# Lecture 2

## Four Abstractions

We will be looking at the following four abstractions:

- Thread: Execution Context
  - Fully describes the program state
  - Program counter, registers, etc
- Address Space
  - Set of memory addrs. which are accessible to the program
- Process: an instance of a running program
  - Protected addr. space, and one or more threads
- Dual Mode Operation/Protection

### 61C Review

- CALL
  - First write then compile, then link libraries and load all text/data segments into memory
  - Then create stack, which goes from high to low and includes function args, return addrs, etc.
  - And heap, which goes from low to high and is filled when we `malloc`
  - Then transfer control of resources to program, and provide services to the program
- Recall that we fetch inst. from memory, decode, then execute

## Threads

A **thread** is a single unique execution, with its own PC, registers, exec flags, stack, and memory state.

Note a thread is executing on a core of a processor when it is **resident** in the processor registers, meaning

- Registers hold the root state of thread
- Includes PC and currently executing instruction

## The illusion of multiple processors

- Can split the CPU into time-slices. The thread is either on the real core, or on a virtual core called the Thead Control Block (TCB)
- Switch triggered by
    - Timer, voluntary yield, I/O, other things to be discussed later. [create backlink here]
- TCB
    - Holds contents of registers when thread not running
    - For now, we say it's stored in the kernel

# Address Space

The **address space** is the set of acceptable addresses and states associated with them.

Recall that our operating system must protect itself from user programs, and must protect user programs from one another.

- reliability
- security
- privacy
- fairness