# Operating System Concepts

Lecture 1: Course Logistics and Introduction

Omid Ardakanian
oardakan@ualberta.ca
University of Alberta

## Today's class

- Course organization and outline
- Assignments and exams
- Policies
- Introduction to Operating System

## General information

- Course webpage: https://canvas.ualberta.ca/courses/27956
  - slides, supplemental readings, assignments, discussion forum, etc.
- Contact information
  - Instructor: Omid Ardakanian, PhD
  - Email: ardakanian@ualberta.ca
  - Instructor/TA list: cmput379@ualberta.ca (for anything that can't be posted on the discussion forum)
  - Research interests: sensors networks, embedded and cyber-physical systems, performance evaluation, sustainable computing

# Teaching Assistants & Contact policy

**Akemi Izuko**

**Armaan Katyal**

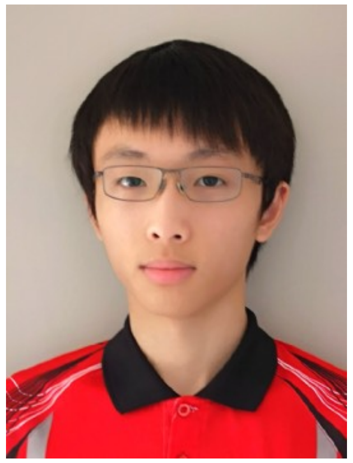
**Ellis McDougald**

**Han Yang**

**Patrick Zijlstra**

**Shasta Johnsen-Sollos**

**Steven Oufan Ha**


**Zhaoyu Li**

If your question can be raised on the **Canvas Discussion Forum**, then you should use the forum. This is the primary and quickest way to ask questions about anything related to the course.

Otherwise, you might send an email to the instructor/TA address: cmput379@ualberta.ca

## Course objectives

- Enhance your knowledge and skills in developing programs that
  - utilize operating system (OS) services
  - support concurrent operations
  - perform inter-process communication via shared memory, signals, pipes, and sockets
  - interact with the Internet
- Introduce fundamental ideas underlying the design and implementation of modern operating systems

## How does it fit in our curriculum?

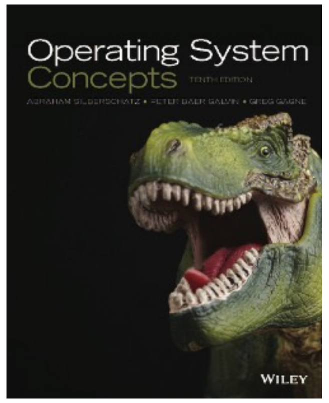
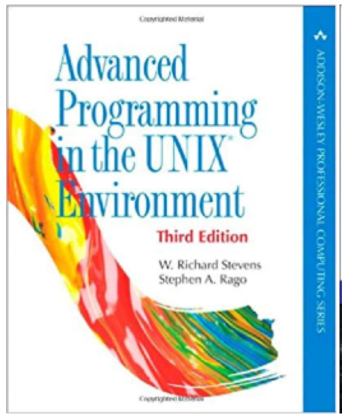
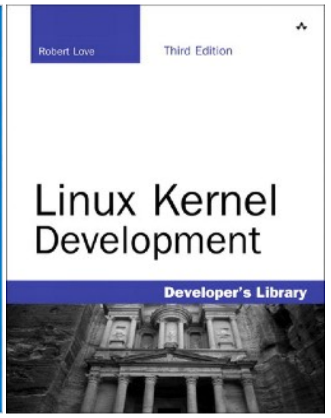
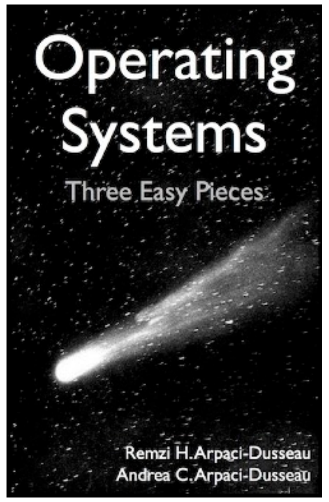
- CMPUT 229: Computer Organization and Architecture I
  - organization of the hardware architecture
  - fundamentals behind program execution
- CMPUT 379: Operating System Concepts
  - operating system structure and services
  - programming in the UNIX environment
- CMPUT 313: Computer Networks
  - network architecture, protocols, and applications

## Prerequisites

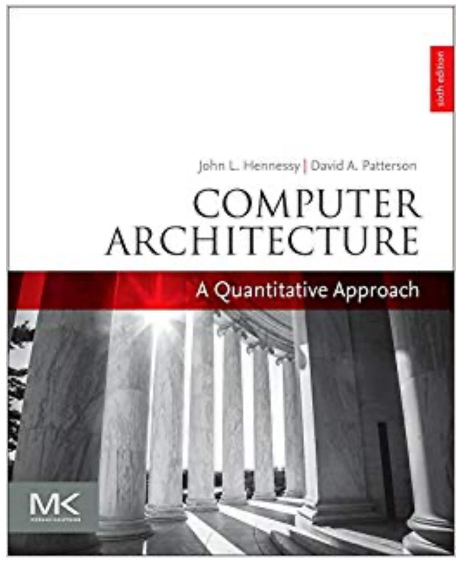
- CMPUT 201 and 204, or 275; one of CMPUT 229, E E 380 or ECE 212
- This means that you must be proficient in C programming and have a good grasp of data structures and algorithms, assembly language and computer architecture

# Textbook

1. A. Silberschatz, P. Galvin, and G. Gagne, Operating System Concepts, 10th Edition, John Wiley, 2018 (**required**).
2. W. Stevens, and S. Rago, Advanced Programming in the Unix Environment, 3rd Edition, Addison-Wesley, 2013 (**highly recommended**).
3. R. Love, Linux Kernel Development, 3rd Edition, Addison-Wesley, 2010 (**highly recommended**).
4. R. H. Arpaci-Dusseau, and A. C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, 1st Edition, ArpaciDusseau Books, 2018 (**highly recommended**).
   http://pages.cs.wisc.edu/~remzi/OSTEP/

Also keep your architecture book handy.



# Evaluation

CMPUT 379 has two main components:

- conceptual component: covered in class and tested in exams
- hands-on programming component: covered in lab sessions and tested in the assignments

- 45% Programming assignments (collaboration policy: **solo effort**)
  - C programming in the UNIX environment
  - small-scale software development and simulating parts of an operating system
  - rubrics will be posted online
  - submissions within 24 hours after the deadline are subject to a 20% penalty
- 20% Midterm exam (October 24, during class)
- 35% Final exam

# Assignments

- 3 programming assignments that must be completed in C and submitted via GitHub Classroom
    - you cannot **discuss** your solution with other people, **borrow** code from someone or just **look at** their code, **hire** someone to write code for you
    - you may search online for generic algorithms but you cannot use code downloaded from the Internet in your submission
    - you cannot submit code generated by AI tools
    - you must cite all resources that you used
    - you might consult TAs and instructor, but not anyone else
- it's okay to write code on your own machine but **you must ensure that it runs on a lab machine**
- strict late policies and policies on cheating (plagiarism detection software will be used)
- see Canvas for assignment release and due dates

# OS labs

- Three lab sessions per week, starting from the third week
  - Tuesday 17:00 – 19:50. @ CCIS L1-160
  - Thursday 14:00 – 16:50. @ UCOMM 2-070
  - Friday 14:00 – 16:50. @ NRE 2-001
  - those registered in a particular lab will get priority
- Lab attendance is not mandatory, but **highly recommended**
  - we won't have enough time to implement some of the concepts in class
  - TAs will help you with programming assignments and course topics

## Other policies

- Cell phones must be off or on silent during lectures and lab sessions
- Slides will be posted on Canvas shortly before each lecture
- Always compile and run your code on **Linux lab machines** before submitting it!

# Overview of 379 topics

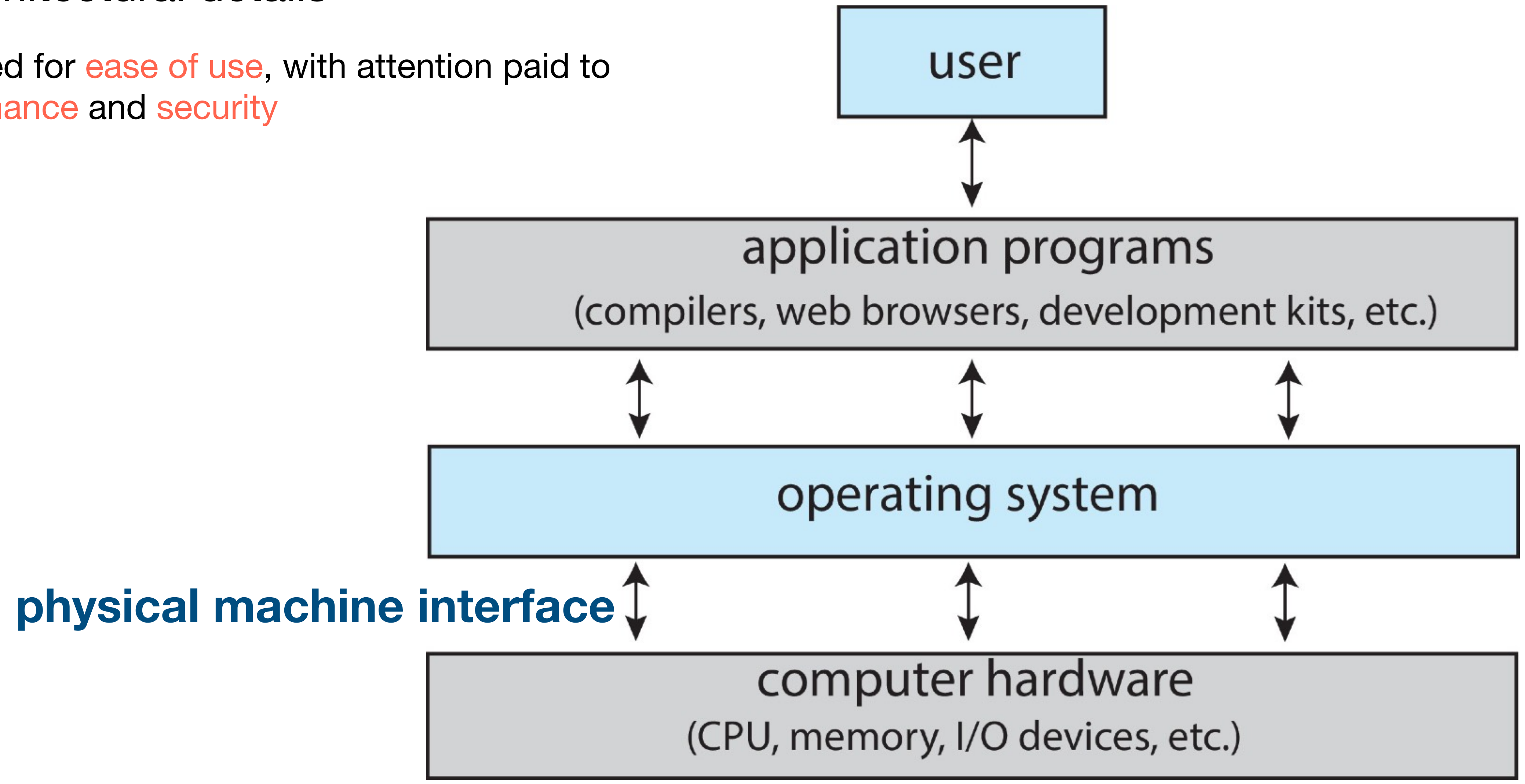
- Introduction
  - OS roles, services, structure
- Process management
  - creation, suspension, resumption, and termination
  - scheduling processes and threads on CPUs
  - mechanisms for interprocess communication
- Concurrency management
  - threading models
  - thread libraries
- Process synchronization
  - critical section and race conditions
  - synchronization primitives
  - deadlock detection, avoidance, and recovery

# Overview of 379 topics

- Memory management
  - keeping track of used/unused memory
  - allocating and deallocating memory as needed
  - deciding on which processes and data to move in and out of memory
- Storage and I/O management
  - mounting and unmounting disk
  - free space management
  - disk scheduling
  - partitioning and protection
- File system management
  - creating and deleting files and directories
  - mapping files onto mass storage
  - backing up files on nonvolatile storage media
- Virtual machine monitors

# What's an Operating System?

- Software that manages a computer's hardware and coordinates its use among various application programs (system view)
  - allocates available resources (CPU time, memory and disk space, I/O devices, etc.) efficiently and fairly
- Software that acts as intermediary between the computer user(s) and hardware hiding architectural details
  - designed for ease of use, with attention paid to performance and security


user
application programs
(compilers, web browsers, development kits, etc.)
operating system
physical machine interface
computer hardware
(CPU, memory, I/O devices, etc.)

# OS virtualizes hardware resources

- Definition: transforming a physical resource into a more general and powerful virtual form of itself
- Virtualizing CPU
  - allowing many programs to **seemingly** run at once on many **virtual** CPUs
- Virtualizing Memory
  - each running program (i.e., process) accesses its own private address space; hence, a memory reference by one does not affect other ones
  - two running programs may perform an update at address 0x200000 independently!
  - OS maps these address spaces onto the physical memory

# OS roles

- as a referee
  - allocates resources among users/applications
    - an infinite loop doesn't require a reboot!
  - protects each user/application from others
  - coordinates communication between users/applications
- as an illusionist
  - gives the illusion of having a dedicated machine — with **infinite** resources (virtual CPUs, memory, etc.) and **reliable** storage and network transport — to each application
  - masks physical limitations and details
- as a glue
  - provides an execution environment with a standard library and common services to applications, …

## What’s part of an OS?

- kernel
  - a program running at all times in the kernel mode, aka supervisor/privileged mode
  - located and loaded into memory by the **bootstrap** program at system boot time
- system programs/daemons
  - programs that are shipped with the operating system, but are not part of the kernel
  - the first system daemon in Linux is `systemd` that starts many other daemons
    - examples of other daemons are `crond`, `syslogd`, and `sshd`
- middleware
  - a set of software frameworks that provide additional services to application developers, for example supporting databases, multimedia, graphics

## Homework

- Watch "A Narrative History of BSD" on YouTube