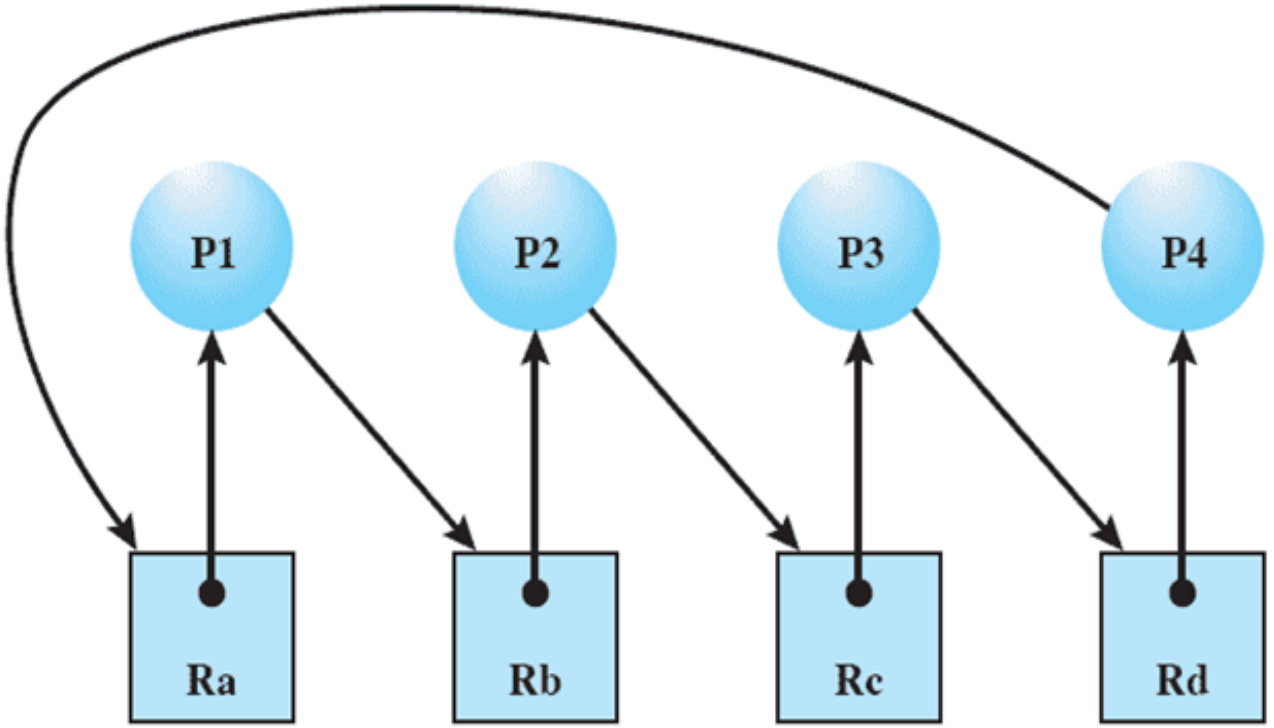
CIS 5512 - Operating Systems
Scheduling

Professor Qiang Zeng

Resource Allocation Graph describing the traffic jam



Resource Allocation Graph

Conditions for Deadlock

Mutual Exclusion	Hold-and-Wait	No Pre-emption	Circular Wait
• A process cannot access a resource that has been allocated to another process	• a process may hold allocated resources while awaiting assignment of others	• no resource can be forcibly removed from a process holding it	• a closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain

Dealing with Deadlock

- Three general approaches exist for dealing with deadlock:

Prevent Deadlock

- adopt a policy that eliminates one of the conditions
- E.g., numbering resources, and request from low to high

Avoid Deadlock

- make the appropriate dynamic choices based on the current state of resource allocation
- E.g., banker's algorithm

Detect Deadlock and Recover

- attempt to detect the presence of deadlock and take action to recover

Main Points

- Scheduling policy: what to do next, when there are multiple threads ready to run
  - Or multiple packets to send, or web requests to serve, or …
- Definitions
  - response time, throughput
- Uniprocessor policies
  - FIFO
  - Shortest Job First
  - Round robin
  - Multilevel feedback queue
- Multiprocessor policies

Some of the slides are courtesy of Dr. Thomas Anderson

Example

- You manage a restaurant, and the customers complain that they wait forever and starve. What will you do?
- You manage a web site, that suddenly becomes wildly popular. Do you?
  - Buy more hardware?
  - Turn away some users?
  - Implement a different scheduling policy?

Non-preemptive vs Pre-emptive

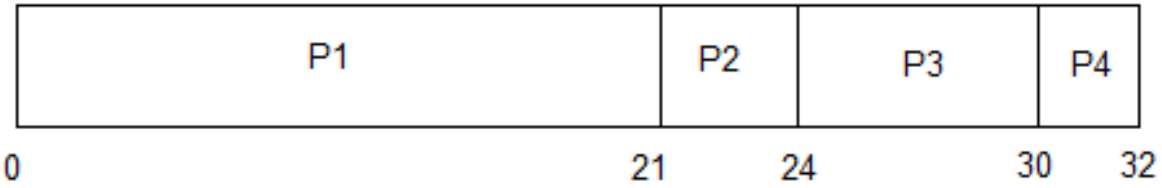
- A preemptive scheduling means that the scheduler can take resources (e.g., CPU) away from the process
- A non-preemptive scheduling means a process occupies the resources until it voluntarily relinquishes the resources

Metrics to evaluate a scheduler

- Response time
  - Time elapsed from the time of submission to the first response
- Throughput
  - # of tasks can be done per unit of time?
- Turnaround time
  - Time elapsed from the time of submission to completion
- Wait time
  - Time spent on waiting in the ready queue
- Predictability (low variance)
  - How consistent is the performance over time?
- Fairness

Example

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2



This is the GANTT chart for the above processes

First Come First Serve (FCFS)

- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor
- Easy to implement; very small overhead due to scheduling
- The scheduler ignores the property of tasks, so the overall performance, e.g., throughput, is usually poor

Shortest-Process-First (SPF) Scheduling

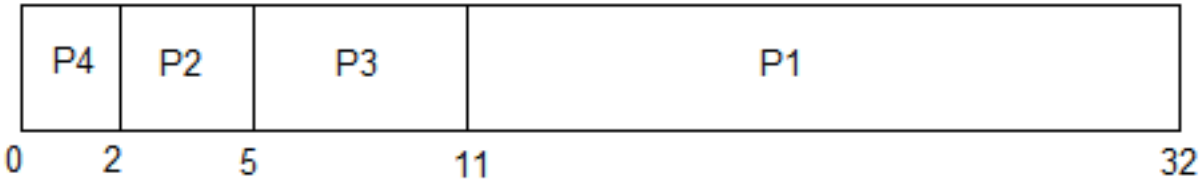
- Scheduler selects process with smallest time to finish
- Advantages: low average wait time
- Disadvantages:
  - Potentially large variance in wait times
    - Long task can be affected by short tasks once and again
    - Evan, starvation
  - Relies on estimates of time-to-completion
    - Can be inaccurate or unrealistic

Example

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

Round Robin

- Each task gets resource for a fixed period of time (time quantum)
  - If task doesn't complete, it goes back in line
- Need to pick a time quantum
  - What if time quantum is too long?
    - Becomes FIFO
  - What if time quantum is too short?
    - Large overhead for context switch
- Advantage: fairness
- Disadvantage: many context switches bring a large overhead; large wait time

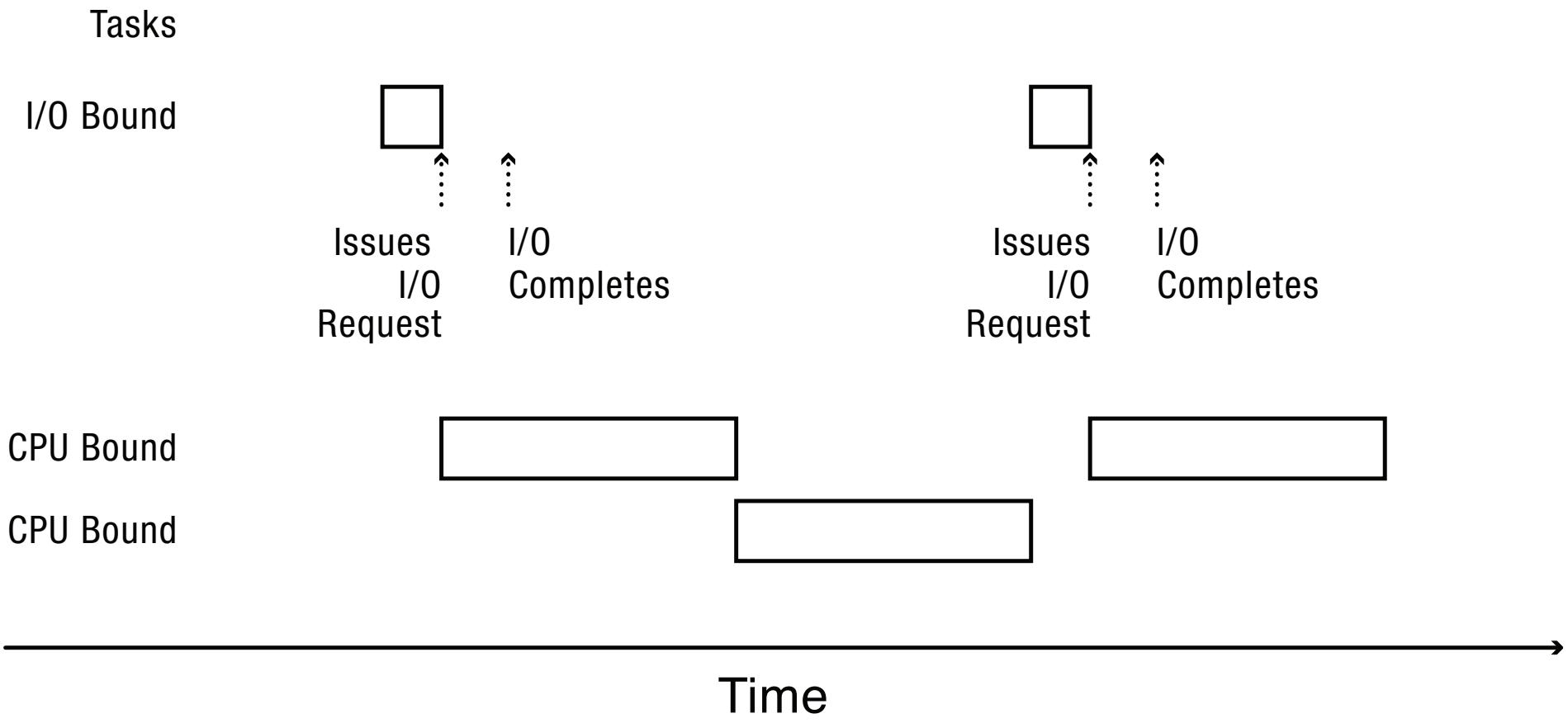
Round Robin = Fairness?

- Is Round Robin always fair?
  - No! See the next slides

CPU bound vs I/O bound

- If a process's speed is mainly determined by the CPU speed, the process is CPU-bound
  - E.g., multiply matrix
- If a process's speed is mainly determined by the I/O speed (i.e., most of the time the process blocks for I/O), the process is I/O-bound
  - E.g., emacs

Mixed Workload



Multi-level Feedback Queue (MFQ)

- Used in Linux, Windows, MacOS, and Solaris
  - Each system adopts the algorithm with some little modifications
- First developed by Corbato et al; led to Turing Award

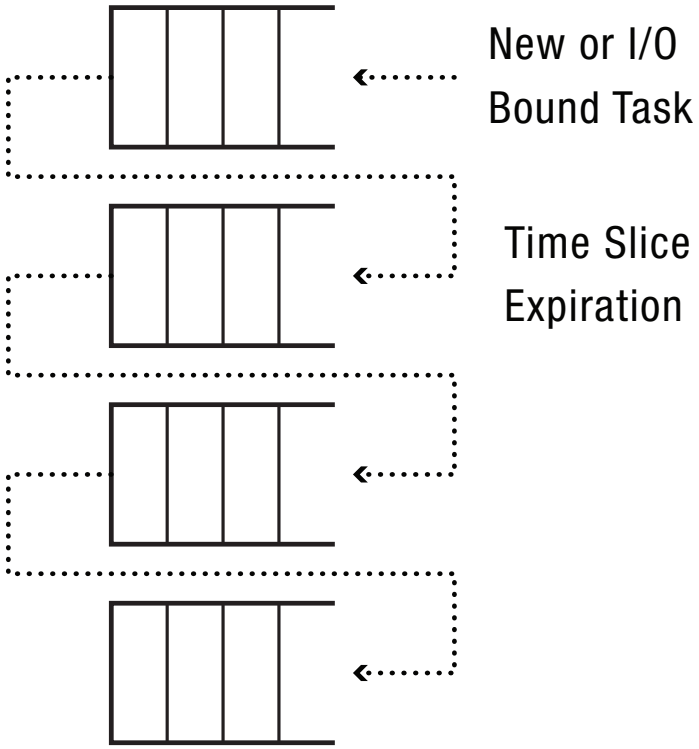
MFQ

- Has a set of Round Robin queues
  - Each queue has a separate priority
- High priority queues have short time slices, while low priority queues have long time slices
- Scheduler picks the first thread in highest-priority non-empty queue
- When a process is scheduled out, it is inserted in queues following the two rules
  - If time slice expires, task drops one level
  - If the process relinquishes the slice due to I/O, it is kept in the current priority queue
- Optionally, for a process in the base level queue that becomes I/O bound, it can be promoted to the next-higher queue

MFQ

Priority	Time Slice (ms)
1	10
2	20
3	40
4	80

Round Robin Queues



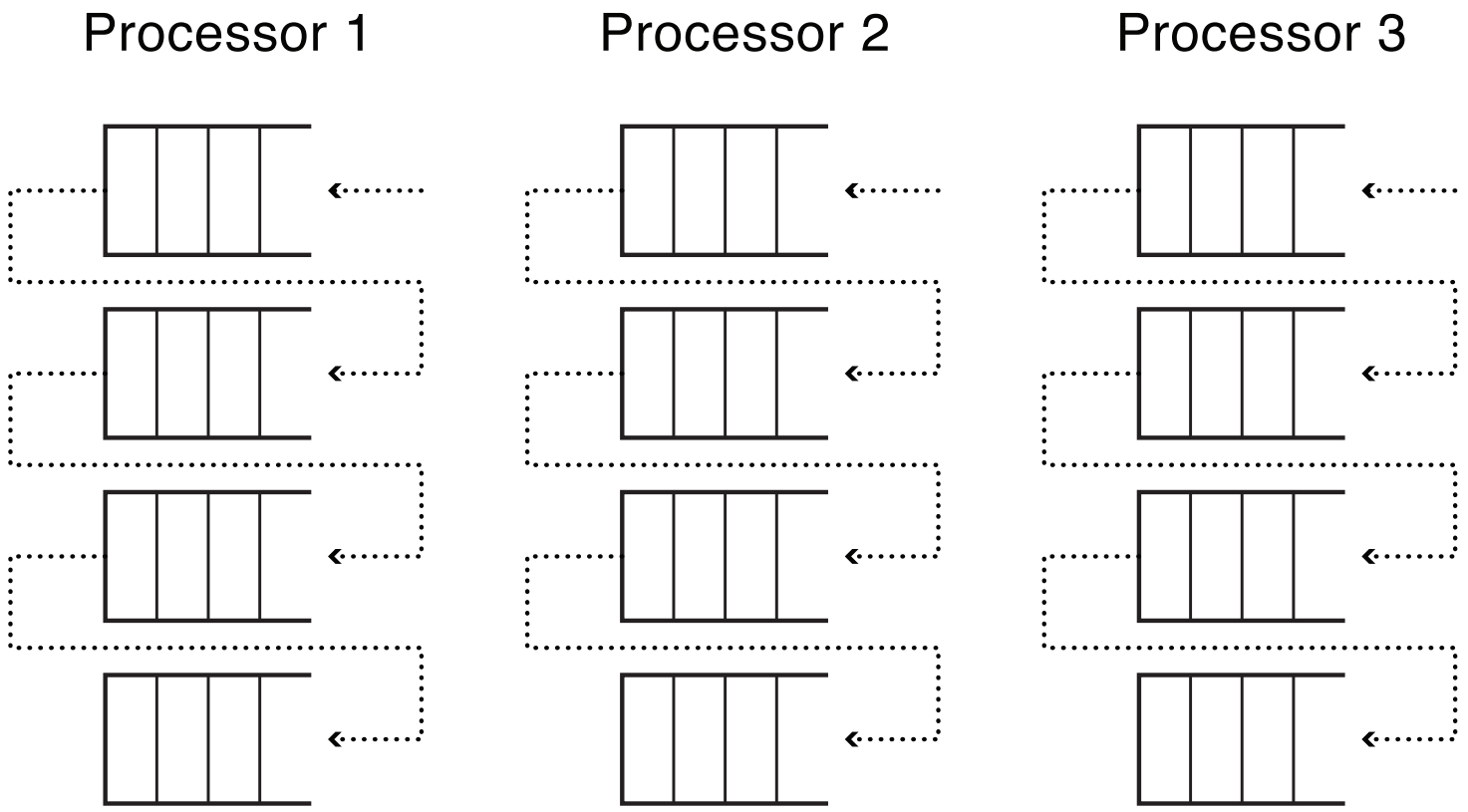
Multiprocessor Scheduling

- What would happen if we used MFQ on a multiprocessor?
  - Contention for scheduler spinlock
  - Poor CPU cache reuse

Per-Processor Affinity Scheduling

- Each processor has its own MFQ
  - Protected by a per-processor spinlock
- When the system puts threads back on the ready list, they are put back where they had most recently run
- But idle processors can steal work from other processors

Per-Processor Multi-level Feedback with Affinity Scheduling



Summary

- Scheduling policy: what to do next, when there are multiple threads ready to run
- Response time, throughput, wait time
- Uniprocessor policies
  - FIFO, Shortest Job First
  - round robin
  - multilevel feedback as approximation of optimal
- Multiprocessor policies