Scheduling
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?
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?
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?
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
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?
Shortest Job First (SJF)

• Always do the task that has the shortest remaining amount of work to do
  – Often called Shortest Remaining Time First (SRTF)

• Suppose we have five tasks arrive one right after each other, but the first one is much longer than the others
  – Which completes first in FIFO? Next?
  – Which completes first in SJF? Next?
FIFO vs. SJF

FIFO

Tasks
(1)
(2)
(3)
(4)
(5)

SJF

Tasks
(1)
(2)
(3)
(4)
(5)

Time
Question

• Claim: SJF is optimal for average response time
  – Why?

• Does SJF have any downsides?
Question

• For what workloads is FIFO optimal?

• Pessimal?
Starvation and Sample Bias

• Suppose you want to compare FIFO and SJF
  – Create some infinite sequence of arriving tasks
  – Stop at some point
  – Compute average response time as the average for completed tasks
• Is this valid or invalid?
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?
    • Infinite?
  – What if time quantum is too short?
    • One instruction?
Round Robin

**Tasks**

1. Round Robin (1 ms time slice)
   - Rest of Task 1

2. Round Robin (100 ms time slice)
   - Rest of Task 1
Round Robin vs. FIFO

- Assuming zero-cost time slice, is Round Robin always better than FIFO?
Round Robin vs. FIFO

Round Robin (1 ms time slice)

FIFO and SJF
Round Robin vs. Fairness

- Is Round Robin always fair?
Mixed Workload

Tasks

I/O Bound

Issues
I/O Request

I/O Completes

Issues
I/O Request

I/O Completes

CPU Bound

CPU Bound

Time
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 uses less than an equal share, schedule the smallest of these first
  – Split the remaining time using max-min
  – If all remaining tasks use at least equal share, split evenly
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)
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
<table>
<thead>
<tr>
<th>Priority</th>
<th>Time Slice (ms)</th>
<th>Round Robin Queues</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

MFQ

New or I/O Bound Task

Time Slice Expiration
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.
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.
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.
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
Per-Processor Multi-level Feedback: Affinity Scheduling
Scheduling Parallel Programs

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

px.y = Thread y in process x
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?
Bulk Synchronous Parallel Program
Gang Scheduling

px.y = Thread y in process x
Number of Processors

Performance (Inverse Response Time)

- Perfectly Parallel
- Diminishing Returns
- Limited Parallelism

Number of Processors
Scheduler activations: kernel informs user-level library as to # of processors assigned to that application, with upcalls every time the assignment changes.
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?
Queueing Model

Assumption: average performance in a stable system, where the arrival rate ($\lambda$) matches the departure rate ($\mu$)
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 (\( \lambda \))
• Throughput (X): rate of task completions
  – If no overload, throughput = arrival rate
Little’s Law

\[ N = X \times R \]

N: number of tasks in the system

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

• Suppose a system has throughput (X) of 100 tasks/sec, and a mean response time (R) of 50 ms/task, how many tasks are in the system on average?

• If the server takes 5ms/task, what is its utilization?

• What is the average response time and average number of queued tasks?
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?
Queueing

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

• What is the worst case scenario?
Queueing: Best Case

- **Arrival Rate** ($\lambda$)
- **Throughput** ($X$)
- **Response Time** ($R$)

**Max throughput**

- $\lambda < \mu$
  - no queuing
  - $R = S$
- $\lambda > \mu$
  - growing queues
  - $R$ undefined

![Graphs](Image)
Response Time: Best vs. Worst Case

- \( \lambda < \mu \) (evenly spaced arrivals)
  - Queuing depends on burstiness

- \( \lambda \geq \mu \) (bursty arrivals)
- \( \lambda > \mu \) (growing queues, \( R \) undefined)

Arrivals Per Second (\( \lambda \))

Response Time (\( R \))
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
Exponential Distribution

The probability density function of the exponential distribution is given by:

\[ f(x) = \lambda e^{-\lambda x} \]
Exponential Distribution

Permits closed form solution to state probabilities, as function of arrival rate and service rate
Response Time vs. Utilization

\[ R = \frac{S}{1-U} \]
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?
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?
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
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
Highway Congestion (measured)
Why Do Metro Buses Cluster?

• Suppose two Metro buses start 15 minutes apart
  – Why might they arrive at the same time?