CSc 360
Operating Systems
CPU Scheduling

Wenjun Yang

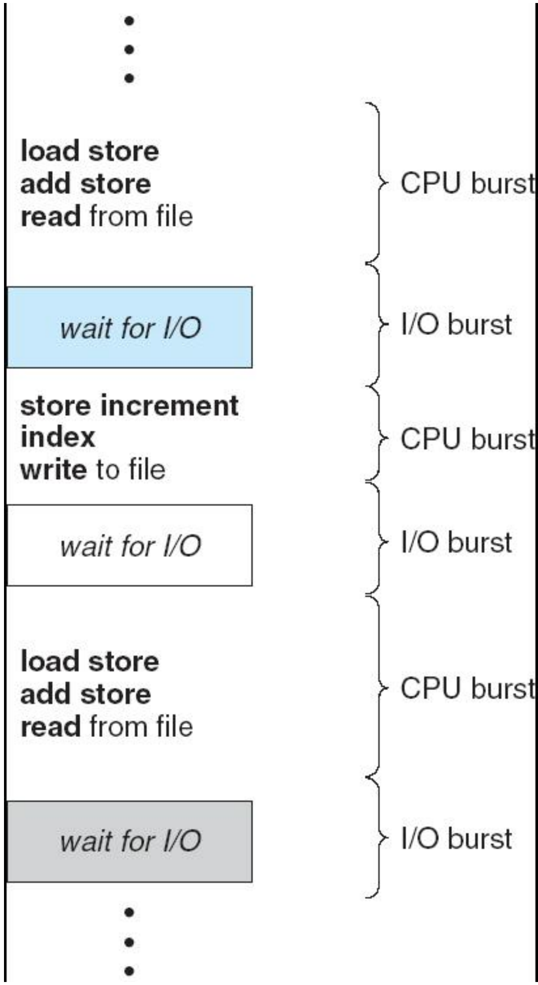
Fall 2025

Review: process and thread

- Uniprogramming
- Multiprogramming
- Multitasking (multiprocessing)
- Multithreading
- How to handle ***many*** processes/threads?
  - state: ready, running, blocked
  - scheduling (PCB, TCB)

CPU scheduling

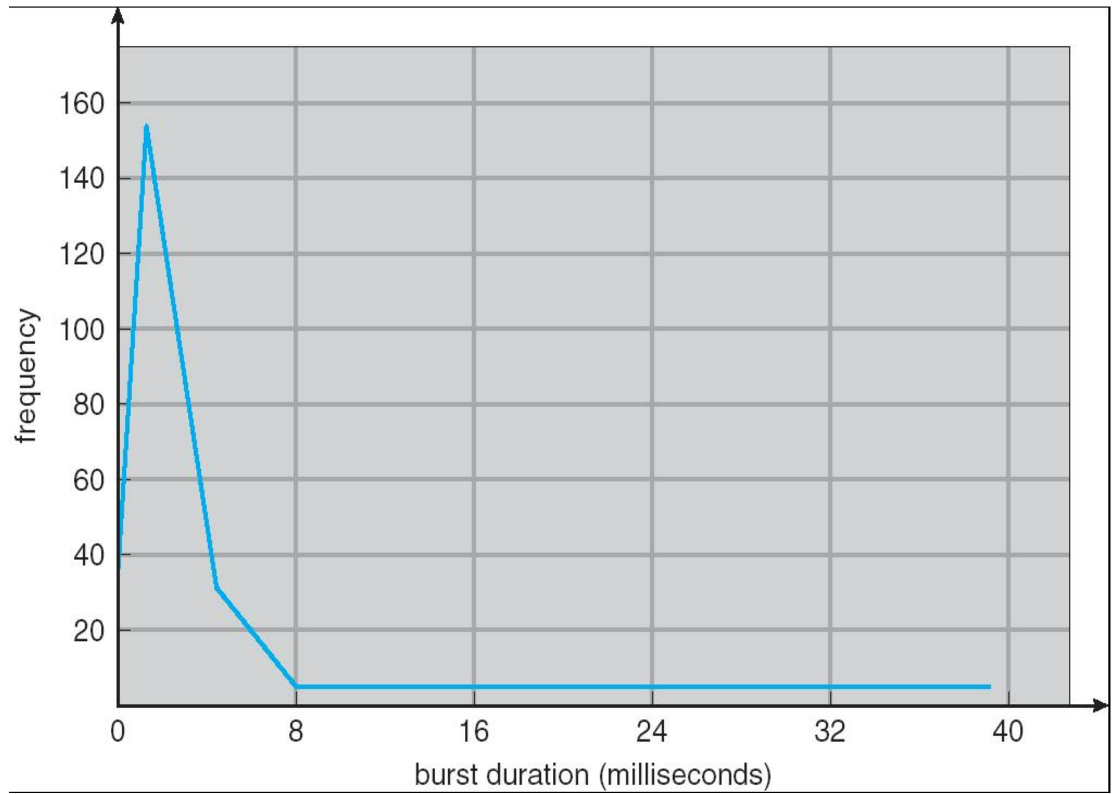
- Goal
  - maximize resource utilization
    - CPU, memory, storage
  - improve system responsiveness
- Example
  - CPU burst
  - I/O burst



Q: why scheduling?

CPU burst distribution

- Observation
  - many short bursts
  - a few long bursts
- Why is this important?



CPU scheduler

- CPU scheduler
  - short-term scheduling: *who's next*?
- CPU scheduling
  - switch from running to waiting (blocked)
  - switch between running and ready
  - switch from waiting to ready
  - terminate (i.e., leave the system)
- Preemptive vs non-preemptive

state diagram on blackboard

CPU dispatcher

- Dispatcher
  - give control to the one selected by scheduler
- Procedures
  - context switching (save old, load new, etc)
  - mode switching (e.g., switch to user mode)
  - start to execute from the newly loaded PC
- Performance measure
  - dispatch latency/overhead

context switching diagram

Scheduling criteria

- “Who’s next?”
  - CPU utilization: keep CPU as busy as possible
  - throughput: # of processes done per unit time
  - turnaround time: from start to finish
  - waiting time: time spent in ready state
  - response time: interactive, request-reply
- Goal
  - max {…}, min {…}

Q: conflicting goals?

Scheduling algorithms

- First come, first serve (FCFS)
  – served by the order of arrival
- Example
  – P1(24), P2(3), P3(3)
  – schedule A (unlucky arrival order)
    - P1, P2, P3
  – schedule B (lucky arrival order)
    - P2, P3, P1

Gantt chart on blackboard

Q: which schedule is better?

This lecture so far

- CPU scheduling
  – who's who
    - scheduler, dispatcher
  – who's next
    - FCFS (and many more)

- Explore further
  – what's the best way (data structure + algorithm) to implement an FCFS scheduler?

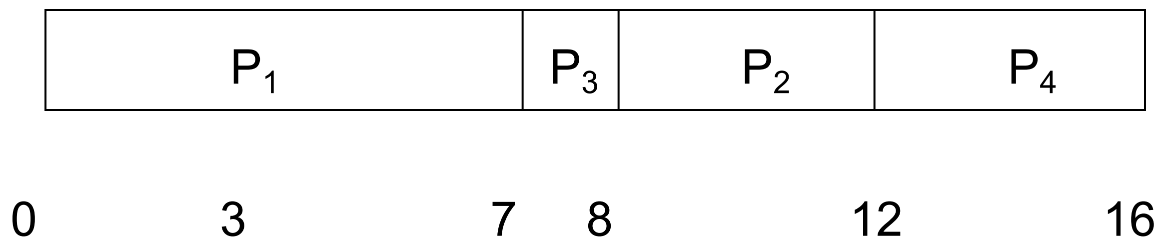
Shortest job first

- SJF: based on the length of *next* CPU burst
  – non-preemptive
    - the job with the smallest burst length scheduled first
  – or preemptive
    - i.e., always the shortest remaining time first
- SJF is optimal in average waiting time
  – reduce the total waiting time for all jobs
  – why ***is*** SJF optimal?

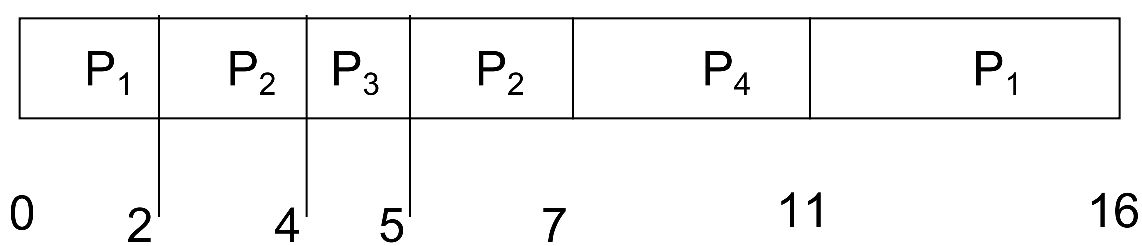
Q: possible problems with SJF?

SJF: example

- Example
  - P1: 0 (arrival time); 7 (burst time)
  - P2: 2; 4
  - P3: 4; 1
  - P4: 5; 4



- Non-preemptive
  - P1, P3, P2, P4



- Preemptive
  - P1, P2, P3, P2, P4, P1

Shortest Remaining Time First (SRTF)

Q: turnaround time?

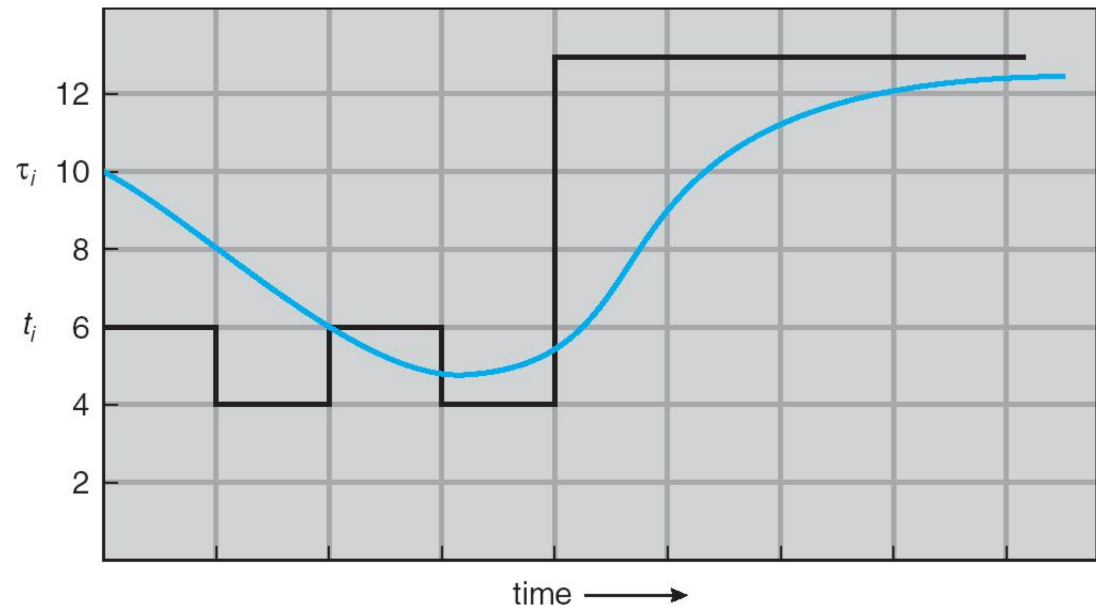
SJF: more

- Determine the *next* burst length
  - how to predict the future?
    - if history is of any indication …
  - use the last burst length
  - use the average so far
  - use the moving average
  - use the weighted moving average
  - the exponentially weighted moving average
    - i.e., $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$.

Q: storage requirement?

EWMA: example

- Exponentially weighted moving average
  - τ_0=10
  - α = 0.5
    - normally (0,1)



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Q: when α=0 or 1?

Priority scheduling

- Priority
  - the job with the highest priority scheduled first
    - SJF: shorter CPU burst, higher priority
    - FCFS: arrival earlier, higher priority
  - static priority: starvation
    - e.g., SJF
  - dynamic priority
    - e.g., aging
- Non-preemptive vs preemptive

Round-robin scheduling

- Discrete processor sharing
  - CPU time quantum
    - usually 10~100 ms
  - for a process
    - either yield after a CPU burst
    - or be preempted after using up a time quantum
  - a FIFO queue
    - all ready processes
- Weighted round-robin

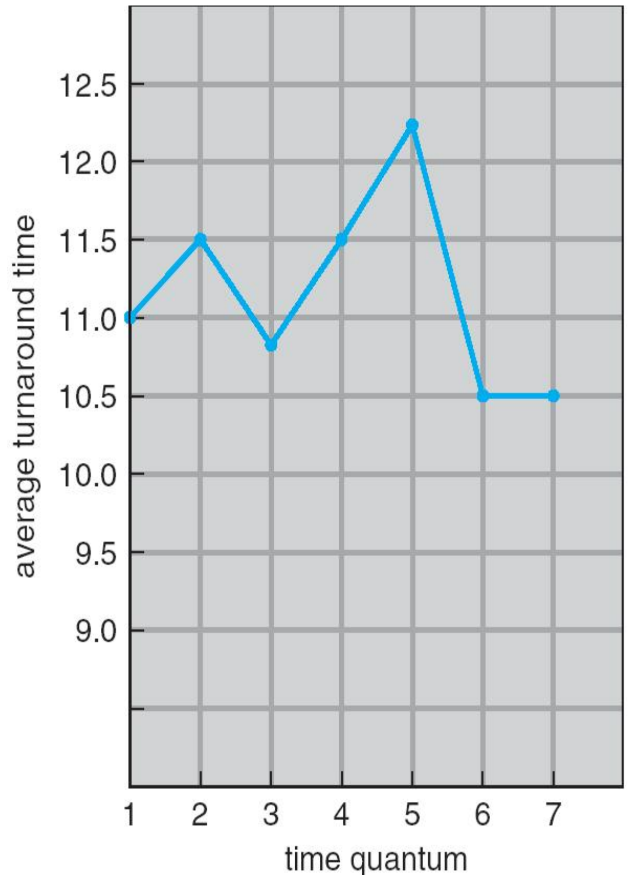
RR: example

- Example
  - P1: 0 (arrival time); 7 (burst time)
  - P2: 2; 4
  - P3: 4; 1
  - P4: 5; 4
- Time quantum
  - e.g., 1 quantum = 1 time unit
  - how about 1 quantum = 4 time units

Q: turnaround time?

Time quantum

- Large quantum
  - => FCFS
- Small quantum
  - better responsiveness
  - be aware of overhead
    - context switching
  - "80%" rule



process	time
P_1	6
P_2	3
P_3	1
P_4	7

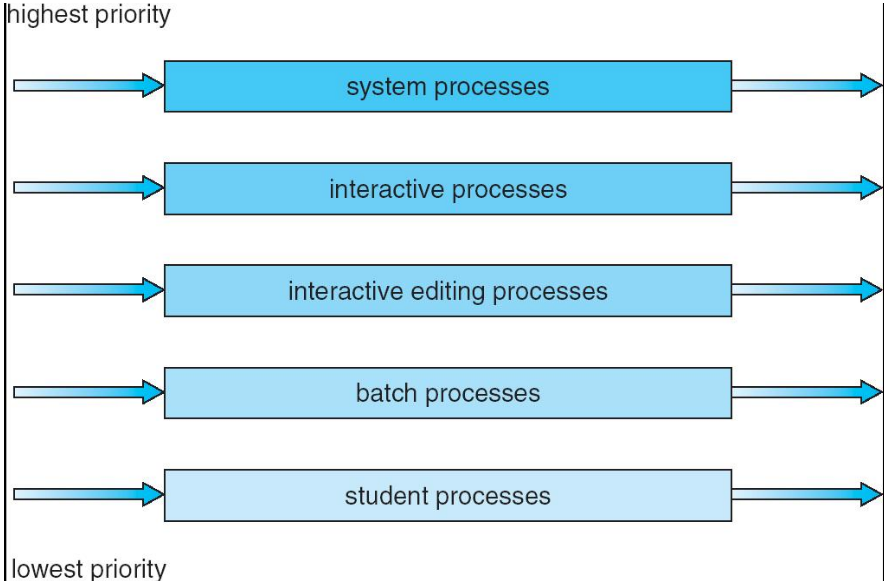
The 2nd half of this lecture

- Scheduling algorithms
  - FCFS
  - SJF, priority, RR
  - preemptive and non-preemptive
- Explore further
  - evaluate average waiting time, average turnaround time per unit job for these algorithms
  - Q's on slides and in the textbook

Next in the lecture

- Next in the lecture
  - more on scheduling

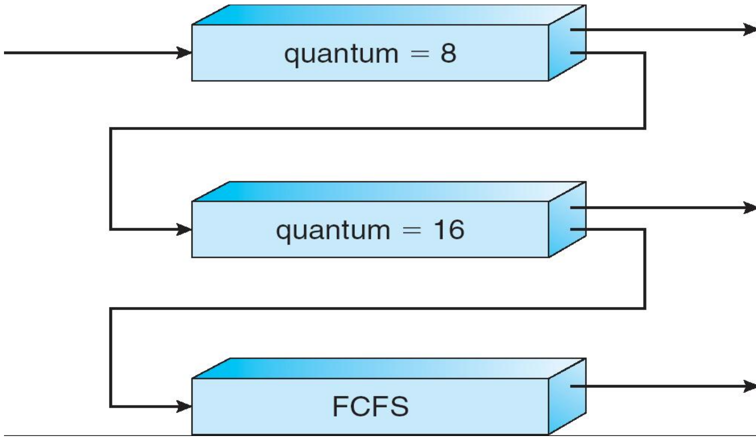
Multi-queue scheduling



- “No one fits all”
- Multi-queue approach
  - foreground queue
    - e.g., RR; better fairness
  - background queue
    - e.g., FCFS; more efficient
- Inter-queue scheduling
  - priority, time sharing (e.g., “80% rule”)

Multi-queue with feedback

- Multi-queue
  - number of queues
  - queuing algorithm for each queue
- Multi-queue with feedback
  - promote jobs
  - demote jobs
  - example



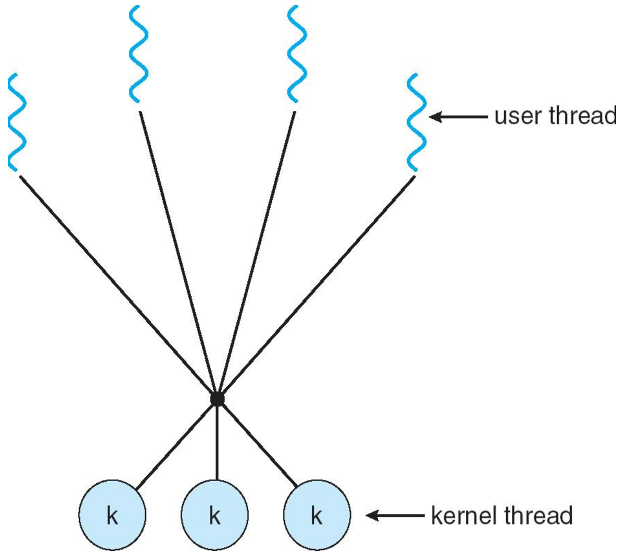
Multi-processor queuing

- Load balance between processors
  - cooperation and communication
- Asymmetric multiprocessing
  - one master scheduler
- Symmetric multiprocessing
  - cooperative schedulers
    - processor affinity: try to stick with one
    - load balancing: push or pull migration

* join the shortest queue? join the shorter of two queues? Q: multi-core?

More on scheduling

- Thread scheduling
  - local: user -> kernel thread
    - e.g., within a process
  - global: kernel thread -> CPU
    - e.g., across the system
- Algorithm evaluation
  - queuing analysis
    - Little’s law: $n = \lambda * W$
  - simulation



Pthread scheduling

- pthread_attr_setschedpolicy ();
  - regular, non-realtime (nice-able)
  - realtime, round-robin (preemptive, privileged)
  - realtime, FCFS (non-preemptive, privileged)
- pthread_attr_setschedparam ();
- pthread_attr_setscope ();
  - scheduling within a process
  - scheduling for the entire system

This lecture finally

- More on scheduling
  – multi-queue scheduling
  – multiprocessor scheduling
  – scheduling evaluation
    - Little's law, simulation
- Explore further
  – list process priority: /usr/bin/top
  – change priority: /bin/nice

Next lecture

- Process synchronization
  - read OSC7 Chapter 6 (or OSC6 Chapter 7)