- 1. Reading 1: Operating Systems Principles
  - 1.1. Part 1: Introduction
  - 1.2. Part 2: Complexity Management Principles
    - 1.2.1. 2.1 Layered Structure and Hierarchical Decomposition
    - 1.2.2. 2.2 Modularity and Functional Encapsulation
    - 1.2.3. 2.3 Appropriately Abstracted Interfaces and Information Hiding
    - 1.2.4. 2.4 Powerful Abstractions

Week 1 Readings notes

Grand Summary:

1. Reading 1: Operating Systems Principles

Summary:

1.1. Part 1: Introduction

More size and complexity in software -> greater problems

Operating Systems: Super complex software to take care of this complexity. Some of which include:

- async interactions
- sharing of stateful resources
- coordinating actions with mix and matched components
- evolution to new tech and methodologies
- portability to any computer

The study of operating systems and their history allows us to have a niche field to tackle these hard problems

1.2. Part 2: Complexity Management Principles

1.2.1. 2.1 Layered Structure and Hierarchical Decomposition

Heiarchichal Decomposition: the process of decomposing a system in a top-down fashion

- Lets you hash out the mission of each group and its interaction with other grouops

1.2.2. 2.2 Modularity and Functional Encapsulation

Abitrarily assigning sub groups does not reduce the complexity. We must choose each group intentionally, such that:

- Each group has a coherent purpose
- Functions can be performed entirely within that group
- The union of groups is able to achieve the system's grand purpose

How to look at groups: We want to isolate these sub groups so we can look at that group alone and not worry about the functions of every group

- Look at that groups role + that group's internal structure + that group's operating rules

  In more Depth:

  - Smaller components is easier to understand
  - Grouping components to combine all closely related operations is better to combat side effects between groups
  - Big components are more efficient, however:
    - less communication between components reduces overhead
    - decreased opportunities for confusion because the component is a black box that does everything
    - increased dependencies increases errors and confusion

  Cohesion: modular design with the smallest possible modules but grouped effectively in their co-locations. A module that has this characteristic is *cohesive*

1.2.3. 2.3 Appropriately Abstracted Interfaces and Information Hiding

We want well-abstracted implementations so that the users can easy use them, but also not be forced to understand alot about the implementation

Well abstracted interfaces are:

- opaque: can't see or know the implemetnation
- information hiding: not exposing implementation details or unwanted information
- good in flexibity to the implmentters
  - ex. Instead of a cold and hot knob for faucet just a dial for temperature control

1.2.4. 2.4 Powerful Abstractions