Process Scheduling

B.Ramamurthy

Introduction

- An important aspect of multiprogramming is scheduling. The resources that are scheduled are IO and processors.
- The goal is to achieve
  - High processor utilization
  - High throughput
    - number of processes completed per unit time
  - Low response time
    - time elapse from the submission of a request to the beginning of the response

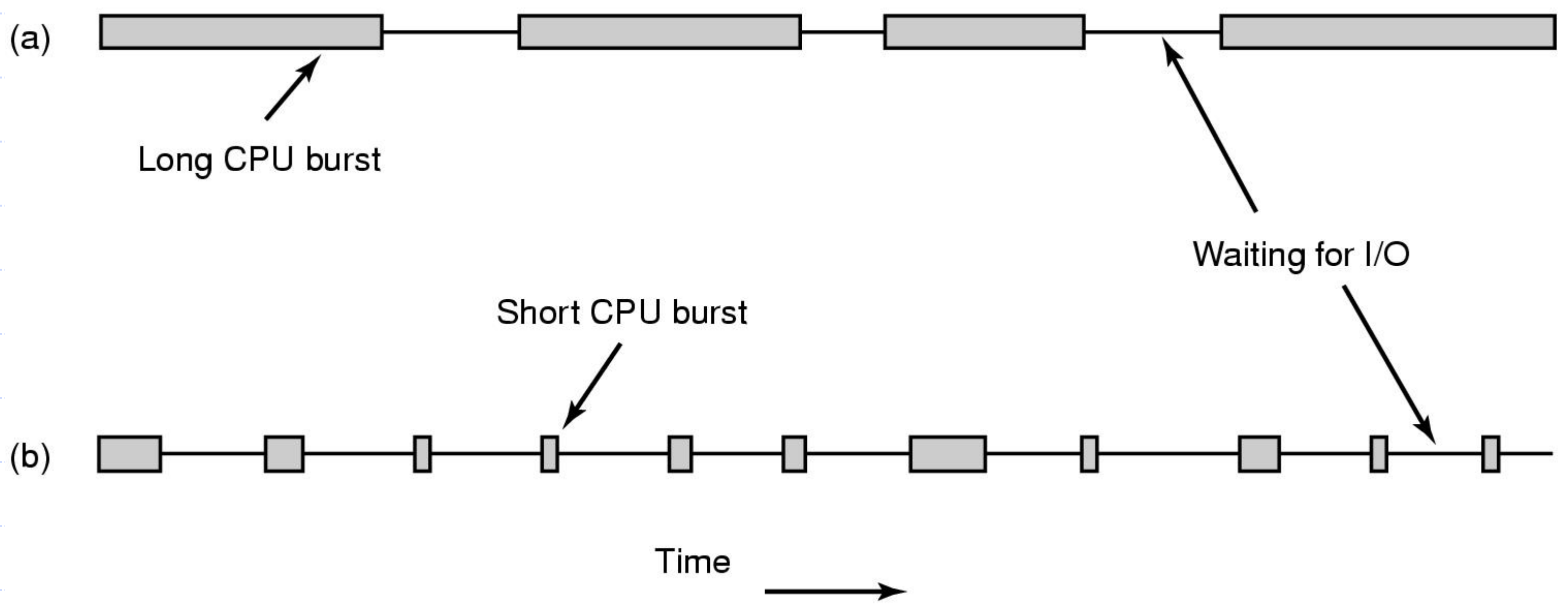
Topics for discussion

- Motivation
- Types of scheduling
- Short-term scheduling
- Various scheduling criteria
- Various algorithms
  - Priority queues
  - First-come, first-served
  - Round-robin
  - Shortest process first
  - Shortest remaining time and others
- Queuing Model and Performance Analysis

The CPU-I/O Cycle

- We observe that processes require alternate use of processor and I/O in a repetitive fashion
- Each cycle consist of a CPU burst (typically of 5 ms) followed by a (usually longer) I/O burst
- A process terminates on a CPU burst
- CPU-bound processes have longer CPU bursts than I/O-bound processes

CPU/IO Bursts



- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process
  - an I/O bound process

Motivation

- Consider these programs with processing-component and IO-component indicated by upper-case and lower-case letters respectively.

A1 a1 A2 a2 A3

0 30 50 80 120 130 ===> JOB A

B1 b1 B2

0 20 40 60 ====> JOB B

C1 c1 C2 c2 C3 c3 C4 c4 C5

0 10 20 60 80 100 110 130 140 150

==>JOB C

Motivation

- The starting and ending time of each component are indicated beneath the symbolic references (A1, b1 etc.)
- Now lets consider three different ways for scheduling: no overlap, round-robin, simple overlap.
- Compare utilization U = Time CPU busy / Total run time

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)

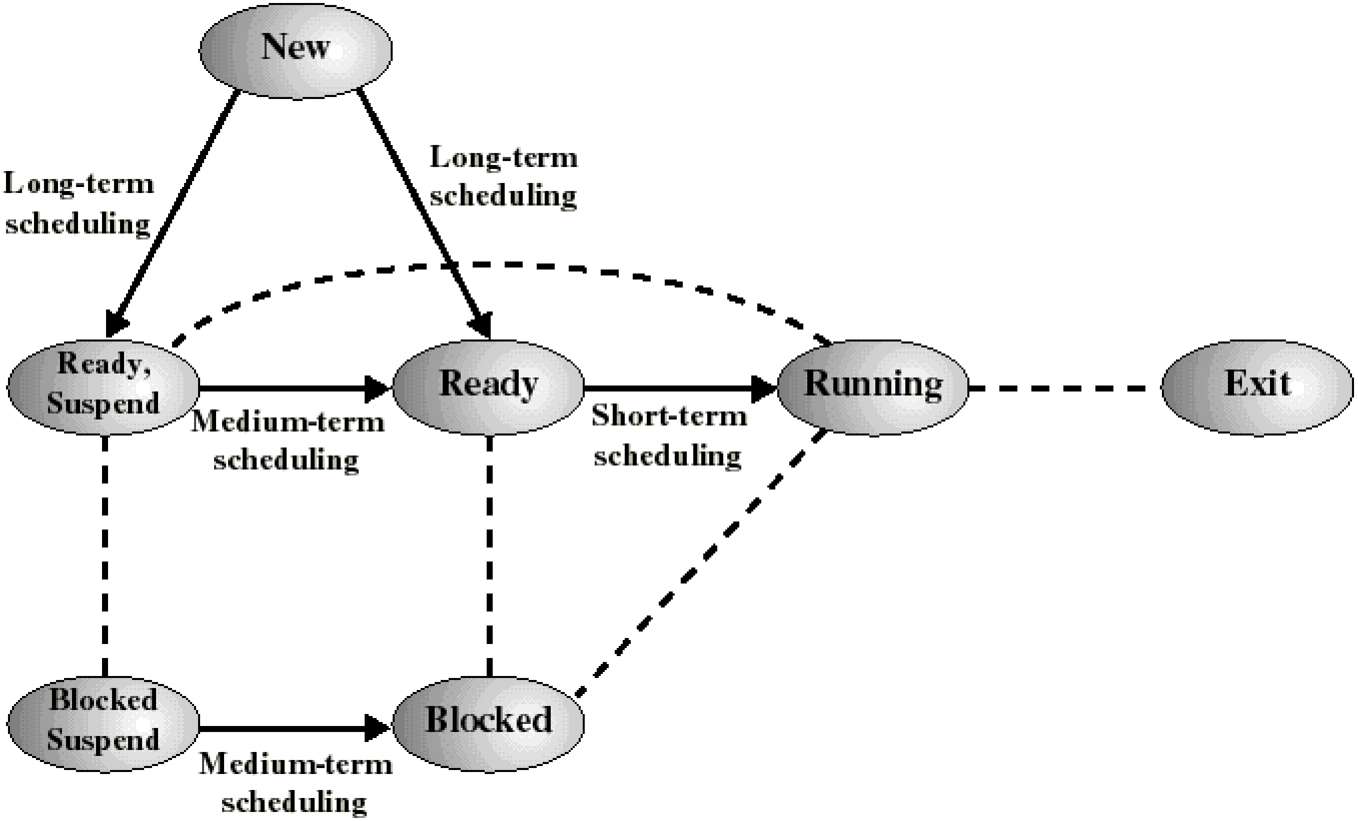
Optimization Criteria

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

Types of scheduling

- Long-term : To add to the pool of processes to be executed.
- Medium-term : To add to the number of processes that are in the main memory.
- **Short-term** : Which of the available processes will be executed by a processor?
- IO scheduling: To decide which process's pending IO request shall be handled by an available IO device.

Classification of Scheduling Activity



- Long-term: which process to admit
- Medium-term: which process to swap in or out
- Short-term: which ready process to execute next

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 $P_1 = 0$; $P_2 = 24$; $P_3 = 27$

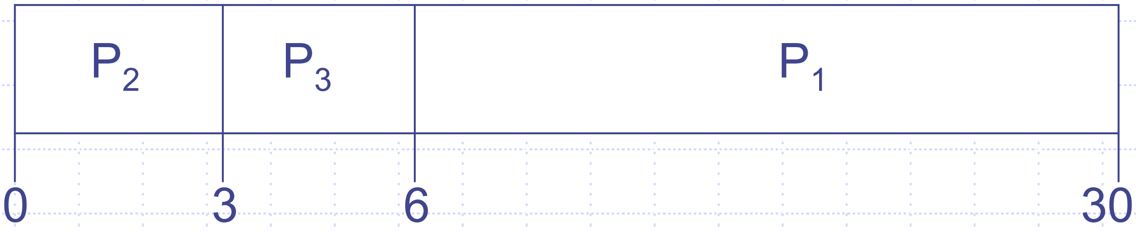
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1 .$$

- The Gantt chart for the schedule is:



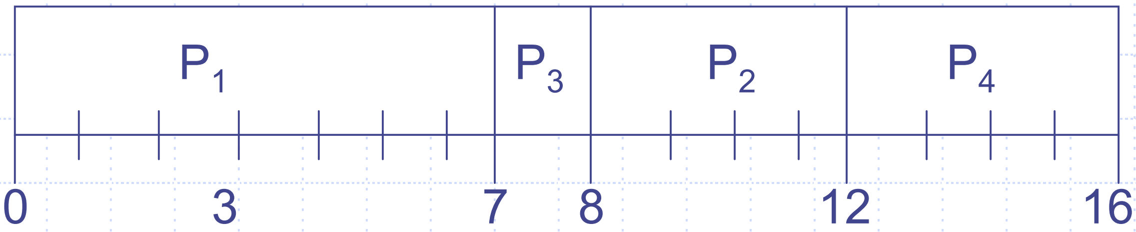
- Waiting time for $P_1 = 6; P_2 = 0_; P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.
- *Convoy effect* short process behind long process

Shortest-Job-First (SJR) 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.

Example of Non-Preemptive SJF

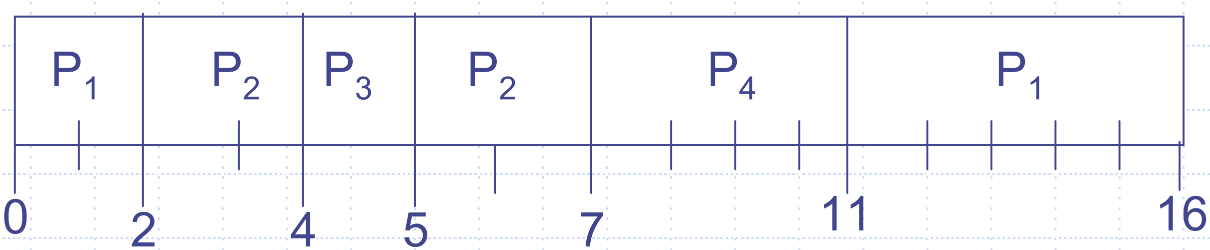
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- 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



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

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. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n .$$

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:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\,\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j\,\alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1}\,\tau_1$$

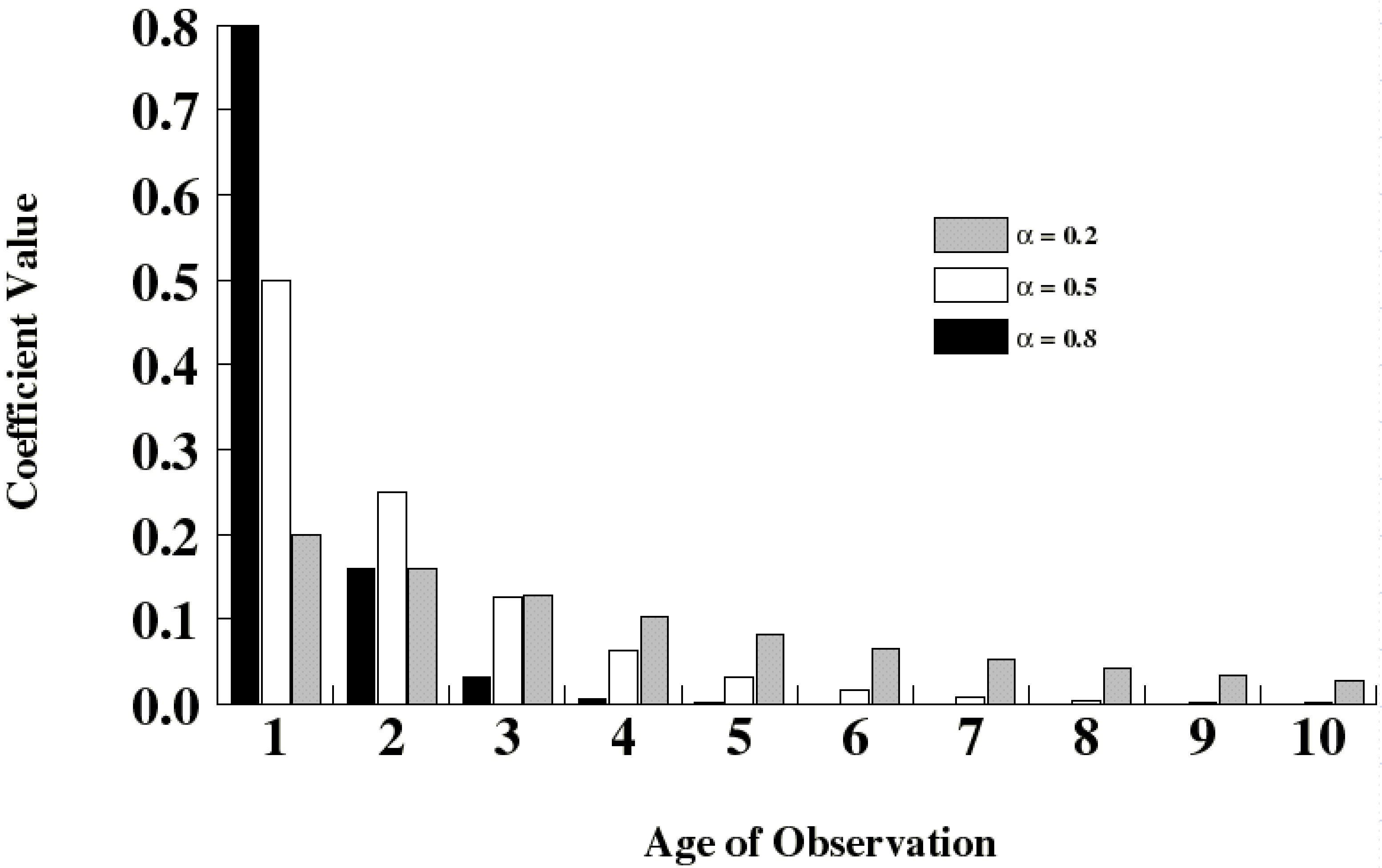
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

More on Exponential Averaging

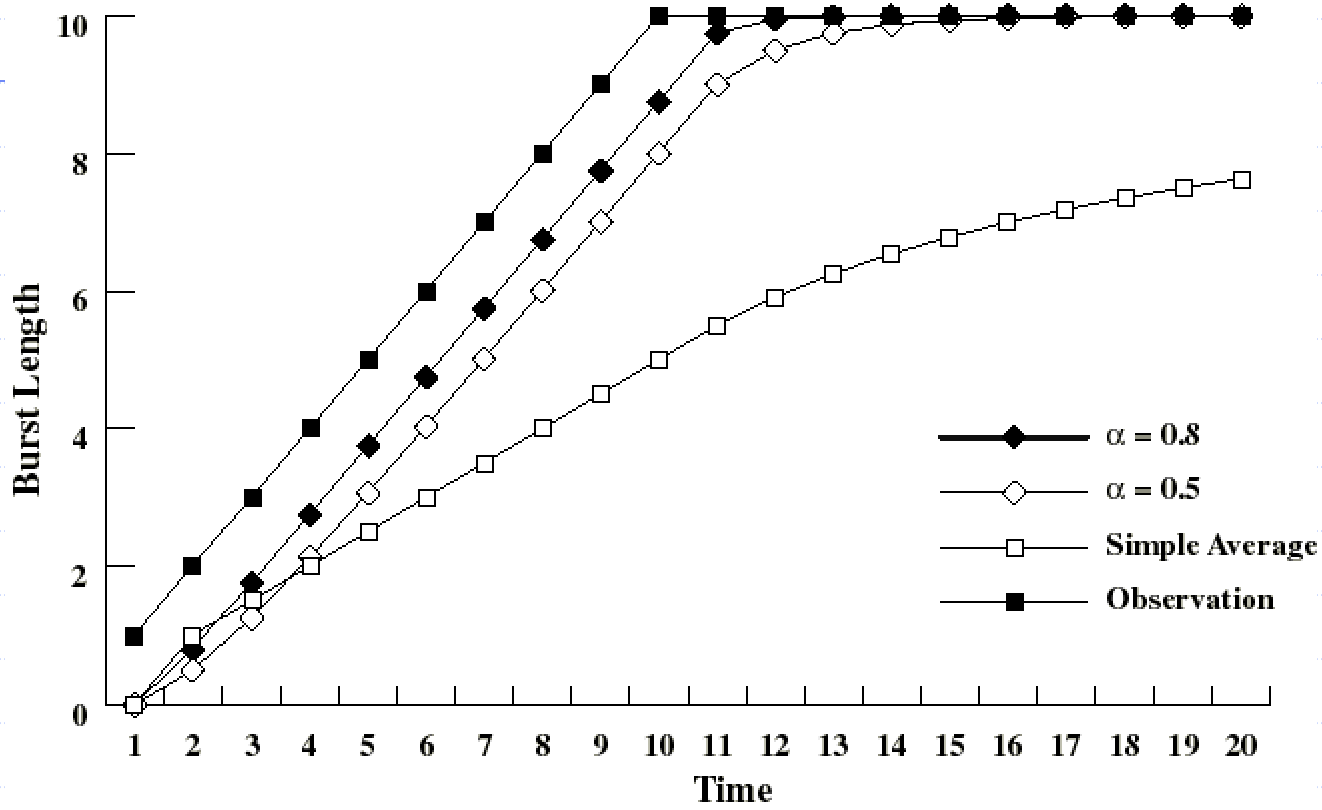
- S[n+1] next burst, s[n] current burst
  - $S[n+1] = \alpha\ T[n] + (1-\alpha)\ S[n]$; $0 < \alpha < 1$
  - more weight is put on recent instances whenever $\alpha > 1/n$
- By expanding this eqn, we see that weights of past instances are decreasing exponentially
  - $S[n+1] = \alpha T[n] + (1-\alpha)\alpha T[n-1] + \ldots (1-\alpha)^i \alpha T[n-i] +$

    $\ldots + (1-\alpha)^n S[1]$
  - predicted value of 1st instance S[1] is not calculated; usually set to 0 to give priority to to new processes

Exponentially Decreasing Coefficients



Exponentially Decreasing Coefficients



- Here S[1] = 0 to give high priority to new processes
- Exponential averaging tracks changes in process behavior much faster than simple averaging

Shortest Process Next: critique

- Possibility of starvation for longer processes as long as there is a steady supply of shorter processes
- Lack of preemption is not suited in a time sharing environment
  - CPU bound process gets lower priority (as it should) but a process doing no I/O could still monopolize the CPU if he is the first one to enter the system
- SPN implicitly incorporates priorities: shortest jobs are given preferences
- The next (preemptive) algorithm penalizes directly longer jobs

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.

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*.

Scheduling in Real-Time Systems

Schedulable real-time system

- Rate Monotonic Scheduling:
- Given
  - *m* periodic events
  - event *i* occurs within period P_i and requires C_i seconds
- Then the load can only be handled if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

Summary

- Scheduling is important for improving the system performance.
- Methods of prediction play an important role in Operating system and network functions.
- Simulation is a way of experimentally evaluating the performance of a technique.

One more IPC

- Sleeping Barbar

The Sleeping Barber Problem

The Sleeping Barber Problem

```
#define CHAIRS 5                    /* # chairs for waiting customers */

typedef int semaphore;              /* use your imagination */

semaphore customers = 0;            /* # of customers waiting for service */
semaphore barbers = 0;              /* # of barbers waiting for customers */
semaphore mutex = 1;                /* for mutual exclusion */
int waiting = 0;                    /* customers are waiting (not being cut) */

void barber(void)
{
     while (TRUE) {
          down(&customers);         /* go to sleep if # of customers is 0 */
          down(&mutex);             /* acquire access to 'waiting' */
          waiting = waiting – 1;    /* decrement count of waiting customers */
          up(&barbers);             /* one barber is now ready to cut hair */
          up(&mutex);               /* release 'waiting' */
          cut_hair( );              /* cut hair (outside critical region) */
     }
}


void customer(void)
{
     down(&mutex);                  /* enter critical region */
     if (waiting < CHAIRS) {        /* if there are no free chairs, leave */
          waiting = waiting + 1;    /* increment count of waiting customers */
          up(&customers);           /* wake up barber if necessary */
          up(&mutex);               /* release access to 'waiting' */
          down(&barbers);           /* go to sleep if # of free barbers is 0 */
          get_haircut( );           /* be seated and be serviced */
     } else {
          up(&mutex);               /* shop is full; do not wait */
     }
}
```

Solution to sleeping barber problem.