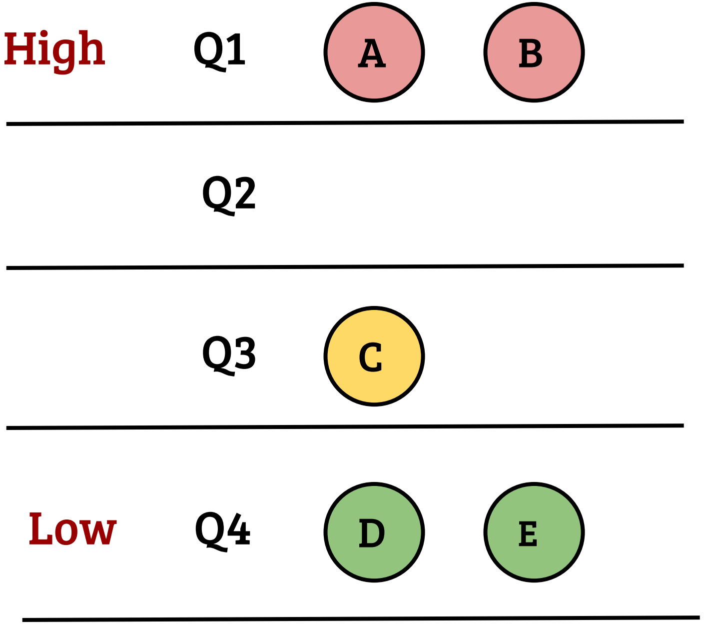
CS330: Operating Systems

Process scheduling policies

Recap: basic scheduling policies

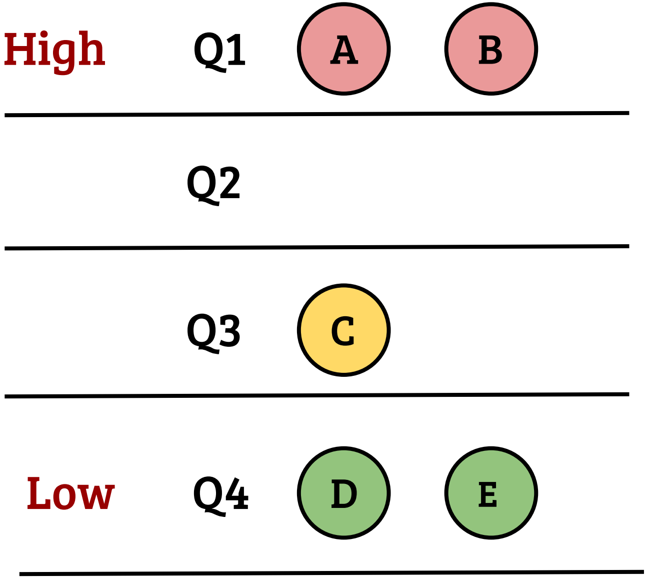
- Scheduling metrics: turnaround time, waiting time, response time
- Fast come first serve (FCFS)
  - Simple but inefficient (convoy effect)
- Shortest job first (SJF) and Shortest time to completion first (STCF)
  - Optimal and efficient. Issues: unrealistic, starvation
- Round robin (RR)
  - Good response time, Issues: scheduling overheads
- Priority scheduling
  - Starvation

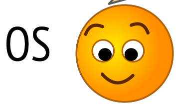
Static priority based scheduling



- Processes are assigned to different queues based on their priority
- Process from the non-empty highest priority queue is always picked
- Different queues may implement different schemes within a queue
- Main concern: Starvation
    - Ex: Low priority processes hug the CPU

Multilevel feedback queue

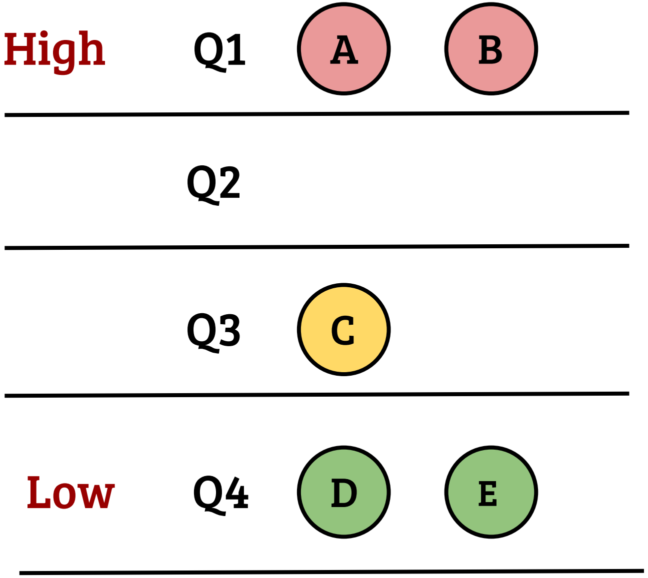




Dynamically adjust priorities such that

1. Interactive applications are responsive
2. Short jobs do not suffer
3. No starvation
4. No user can trick the scheduler

Multilevel feedback queue



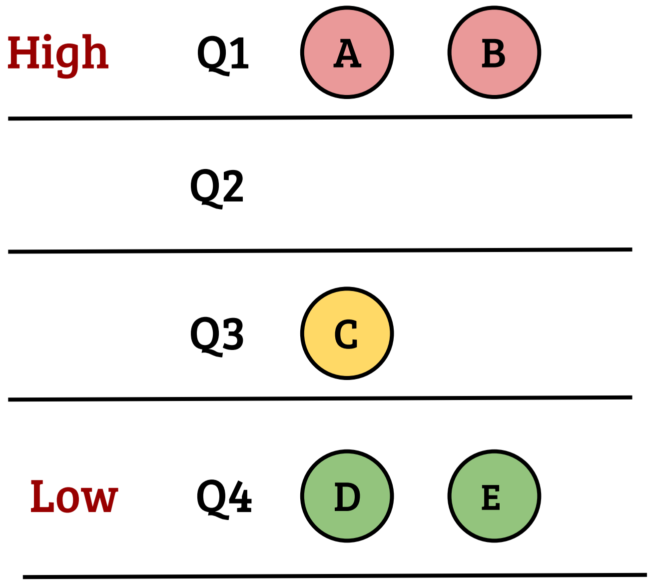
OS 

Dynamically adjust priorities such that

1. Interactive applications are responsive
2. Short jobs do not suffer
3. No starvation
4. No user can trick the scheduler

- Basic multi level strategy
  - Pick a process from highest priority queue
  - Within a queue, apply RR

Multilevel feedback queue: Dynamic priorities



- A process is assigned the highest priority when it is created
- If the process consumes the slice (scheduler invoked because of timer), its priority is reduced
- If the process relinquishes the CPU (I/O wait etc.), its priority remain the same

Multilevel feedback queue: Dynamic priorities

- A process is assigned the highest priority when

- How does this strategy work for short jobs?
- How does the strategy work for interactive jobs?
- Does it avoid starvation?
- Can a user trick the scheduler?

etc.), its priority remain the same

MLFQ: Approximation of SJF

- MLFQ can approximate SJF because
  - Long running jobs are moved to low priority queues
  - New jobs are added to highest priority queue
- A shorter job may not get a chance to execute for a small duration. What is the upper bound?

MLFQ: Approximation of SJF

- MLFQ can approximate SJF because
  - Long running jobs are moved to low priority queues
  - New jobs are added to highest priority queue
- A shorter job may not get a chance to execute for a small duration. What is the upper bound?
- (# of jobs in the highest priority queue + 1) X (time quantum)

Multilevel feedback queue: Dynamic priorities

High Q1 A B

- A process is assigned the highest priority when

- How does this strategy work for short jobs?
- Works nicely, approximates SJF
- How does the strategy work for interactive jobs?
- Does it avoid starvation?
- Can a user trick the scheduler?

etc.), its priority remain the same

MLFQ: Interactive jobs

- MLFQ favors interactive jobs because
  - Interactive jobs maintain the highest priority as they relinquish the CPU before quantum expires
  - Long running jobs are moved to low priority queues

MLFQ: Interactive jobs

- MLFQ favors interactive jobs because
    - Interactive jobs maintain the highest priority as they relinquish the CPU before quantum expires
    - Long running jobs are moved to low priority queues
- Conclusion: In a steady state, interactive jobs compete with short and other interactive jobs

Multilevel feedback queue: Dynamic priorities

- How does this strategy work for short jobs?
- Works nicely, approximates SJF
- How does the strategy work for interactive jobs?
- Works pretty well as interactive jobs retain priority
- Does it avoid starvation?
- Can a user trick the scheduler?

MLFQ: Starvation and other issues

- Long running processes may starve with the proposed scheme
- Additionally, permanent demotion of priority hurts processes which change their behavior
  - Example: A process performing a lot of computation only at start gets pushed to a low priority queue permanently
- How to avoid the above issues?

MLFQ: Starvation and other issues

- Long running processes may starve with the proposed scheme
- Additionally, permanent demotion of priority hurts processes which change their behavior
  - Example: A process performing a lot of computation only at start gets pushed to a low priority queue permanently
- How to avoid the above issues?
  - Periodic priority boost: all processes moved to high priority queue
  - Priority boost with aging: recalculate the priority based on scheduling history of a process

Multilevel feedback queue: Dynamic priorities

- How does this strategy work for short jobs?
- Works nicely, approximates SJF
- How does the strategy work for interactive jobs?
- Works pretty well as interactive jobs retain priority
- Does it avoid starvation?
- No. Requires additional mechanism like priority boost.
- Can a user trick the scheduler?

MLFQ: The tricky user

- A smart user can maintain highest priority for long running processes by exploiting the scheduling strategy. How?

MLFQ: The tricky user

- A smart user can maintain highest priority for long running processes by exploiting the scheduling strategy. How?
- Assumption: user knows the time quantum

MLFQ: The tricky user

- A smart user can maintain highest priority for long running processes by exploiting the scheduling strategy. How?
- Assumption: user knows the time quantum
- Strategy: Voluntarily release the CPU before time quantum expires
- Result: Batch process competes with other interactive processes!

MLFQ: The tricky user

- A smart user can maintain highest priority for long running processes by exploiting the scheduling strategy. How?
- Assumption: user knows the time quantum
- Strategy: Voluntarily release the CPU before time quantum expires
- Result: Batch process competes with other interactive processes!
- Core of the issue: binary history regarding a process

MLFQ: The tricky user

- A smart user can maintain highest priority for long running processes by exploiting the scheduling strategy. How?
- Assumption: user knows the time quantum
- Strategy: Voluntarily release the CPU before time quantum expires
- Result: Batch process competes with other interactive processes!
- Core of the issue: binary history regarding a process
    - MLFQ: Process consumed or not consumed the quantum
    - Advanced MLFQ: Better accounting, variable quantums

Multilevel feedback queue: Dynamic priorities

- How does this strategy work for short jobs?
- Works nicely, approximates SJF
- How does the strategy work for interactive jobs?
- Works pretty well as interactive jobs retain priority
- Does it avoid starvation?
- No. Requires additional mechanism like priority boost.
- Can a user trick the scheduler?
- Yes. Additional history regarding execution is required to be maintained