EEL-4736/5737
Principles of Computer System Design

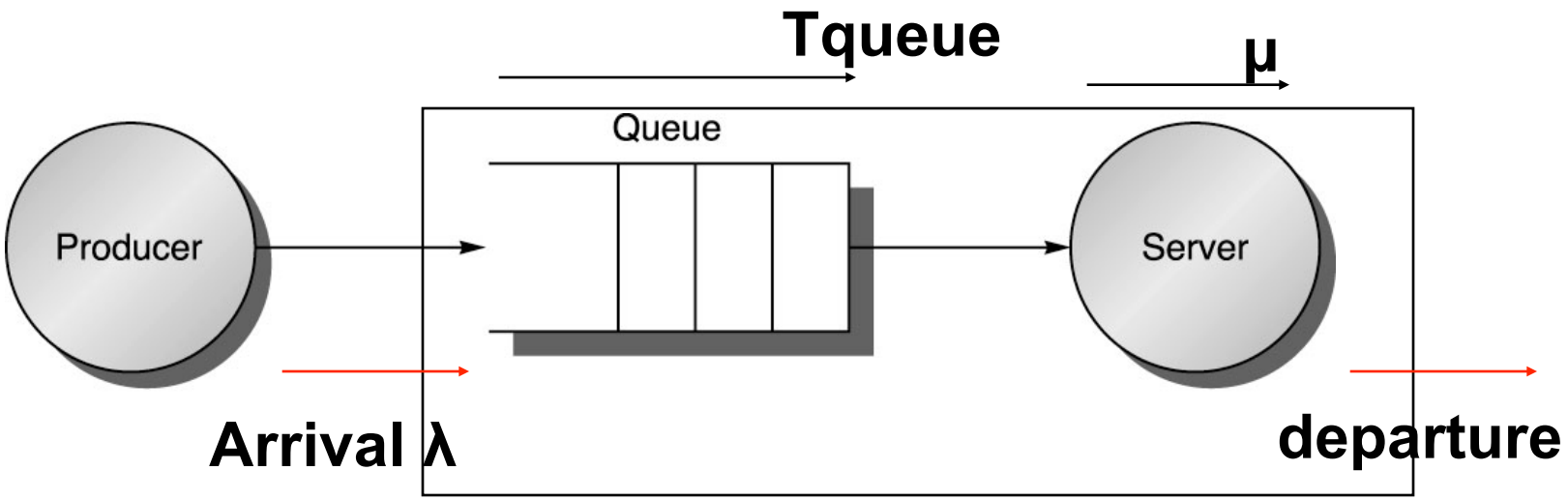
Lecture Slides 14

Textbook Chapter 6

Scheduling

Introduction

- Our simple model based on first come, first serve policy
- In general,
  - Different requests may have different priorities/deadlines
  - Different ordering can lead to better performance
  - We will overview systematic approaches to the general question of *resource scheduling*



Resource scheduling

- Collection of entities
  - Threads, address spaces, clients, services
- Collection of resources
  - Processor time, memory space, network bandwidth, bus time
- Scheduling: set of *policies* and dispatch *mechanisms* to allocate resources to entities
  - Scheduler: module that implements a policy

Examples

- Supermarket lines
  - Policy: independent first-come, first-serve lines, with separate fast service lines
  - Mechanism: Multiple physical lanes, one cashier per lane; enforce <10 items and no check on fast lanes
- Flight boarding
  - Policy: board by zones, priority boarding for first class/frequent miles
  - Mechanism: Single lane, broadcast zone number, check boarding pass/ticket class

Examples

- Thread scheduling
  - Example policy: each thread has a unique service level and minimum CPU time guaranteed per period
    - Threads scheduled in round-robin fashion according to service level
  - Mechanism: timer interrupts, pre-emption, admission control, thread queue
- Disk scheduling may dally, batch, reorder requests to achieve high throughput
  - Goals may conflict with thread scheduling

Scheduling metrics

- Turnaround time
  - Length of time from request arriving at service to completion
- Response time
  - Length of time from request arriving at service to begin *producing output*
    - E.g. Web browser begin to render page with incomplete data
- Waiting time
  - Length of time from request arriving at service to when service starts *processing*

Scheduling metrics

- Which metric is important depends on nature of application
  - Batch job processing:
    - Average turnaround time
  - Interactive user applications:
    - Response time; depends on human's perception of good service
    - Optimizing for a low variance (more predictability) can be more important than for low mean
    - If faster than human reaction time, no need to further optimize
- Will use "job" as a unit of work
  - Threads: series of jobs, each job a burst of activity

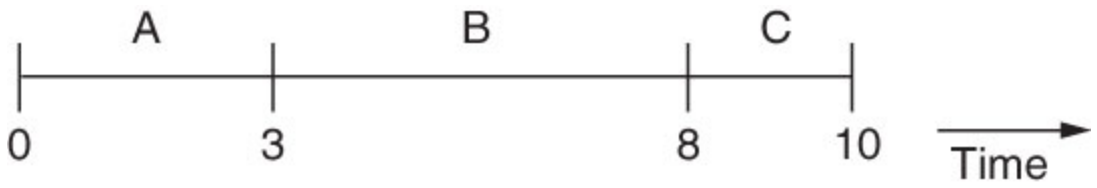
Fairness

- Scheduling policies may desire to provide a degree of fairness in handling requests
  - Starving a request to serve others – unfair
  - Not necessarily a “bad” scheduler – may have higher throughput than fair scheduler
- Cannot optimize for fairness, and response time, and throughput simultaneously
  - Different algorithms optimize along different dimensions
  - Trade-offs depending on application domain

Policies

- First-come, first-served (FCFS)
  - Implementation: FIFO ready queue
  - Add requests to end of queue, dispatch requests from beginning of queue

Job	Arrival time	Work amount
A	0	3
B	1	5
C	3	2

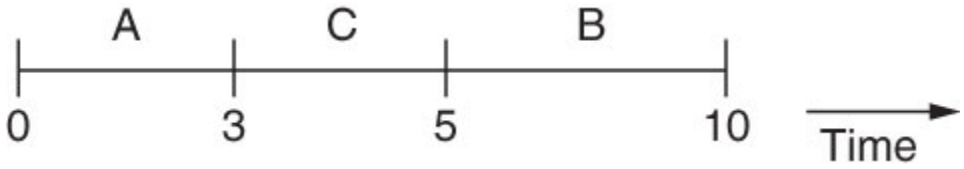


C's waiting time?

Policies

- Shortest-job-first
  - At dispatch time, choose job with shortest expected time
  - Need prediction of job request time

Job	Arrival time	Work amount
A	0	3
B	1	5
C	3	2



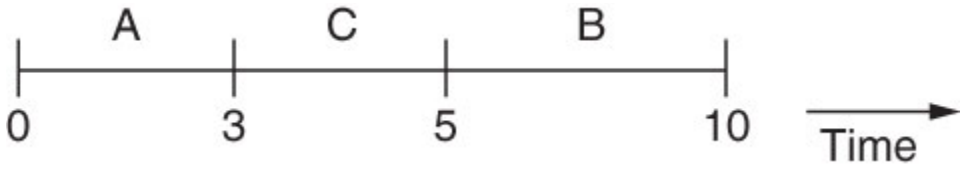
C is considered at time t=3; shorter than B

FCFS vs SJF

- Let's look at waiting times:

Job	Arrival time	FCFS wait	SJF wait
A	0	0	0
B	1	2	4
C	3	5	0

Reduced total wait time
Same total time



Discussion

- SJF requires some form of prediction
  - Based on model accounting for job class, inputs
  - Based on history of past similar requests
- SJF may lead to *starvation*
  - In pathological case, several short jobs may prevent a long job from ever making progress
    - Enhanced policies based on SJF can address starvation problem

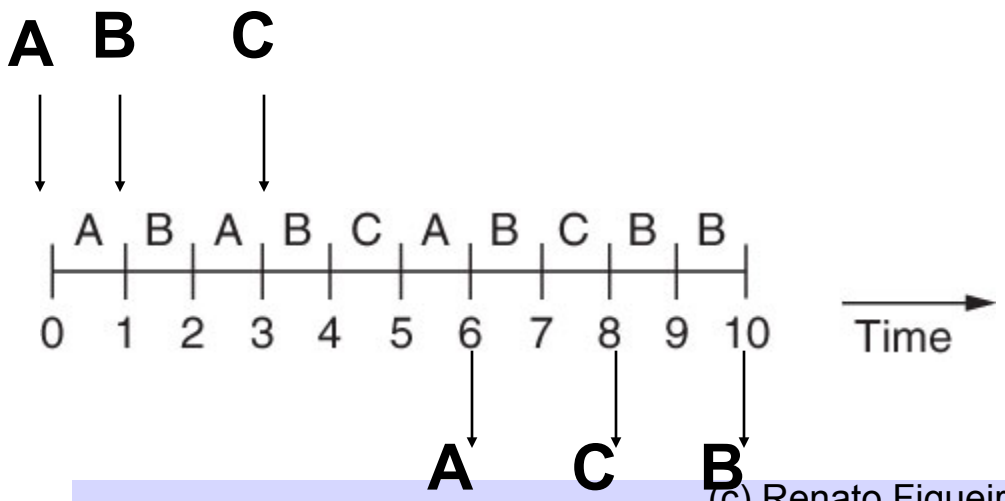
Pre-emption, round-robin

- An approach of dealing with jobs of different sizes: divide into smaller jobs
  - Pre-emptive scheduling – enforces modularity, allows division of jobs
- Simple pre-emptive policy: round-robin
  - Timer programmed to tick at “quantum” and interrupt/call YIELD
  - Move current job to end of queue, dispatch from head of queue

FCFS vs SJF vs RR

- 1 second quantum; wait times

Job	Arrival	FCFS wait	SJF wait	RR wait
A	0	0	0	0
B	1	2	4	1
C	3	5	0	2

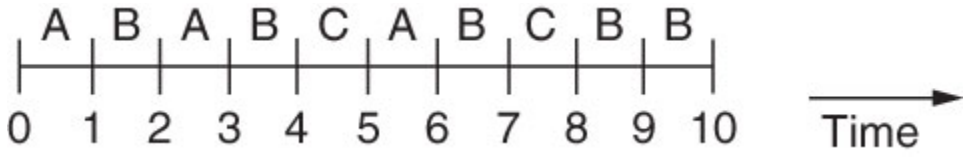


Important in interactive systems with a mix of short and long jobs

FCFS vs SJF vs RR

- 1 second quantum; turn-around times
  - Switching overhead not accounted for

Job	Arrival	FCFS t.a.	SJF t.a.	RR t.a.
A	0	3	3	6
B	1	7	9	9
C	3	7	2	5



Priority scheduling

- FCFS, RR did not distinguish job types
  - SJF distinguished based on predicted job time
- Priority scheduling
  - Flexibility to provide a means of differentiating jobs
  - Examples:
    - Static, coarse-grain: system vs user jobs
    - Static, fine-grain: priorities assigned per thread at creation time
    - Dynamic: priorities re-evaluated during run-time

Real-time schedulers

- Many systems have tasks that need to deliver results by a deadline
  - Hard real-time:
    - Missing deadlines can have catastrophic consequences – loss of property, life
    - Healthcare, automotive, defense, …
  - Soft real-time:
    - Missing deadlines compromises quality of service
    - Multi-media streaming, gaming

Real-time schedulers

- Hard real-time systems require planning and guaranteeing deadlines in worst-case scenario, without exceptions
  - Enhancements where performance cannot be deterministically determined are not helpful
    - E.g. speculation, caching
    - Interrupts vs. polling
  - Schedulability of jobs analyzed statically based on well-understood resource configuration and pattern of arrival and departure of jobs
- Soft real-time systems attempt to guarantee deadlines, but can live with occasional missed deadlines

Earliest-deadline first

- Widely-used heuristic policy in soft RT systems
  - Keep job queue sorted by deadline, pick first from queue
  - Attempts to minimize summed “lateness” of all jobs
- Total amount of work must be less than capacity of the system

EDF

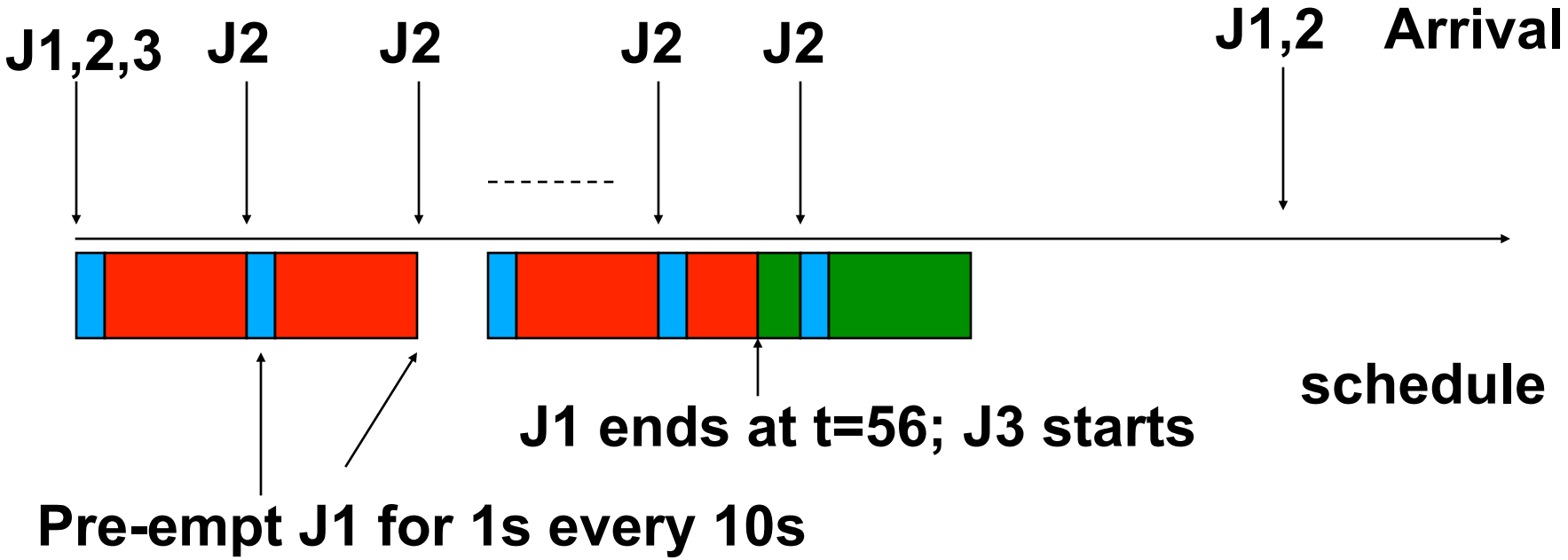
- “n” periodic jobs with period Pi, each requiring Ci seconds
  - Necessary (but not sufficient) condition: Sum(j=1,n)(Ci/Pi) <= 1
- Example
  - Job 1: P=100, C=50
  - Job 2: P=10, C=1
  - Job 3: P=1000, C=400
    - Sum = 50% + 10% + 40% = 100%
  - Admission control used to prevent the addition of more jobs into the system and enforce necessary condition

Rate monotonic scheduler

- Jobs have statically-defined priorities, and are scheduled dynamically
  - At design time, analysis of jobs' periods determine priorities
    - The shorter the cycle, the higher the priority
    - Job deadline is the period
    - Assume jobs do not share resources (e.g. locks)
  - At run time, always run job with highest priority, pre-empting a running job if necessary

Example

- Three periodic jobs
  - – Job 1: P=100, C=50 – priority 2
  - – Job 2: P=10, C=1 – priority 1 (highest)
  - – Job 3: P=200, C=80 – priority 4 (lowest)



RMS

- Schedulability:

  U = Sum(j=1,n)(Ci/Pi) <= n*[2^(1/n)-1]

  n = 1 : U <= 1

  n = 2 : U <= 0.828

  n -> infinity : U <= ln(2) ~ 0.693

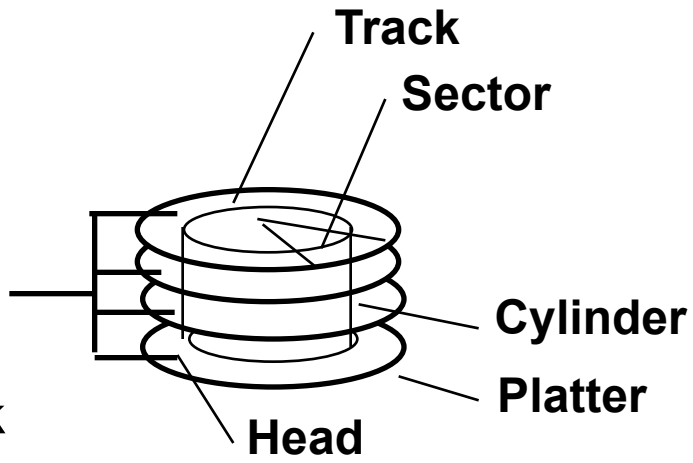
- Can guarantee schedulability in general if CPU utilized at 69.3%

Pitfall – priority inversion

- If jobs share resources (e.g. lock), lower-priority jobs may block higher-priority jobs
- Example:
  - T1 (low), T2, …, Tn-1 (medium), Tn (high)
  - T1, Tn share a lock
  - Assume T1 acquires a lock, then is pre-empted by Tn (while holding lock)
  - Tn tries to acquire lock and blocks
    - Other threads can prevent T1 from being scheduled, and therefore, Tn from making progress
  - Priority inheritance – mechanism to allow temporarily lending priority level to holder of lock

Example: Disk Arm Scheduling

- Rotational Latency:
  - Most disks rotate at 5K-15K RPM
  - Approximately 4-12ms per revolution
  - An average latency to the desired information is halfway around the disk



- Transfer Time is a function of :
  - Transfer size (usually a sector): 512B-4KB / sector
  - Rotation speed (5K-15K RPM)
  - Recording density: typical diameter ranges from 2 to 3.5 in
  - Typical values: 30-80 MB per second
    - Caches near disk; higher bandwidth (320MB/s)

Disk arm scheduling

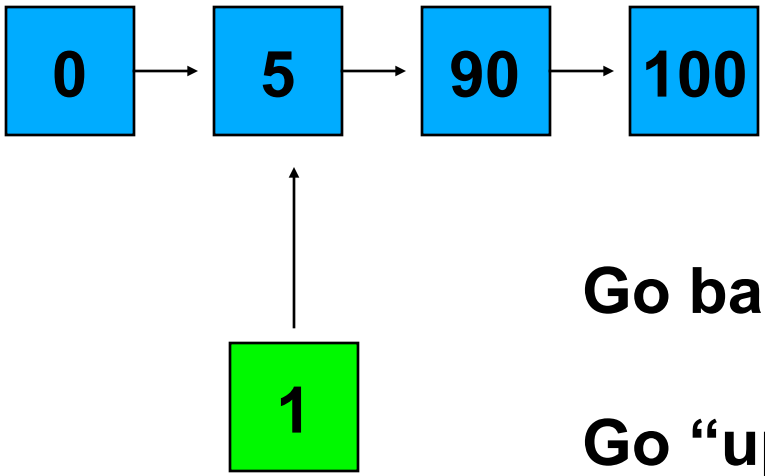
- Need to account for seek time – current track position and where it needs to go
- Example:
  - Consider a simple seek time model:
    - Seek from track x to x±n takes n*t seconds, where t is the time for the disk step motor to move read/write head one step
  - Head in track “0”; four requests that require data from tracks 0, 90, 5, and 100

Disk arm scheduling

- Example:
  - Head in track “0”; four requests that require data from tracks 0, 90, 5, and 100
- In-order scheduling:
  - 0t + 90t + 85t + 95t = 270t
- Sort requests by distance
  - Assume all requests arrive while at track 0
  - New order: 0, 5, 90, 100
  - 0t + 5t + 85t + 10t = 100t

Disk arm scheduling

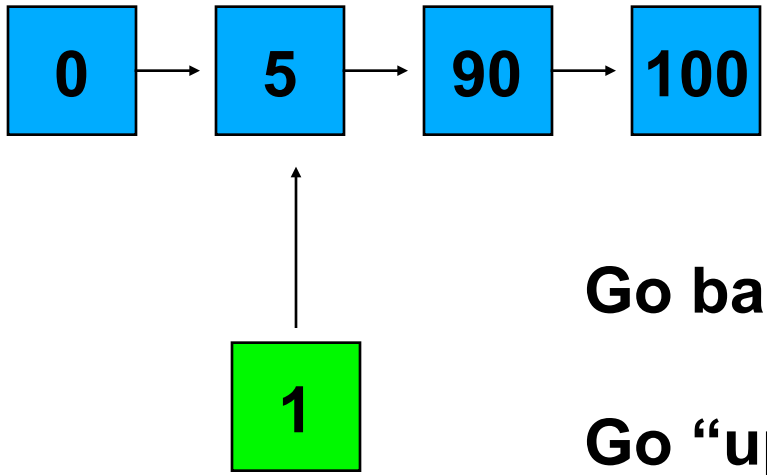
- Requests continue to arrive while the disk is handling outstanding requests



Go back to track 1?
Shortest-seek first; 108t
Go "up" to track 100 then down?
Elevator algorithm; 199t

Disk arm scheduling

- Shortest-seek first has less seek time, but can starve far-away requests
- Combined policy may have a bound time or number of shortest-seek first policy decision then an elevator policy decision



Go back to track 1?
Shortest-seek first; 108t
Go "up" to track 100 then down?
Elevator algorithm; 199t