

# Zeroth Presentation Speaker Notes

---

## Introduction (Nihal)

- We want to build a 32-bit operating system/ kernel
- We will use the C programming language and x86 assembly
  - The reasoning behind why we did not choose a modern programming language like Rust will be explained in the later slides.
- Our three basic goals are:

### 1. Implement Display

- This involves printing text to the screen and drawing geometric shapes such as squares, triangles and so on ...
- Drawing text to the screen
- In our current design, we are using VGA text-mode and graphics-mode.  
This involves writing 2 bytes to memory location `0xB8000` to represent a character.

### 2. Input Drivers

- This involves implementing both keyboard and mouse drivers from scratch.
- Implementing keyboard driver involves reading scan codes from the PS/2 controller
- Implementing mouse driver involves handling interrupts

### 3. C standard library

- The functions that we take for granted - like `strlen` and `printf`, `scanf`, `malloc` are all parts of the C standard library.
  - They are not provided by the compiler.
  - It is upto the kernel to provide these libraries
  - We will implement some parts of the `stdlib` (like `string.h` and `malloc` function)
- Since it is not really practical, we will exclude things like networking.

## Motivation (Antony)

- Writing low level code is hard and we wanted to do something **hard**.
  - *Illustrate the difficulty by mentioning the steps needed to setup a compiler:*
    1. First install 8 different dependencies.
    2. Download and **compile** `binutils` (for GNU linker and assembler).
    3. Download and **compile** GCC as a cross-compiler.
    4. Get a bootloader (eg: GRUB).
    5. Write custom linker script.
    6. Build a bootable image of your kernel.
    7. Now you have a clean slate. No printing, no keyboard, no mouse functionality.

- Makes you appreciate the built-in abstractions.  
Example:
  - We call a function and text magically appears on the screen
  - When the user presses a key, an event is triggered
- We want this project to be used in conjunction with Operating System classes (CS204).
  - Students end up having a good theoretical understanding but have no idea how to implement the concepts in code.
  - Students can examine the codebase to understand how the concepts are implemented.
  - This can encourage to students to contribute code to linux kernel and other open source projects.
- Additionally we want students to understand how to write clean code and design good system architectures.
- Students can also be introduced to basic concepts of devOps such as:
  - How to use source control and proper etiquettes to follow when committing and so on ...
  - How to do unit testing
  - How to write testable code
  - How to design workflows that shorten development cycles etc ...

## Existing Technology (Mathew)

- Linux Kernel
  - Extremely large; contains about 27.8 million lines of code
  - Code is complicated since it supports a large variety of architecturally dependent devices
  - Compiling takes a long time - about 1 hour
- Windows Research Kernel
  - Not open source
  - Difficult to get access
    - Needs to register through University
  - Outdated - Not in development
    - Based on older windows XP / Server 2003 code ; Therefore outdated
- Minix microkernel
  - The best introductory kernel out of all the ones mentioned here
  - Developed by Andrew S. Tanenbaum  
(Fun Fact: If the name sounds familiar it's because he authored our computer networks textbook)
  - However, the kernel has a micro-kernel design instead of the familiar monolithic kernel
    - Traditional unix based kernels (eg: linux) follow monolithic design
    - This has led to the famous [Tanenbaum–Torvalds debate](#).

- Explain [micro-kernel](#) & [monolithic kernel](#)
- Redox
  - Similar to MINIX (based on microkernel design)
  - Written in Rust programming language
    - Rust is a systems programming language with guaranteed safety
  - Has all the disadvantage of MINIX
  - Additionally students have to learn the Rust Programming language
- Proposed System
  - The system we propose will be written primarily in C; a language that is familiar to all undergraduate students
  - It will have a monolithic design; so that the knowledge and experience gained by tinkering with our project can directly translate when the student starts to play with the Linux kernel
  - Open source
  - Plenty of documentation