COMP310/ECSE427 Study guide

Francis Piche

September 17, 2018

Contents

Part I
Preliminaries

1 Disclaimer

These notes are curated from Professor Muthucumaru Maheswaran COMP310/ECSE427 lectures at McGill University, and *A. Tenenbaum and H. Bos, Modern Operating Systems, 4th Edition, Pearson, 2015.* They are for study purposes only. They are not to be used for monetary gain.

2 About This Guide

I make my notes freely available for other students as a way to stay accountable for taking good notes. If you spot anything incorrect or unclear, don't hesitate to contact me via Facebook or e-mail at `http://francispiche.ca/contact/`.

Part II
Overview of OS Concepts

3 Introduction

3.1 What is an OS?

An operating system is a trusted software which interfaces between the hardware and user applications to provide:

- Security
- Usability
- Efficiency
- Abstractions
- Resource management

They attempt to solve the problem of maximizing utilization, and minimizing idle time, to maximize throughput.

3.2 Design Concerns For Different OS's

For a personal/embedded system: response time should be minimal

For a time-sharing system: there should be *fair* time sharing

In batch-processing systems: goal is to maximize throughput.

3.3 Booting

First, the hardware is powered and the CPU is in ”real-mode” which is essentially ”trust everything mode”. From here, the BIOS are loaded from ROM, and the CPU switches to ”protected” or ”user” mode. Finally the Kernel finishes initialization and the kernel services (OS) are started.

3.4 Processes

A process is a running program. Each process has an ”address space” or ”**core image**” in memory which it is allowed to use.

This address space contains the program's:

- executable code
- data (variables etc)
- call stack

The **process table** is an array of structures containing each process in existence. This includes all of the state information about each program, even if it is in a suspended or background state. (Some programs may run periodically or in the background).

A **child process** is a process that was created by another process.

Each person using a system is assigned a **UID**. Each process has the UID of the person who started it. Child processes have the UID of their parent.

Multiple processes allows for better utilization since while one process is idle (for example waiting for I/O) another process can work.

3.5 Memory Management

Processes need to be kept separate from other processes, and memory must also allow more than one process to exist in RAM at the same time.

We might also need to have these processes communicate.

Thanks to virtual memory, the address space can be larger than the actual amount of physical memory addresses. For more on virtual memory, see my COMP273 guide (or this guide in the later sections.) This also allows for better ”chunking” of memory to keep processes separate. We can also provide shared memory spaces for inter-process communcation. Virtual memory also allows for processes to not care where in memory they actually are, which is useful since they may be moved around when they get ”kicked out” by another process.

3.6 Storage

Need persistent storage, but it's slow. A hard disk is broken up into blocks which contain binary data. So when you use files, for example, they are saved as a series of blocks of binary data on the secondary storage.

3.7 OS and Interrupts

There are two kinds of Interrupts, hardware and software. Hardware is when a device sends a signal to the CPU, for example, a mouse is moved and needs to be processed. Software is when a program (OS or otherwise) throws an exception (trap). This alters the regular flow of the CPU and is therefore an interrupt.

3.8 Dual-Mode OS

Dual mode is the idea of keeping Users (unprivileged) and the kernel (privileged) separate. This provides greater security since this ensures on the trusted OS can make potentially dangerous operations on the core of the computer.

When a program running in user mode sends a system call, the OS then switches to kernel mode to complete the operation, and back when complete.

3.9 Monolith vs Micro-Kernel

There are two main architectures for an OS, the Monolith (one big program that handles everything) and the Micro-kernel. The latter's kernel is just the bare minimum inter-process communication, while the rest is a series of micro-services each capable of executing one OS task.

Part III
Scheduling

Part IV
Memory & Virtualization

Part V
Security