Scheduling

Algorithms

Multilevel Feedback Queueing

Scheduling

Algorithms

Multilevel Feedback Queueing

Can be used when we have no estimates on run times

Scheduling

Algorithms

Multilevel Feedback Queueing

Can be used when we have no estimates on run times

- There are multiple FIFO run queues, RQ_0, RQ_1, ... RQ_n. with RQ_0 the highest priority, RQ_n, the lowest

Scheduling

Algorithms

Multilevel Feedback Queueing

Can be used when we have no estimates on run times

- There are multiple FIFO run queues, RQ_0, RQ_1, ... RQ_n. with RQ_0 the highest priority, RQ_n, the lowest
- Queues are processed in FIFO fashion, in priority order, so RQ_1 does not get a look-in until RQ_0 has emptied

Scheduling

Algorithms

Multilevel Feedback Queueing

Can be used when we have no estimates on run times

- There are multiple FIFO run queues, RQ_0, RQ_1, ... RQ_n. with RQ_0 the highest priority, RQ_n, the lowest
- Queues are processed in FIFO fashion, in priority order, so RQ_1 does not get a look-in until RQ_0 has emptied
- Each process is allocated a *quantum* of time (a timeslice)

Scheduling

Algorithms

Multilevel Feedback Queueing

Can be used when we have no estimates on run times

- There are multiple FIFO run queues, RQ_0, RQ_1, ... RQ_n. with RQ_0 the highest priority, RQ_n, the lowest
- Queues are processed in FIFO fashion, in priority order, so RQ_1 does not get a look-in until RQ_0 has emptied
- Each process is allocated a *quantum* of time (a timeslice)
- A new process is admitted to the end (last) of RQ_0

Scheduling

Algorithms

Multilevel Feedback Queueing

Can be used when we have no estimates on run times

- There are multiple FIFO run queues, RQ_0, RQ_1, ... RQ_n. with RQ_0 the highest priority, RQ_n, the lowest
- Queues are processed in FIFO fashion, in priority order, so RQ_1 does not get a look-in until RQ_0 has emptied
- Each process is allocated a *quantum* of time (a timeslice)
- A new process is admitted to the end (last) of RQ_0
- When the running process has used its quantum of time, it is interrupted and placed at the end of the next lower queue (demoted)

Multilevel Feedback Queueing

- If the running process relinquishes voluntarily before the end of the quantum, it gets placed back at the end of the *same* queue

Multilevel Feedback Queueing

- If the running process relinquishes voluntarily before the end of the quantum, it gets placed back at the end of the *same* queue
- If it blocks for I/O, it will be promoted and placed at the end of the next higher queue (when ready to run)

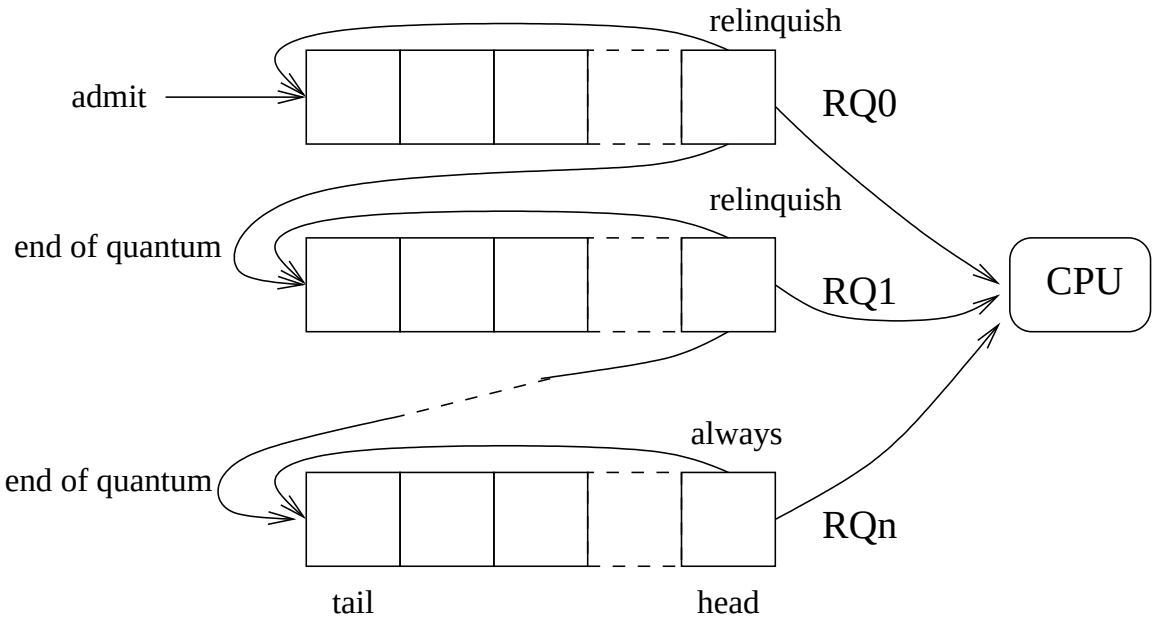
Scheduling

Algorithms

Multilevel Feedback Queueing

- If the running process relinquishes voluntarily before the end of the quantum, it gets placed back at the end of the *same* queue
- If it blocks for I/O, it will be promoted and placed at the end of the next higher queue (when ready to run)
- Demoted processes in RQ_n get placed back at the end of RQ_n

Multilevel Feedback Queueing



Multilevel Feedback Queueing

Scheduling

Algorithms

Multilevel Feedback Queueing

- This gives newer, shorter processes priority over older, longer ones

Scheduling

Algorithms

Multilevel Feedback Queueing

- This gives newer, shorter processes priority over older, longer ones
- I/O processes tend to rise, getting more priority

Scheduling

Algorithms

Multilevel Feedback Queueing

- This gives newer, shorter processes priority over older, longer ones
- I/O processes tend to rise, getting more priority
- Compute processes tend to sink, getting less

Scheduling

Algorithms

Multilevel Feedback Queueing

- This gives newer, shorter processes priority over older, longer ones
- I/O processes tend to rise, getting more priority
- Compute processes tend to sink, getting less

Old processes tend to starve with this, so a variant doubles the quantum for each level: RQ_0 gets 1, RQ_1 gets 2, RQ_2 gets 4, and so on

Scheduling

Algorithms

Multilevel Feedback Queueing

- This gives newer, shorter processes priority over older, longer ones
- I/O processes tend to rise, getting more priority
- Compute processes tend to sink, getting less

Old processes tend to starve with this, so a variant doubles the quantum for each level: RQ_0 gets 1, RQ_1 gets 2, RQ_2 gets 4, and so on

So compute intensive processes get a big bite, whenever they get a chance, at the potential cost of responsiveness to a new process

Scheduling

Algorithms

Another advantage of MFQ is that it does not need to do any arithmetic: it just moves processes between queues

□

Scheduling

Algorithms

Another advantage of MFQ is that it does not need to do any arithmetic: it just moves processes between queues

Remember, in early machines, arithmetic was a lot more time-consuming than it is now

Scheduling

Algorithms

Another advantage of MFQ is that it does not need to do any arithmetic: it just moves processes between queues

Remember, in early machines, arithmetic was a lot more time-consuming than it is now

This scheme was used by Windows NT and Unix derivatives, as we shall see next