CS 422/522 Design & Implementation of Operating Systems

Lecture 11: CPU Scheduling

Zhong Shao
Dept. of Computer Science
Yale University

1

CPU scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  1. switches from running to waiting state.
  2. switches from running to ready state.
  3. switches from waiting to ready.
  4. terminates.
- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive.*

2

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, predictability
- Uniprocessor policies
  - FIFO, round robin, optimal
  - multilevel feedback as approximation of optimal
- Multiprocessor policies
  - Affinity scheduling, gang scheduling
- Queueing theory
  - Can you predict/improve a system's response time?

3

Example

- You manage a web site, that suddenly becomes wildly popular. Do you?
  - Buy more hardware?
  - Implement a different scheduling policy?
  - Turn away some users? Which ones?
- How much worse will performance get if the web site becomes even more popular?

4

Definitions

- Task/Job
  - User request: e.g., mouse click, web request, shell command, …
- Latency/response time
  - How long does a task take to complete?
- Throughput
  - How many tasks can be done per unit of time?
- Overhead
  - How much extra work is done by the scheduler?
- Fairness
  - How equal is the performance received by different users?
- Predictability
  - How consistent is the performance over time?

5

More definitions

- Workload
  - Set of tasks for system to perform
- Preemptive scheduler
  - If we can take resources away from a running task
- Work-conserving
  - Resource is used whenever there is a task to run
  - For non-preemptive schedulers, work-conserving is not always better
- Scheduling algorithm
  - takes a workload as input
  - decides which tasks to do first
  - Performance metric (throughput, latency) as output
  - Only preemptive, work-conserving schedulers to be considered

6

Scheduling policy goals

- ***minimize response time*** : elapsed time to do an operation (or job)
  - Response time is what the user sees: elapsed time to
    - echo a keystroke in editor
    - compile a program
    - run a large scientific problem
- ***maximize throughput*** : operations (jobs) per second
  - two parts to maximizing throughput
    - minimize overhead (for example, context switching)
    - efficient use of system resources (not only CPU, but disk, memory, etc.)
- ***fair*** : share CPU among users in some equitable way

7

First In First Out (FIFO)

- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor
- Example: memcached
  - Facebook cache of friend lists, …
- On what workloads is FIFO particularly bad?

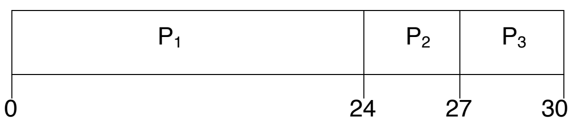
8

FIFO scheduling

- Example:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



- Waiting time for P_1 = 0; P_2 = 24; P_3= 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FIFO scheduling (cont' d)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

- The Gantt chart for the schedule is:

  - Waiting time for P_1 = 6; P_2 = 0; P_3 = 3
  - Average waiting time: (6 + 0 + 3)/3 = 3
  - Much better than previous case.

- FIFO Pros: simple; Cons: short jobs get stuck behind long jobs

Shortest-Job-First (SJF) scheduling

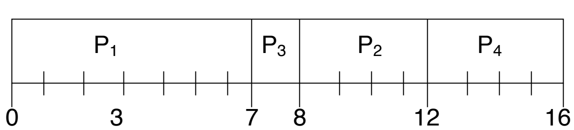
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – once given CPU it cannot be preempted until completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. A.k.a. Shortest-Remaining-Time-First (SRTF).
- SJF is optimal but unfair
  - pros: gives minimum average response time
  - cons: long-running jobs may starve if too many short jobs;
  - difficult to implement (how do you know how long it will take)

11

Example of non-preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



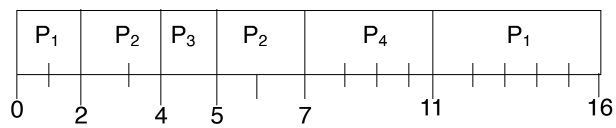
- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

12

Example of preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

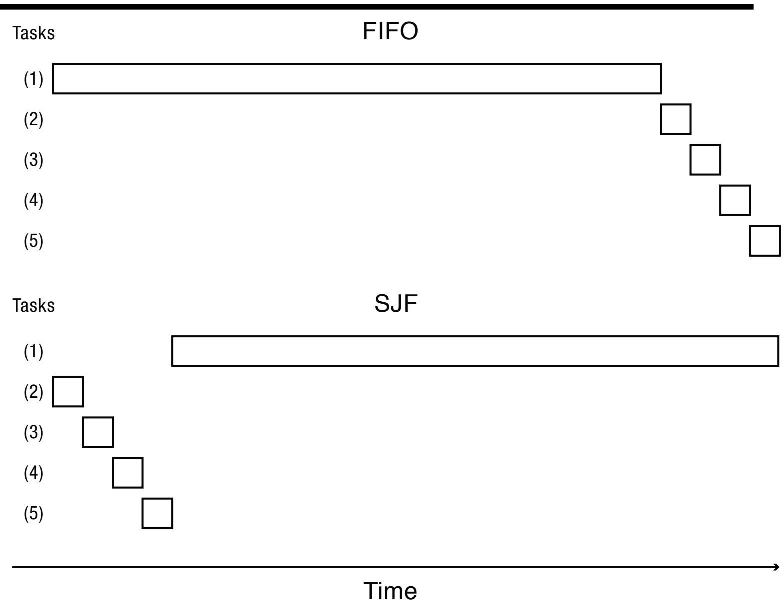
- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

13

FIFO vs. SJF



14

Starvation and sample bias

- Suppose you want to compare two scheduling algorithms
  - Create some infinite sequence of arriving tasks
  - Start measuring
  - Stop at some point
  - Compute average response time as the average for completed tasks between start and stop
- Is this valid or invalid?

15

Sample bias solutions

- Measure for long enough that
  - # of completed tasks >> # of uncompleted tasks
  - For both systems!
- Start and stop system in idle periods
  - Idle period: no work to do
  - If algorithms are work-conserving, both will complete the same tasks

16

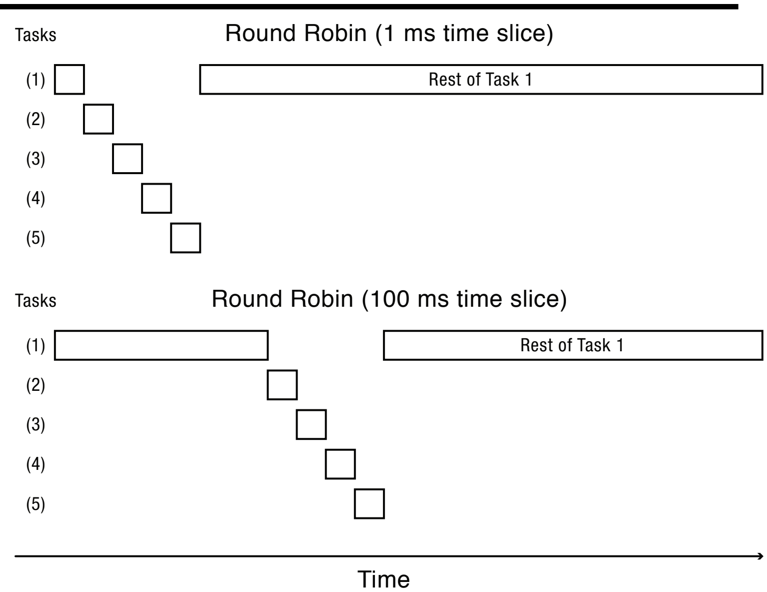
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*). After time slice, it is moved to the end of the ready queue.

  Time Quantum = 10 - 100 milliseconds on most OS
- *n* processes in the ready queue; time quantum is *q*
  - each process gets 1/*n* of the CPU time in *q* time units at once.
  - no process waits more than $(n-1)q$ time units.
  - each job gets equal shot at the CPU
- Performance
  - *q* large ⇒ FIFO
  - *q* too small ⇒ throughput suffers. Spend all your time context switching, not getting any real work done

17

Round Robin



18

Example: RR with time quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 37 57 77 97 117 121 134 154 162

- Typically, higher average turnaround than SJF, but better *response*.

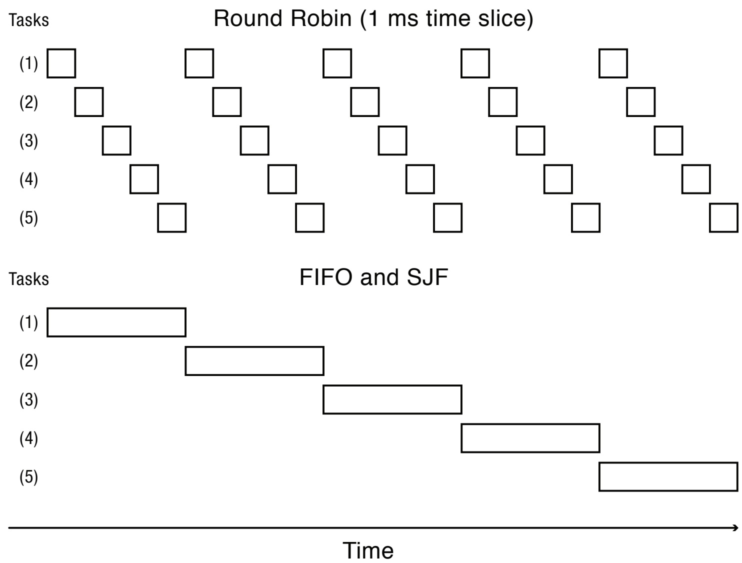
19

RR vs. FIFO

- Assuming zero-cost time slice, is RR always better than FIFO?
  - 10 jobs, each take 100 secs, RR time slice 1 sec
  - what would be the average response time under RR and FIFO ?
- RR
  - job1: 991s, job2: 992s, ... , job10: 1000s
- FIFO
  - job 1: 100s, job2: 200s, ... , job10: 1000s
- Comparisons
  - RR is much worse for jobs about the same length
  - RR is better for short jobs

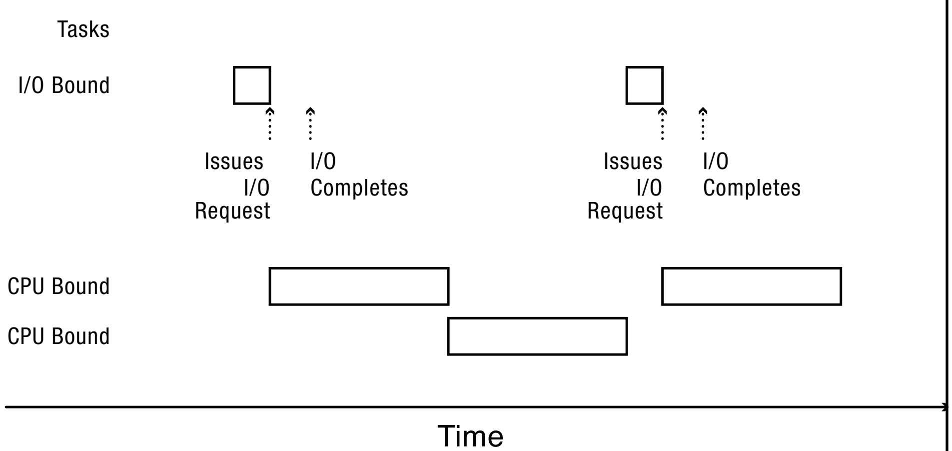
20

RR vs. FIFO (cont'd)



21

Mixed workload



22

Max-Min Fairness

- How do we balance a mixture of repeating tasks:
  - Some I/O bound, need only a little CPU
  - Some compute bound, can use as much CPU as they are assigned
- One approach: maximize the minimum allocation given to a task
  - If any task needs less than an equal share, schedule the smallest of these first
  - Split the remaining time using max-min
  - If all remaining tasks need at least equal share, split evenly
- Approximation: every time the scheduler needs to make a choice, it chooses the task for the process with the least accumulated time on the processor

23

Multi-level Feedback Queue (MFQ)

- Goals:
  - Responsiveness
  - Low overhead
  - Starvation freedom
  - Some tasks are high/low priority
  - Fairness (among equal priority tasks)
- Not perfect at any of them!
  - Used in Linux (and probably Windows, MacOS)

24

MFQ

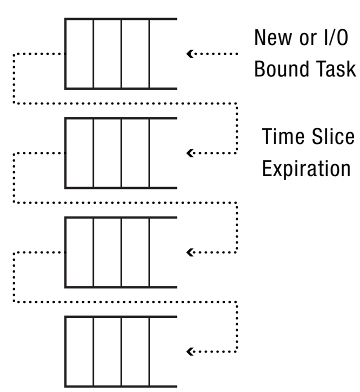
- Set of Round Robin queues
  - Each queue has a separate priority
- High priority queues have short time slices
  - Low priority queues have long time slices
- Scheduler picks first thread in highest priority queue
- Tasks start in highest priority queue
  - If time slice expires, task drops one level

25

MFQ

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

Round Robin Queues



26

Uniprocessor summary (1)

- FIFO is simple and minimizes overhead.
- If tasks are variable in size, then FIFO can have very poor average response time.
- If tasks are equal in size, FIFO is optimal in terms of average response time.
- Considering only the processor, SJF is optimal in terms of average response time.
- SJF is pessimal in terms of variance in response time.

27

Uniprocessor summary (2)

- If tasks are variable in size, Round Robin approximates SJF.
- If tasks are equal in size, Round Robin will have very poor average response time.
- Tasks that intermix processor and I/O benefit from SJF and can do poorly under Round Robin.

28

Uniprocessor summary (3)

- Max-Min fairness can improve response time for I/O-bound tasks.
- Round Robin and Max-Min fairness both avoid starvation.
- By manipulating the assignment of tasks to priority queues, an MFQ scheduler can achieve a balance between responsiveness, low overhead, and fairness.

29

Multiprocessor scheduling

- What would happen if we used MFQ on a multiprocessor?
  - Contention for scheduler spinlock
  - Cache slowdown due to ready list data structure pinging from one CPU to another
  - Limited cache reuse: thread's data from last time it ran is often still in its old cache

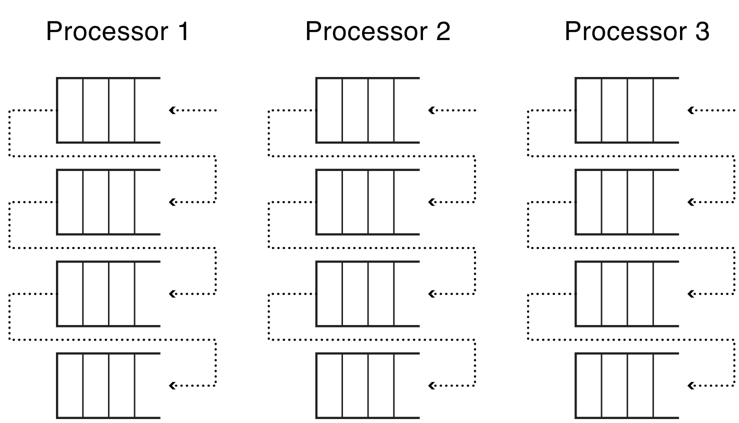
30

Per-processor affinity scheduling

- Each processor has its own ready list
  - Protected by a per-processor spinlock
- Put threads back on the ready list where it had most recently run
  - Ex: when I/O completes, or on Condition->signal
- Idle processors can steal work from other processors

31

Per-processor Multi-level Feedback with affinity scheduling



32

Scheduling parallel programs

- What happens if one thread gets time-sliced while other threads from the same program are still running?
  - Assuming program uses locks and condition variables, it will still be correct
  - What about performance?

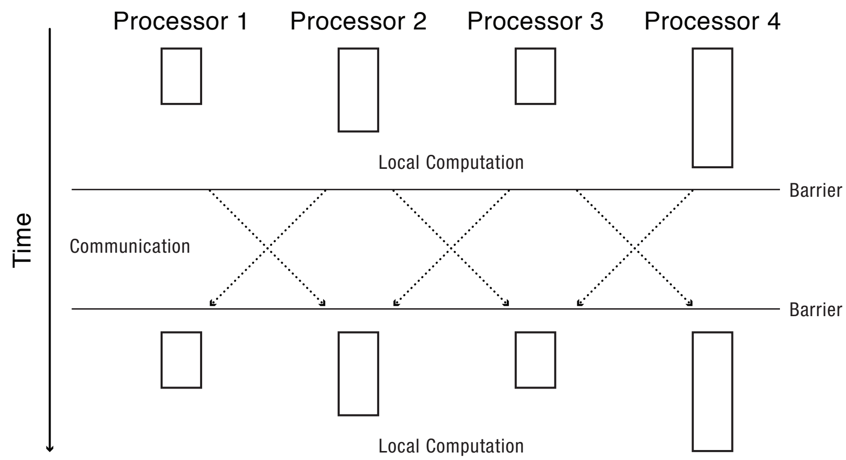
33

Bulk synchronous parallelism

- Loop at each processor:
  - Compute on local data (in parallel)
  - Barrier
  - Send (selected) data to other processors (in parallel)
  - Barrier
- Examples:
  - MapReduce
  - Fluid flow over a wing
  - Most parallel algorithms can be recast in BSP
    - Sacrificing a small constant factor in performance

34

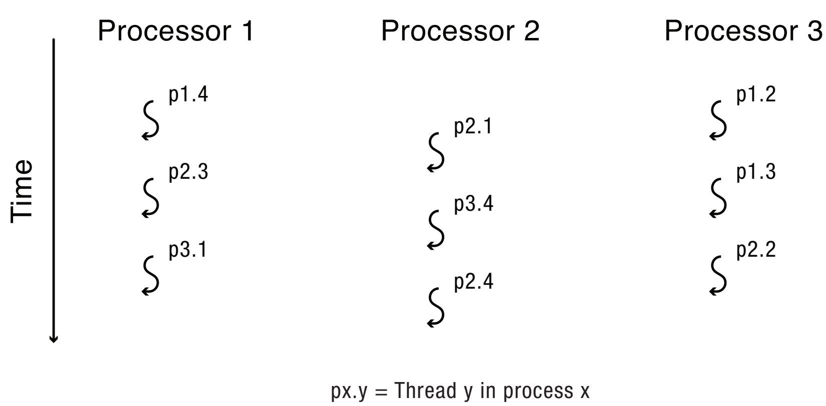
Tail latency



35

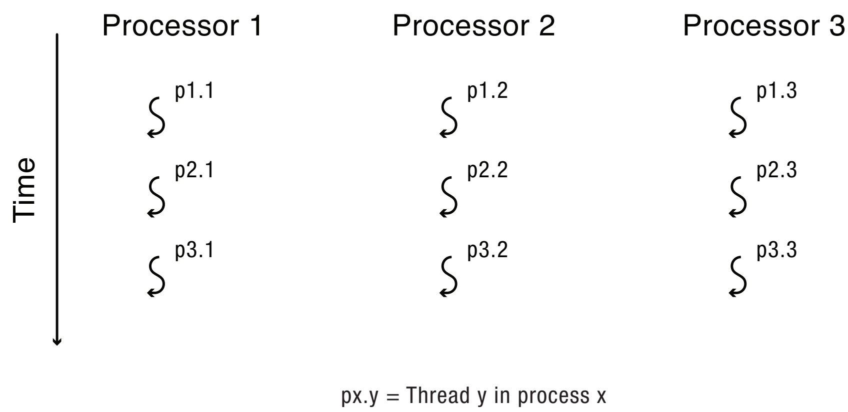
Scheduling parallel programs

Oblivious: each processor time-slices its ready list independently of the other processors



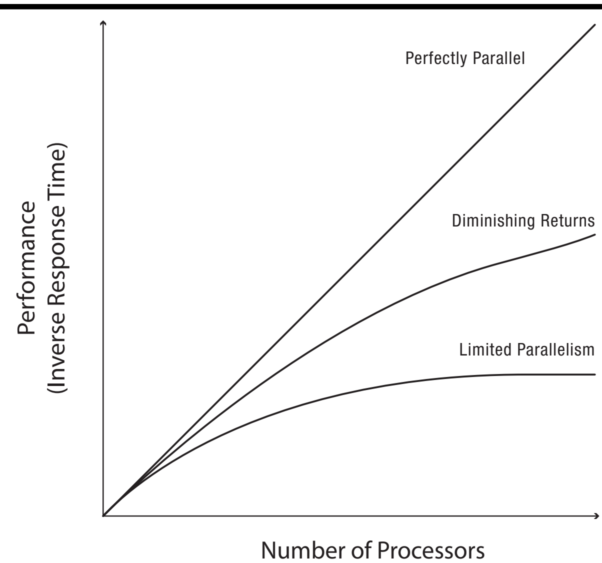
36

Gang scheduling



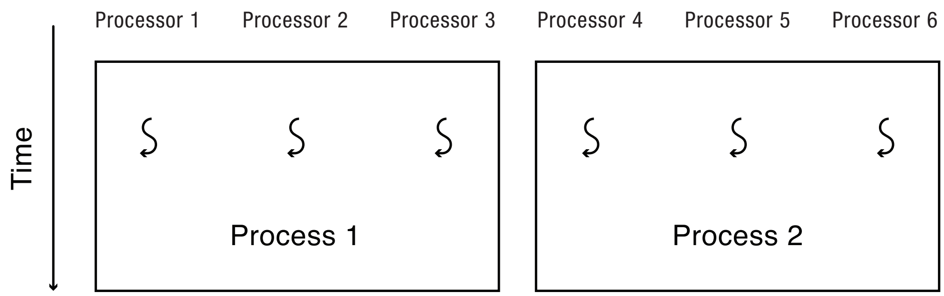
37

Parallel program speedup



38

Space sharing



Scheduler activations: kernel tells each application its # of processors with upcalls every time the assignment changes

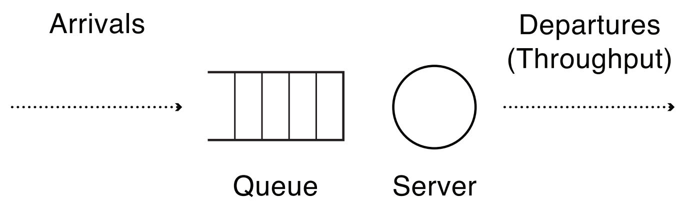
39

Queueing theory

- Can we predict what will happen to user performance:
  - If a service becomes more popular?
  - If we buy more hardware?
  - If we change the implementation to provide more features?

40

Queueing model



Assumption: average performance in a stable system, where the arrival rate (λ) matches the departure rate (μ)

41

Definitions

- Queueing delay (W): wait time
  - Number of tasks queued (Q)
- Service time (S): time to service the request
- Response time (R) = queueing delay + service time
- Utilization (U): fraction of time the server is busy
  - Service time * arrival rate (λ)
- Throughput (X): rate of task completions
  - If no overload, throughput = arrival rate

42

Little's law

N = X * R

N: number of tasks in the system

Applies to *any* stable system – where arrivals match departures.

43

Question

Suppose a system has throughput (X) = 100 tasks/s, average response time (R) = 50 ms/task

- How many tasks are in the system on average?
- If the server takes 5 ms/task, what is its utilization?
- What is the average wait time?
- What is the average number of queued tasks?

44

Question

- From example:
  X = 100 task/sec
  R = 50 ms/task
  S = 5 ms/task
  W = 45 ms/task
  Q = 4.5 tasks
- Why is W = 45 ms and not 4.5 * 5 = 22.5 ms?
  - Hint: what if S = 10ms? S = 1ms?

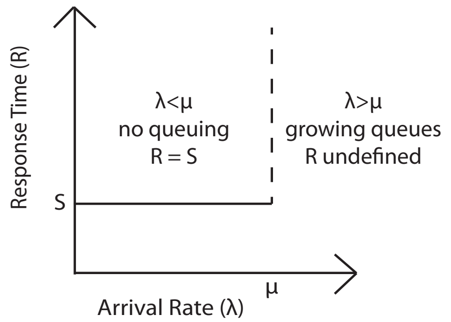
45

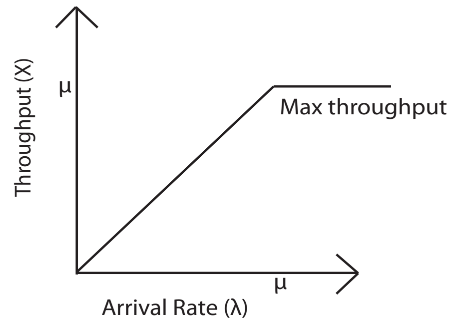
Queueing

- What is the best case scenario for minimizing queueing delay?
  - Keeping arrival rate, service time constant
- What is the worst case scenario?

46

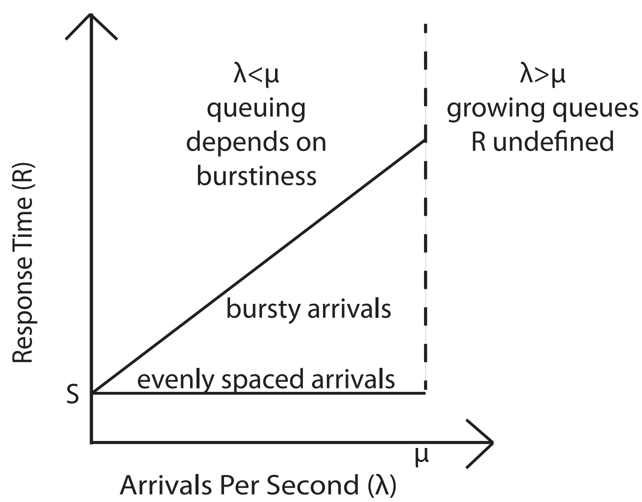
Best case: evenly spaced arrivals





47

Response time: best vs. worst case



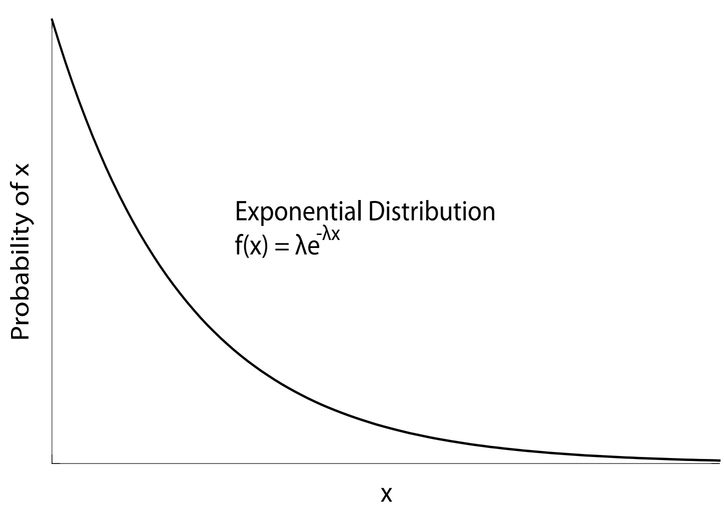
48

Queueing: average case?

- What is average?
  - Gaussian: Arrivals are spread out, around a mean value
  - Exponential: arrivals are memoryless
  - Heavy-tailed: arrivals are bursty
- Can have randomness in both arrivals and service times

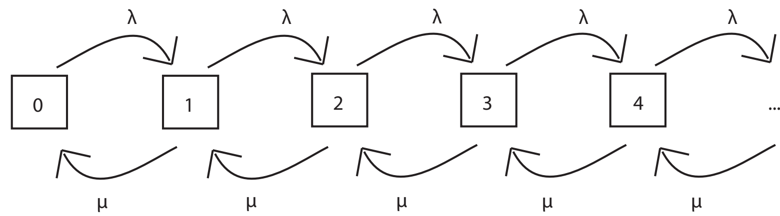
49

Exponential distribution



50

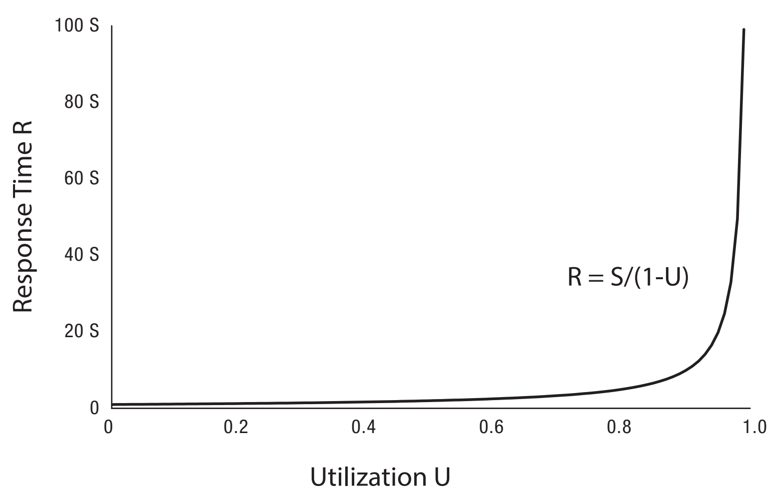
Exponential distribution



Permits closed form solution to state probabilities, as function of arrival rate and service rate

51

Response time vs. utilization



52

Question

- Exponential arrivals: R = S/(1-U)
- If system is 20% utilized, and load increases by 5%, how much does response time increase?
- If system is 90% utilized, and load increases by 5%, how much does response time increase?

53

Variance in response time

- Exponential arrivals
  - Variance in R = S/(1-U)^2
- What if less bursty than exponential?
- What if more bursty than exponential?

54

What if multiple resources?

- Response time =
  Sum over all i
    Service time for resource i /
      (1 – Utilization of resource i)
- Implication
  - If you fix one bottleneck, the next highest utilized resource will limit performance

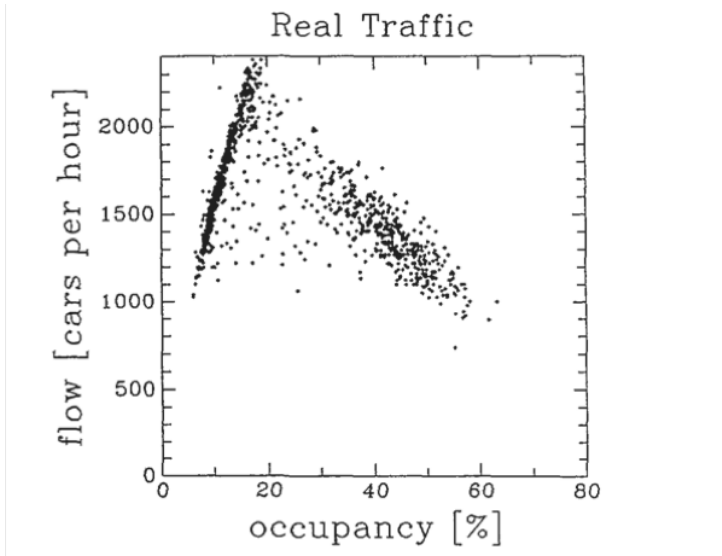
55

Overload management

- What if arrivals occur faster than service can handle them
  - If do nothing, response time will become infinite
- Turn users away?
  - Which ones? Average response time is best if turn away users that have the highest service demand
  - Example: Highway congestion
- Degrade service?
  - Compute result with fewer resources
  - Example: CNN static front page on 9/11

56

Highway congestion (measured)



57