Review – we considered two scheduling metrics:

Context: we generally have two types of jobs:

Today’s Goal:

Algorithm:

How do we choose which job to run?
Rule 1:
Rule 2:

See a problem?

How do we fix?
Rule 5:

Ok that’s up what about ________
Rule 4:
What’s Missing?

Rule 3:

Can you explain how these five rules work together to provide good response time?

Why does this algorithm work well?

- Prioritizes short jobs yielding fast response time for I/O bound jobs
- New jobs start at highest level – ensures immediate initial response
- If a job uses little bursts of CPU it’s interactive (and consumes its allotment slowly)
- If a job doesn’t give up the CPU it’s probably CPU bound – move down for big long turn
- Rule 4’s allotment prevents process from gaming the system by doing an I/O just before their time slice ends, keeping the process in the highest priority queue.
- Rule 5 prevents too many interactive jobs from starving non-interactive jobs
- Jobs may vacillate between interactive or non-interactive (Rule 5 also account for this)

Design Issues

- How many queues?
- Time slice per queue? Why might longer time slice for low priority queues make sense?
- Frequency of priority boost? (Alas, setting $S$ can be hard. No answer that is always right.)
- Might reserve highest priority for OS
- Allow user to give the OS advice for a job’s priority [man nice]

Summary:

MLFQ allows us to have good response and turnaround time without being omniscient

It uses the following rules:

1. If Priority(A) > Priority(B), run A
2. If Priority(A) = Priority(B), run with RR
3. New jobs start at highest priority
4. Once a job uses up its time allotment at its priority level, it is move down a queue
5. After time $S$, all jobs move back to highest priority

Actual OSes that you have used, use a form of this scheduler!