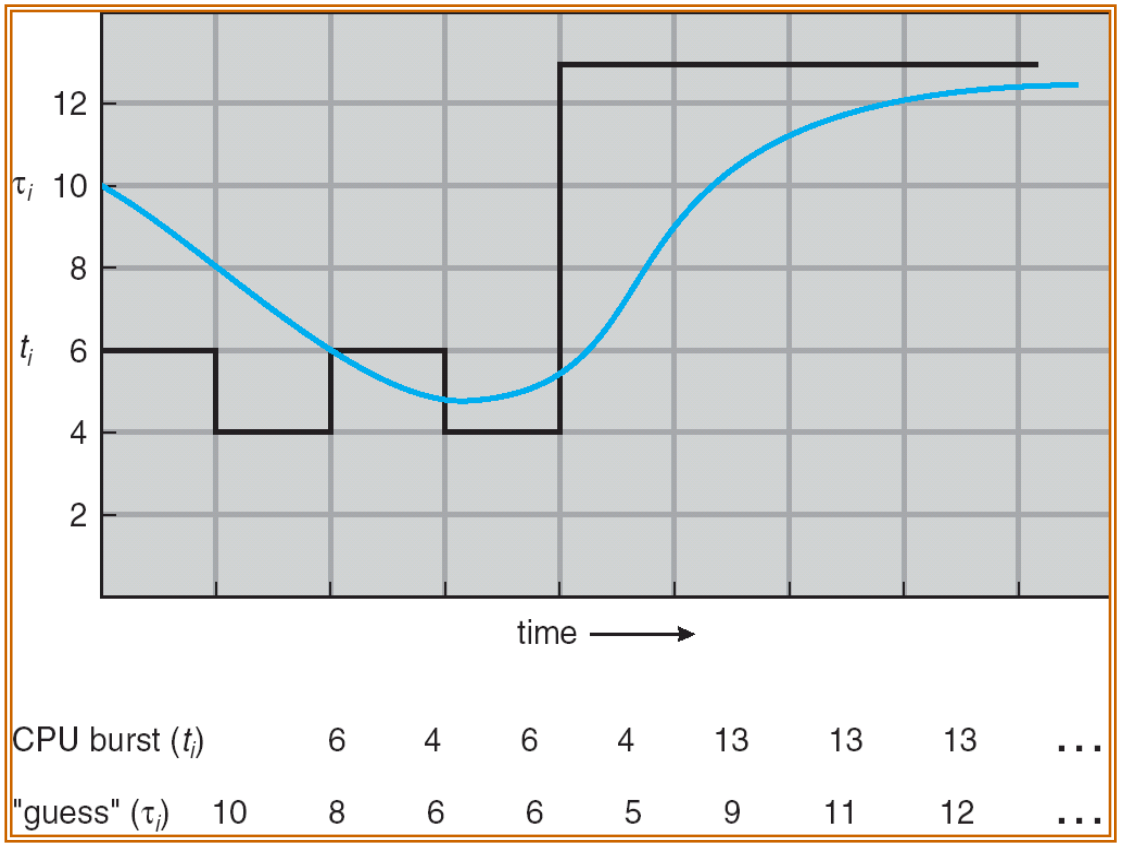


Fig. Prediction of the length of the next CPU burst

2.5.2 Examples of Exponential Averaging

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = \alpha\, t_n$
  - 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

2.6 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

3. Scheduling Algorithms

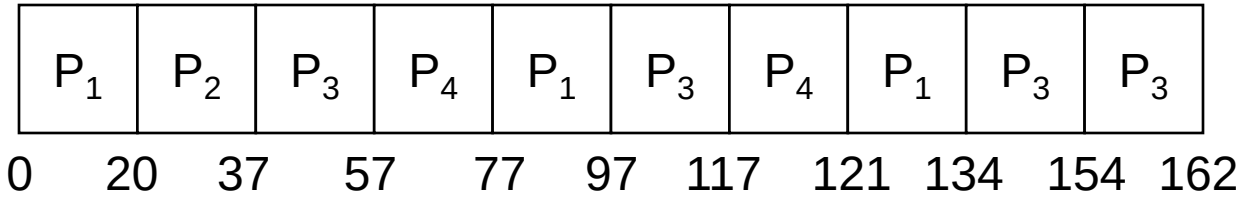
3.1 Round Robin (RR)

- 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 --> FIFO
  - q small -> q must be large with respect to context switch, otherwise overhead is too high

3.1.1 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:



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

3.1.2 Time Quantum and Context Switch Time

In software, we need also to consider the effect of context switching on the performance of RR scheduling. Let us assume that we have only one process of 10 time units. If the quantum is 12 time units, the process finishes in less than 1 time quantum, with no overhead.

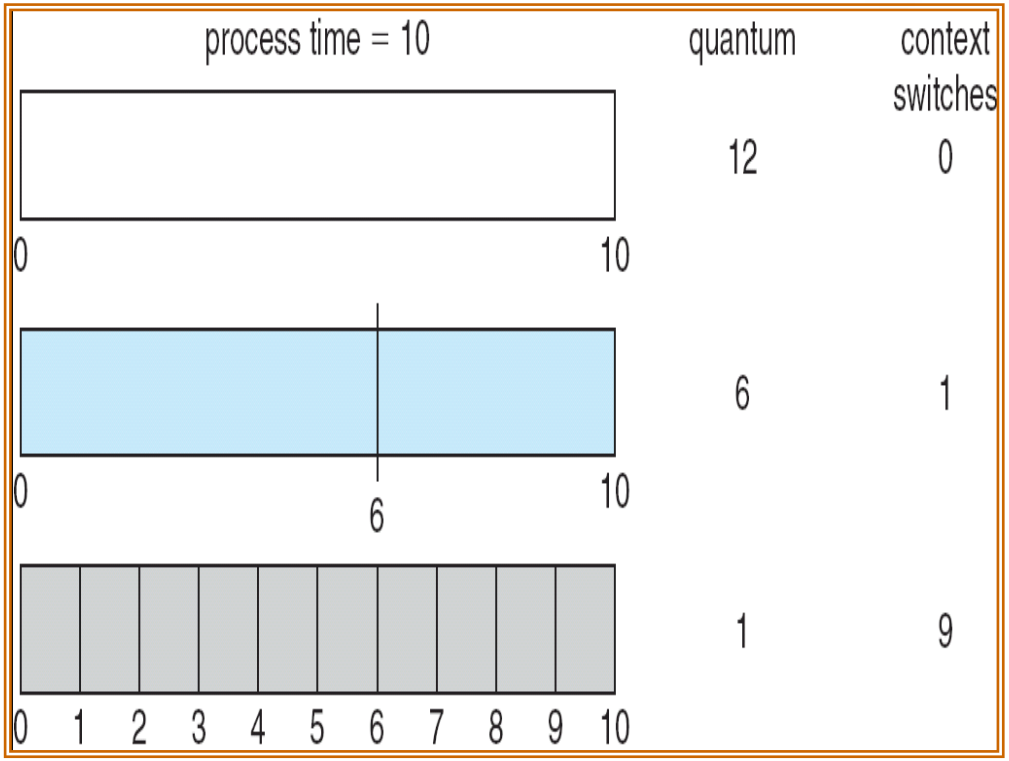


Fig. The way in which a smaller time quantum increases context switches.

3.1.3 Turnaround Time Varies With The Time Quantum

Turnaround time also depends on the size of the time quantum. the average turnaround time of a set of processes does not necessarily improve as the time-quantum size increases.

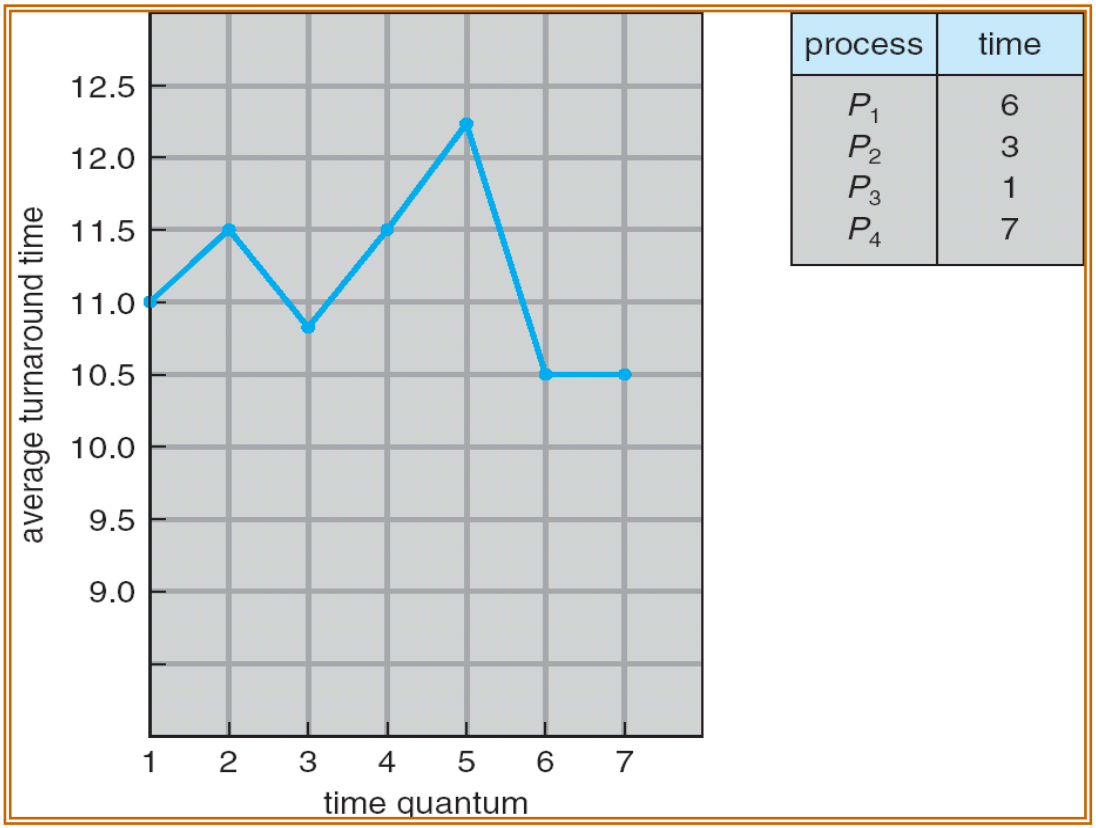


Fig. The way in which turnaround time varies with the time quantum

3.2 Multilevel Queue

- Ready queue is partitioned into separate queues: foreground (interactive) background (batch)
- 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). 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

3.2.1 Multilevel Queue Scheduling

A multilevel queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.

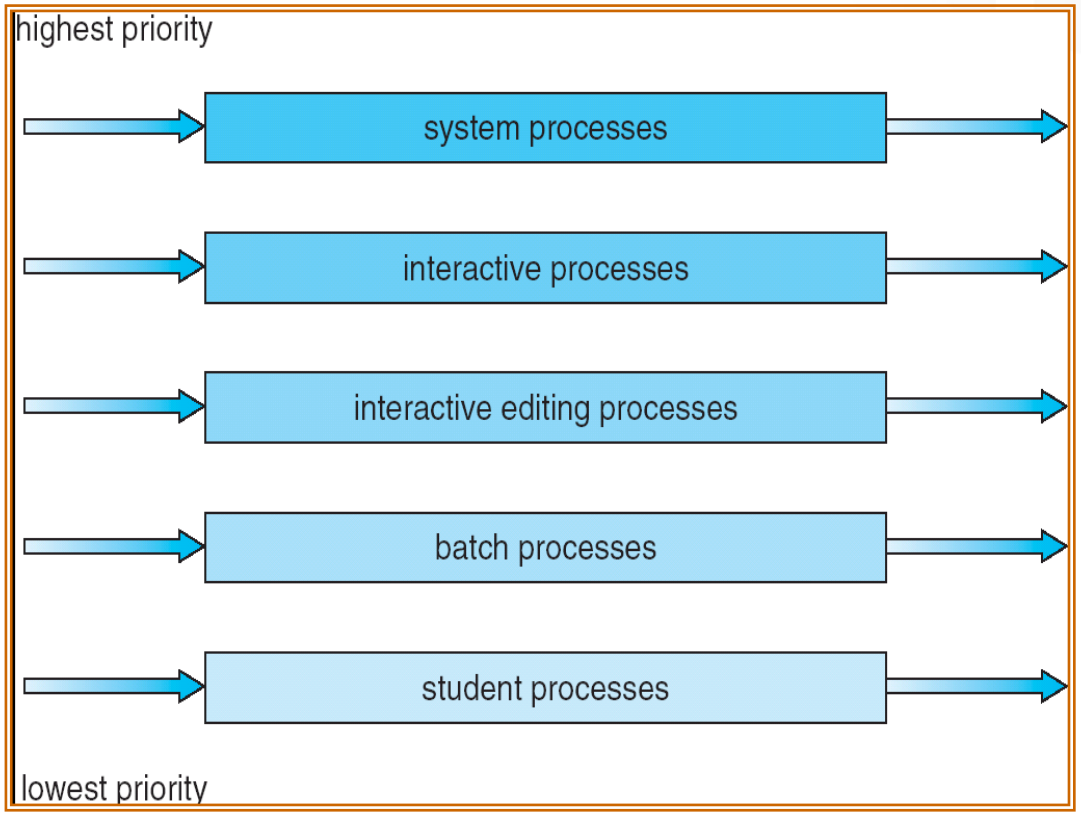


Fig. Multilevel queue scheduling.

3.3 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

3.3.1 Example of Multilevel Feedback Queue

- Three queues:
  - Q0 – RR with time quantum 8 milliseconds
  - Q1 – RR time quantum 16 milliseconds
  - Q2 – FCFS
- Scheduling
  - A new job enters queue Q0 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 Q1.
  - At Q1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q2.

3.3.2 Multilevel Feedback Queues

The multilevel feedback-queue scheduling algorithm, in contrast, allows a process to move between queues.
Consider a multilevel feedback-queue scheduler with three queues, numbered from 0 to 2.

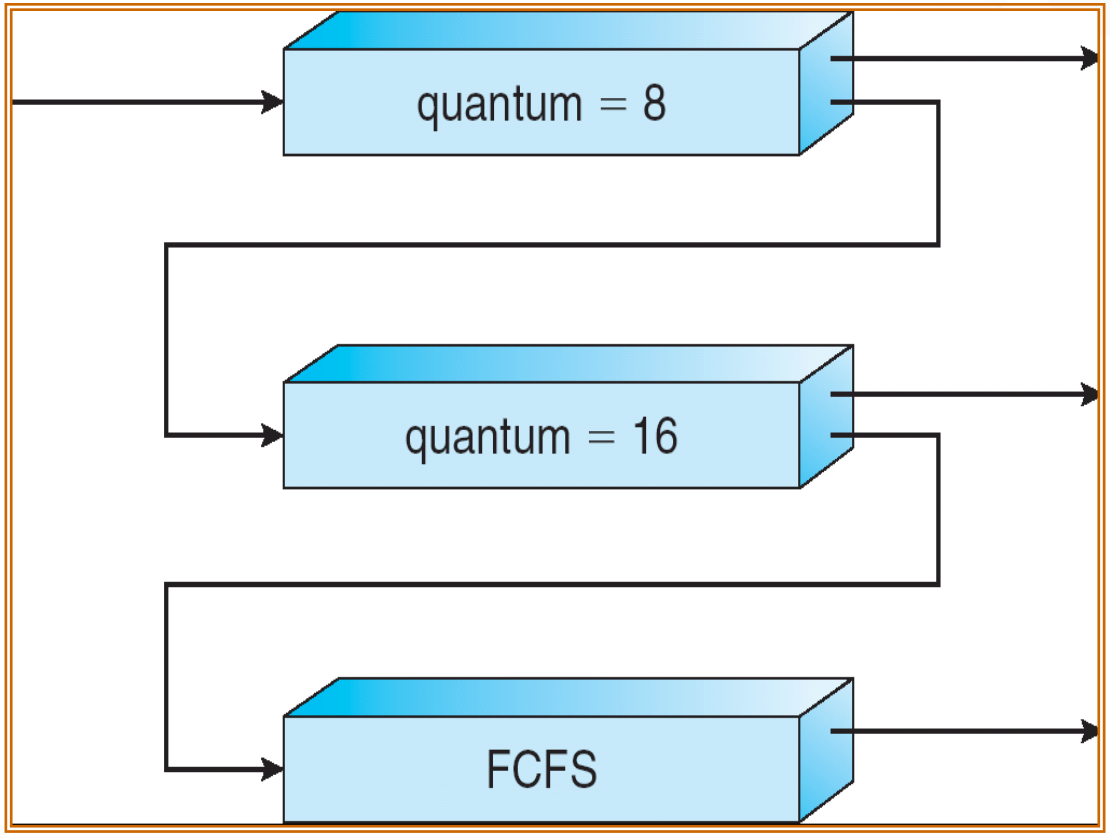


Fig. Multilevel feedback queues

3.4 Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- Homogeneous processors within a multiprocessor
- Load sharing
- Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing

3.5 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

ubm
Universitas Bunda Mulia

4. Thread Scheduling

Thread Scheduling

- Local Scheduling – How the threads library decides which thread to put onto an available LWP
- Global Scheduling – How the kernel decides which kernel thread to run next

4.1 Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM THREADS 5
int main(int argc, char *argv[])
{
        int i;
       pthread t tid[NUM THREADS];
       pthread attr t attr;
       /* get the default attributes */
       pthread attr init(&attr);
       /* set the scheduling algorithm to PROCESS or SYSTEM */
       pthread attr setscope(&attr, PTHREAD SCOPE SYSTEM);
       /* set the scheduling policy - FIFO, RT, or OTHER */
       pthread attr setschedpolicy(&attr, SCHED OTHER);
       /* create the threads */
       for (i = 0; i < NUM THREADS; i++)
                pthread create(&tid[i],&attr,runner,NULL);
```

4.1 Pthread Scheduling API (Cont.)

```
        /* now join on each thread */
        for (i = 0; i < NUM THREADS; i++)
                pthread join(tid[i], NULL);
}
 /* Each thread will begin control in this function */
void *runner(void *param)
{
        printf("I am a thread\n");
        pthread exit(0);
}
```

5. Multi-Processor Scheduling

Operating System Examples

- Solaris scheduling
- Windows XP scheduling
- Linux scheduling

5.1 Solaris 2 Scheduling

Solaris uses priority-based thread scheduling. It has defined four classes of scheduling, which are, in order of priority

1. Real time
2. System
3. Time sharing
4. Interactive

Within each class there are different priorities and different scheduling algorithms.

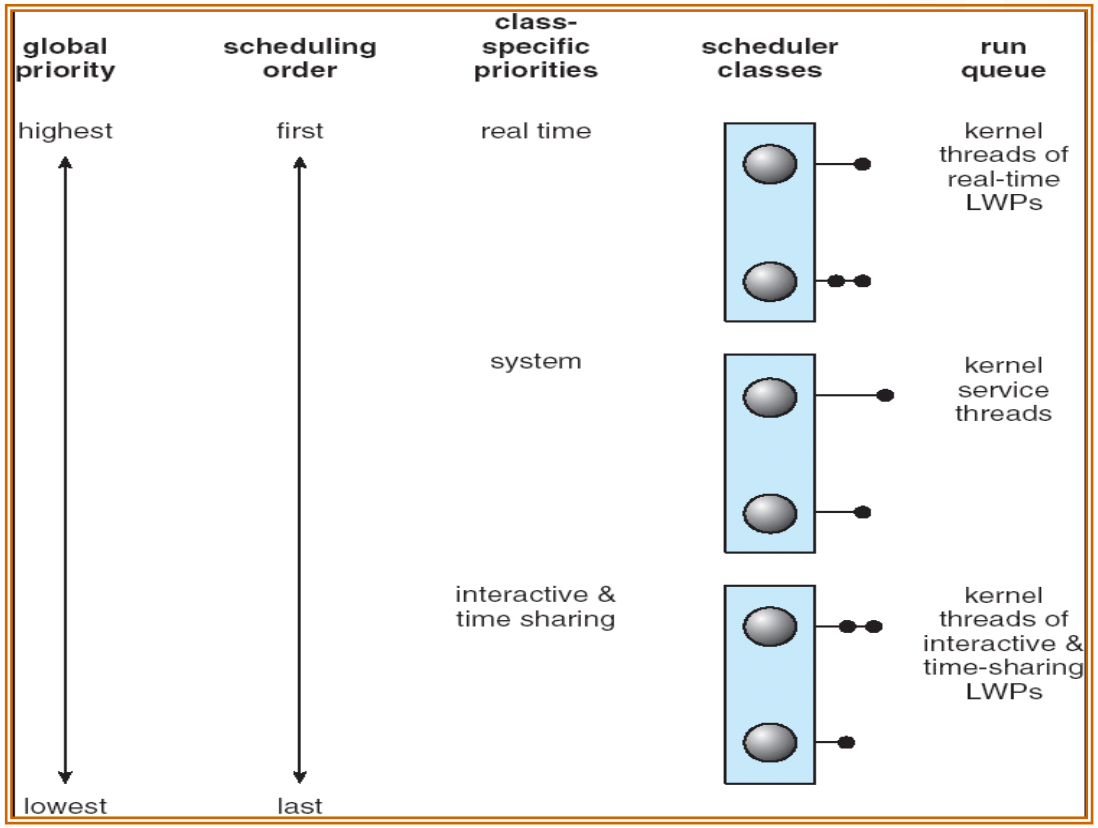


Fig. Solaris scheduling

5.1.1 Solaris Dispatch Table

The dispatch table for scheduling interactive and time_x0002_sharing threads. These two scheduling classes include 60 priority levels, but for brevity, we display only a handful.

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

Fig. Solaris dispatch table for interactive and time-sharing threads.

5.2 Windows XP Priorities

The priority of each thread is based on the priority class it belongs to and its relative priority within that class. The values of the priority classes appear in the top row. The left column contains the values for the relative priorities.

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Fig. Windows XP priorities

6. Real-Time CPU Scheduling

6.1 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

6.2 The Relationship Between Priorities and Time-slice length

A runnable task is considered eligible for execution on the CPU as long as it has time remaining in its time slice. When a task has exhausted its time slice, it is considered expired and is not eligible for execution again until all other tasks have also exhausted their time quanta.

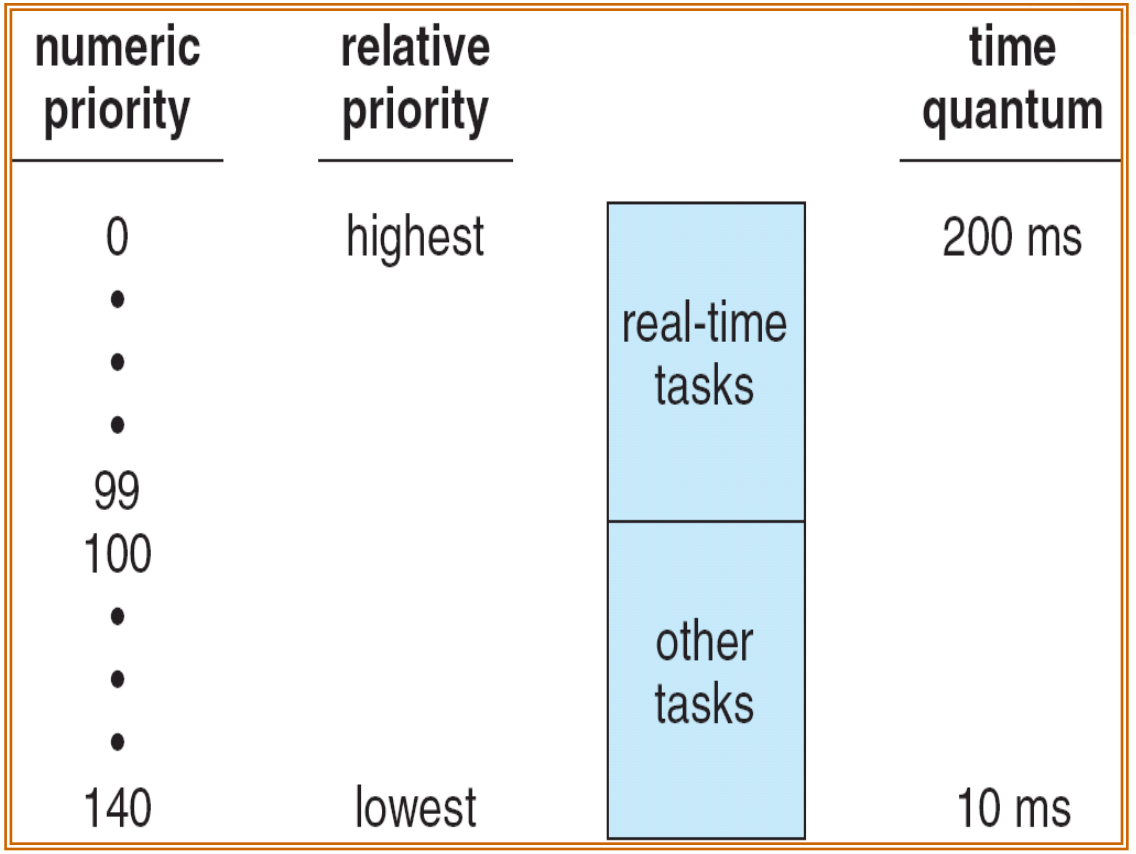


Fig. The relationship between priorities and time-slice length.

6.3 List of Tasks Indexed According to Prorities

The active array contains all tasks with time remaining in their time slices, and the expired array contains all expired tasks. Each of these priority arrays contains a list of tasks indexed according to priority

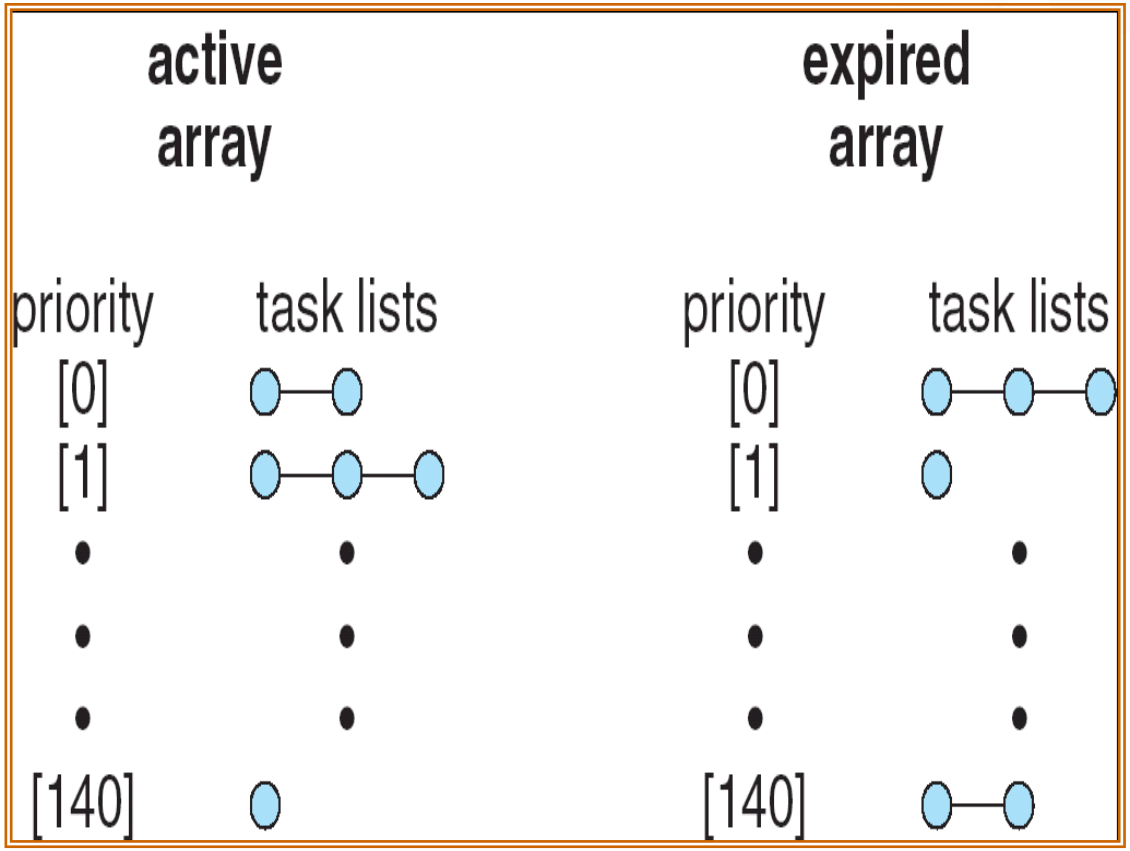


Fig. List of tasks indexed according to priority.

7. Algorithm Evaluation

Algorithm Evaluation

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

Summary

- CPU scheduling decisions may take place when a process:
  - Switches from running to waiting state
  - Switches from running to ready state
  - Switches from waiting to ready
  - Terminates
- Scheduling Criteria
  - CPU utilization – keep the CPU as busy as possible
  - Throughput – # of processes that complete their execution per time unit
  - Turnaround time – 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)

ubm
Universitas Bunda Mulia

Thank You