Notes from *Operating Systems Three Easy Pieces*

Sachidananda Urs

May 24, 2021

Abstract

This document contains the introductory notes from the book OS Three Easy Pieces. The book aims to teach operating systems by presenting two types of projects "system programming" projects and "XV6" based projects. See https://github.com/remzi-arpacidusseau/ostep-projects for more details.

Introduction

The three pieces the book aims to teach are *virtualization*, *concurrency*, and *persistence*.

When we talk about *virtualization*, we look at the virtualization of **CPU** and **memory**.

When we talk about *concurrency*, we look at threads, locks, condition variables, semaphores, etc. And common problems in concurrency, event based concurrency, ...

When we talk about *persistence* we talk about I/O devices, hard disk drives, RAID, files and directories, file system implementation ...

We also look into the basics of distributed systems, NFS, and AFS.

Virtualization

We try to understand the virtualization of *CPU* and *Memory*

CPU Virtualization

We study the abstraction (a process), APIs, states of a process, the data structures involved, limited direct execution, and scheduling. Scheduling is studied extensively.

Memory Virtualization

We study the abstraction of memory (address space), APIs, address translation, segmentation, free space management, paging and TLBs, swap space, policies, case studies (VAX/VMS, Linux, illumos).

Concurrency

We study the importance of concurrency and why we even bother about it. And emphasize on threads, thread APIs, locks, condition variables, semaphores, ...

Persistence

We study the I/O devices, viz hard disk drives, SSDs, and RAIDs. And the related software concepts of files and directories, file system implementation case studies, FFS, data integrity, so on.

Introduction to Operating Systems

We try to answer a simple question *"How does the operating system virtualize resources?"*

Breaking the question even further we ask:

1. What mechanism and policies are implemented by the operating system to attain virtualization?
2. How does the operating system do it efficiently?
3. What hardware support is needed?

On Virtualization

Virtualization is a technique where operating system takes a physical resource (processor, memory, or disk) and transforms it into a more general, powerful, and easy-to-use virtual form of itself.

Thus operating system is sometimes referred to as a virtual machine.

Now the question arises, how to use these resources? Or how to interact with the operating system? This is done through a set of APIs called "System calls".

Summarily, virtualization allows many programs to run (sharing CPU), many programs can concurrently access their data and instructions (sharing memory), many programs can access devices (by sharing disks, network etc).

We examine some of the examples for:

1. Virutalizing the CPU
2. Virtualizing the memory
3. Concurrency
4. Persistence

Virtualizing the CPU

Consider a program which sleeps for a second and prints a string. Now, if the program is run multiple times parallelly, it appears to work simultaneously though we have a single CPU. How is this achieved? The OS provides an illusion of multiple virtual CPUs. The operating system virtualizes the CPU.

Now, since the operating system runs multiple programs "simultaneously", if two programs want to run at a particular time, which one should run first?

Virtualizing the memory

We try to answer the question, *"What is memory virtualizing?"*

Each process accesses its own private *virtual address space*, which the operating system somehow maps to physical memory of the machine. Memory

reference within one running program does not affect the address space of the other process. As far as running program is concerned it has the entire memory for itself. However, the physical memory is a shared resource, managed by the operating system.

How does the operating system manage all this stuff is the concept called the virtual memory.

Concurrency

If a threaded program is written carelessly, concurrency leads to all sorts of problems. The program does not behave as expected. By understanding concurrency we can write these programs in a much better way.

Persistence

When we talk about persistence we mean file systems which manage the data storage and retrieval. We try to understand:

1. How is a file system written?
2. What techniques are needed to manage persistent data correctly?
3. How is high performance achieved?
4. How is reliability achieved?