# Scheduling policies

## Scheduling Criteria

- CPU-bound processes
  - Use all available processor time
- I/O-bound
  - Generates an I/O request quickly and relinquishes processor
- Batch processes
  - Contains work to be performed with no user interaction
- Interactive processes
  - Requires frequent user input

## 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 and blocked 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 throughput
  - Jobs per second
  - Throughput related to response time, but not identical
    - Minimizing response time will lead to more context switching than if you maximized only throughput
  - Minimize overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.)

- Min response time
  - Elapsed time to do an operation (job)
  - Response time is what the user sees
    - Time to echo keystroke in editor
    - Time to compile a program
    - Real-time Tasks: Must meet deadlines imposed by World

- Min turnaround time
- Min waiting time
- Max CPU utilization
- Fairness
  - Share CPU among users in some equitable way
  - Not just minimizing average response time

## Selection Function

- Determines which process is selected for execution.
- If it is based on execution characteristics then important quantities are:
  - ***w*** = time spent in system so far, waiting
  - ***e*** = time spent in execution so far
  - ***s*** = total service time required by the process, including *e*

## Decision Mode

- Specifies the instants in time at which the selection function is exercised.
- Two categories:
  - Non-preemptive
  - Preemptive

# Non-preemptive vs Preemptive

- Non-preemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O or OS service.
  - i.e. Run until completion or until they yield control of a processor
  - Unimportant processes can block important ones indefinitely
- Preemptive
  - Currently running process may be interrupted and moved to ready state by the OS.
  - Preemption may occur when new process arrives, on an interrupt, or periodically.
  - Can be removed from their current processor
  - Can lead to improved response times
  - Important for interactive environments
  - Preempted processes remain in memory

# Priorities

- Static priorities
  - Priority assigned to a process does not change
  - Easy to implement
  - Low overhead
  - Not responsive to changes in environment
- Dynamic priorities
  - Responsive to change
  - Promote smooth interactivity
  - Incur more overhead than static priorities
    - Justified by increased responsiveness

## Scheduling Objectives

- Different objectives depending on system
  - Maximize throughput
  - Maximize number of interactive processes receiving acceptable response times
  - Minimize resource utilization
  - Avoid indefinite postponement
  - Enforce priorities
  - Minimize overhead
  - Ensure predictability

## Scheduling Objectives

- Several goals common to most schedulers
  - Fairness
  - Predictability
  - Scalability

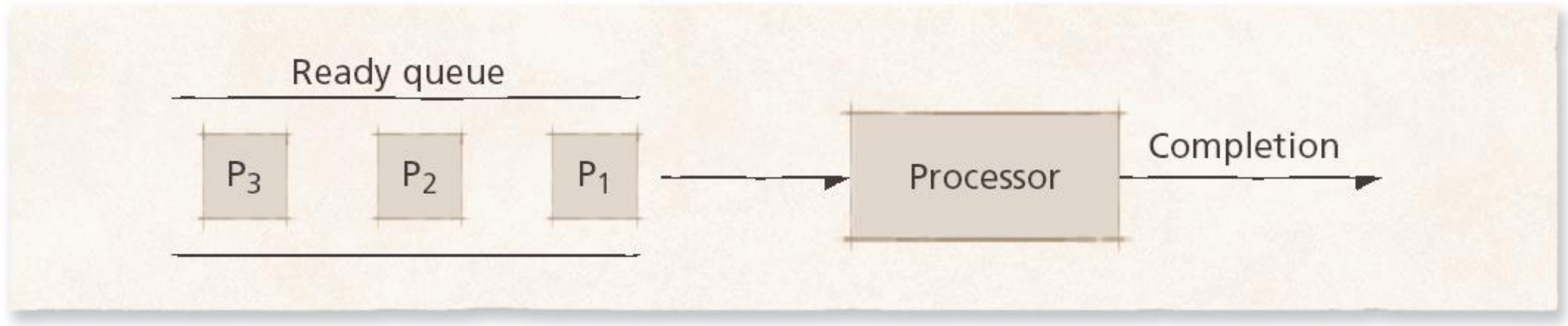
## Scheduling Algorithms

- Scheduling algorithms
  - Decide when and for how long each process runs
  - Make choices about
    - Preemptibility
    - Priority
    - Running time
    - Run-time-to-completion
    - fairness

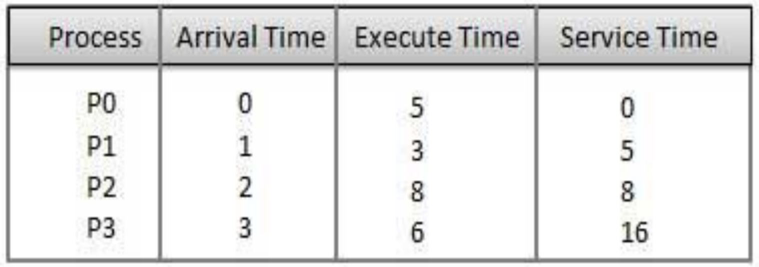
## First-In-First-Out (FIFO)/ First Come First Serve (FCFS) Scheduling

- Simplest scheme: Jobs are executed on first come, first serve basis
- Processes dispatched according to arrival time
- It is a non-preemptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue
- Poor in performance as average wait time is high
- Rarely used as primary scheduling algorithm
- Convoy effect

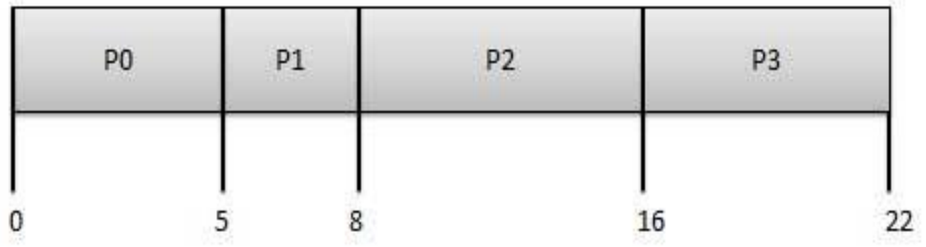
# First-In-First-Out (FIFO) Scheduling



## First Come First Serve (FCFS)



| Process | Arrival Time | Execute Time | Service Time |
|---|---|---|---|
| P0 | 0 | 5 | 0 |
| P1 | 1 | 3 | 5 |
| P2 | 2 | 8 | 8 |
| P3 | 3 | 6 | 16 |



## **Turnaround time** of each process is as follows –

| Process | Turnaround Time |
|---|---|
| P0 | 5 - 0 = 5 |
| P1 | 8 - 1 = 7 |
| P2 | 16 - 2 = 14 |
| P3 | 22 - 3 = 19 |

Average Turnaround Time: (5+7+14+19) / 4 = 11.25

## **Wait time** of each process is as follows –

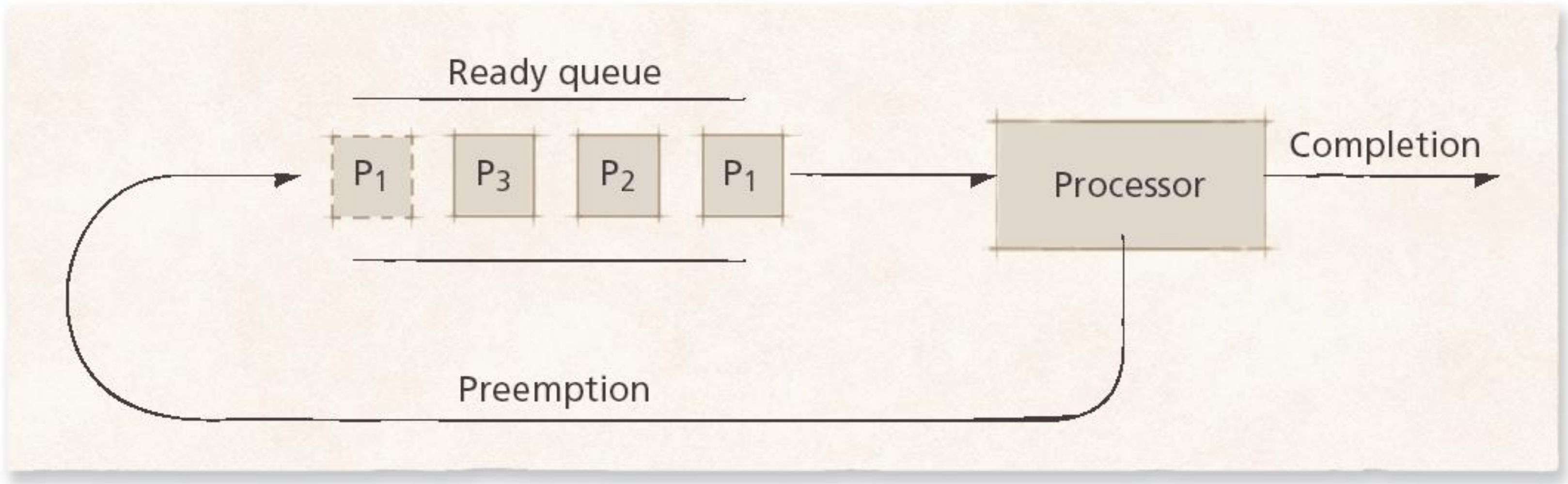
| Process | Wait Time |
|---|---|
| **P0** | 0 - 0 = 0 |
| **P1** | 5 - 1 = 4 |
| **P2** | 8 - 2 = 6 |
| **P3** | 16 - 3 = 13 |

Average Wait Time: (0+4+6+13) / 4 = 5.75

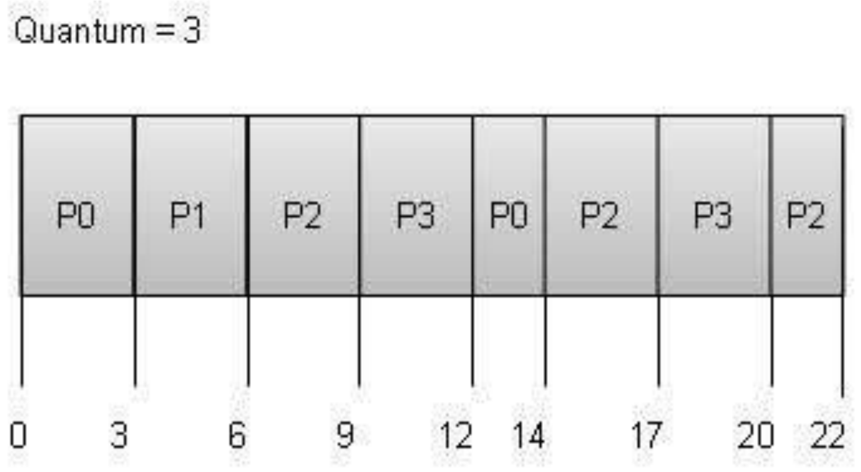
## Round-Robin (RR) Scheduling

- Round-robin scheduling
  - Based on FIFO
  - Processes run only for a limited amount of time called a time slice or quantum
  - Preemptible
  - Requires the system to maintain several processes in memory to minimize overhead
  - Context switching is used to save states of preempted processes.
  - Often used as part of more complex algorithms

# Round-Robin (RR) Scheduling



Assumption at time = 3, P3 is first put in the ready queue then P0.



## **Turnaround time** of each process is as follows :

| Process | Turnaround Time |
|---|---|
| P0 | 14-0 = 14 |
| P1 | 6 -1 =5 |
| P2 | 22-2 = 20 |
| P3 | 20 – 3 = 17 |

Average Turnaround Time:  (14 + 5 +20 + 17) / 4 = 14

## **Wait time** of each process is as follows

| Proces s | Wait Time |
|---|---|
| P0 | (0 - 0) + (12 - 3) = 9 |
| P1 | (3 - 1) = 2 |
| P2 | (6 - 2) + (14 - 9) + (20 - 17) = 12 |
| P3 | (9 - 3) + (17 - 12) = 11 |

Average Wait Time: (9+2+12+11) / 4 = 8.5

# Round-Robin (RR) Scheduling

Variants of RR:

- State dependent RR: same as RR but Q is varied dynamically depending on the state of the system.
- External priorities/ Weighted RR:  RR but a user can pay more and get bigger Q.
- Selfish RR:
  - A new process starts at priority 0; its priority increases at rate a>=0. It becomes an accepted process when its priority reaches that of an accepted process (or until there are no accepted processes).  At any time all accepted processes have same priority.
  - Increases priority as process ages
  - Two queues
    - Active
    - Holding
  - Favors older processes to avoids unreasonable delays

## Round-Robin (RR) Scheduling

- Quantum size
    - Determines response time to interactive requests
    - Very large quantum size
        - Processes run for long periods
        - Degenerates to FIFO
    - Very small quantum size
        - System spends more time context switching than running processes
    - Middle-ground
        - Long enough for interactive processes to issue I/O request
        - Batch processes still get majority of processor time

## Shortest-Process-First/Next (SPF/N) / Shortest Job First (SJF) Scheduling

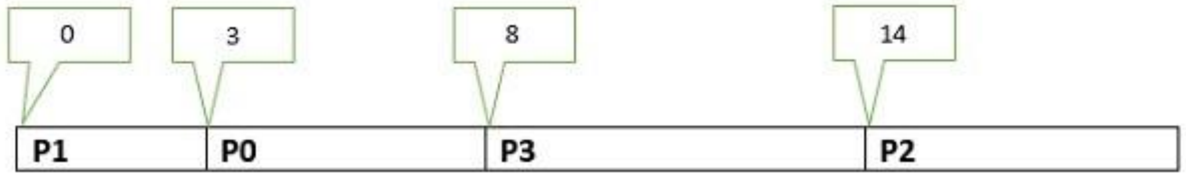
- Scheduler selects process with smallest time to finish
  - Lower average wait time than FIFO
    - Reduces the number of waiting processes
  - Potentially large variance in wait times
  - Non-preemptive
    - Results in slow response times to arriving interactive requests
  - Best approach to minimize waiting time.
  - Relies on estimates of time-to-completion
    - Can be inaccurate or falsified
  - Easy to implement in Batch systems where required CPU time is known in advance.
  - Impossible to implement in interactive systems where required CPU time is not known. The processer should know in advance how much time process will take.
  - Unsuitable for use in modern interactive systems

## Shortest Job First

| Process | Arrival Time | Execute Time |
|---|---|---|
| **P0** | 0 | 5 |
| **P1** | 0 | 3 |
| **P2** | 0 | 8 |
| **P3** | 0 | 6 |

## Gantt Chart

When arrival time of all processes is zero



## Turnaround Time of each process is as follows:

| Process | Turnaround Time |
|---|---|
| P0 | 8 |
| P1 | 3 |
| P2 | 22 |
| P3 | 14 |

Average Turnaround Time: (8+3+22+14) / 4 = 11.75

## Wait Time of each process is as follows:

| Process | Wait Time |
|---|---|
| P0 | 3 |
| P1 | 0 |
| P2 | 14 |
| P3 | 8 |

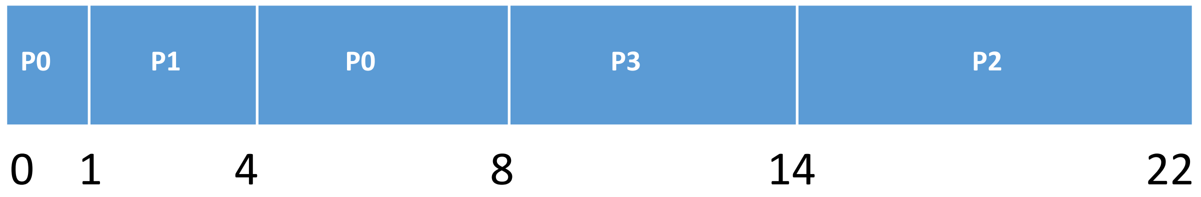
Average Wait Time: (3+14+8) / 4 = 6.25

## Shortest-Remaining-Time (SRT) Scheduling

- SRT scheduling
  - Preemptive version of SPF
  - Shorter arriving processes preempt a running process
  - Very large variance of response times: long processes wait even longer than under SPF
  - Not always optimal
    - Short incoming process can preempt a running process that is near completion
    - Context-switching overhead can become significant

# Shortest Remaining Time

| Process | Arrival Time | Execute Time |
|---|---|---|
| P0 | 0 | 5 |
| P1 | 1 | 3 |
| P2 | 2 | 8 |
| P3 | 3 | 6 |


P0
P1
P0
P3
P2
0
1
4
8
14
22

## Turnaround Time of each process is as follows:

| Process | Turnaround Time |
|---|---|
| P0 | 8-0 = 8 |
| P1 | 4-1 = 3 |
| P2 | 22-2 =20 |
| P3 | 14-3 = 11 |

Average Turnaround Time: (8+3+20+11) / 4 = 10.5

## Wait Time of each process is as follows:

| Process | Wait Time |
|---|---|
| P0 | 4-1 =3 |
| P1 | 0 |
| P2 | 14-2 =12 |
| P3 | 8-3 =5 |

Average Wait Time: (3+12+5) / 4 = 5

## Highest-Response-Ratio-Next (HRRN) Scheduling

- The discrimination towards long jobs in SJF is reduced by the strategy HRRN. Response ratio is the sum of wait time and Job time divided by the Job time. The job with the highest response time is chosen for scheduling. To start with long jobs suffer but as their wait time increases the response time ratio also increases.
- HRRN scheduling
  - Improves upon SPF scheduling
  - Still nonpreemptive
  - Considers how long process has been waiting
  - Prevents indefinite postponement

| Process ID | Arrival Time | Burst Time |
|---|---|---|
| 0 | 0 | 3 |
| 1 | 2 | 5 |
| 2 | 4 | 4 |
| 3 | 6 | 1 |
| 4 | 8 | 2 |

P0 is executed for 3 units

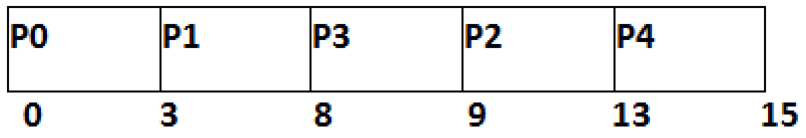
P1 is executed for 5 units

RR (P2) = ((8-4) +4)/4 = 2
RR (P3) = (2+1)/1 = 3
RR (P4) = (0+2)/2 = 1

P3 is scheduled for 1 unit.

RR ( P2) = (5+4)/4 = 2.25
RR (P4) = (1+2)/2 = 1.5

P2 will be scheduled, P4 will be scheduled



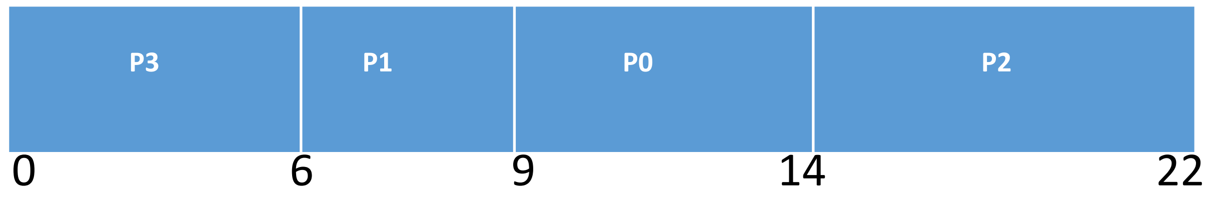
## Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

## Priority Scheduling

- Ready queue is maintained in priority order, where priority of a process is:
  - determined by OS (e.g., run system processes before user processes use the CPU,  average time of last CPU bursts, number of open files, memory size etc. )
  - purchased by user
  - based on corporate policy (e.g. give high priority to some project viewed as very important to company)

| Process | Arrival Time | Execute Time | Priority |
|---|---|---|---|
| P0 | 0 | 5 | 1 |
| P1 | 0 | 3 | 2 |
| P2 | 0 | 8 | 1 |
| P3 | 0 | 6 | 3 |


P3
P1
P0
P2
0
6
9
14
22

## Turnaround Time of each process is as follows:

| Process | Turnaround Time |
|---|---|
| P0 | 14 |
| P1 | 9 |
| P2 | 22 |
| P3 | 6 |

Average Turnaround Time: (14+9+22+6) / 4 =12.75

## Wait Time of each process is as follows:

| Process | Wait Time |
|---|---|
| P0 | 9 |
| P1 | 6 |
| P2 | 14 |
| P3 | 0 |

Average Wait Time: (9+6+14+0) / 4 = 7.25

## Multilevel Queues (MLQ)

- The basic idea is to put different classes of processes in different queues.
- Processes do not move one queue to another. Different queues may follow different policies.
- There should be a policy among queues. ( e.g.  one queue for the foreground processes, another for background processes etc. )

## Multilevel Feedback Queues

- Different processes have different needs
  - Short I/O-bound interactive processes should generally run before processor-bound batch processes
  - Behavior patterns not immediately obvious to the scheduler
- Multilevel feedback queues
  - Arriving processes enter the highest-level queue and execute with higher priority than processes in lower queues
  - Long processes repeatedly descend into lower levels
    - Gives short processes and I/O-bound processes higher priority
    - Long processes will run when short and I/O-bound processes terminate
  - Processes in each queue are serviced using round-robin
    - Process entering a higher-level queue preempt running processes

## Multilevel Feedback Queues

- Algorithm must respond to changes in environment
  - Move processes to different queues as they alternate between interactive and batch behavior
- Example of an adaptive mechanism
  - Adaptive mechanisms incur overhead that often is offset by increased sensitivity to process behavior

# Multilevel Feedback Queues

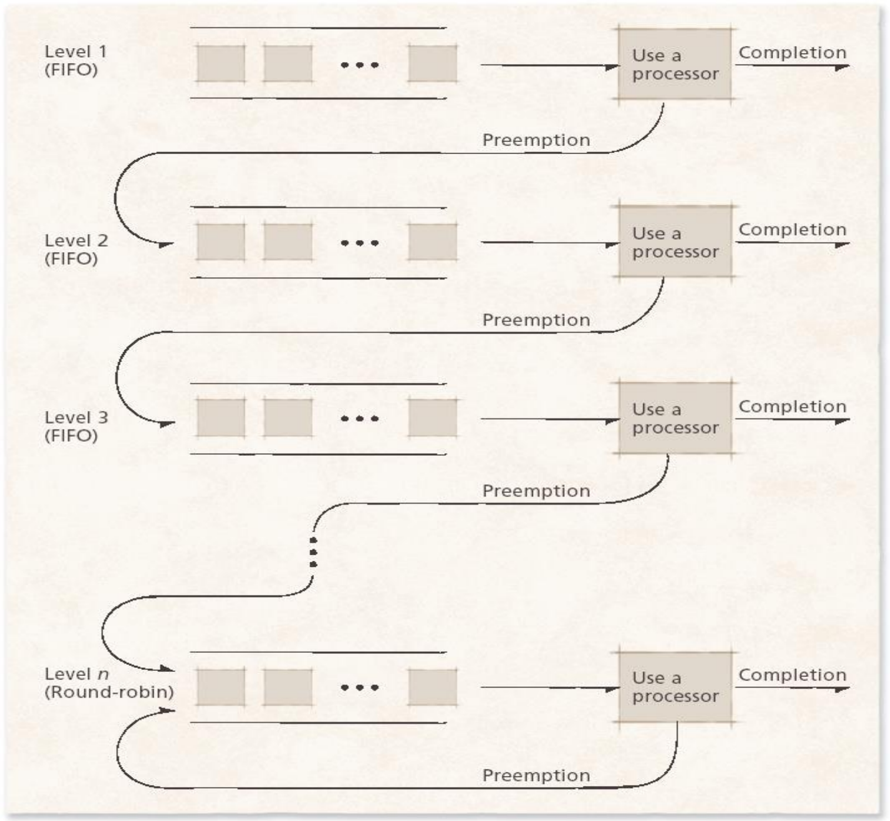


**Table 9.3** Characteristics of Various Scheduling Policies

| | **FCFS** | **Round robin** | **SPN** | **SRT** | **HRRN** | **Feedback** |
|---|---|---|---|---|---|---|
| **Selection function** | max[w] | constant | min[s] | min[s − e] | $\max\left(\frac{w+s}{s}\right)$ | (see text) |
| **Decision mode** | Non-preemptive | Preemptive (at time quantum) | Non-preemptive | Preemptive (at arrival) | Non-preemptive | Preemptive (at time quantum) |
| **Throughput** | Not emphasized | May be low if quantum is too small | High | High | High | Not emphasized |
| **Response time** | May be high, especially if there is a large variance in process execution times | Provides good response time for short processes | Provides good response time for short processes | Provides good response time | Provides good response time | Not emphasized |
| **Overhead** | Minimum | Minimum | Can be high | Can be high | Can be high | Can be high |
| **Effect on processes** | Penalizes short processes; penalizes I/O bound processes | Fair treatment | Penalizes long processes | Penalizes long processes | Good balance | May favor I/O bound processes |
| **Starvation** | No | No | Possible | Possible | No | Possible |

Guaranteed Scheduling:

- System keeps track of how much CPU time each process has had since its creation. It computes the amount of CPU time each is entitled to (time since creation divided by n) and then computes the ratio of actual CPU time consumed to CPU time entitled and run the process with the lowest ratio until its ratio passes its closest competitor.

Lottery Scheduling:

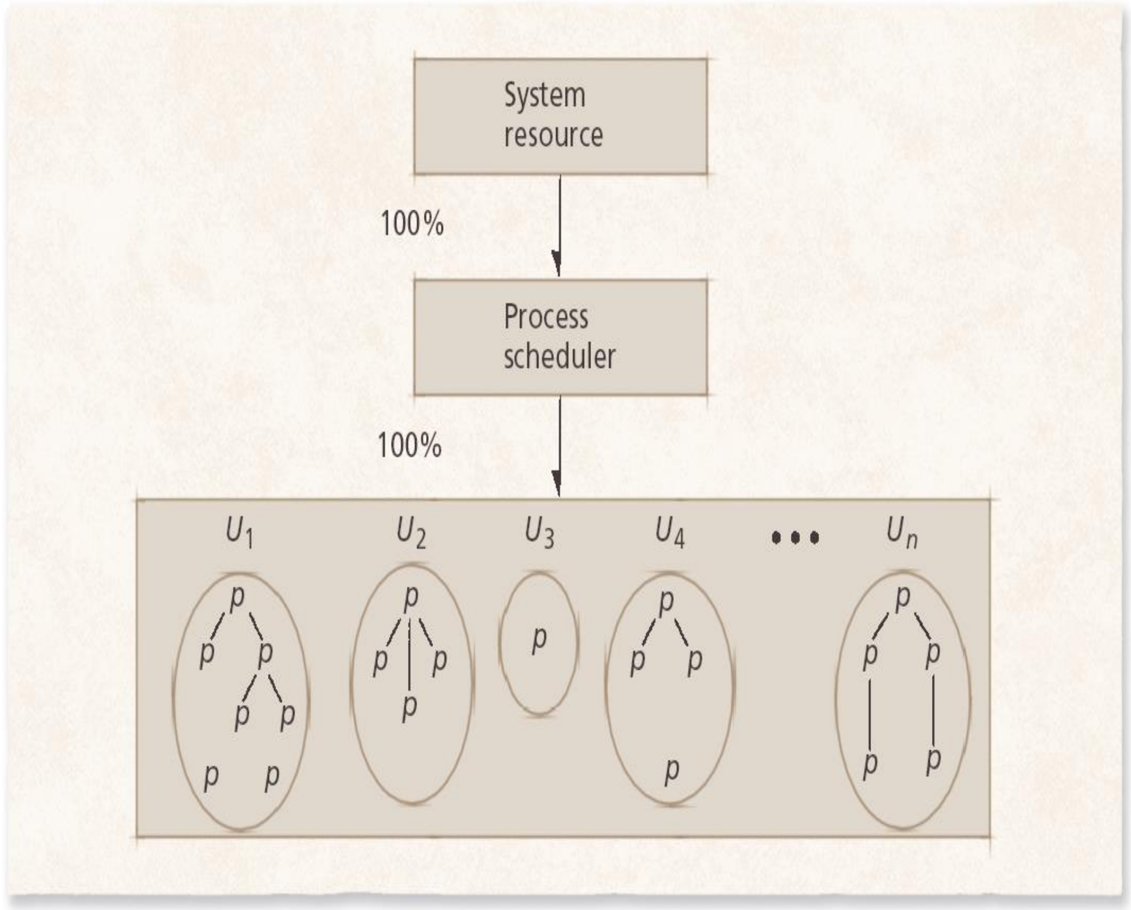
- Processes are given “lottery tickets” for system resources such as CPU time. When scheduling decision needs to be made, lottery ticket is chosen at *random* and the process holding that ticket gets the resource.

# Fair Share Scheduling

- Some systems schedule processes based on usage allocated to each user, independent of the number of processes that user has.
- FSS controls users' access to system resources
  - Some user groups more important than others
  - Ensures that less important groups cannot monopolize resources
  - Unused resources distributed according to the proportion of resources each group has been allocated
  - Groups not meeting resource-utilization goals get higher priority

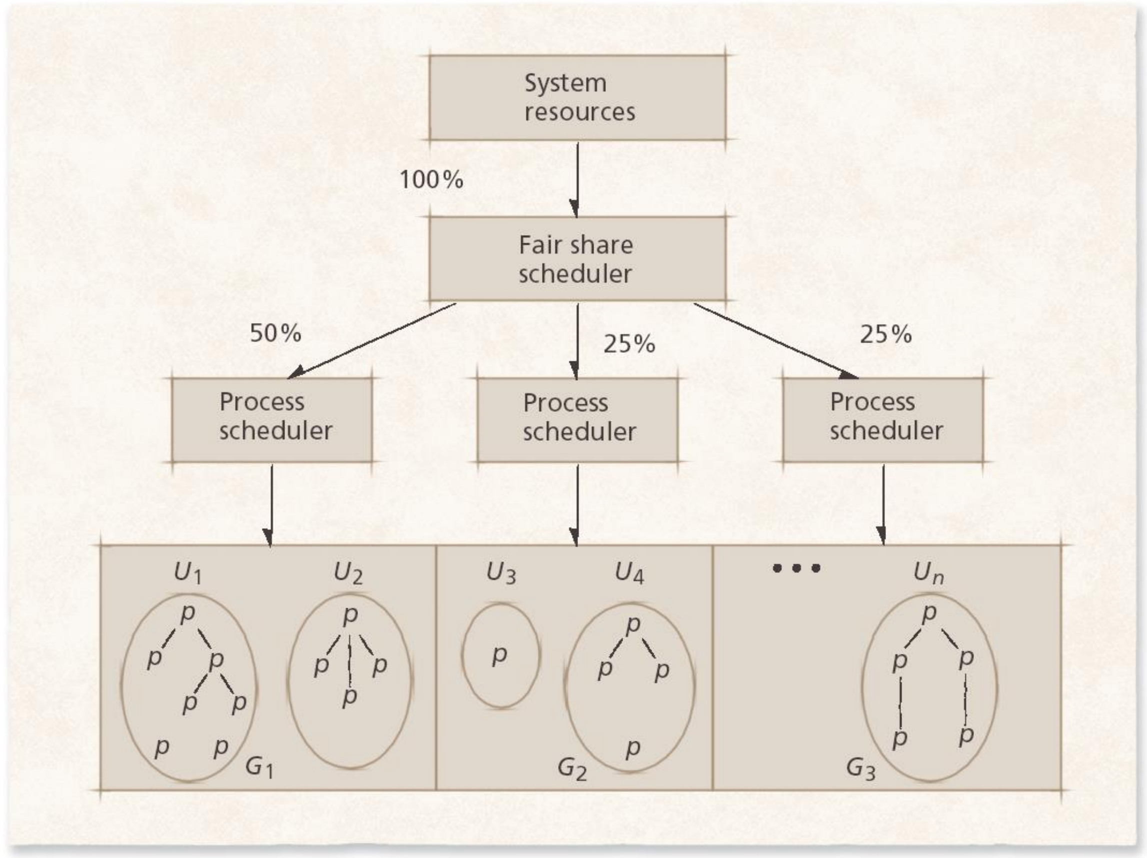
# Fair Share Scheduling

Standard UNIX process scheduler. The scheduler grants the processor to users, each of whom may have many processes.



# Fair Share Scheduling

Fair share scheduler. The fair share scheduler divides system resource capacity into portions, which are then allocated by process schedulers assigned to various fair share groups.



## Deadline Scheduling

- Deadline scheduling
    - Process must complete by specific time
    - Used when results would be useless if not delivered on-time
    - Difficult to implement
        - Must plan resource requirements in advance
        - Incurs significant overhead
        - Service provided to other processes can degrade

# Real-Time Scheduling

- Real-time scheduling
  - Related to deadline scheduling
  - Processes have timing constraints
  - Also encompasses tasks that execute periodically
- Two categories
  - Soft real-time scheduling
    - Does not guarantee that timing constraints will be met
    - For example, multimedia playback
  - Hard real-time scheduling
    - Timing constraints will always be met
    - Failure to meet deadline might have catastrophic results
    - For example, air traffic control

## Real-Time Scheduling

- Static real-time scheduling
  - Does not adjust priorities over time
  - Low overhead
  - Suitable for systems where conditions rarely change
    - Hard real-time schedulers
  - Rate-monotonic (RM) scheduling
    - Process priority increases monotonically with the frequency with which it must execute
  - Deadline RM scheduling
    - Useful for a process that has a deadline that is not equal to its period

# Real-Time Scheduling

- Dynamic real-time scheduling
  - Adjusts priorities in response to changing conditions
  - Can incur significant overhead, but must ensure that the overhead does not result in increased missed deadlines
  - Priorities are usually based on processes' deadlines
    - Earliest-deadline-first (EDF)
      - Preemptive, always dispatch the process with the earliest deadline
    - Minimum-laxity-first
      - Similar to EDF, but bases priority on laxity, which is based on the process's deadline and its remaining run-time-to-completion