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 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
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
Shortest Job First

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

• For what workloads is FIFO optimal?

• Pessimal?

• Does SJF have any downsides?
Starvation and Sample Bias

• Suppose you want to compare FIFO and SJF on some sequence of arriving tasks
  – Compute average response time as the average for tasks that start/end in some window

• 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

Round Robin (1 ms time slice)

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

rest of task 1

Round Robin (100 ms time slice)

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

rest of task 1

Time
Round Robin

(a) A in execution

(b) A completes its time share

- Proportionality: turnaround (time spent in the system) proportional to task length
- Response time upperbounded by the number of processes
  - # ready processes * time share
Round Robin

(a) A in execution

(c) reactivation of A
Round Robin

- End of time share signaled by the timer
  - Timer interrupt causes the activation of the scheduler
  - The scheduler restarts the timer
- Scheduler takes over also in case of suspension of the running process
  - Reassigns the CPU and restarts timer
- In current systems time share around 20-120 msec
Round Robin vs. FIFO

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

Tasks

Round Robin (1 ms time slice)

FIFO and SJF

Time
Round Robin vs. Fairness

• Is Round Robin always fair?
Mixed Workload

- Two kind of processes:
  - CPU-bound – long CPU bursts with sparse I/O
  - I/O bound – short CPU bursts with frequent I/O
Mixed Workload

Tasks

I/O bound

I/O completes

issues I/O request

CPU bound

I/O completes

gets CPU

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
  – Schedule the smallest task first, then split the remaining time using max-min
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
  – Round robin in each 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>new or I/O bound task</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>time slice expiration</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

• Issues with MFQ:
  – Starvation:
    – If all the upper queues are always full of I/O-bound processes
    – A process may change its “habits”: from CPU-bound to I/O-bound and vice-versa
  • Need for policies to raise up the priority of:
    – I/O bound processes
    – CPU-bound processes that are starving
• MFQ usually combined with dynamic priorities
Example: scheduling in Windows
Example: scheduling in Windows

• A new thread starts with priority 8
• Priority raised up if:
  – thread reactivated after I/O operation (disk: +1, Serial line: +6, Keyboard: +8, Audio card: +8, …)
  – thread reactivated after waiting on a mutex/semaphore (+1 if in background, +2 if in foreground)
  – Thread didn’t run for a given amount of time (priority goes to 15 for two time shares)
• Priority lowered if thread uses all time share (-1)
• When a window goes in foreground the time share of its threads is enlarged
Example: scheduling in Windows

- Inversion of priority

1) A (12) executes `wait(sem)` and waits

2) A waiting on `sem` running

3) B (4) ready; Should execute `signal(sem)` But does not have the CPU

4) C (8) ready
Uniprocessor Summary

• 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

- 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.
- 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.