Lecture 6: Scheduling (1)

Instructor: Chengyu Song
Slide contributions from
Nael Abu-Ghazaleh, Harsha Madhyvasta and Zhiyun Qian
Scheduling Overview

- Scheduler runs when we context switching among execution units (jobs/processes/threads) to pick who runs next
  - Under what situation does this occur?
  - What should it do? Does it matter?
- Making this decision is called scheduling
- Now, we’ll look at:
  - The goals of scheduling
  - Starvation
  - Various well-known scheduling algorithms
  - Standard Unix scheduling algorithm
Multiprogramming

- Increase CPU utilization and job throughput by overlapping I/O and CPU activities
  - Doing this requires a combination of mechanisms and policy
- We have covered part of the mechanisms
  - Context switching, how and when it happens
  - Process queues and process states
  - Now meet the scheduler
- Now we’ll look at the policies
  - Who, when, goals, …
- We’ll refer to schedulable entities as jobs (standard usage) – could be processes, threads, people, etc.
Scheduler

- The scheduler is the OS module that manipulates the process queues, moving jobs to and from them.

- The scheduling algorithm determines which jobs are chosen to run next, for how long, and what queues they wait on.

- In general, the scheduler runs:
  - When a job switches from running to waiting
  - When an interrupt occurs
    - Why?
  - When a job is created or terminated
Scheduling Levels

- Scheduling works at two levels in an operating system
  1. Control multiprogramming level – number of jobs loaded into memory
     - Moving jobs to/from memory is often called swapping
     - Long term scheduler: infrequent
  2. To decide what job to run next
     - Does it matter? What criteria?
     - Short term scheduler: frequent
     - We are concerned with this level of scheduling
Scheduling Styles

- Scheduler works differently in different systems
  - In **preemptive** systems the scheduler can interrupt a running job (involuntary context switch)
  - In **non-preemptive** systems, the scheduler waits for a running job to explicitly block (voluntary context switch)

- What about **preemptive kernel**?
  - Non-preemptive kernel disables maskable interrupts during event handling
  - Preemptive kernel allows interrupts to be delivered during event handling → more responsive
Scheduling Goals

- What are some reasonable goals for a scheduler?
- Scheduling algorithms can have many different goals:
  - CPU utilization
  - Job throughput (# jobs/unit time)
  - Turnaround time ($T_{\text{finish}} - T_{\text{start}}$)
    » Normalized turnaround time = Turnaround time/process length
  - Avg Waiting time (Avg($T_{\text{wait}}$): avg time spent on wait queues)
  - Avg Response time (Avg($T_{\text{ready}}$): avg time spent on ready queue)

- Batch systems
  - Strive for job throughput, turnaround time (supercomputers)

- Interactive systems
  - Strive to minimize response time for interactive jobs (PC)
Job Characteristics

- Achieving the goals requires knowing the job
  - Past
    » When it arrived, how much progress has it made, how long has it run, has it been behaving nicely, …
  - Current
    » How many resources it uses, how many are left, …
  - Future
    » How much work is left? …
    » Important for some scheduling algorithm, but can we really know?
  - Type
    » GUI, I/O, realtime, …
  - Priority
  - etc
Starvation

Starvation is a scheduling “non-goal”:

- **Starvation**: a job is prevented from making progress because other jobs have the resource it requires
  - Resource could be the CPU, or a lock (later in synchronization)
- **Starvation usually a side effect of the scheduling**
  - E.g., a high priority process always prevents a low priority process from running on the CPU
- **Starvation can be a side effect of synchronization**
  - E.g., one thread always beats another when acquiring a lock
  - E.g., constant supply of readers always blocks out writers
First In First Out (FIFO)

- Schedule tasks in the order they arrive
  - Continue running them until they complete or give up the processor

- Example: queues
  - Supermarket, banks, drive-through, …

- On what workloads is FIFO particularly bad?
  - Imagine being at supermarket to buy a drink of water, but get stuck behind someone with a huge cart (or two!)
    - …and who pays in pennies!
  - Can we do better?
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

What’s the big deal? Don’t they finish at the same time?
FIFO vs. SJF

- Assuming jobs arrives almost at the same time

\[
\text{ATT} = \frac{(8+(8+4)+(8+4+2))}{3} = 11.33
\]

\[
\text{ATT} = \frac{(4+(4+8)+(4+8+2))}{3} = 10
\]

\[
\text{ATT} = \frac{(4+(4+2)+(4+2+8))}{3} = 8
\]

\[
\text{ATT} = \frac{(2+(2+4)+(2+4+8))}{3} = 7.33
\]
FIFO vs. SJF

- Claim: SJF is optimal for average response time
  - Why?

- For what workloads is FIFO optimal?
  - For what is it pessimal (i.e., worst)?

- Does SJF have any downsides?
SJF

- **Problems?**
  - Impossible to know size of CPU burst
    - Like choosing person in line without looking inside basket/cart
  - How can you make a reasonable guess?
  - Can potentially starve

- **Flavors**
  - Can be either preemptive or non-preemptive
  - Preemptive SJF is called shortest remaining time first (SRTF)
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

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Round Robin (1 ms time slice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>rest of task 1</td>
</tr>
<tr>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td></td>
</tr>
</tbody>
</table>

Round Robin (100 ms time slice)

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Round Robin (100 ms time slice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>rest of task 1</td>
</tr>
<tr>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td></td>
</tr>
</tbody>
</table>

Time
Round Robin vs. FIFO

- Many context switches can be costly

- Other than that, is Round Robin always better than FIFO?
Round Robin vs. FIFO

Is Round Robin always fair?

Tasks | Round Robin (1 ms time slice) | FIFO and SJF
--- | --- | ---
(1) | | (1)
(2) | | (2)
(3) | | (3)
(4) | | (4)
(5) | | (5)
Mixed Workload

Tasks

I/O bound

I/O completes

issues I/O request

CPU bound

CPU bound

I/O completes

gets CPU

Time
Next class

- Advanced scheduling
- Preparation
  - Read Module 7 & 8