Operating Systems

Process Scheduling

Debadatta Mishra, CSE, IITK

Scheduler invocation

- Process enters ready queue → schedule?
- Process termination → Definitely schedule
- Process waits/blocks for an I/O or event → Definitely schedule
- CPU receives an interrupt → After handling, schedule?
- Process exclusively yields the CPU → Definitely schedule

- Schedulers invoked only in second, third and fifth conditions are called non-preemptive or cooperative schedulers

Scheduling strategy

- Objectives
  - Maximize throughput
  - Minimize turnaround time
  - Minimize waiting time in ready queue
  - Minimize response time
- Is the mean of above measures always enough?
- Standard deviation: fairness, predictability

Problem formulation

Process	Arrival Time	CPU bursts	I/O bursts
P1	0	0-3, 7-9, 14-15	3-7,9-14
P2	2	2-10, 12-15	10-12
P3	3	3-4, 10-11	4-10

- Every process goes through a series of CPU and I/O bursts
- Looks complicated, can it be simplified?
- What if each CPU burst is treated as a new process?

First Come First Served (FCFS)

- FIFO queue based non-preemptive scheduling

- Convoy effect
- Not suitable for interactive applications
- Can FCFS be preemptive?

Shortest Job First

- Select the process with shortest CPU burst
- Non-preemptive → Pick the next process only when the current process is finished
- Optimal on waiting time and turnaround time

- Preemptive → Pick the process with shortest remaining time when a new process arrives in the ready queue (SRTF)
- Other preemption points?

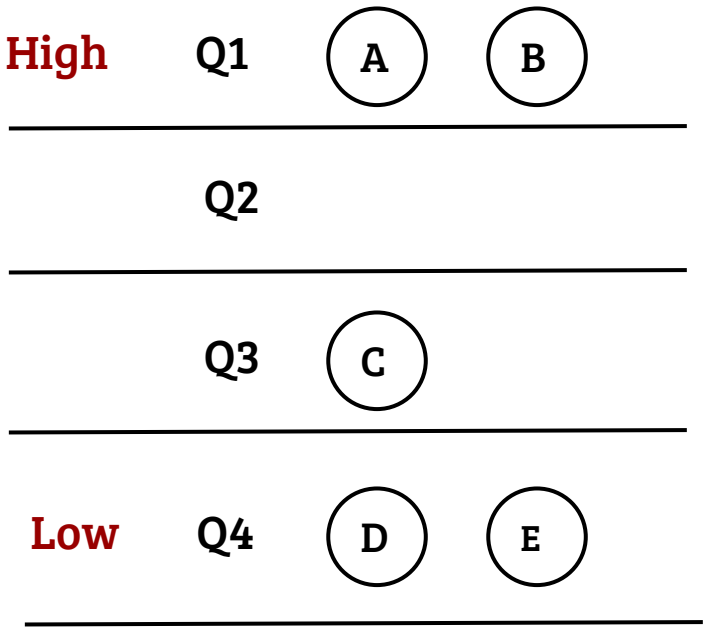
Round-robin scheduling

- Preemptive scheduling with time slicing
- Ready queue is maintained as a circular queue
- At end of the time quantum, If there are other processes in the queue
  - Current process goes to the TAIL of the queue
  - Next process is picked up from the HEAD of the queue
- New processes are added to the TAIL of the queue
- Design choice: size of time quantum

Priority scheduling

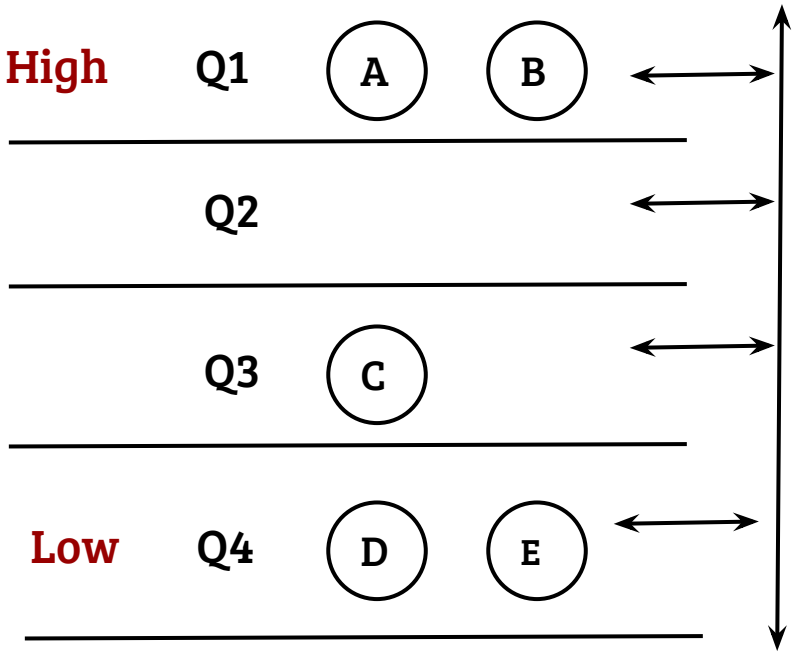
- Select the process with highest priority
- Can be preemptive and non-preemptive
- SJF: priority defined by job length
- Advantages: practical (no assumptions)
- Disadvantages: Starvation
- Solution?

Multilevel queue scheduling



- Processes are assigned to queues based on their priority
- Different queues may implement different schemes
- Process from high priority queue is always chosen
- Issues?

Multilevel feedback queue scheduling



- Process priorities can change dynamically
- But how to change priorities?

Multilevel feedback queue scheduling

- Consider RR scheduling within a queue
- Process is given the highest priority when it enters the system
- If it consumes its time quantum, it is moved to a lower priority. Otherwise remains in the same level

- How does this scheme perform?
  - Interactive applications
  - CPU bound applications
  - I/O bound applications

Multilevel feedback queue

- Possible starvation for CPU bound applications
- Scheduler can be tricked, How?
- What is a good choice for time quantum?

Multilevel feedback queue

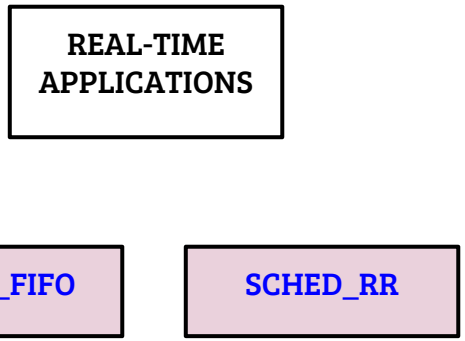
- Possible starvation for CPU bound applications
- Scheduler can be tricked, How?
- What is a good choice for time quantum?

- Periodic priority boost
- Priority lowers when the CPU quantum expires (across multiple scheduling instances)

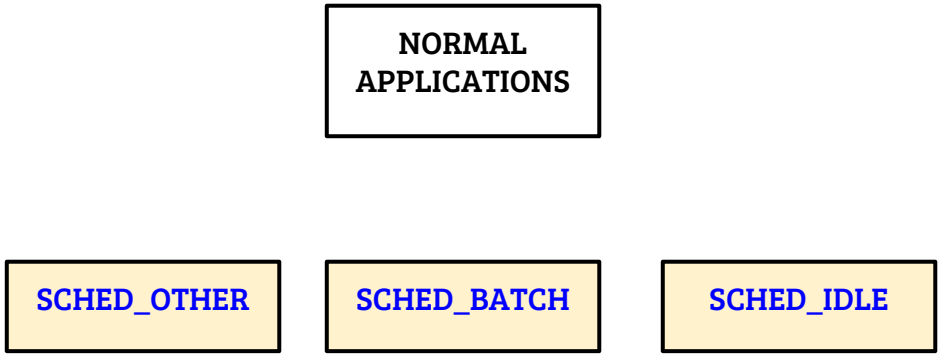
Example: Linux Scheduler

- Meet scheduling need of
  - Real-time processes
  - Interactive processes
  - Batch processes
- General strategy
  - Provide user defined scheduling policies
  - Define priorities (static)
  - But users may be biased, uninformed? So, let the good sense prevail (in kernel).

User control: scheduling classes

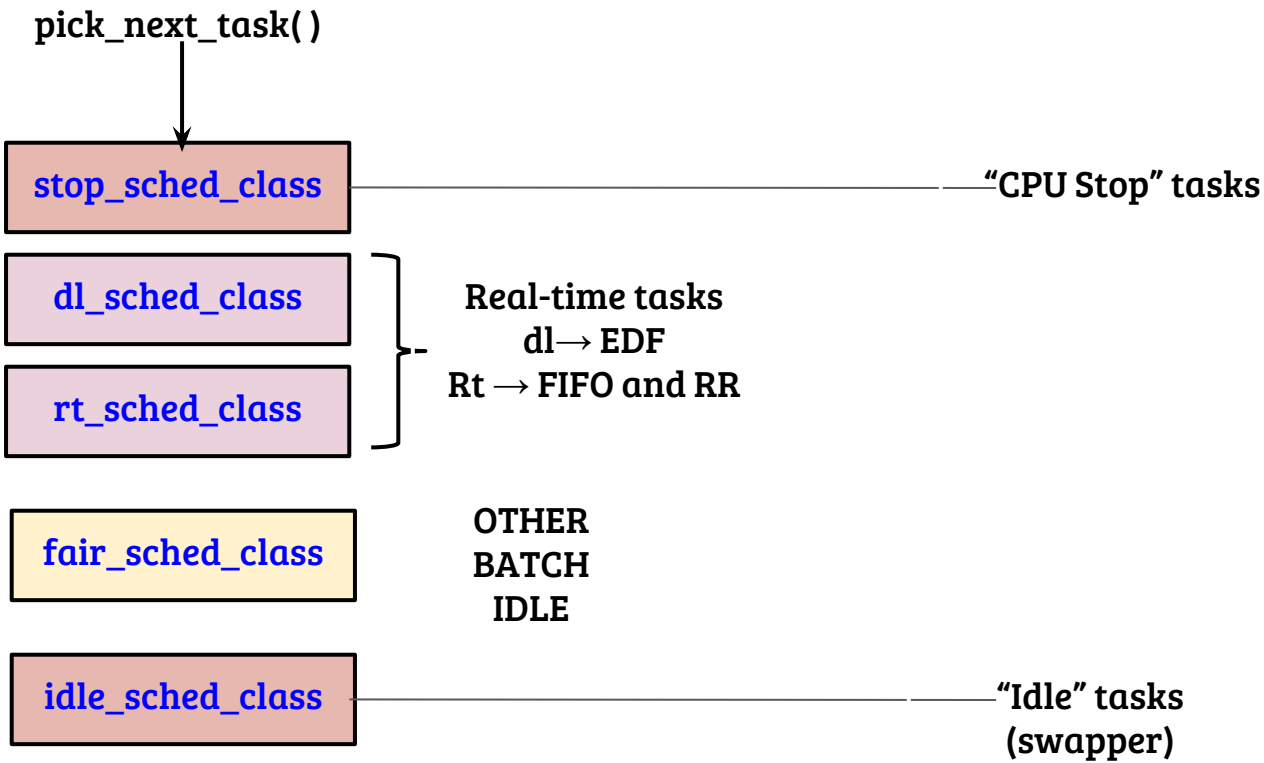


- Always higher priority than normal processes
- Priority value: 1 to 99
- FIFO: run to completion
- RR: Round robin within a priority-level



- SCHED_OTHER is default
- SCHED_BATCH: Assume CPU bound while calculating dynamic priorities
- SCHED_IDLE are for low priority jobs

Scheduling classes (v-4.12.3)



Priority of non-RT processes

- Static priority, Unix “niceness”
  - -20 to 19
  - How “nice” is this process to others?
  - Low value→ not nice to others→ higher priority to myself
- Dynamic priority
  - Calculated by the kernel
  - It is a function of sleep time

Scheduling legacy: O(1) scheduler

- Two sets of run queues, one queue for each priority level
  - Active
  - Expired
- Total 40 dynamic priority levels
  - 40 lists in active and expired
- Select the first task from the highest priority queue
  - Move it to inactive after its time slice is finished
- Waiting tasks (with remaining time slice) come back to active queue
- Swap the lists when active is empty

O(1) scheduler: example