Operating Systems
ECE344

Introduction to Scheduling Policies

Ashvin Goel
ECE
University of Toronto
Overview

- Scheduler overview
- Scheduling goals
  - Batch systems
  - Interactive systems
- Scheduling policies
  - Batch scheduling policies
  - Interactive scheduling policies
Overview of Thread Scheduler

- A thread scheduler implements the threads abstraction
  - Allows running threads concurrently and in parallel
- Scheduler runs when a thread calls threads API
Scheduler Mechanism

- When scheduler runs, it may run another thread
  - Suspends current thread
  - Runs another thread, by resuming this thread
  - Thread switching impl. is called scheduling mechanism

- OS generally implements preemptive scheduling
  - Scheduler can suspend running thread at any time

Pros, Cons?
Scheduling Policy

- Several threads may be ready to run at any time
- Which thread should the scheduler choose to run?
- Choice depends on scheduling policy
  - Chooses next thread to run, e.g., highest priority
  - Decides whether and when to preempt this thread

Many threads are ready to run

One scheduling mechanism, many scheduling policies!
Overview

- Scheduler overview

- Scheduling goals
  - Batch systems
  - Interactive systems

- Scheduling policies
  - Batch scheduling policies
  - Interactive scheduling policies
CPU-Bound and IO-Bound Programs

- Programs alternate between performing computation and IO, called CPU and IO bursts
  - A CPU-bound program has frequent CPU bursts
    - E.g., matrix multiply
  - An IO-bound program has frequent IO bursts
    - E.g., web server

- When a program performs IO
  - Program needs to wait for IO to finish, so CPU is not needed

- Scheduler runs another program to keep CPU busy
  - Improves CPU utilization
Common Computer Systems

- **Batch systems**
  - Long running, CPU-bound programs (jobs)
  - No interactive users, minimal time constraints
  - E.g., report generation, big data processing

- **Interactive (or general-purpose) systems**
  - Short running, IO-bound programs
  - Interactive users, response time constraints
  - E.g., desktop, mobile environments
Scheduling Goals

- Scheduler aims to improve different metrics for each type of system

- Batch systems
  - CPU Utilization: % of time that CPU is busy (not idle)
  - Throughput: number of jobs that complete per unit time
  - Turnaround time: time needed from start to finish of job
    \[ \text{turnaround time} = \text{processing time (running)} + \text{waiting time (not running)} \]

- Interactive systems
  - Response time: time between receiving request and producing response
Summary

- A scheduling mechanism is the method by which a scheduler runs threads
  - Involves switching threads

- A scheduling policy dictates
  - Which thread is run next
  - When the thread stops running

- Scheduling policies aim to improve performance
  - Batch systems, with long-running, CPU bound jobs
    - Improve throughput and reduce waiting time
  - Interactive systems, with short-running IO bound jobs
    - Improve response time

- Next lectures describe scheduling policies
Think Time

- Is there a difference between throughput and CPU utilization? If so, why?
- Why does improving (reducing) the response time reduce throughput as well?
Is there a difference between throughput and CPU utilization? If so, why?

- Throughput is the number of jobs that complete over time, while CPU utilization is the % of time that the CPU is busy. The two are correlated for CPU bound jobs since higher CPU utilization will generally increase throughput. However, this may not always be the case. Consider two schedulers that have 100% CPU utilization, i.e., the CPU is never idle. If one of the schedulers is less efficient than the other, or it performs too many pre-emption, then its throughput will be lower, since more CPU time is being spent on running the scheduler code (either the scheduler algorithm or the thread switching code).
Why does improving (reducing) the response time reduce throughput as well?

- Throughput is improved by running threads for long periods of time because that reduces the number of preemptions and because it increases cache locality (the working set of the process is cached, which improves performance). However, running threads for long periods of time increases response time since it takes time before a thread that has been woken up by an IO event is run. Reducing the response time requires preempting threads frequently so that all threads get to run as soon as they are ready to run or are runnable, but this increases preemption and reduces cache locality, thus reducing performance.