K. J. Somaiya College of Engineering, Mumbai

(A constituent College of Somaiya Vidyavihar University)

Operating System

Module 2. Process Concept and scheduling

Dr. Prasanna Shete

Dept. of Computer Engineering

prasannashete@somaiya.edu

Mobile/WhatsApp 9960452937

Module 2: Process Concept and scheduling

- Scheduling: Uniprocessor Scheduling - Types of Scheduling:
- Preemptive and, Non-preemptive, Scheduling Algorithms:
- FCFS, SJF, SRTN, Priority based, Round Robin, Multilevel Queue scheduling.
- Multi Processor Scheduling
- Introduction to Thread Scheduling
- Linux Scheduling

Process Scheduling Example

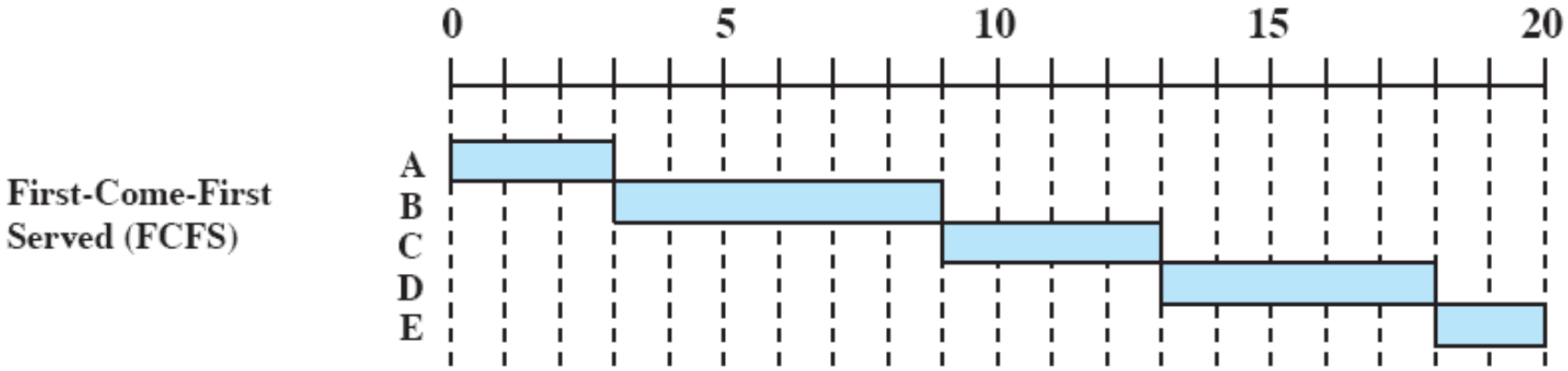
- Service time represents total execution time
- Set of processes, consider each as batch job

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

First-Come-First-Served (FCFS)

- Process that requests CPU first is allotted first
- Each process joins the Ready queue
- When the current process ceases to execute, the longest waiting process in the Ready queue is selected

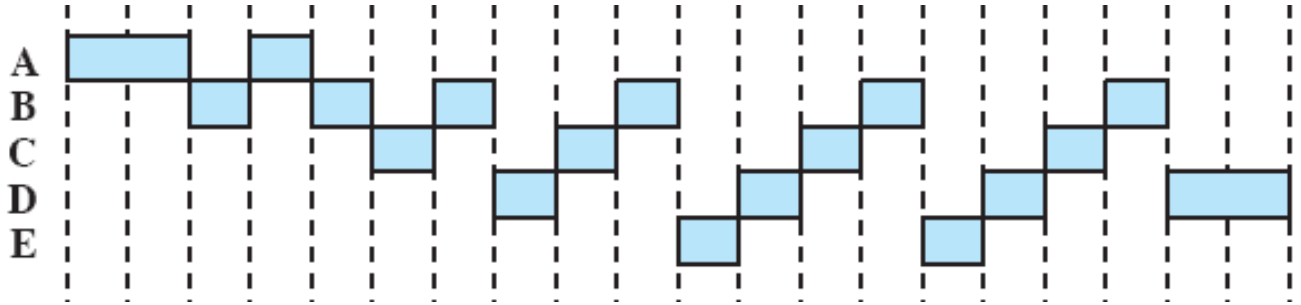


- A short process may have to wait a very long time before it can execute
- Favors CPU-bound processes; –I/O processes have to wait until CPU-bound process completes

Round Robin

- Uses preemption based on a clock
- –also known as time slicing, because each process is given a slice of time before being preempted
- Clock interrupt is generated at periodic intervals
  - When an interrupt occurs, the currently running process is placed in the ready queue
  - Next ready job is selected

Round-Robin (RR), $q = 1$



Round Robin: Effect of Size of Preemption Time Quantum

- Small Quantum:
  - Short processes move through the system relatively quickly
  - BUT involves **more context switches**
  - → increased processing over-head in handling the clock interrupt and performing scheduling and dispatching functions

- Large Quantum:
  - **Quantum > largest service time**
    - **RR → FCFS**
- Time quantum should be slightly greater than the time required for a typical interaction or process function
  - If it is less, then most processes will require at least two time quanta

Round Robin: Limitations

- Processor-bound processes tend to receive an unfair portion of processor time
- Results in:
  - poor performance for I/O-bound processes
  - inefficient use of I/O devices
  - increase in the variance of response time
- Remedy: Virtual RR

Virtual Round Robin

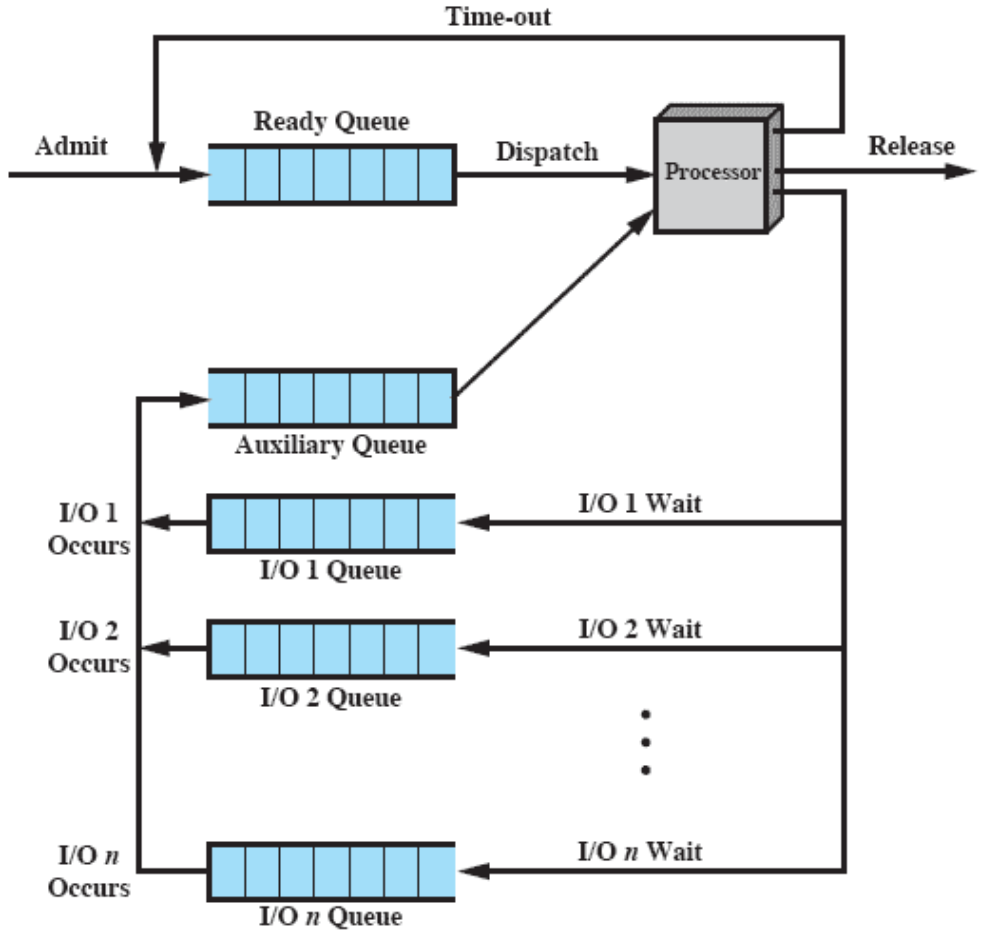


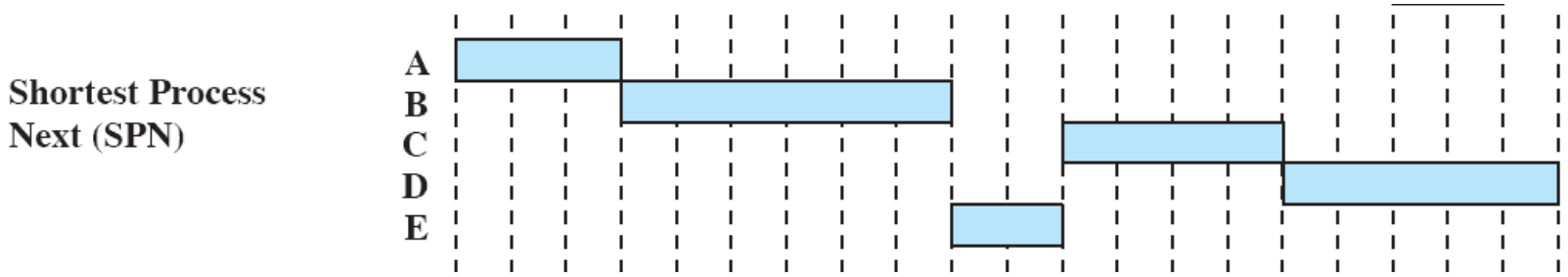
Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

Virtual Round Robin

- FCFS Auxiliary queue
- Processes moved to auxiliary queue after being released from an I/O blocked Q
- Processes in the auxiliary queue get preference over those in the main ready queue in the dispatching decision
- When a process is dispatched from the auxiliary queue, it runs no longer than time equal to the basic time quantum minus total time spent running since it was last selected from the main ready queue

Shortest Process Next

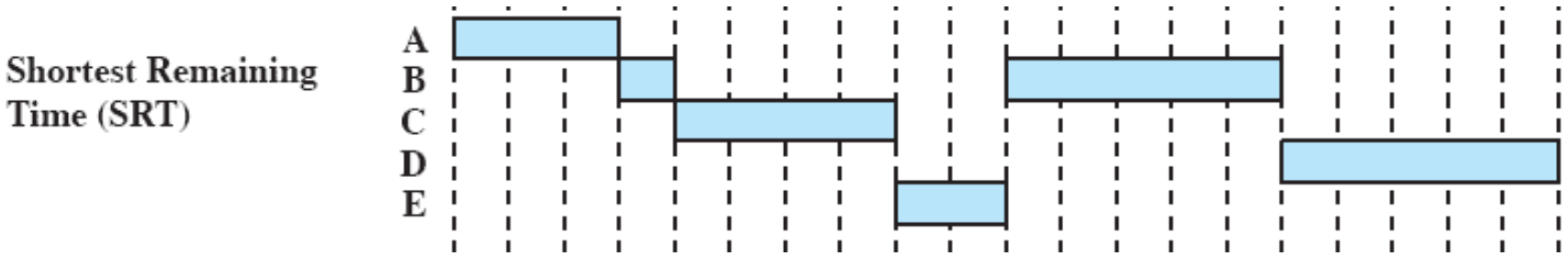
- Non preemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes



- Possibility of starvation for longer processes
- Predictability of longer processes is reduced
  - If estimated time for process not correct, the operating system may abort it

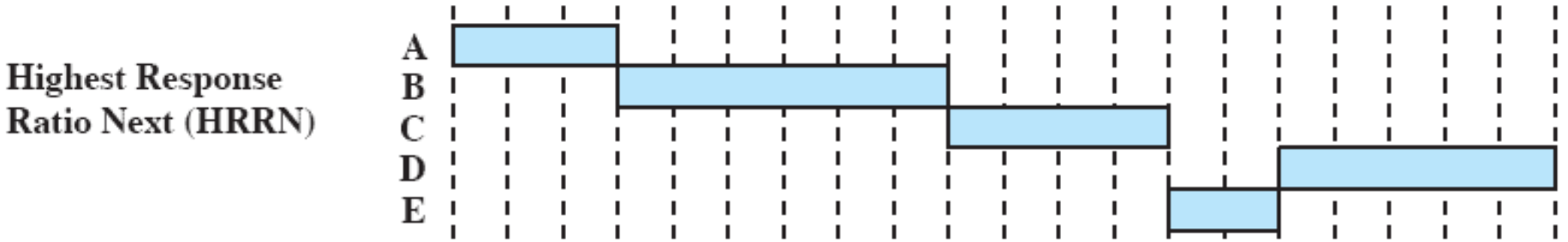
Shortest Remaining Time

- Preemptive version of shortest process next policy
- Must estimate processing time and choose the shortest process



Highest Response Ratio Next

- Choose next ready process with the greatest response ratio (R)
- Response Ratio (R) = $\frac{time\ spent\ waiting\ (w) + expected\ service\ time\ (s)}{expected\ service\ time\ (s)}$
  - Aging without service increases the ratio, so that longer waiting process (*w)* get past competing shorter processes (*s*)
- Expected service time must be estimated → Estimated using FCFS



Feedback Scheduling

- Penalize jobs that have been running longer
  - focus on time spent in execution so far
- Premptive method based on time quantum
- Dynamic priority mechanism

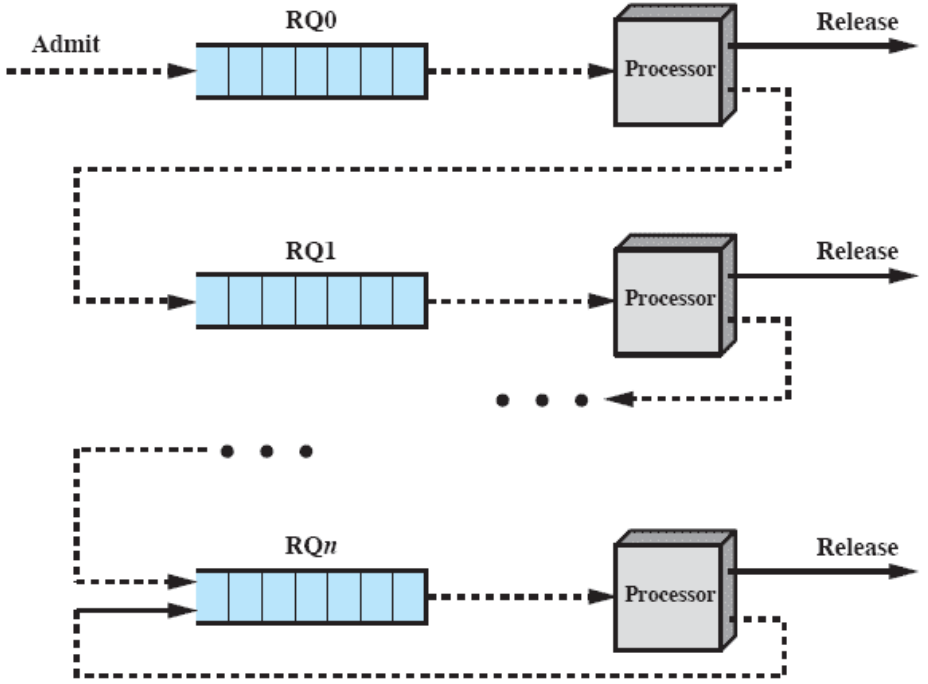


Figure 9.10 Feedback Scheduling

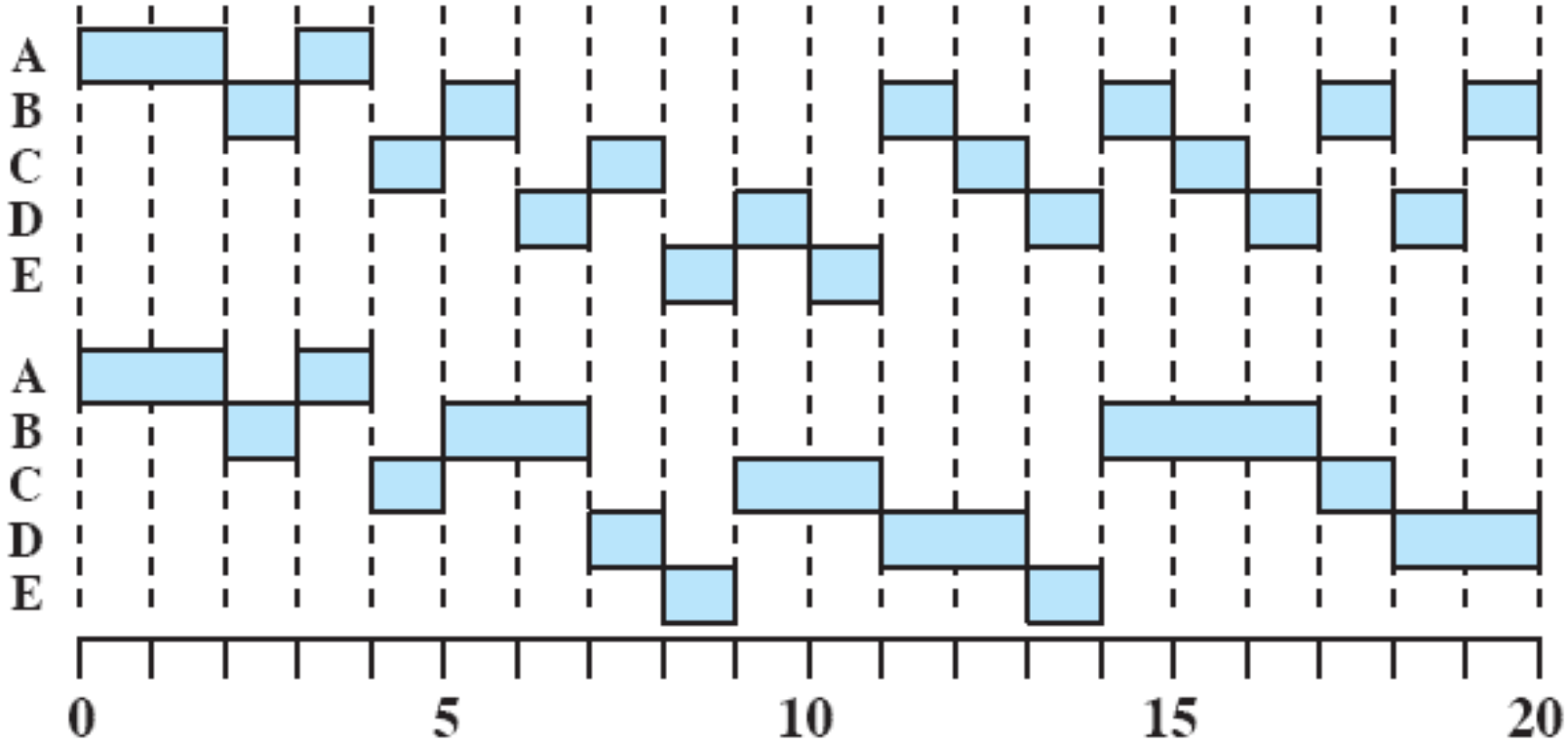
Feedback Scheduling

- A new process is placed in the RQ_0 (Ready queue with highest priority)
- After being executed for given time slice, the process is prempted and demoted to the next lower-priority queue
- All queues are FCFS except the least priority Q which uses RR mechanism
- Problem: In simple MLFQ, Turnaround time of longer processes is stretched → starvation of long processes if new jobs frequently enter the system
- Remedy: Use different time quantum in the queues
- Process scheduled from RQ_i -allowed to execute for 2^i time units before premption
- RQ_0 – 1time unit
  - RQ_1 – 2 time units
  - RQ_0 – 4 time units
  - And so on…

Feedback Scheduling

Feedback
$q = 1$

Feedback
$q = 2^i$



Fair-Share Scheduling

- User's application runs as a collection of processes (and/or threads)
- User is concerned about the performance of the application as a whole and not about performance of a particular process

- Need to make scheduling decisions based on process sets →known as Fair-Share scheduling
  - The concept can be extended to groups of users

Fair-Share Scheduling

- Each user is assigned a weight
  - defines its share of system resources as a fraction of total usage of these resources
- Each user is assigned a share of the processor

- FSS monitors the usage
  - Fever resources to users who have had more than their fair share
  - More resources to those who have had less than their fair share

Fair-Share Scheduling

- The system divides the users into a set of fair share groups and allocates a fraction of the processor usage to each group
  - i.e Each fair-share group is provided with a virtual system
- FSS considers execution history of a related group of processes along with the individual execution history of each process in the scheduling decisions

Fair-Share Scheduling

- Priority Scheduling; takes into account the underlying priority of the process, its recent processor usage, and the recent processor usage of the group to which the process belongs

- Each process assigned base priority
- Priority of the process drops as
  - Process uses the processor and/or
  - The group to which the process belongs uses the processor

Priority = Base Priority + Processor Utilization by Process + Processor Utilization by Group

Fair-Share Scheduling

Priority = Base Priority + Processor Utilization by process + Processor Utilization by Group

$$\boldsymbol{P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4*wk}}$$

Where

$CPU_j(i)$ – Measure of processor utilization by process *j* through interval *i* $= \frac{CPU_j(i-1)}{2}$

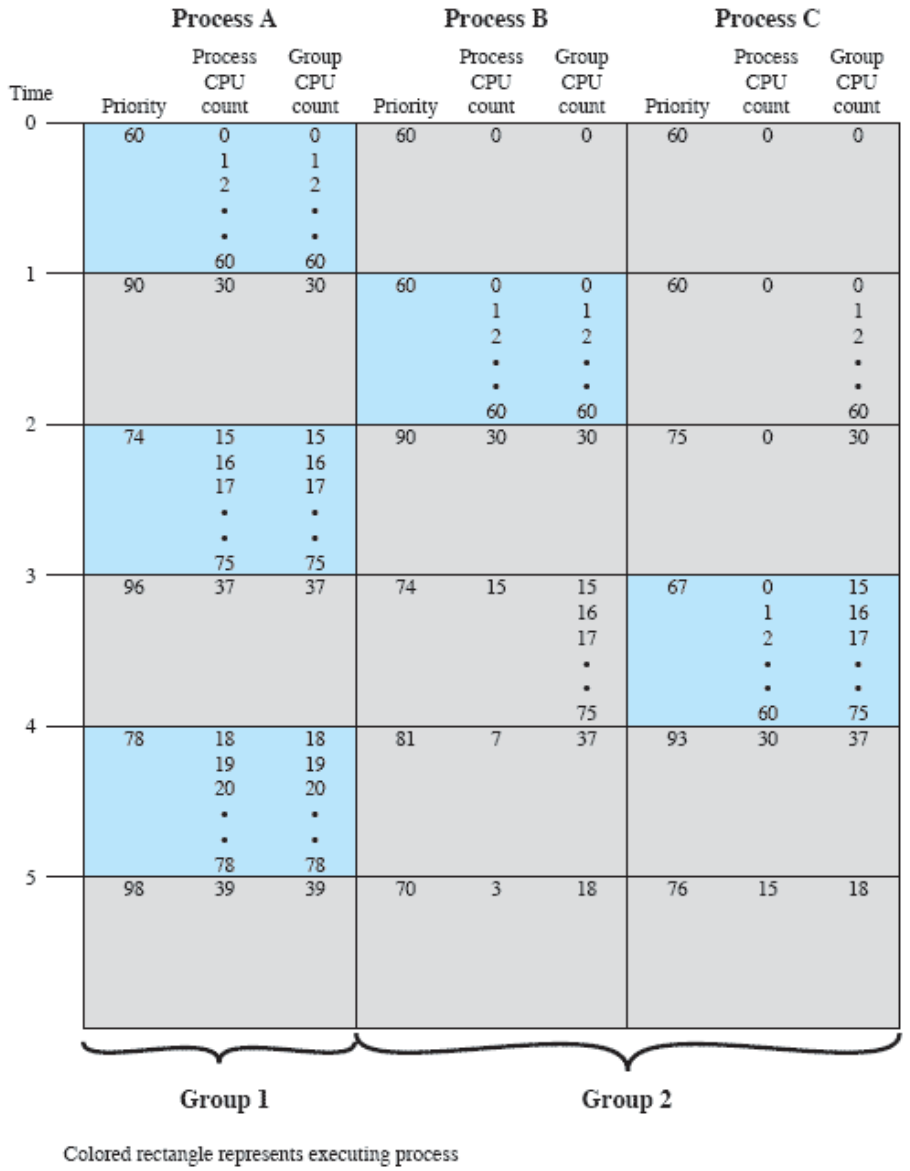
$GCPU_k(i)$ - Measure of processor utilization by group of processes *k* through interval *i*$= \frac{GCPU_k(i-1)}{2}$

$P_j(i)$ - priority of process *j* at beginning of interval *i*

$Base_j$- base priority of process *j*

w_k - weight assigned to group *k*, with $0 < w_k <= 1$ and $\sum_k w_k = 1$

Example



Time	Process A Priority	Process A Process CPU count	Process A Group CPU count	Process B Priority	Process B Process CPU count	Process B Group CPU count	Process C Priority	Process C Process CPU count	Process C Group CPU count
0	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0	60	0	0
1	90	30	30	60	0 1 2 • • 60	0 1 2 • • 60	60	0	0 1 2 • • 60
2	74	15 16 17 • • 75	15 16 17 • • 75	90	30	30	75	0	30
3	96	37	37	74	15	15 16 17 • • 75	67	0 1 2 • • 60	15 16 17 • • 75
4	78	18 19 20 • • 78	18 19 20 • • 78	81	7	37	93	30	37
5	98	39	39	70	3	18	76	15	18

Group 1 Group 2

Colored rectangle represents executing process

Figure 9.16 Example of Fair Share Scheduler—Three Processes, Two Groups