LECTURE 13: SCHEDULING ALGO-2

Chapter 3 of Operating System Concepts- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne

Chapter 7 of OSTEP

Session Outline

- ❖ CPU Scheduling
- ❖ Limitations of Priority Scheduling
- ❖ Priority Inversion
- ❖ Multilevel Feedback Queue Scheduling
- ❖ Lottery Scheduling
- ❖ Summary of Scheduling algorithms

Scheduling Criteria

- Different CPU-scheduling algorithms have different properties.
- Certain characteristics/criteria are used for comparing various CPU scheduling algorithms.
  - CPU Utilization
  - Throughput
  - Turnaround time
  - Waiting Time
  - Response Time

CPU Scheduling Algorithms

Batch Systems

- First-come first-served
- Shortest job first
- Shortest remaining Time next

Interactive Systems

- Round-robin scheduling
- Priority scheduling
- Multiple queues
- Shortest process next
- Guaranteed scheduling
- Lottery scheduling
- Fair-share scheduling

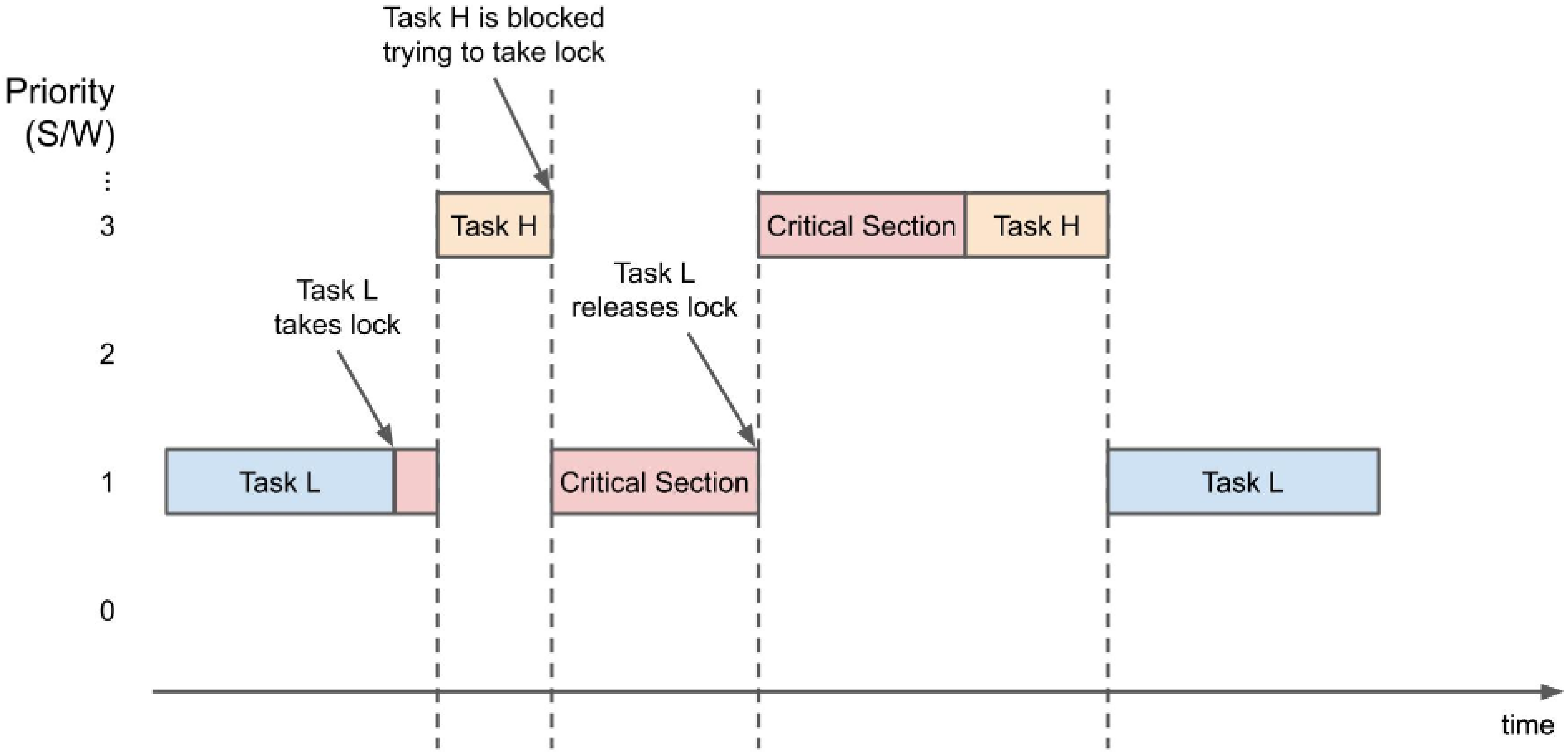
Priority Scheduling

- Each process has a priority number (integer)
- Lower the integer, higher the priority
- Highest priority process is scheduled first; if equal priorities, then FCFS
- It can have 2 variants; non-preemptive and preemptive
- Arrival of a new process with a higher priority can preempt the currently running process.

Issues with Priority Scheduling

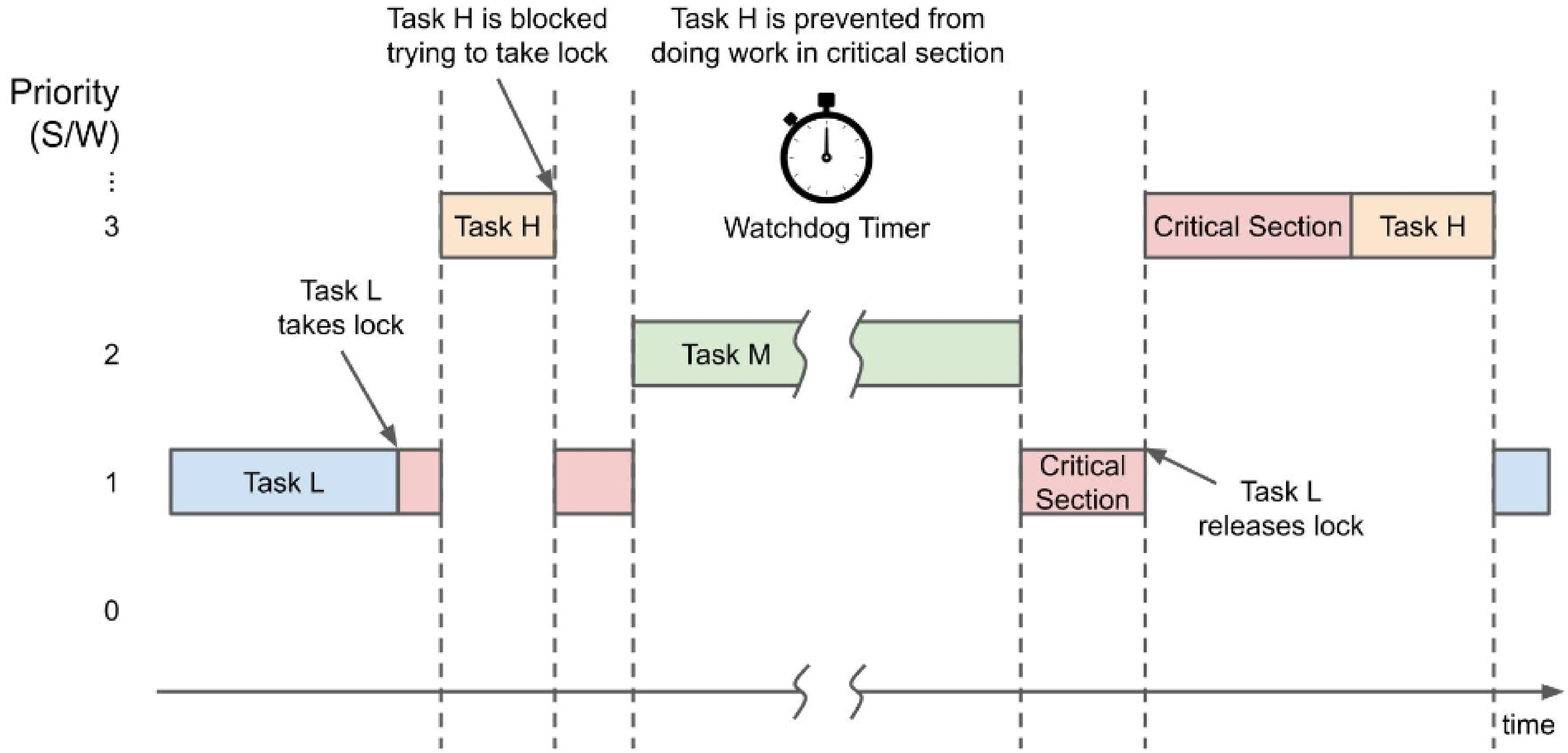
- Priority Inversion
  - Priority inversion is a bug that occurs when a high priority task is indirectly preempted by a low priority task.
  - For example, the low priority task holds a mutex that the high priority task must wait for to continue executing.

Bounded Priority Inversion



As you can see in the diagram above, Task H is blocked so long as Task L holds the lock. The priority of the tasks have been indirectly “inverted” as now Task L is running before Task H.

Unbounded Priority Inversion



Unbounded priority inversion occurs when a medium priority task (Task M) interrupts Task L while it holds the lock. It's called "unbounded" because Task M can now effectively block Task H for any amount of time, as Task M is preempting Task L (which still holds the lock).

Solution To Unbounded Priority Inversion

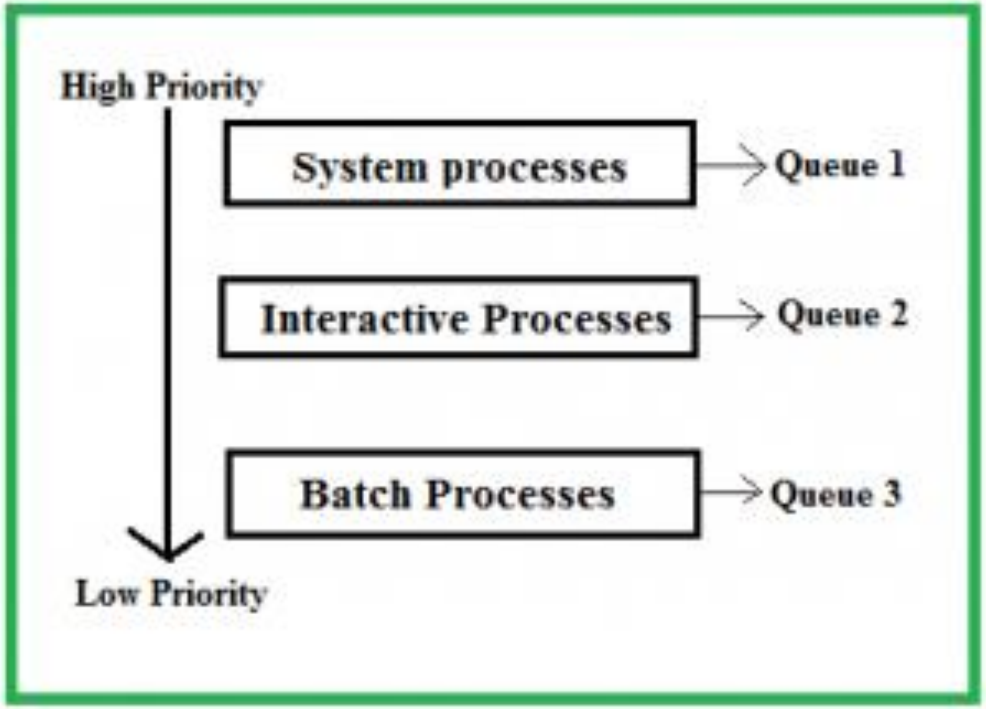
- Process L should, in fact, be temporarily of higher priority than process M, on behalf of process H.
- Process H can donate its priority to process L, which, in this case, would make it higher priority than process M.
- This enables process L to preempt process M and run.
- When process L is finished, process H becomes unblocked.
- Process H, now being the highest priority ready process, runs, and process M must wait until it is finished.

Multilevel Queue

- ❖ Ready queue is partitioned into separate queues:
  - ❖ Foreground, interactive process→ RR scheduling
  - ❖ Background, batch process→ FCFS scheduling
- ❖ A process is permanently assigned to one queue
- ❖ Each queue has its own scheduling algorithm
- ❖ Can be preemptive

Multilevel Feedback Queue Scheduling

- Scheduling must be done between the queues.
  - Fixed priority scheduling
  - Serve all from foreground then from background
  - Possibility of starvation
- Time slice
  - Each queue gets a certain amount of CPU time which it can schedule among its processes
    - i.e.: 80% Vs 20%

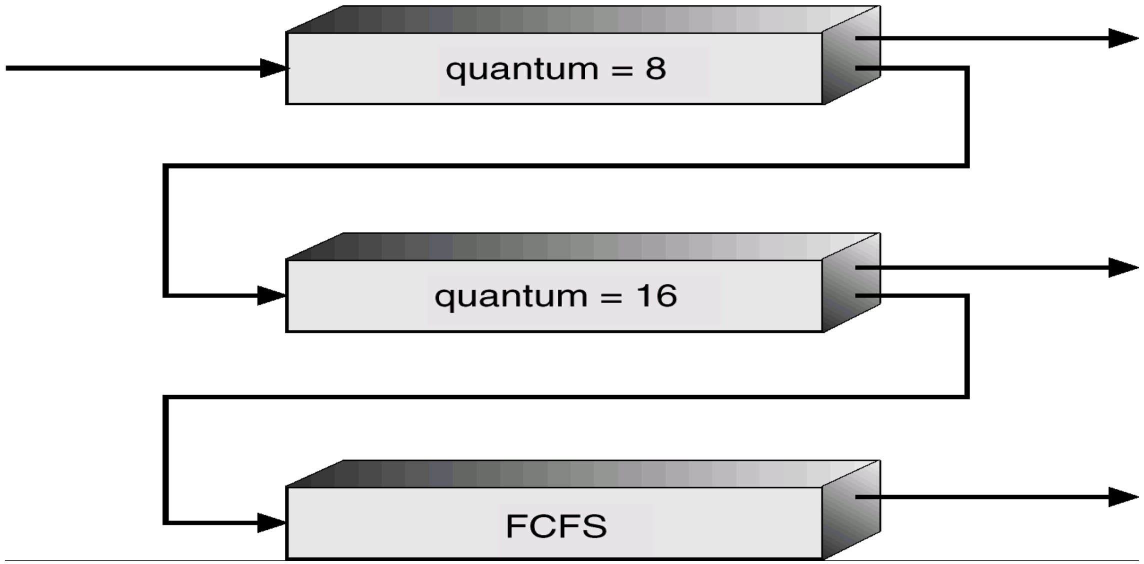


Multilevel Feedback Queue Scheduling

- **A process can move between the various queues. (Aging)**
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q0 – time quantum 8 milliseconds, FCFS
  - Q1 – time quantum 16 milliseconds, FCFS
  - Q2 – FCFS



- A new job enters queue Q0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q1.
- At Q1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q2.

Lottery Scheduling

- Each job some number of lottery tickets are issued
- On each time slice, randomly pick a winning ticket
- On average, CPU time is proportional to number of tickets given to each job over time
- How to assign tickets?
  - To approximate SRTF, short-running jobs get more, long running jobs get fewer
  - To avoid starvation, every job gets at least one ticket (everyone makes progress)

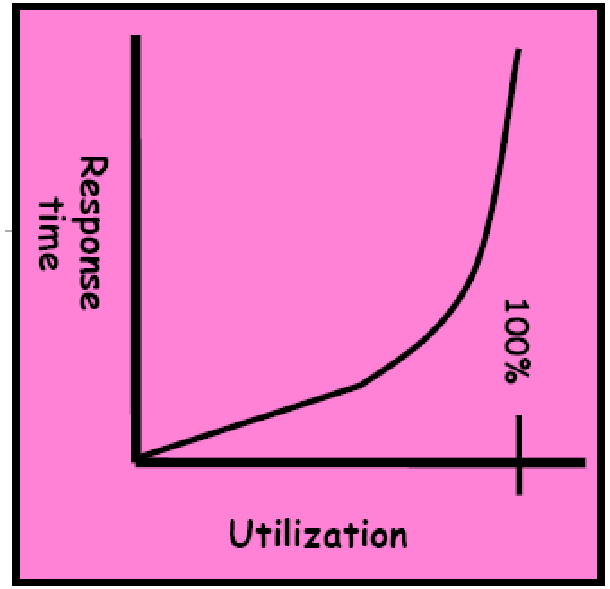
Example: Lottery Scheduling

- Assume short jobs get 10 tickets, long jobs get 1 ticket

# short jobs / # long jobs	% of CPU each short job gets	% of CPU each long job gets
1/1	91%	9%
0/2	N/A	50%
2/0	50%	N/A
10/1	9.9%	0.99%
1/10	50%	5%

Conclusion

- Scheduling: selecting a waiting process from the ready queue and allocating the CPU to it
- When do the details of the scheduling policy and fairness really matter?
  - When there aren't enough resources to go around



Conclusion

- FCFS scheduling, FIFO Run Until Done:
  - Simple, but short jobs get stuck behind long ones
- RR scheduling:
  - Give each thread a small amount of CPU time when it executes, and cycle between all ready threads
  - Better for short jobs, but poor when jobs are the same length
- SJF/SRTF:
  - Run whatever job has the least amount of computation to do / least amount of remaining computation to do
  - Optimal (average response time), but unfair; hard to predict the future

Conclusion

- Multi-Level Feedback Scheduling:
  - Multiple queues of different priorities
  - Automatic promotion/demotion of process priority to approximate SJF/SRTF
- Lottery Scheduling:
  - Give each thread a number of tickets (short tasks get more)
  - Every thread gets tickets to ensure forward progress / fairness
- Priority Scheduling:
  - Preemptive or Non-preemptive
  - Priority Inversion

Thank You