NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4186 - Operating Systems

Course Information

Spring, 2024

Table of Contents

Introduction

This booklet presents the syllabus of TDT4186 – Operating Systems (OSs). The course consists of two parts, **theories** (lectures and exercises) and **practical exercises** (hand-on projects).

The textbook we mainly use for this course is:

> Remzi H Arpaci-Dusseau and Andrea C Arpaci-Dusseau (2021). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books, LLC

Other textbooks can also be a good complement, e.g.,:

> Silberschatz Abraham, Gagne Greg et al. (2014). *Operating System Concepts Essentials.-2nd*

Each section includes the reading chapters and learning objectives for the topics. The reading chapters are the contents we have in our lectures, but other unlisted chapters can also be interesting to read. Learning objectives are organized as a list of questions. There are some contents that are not covered in the textbook, but attending lectures and understanding lectures' slides should be sufficient to understand them.

This booklet is a work in progress. More contents need to be added.

1 Operating systems

This chapter is not covered in the textbook.

1.1 Notes

This chapter mainly discusses the purpose of OSs, where it serves as a resource management layer between underlying hardware and user applications.

Three main structures are discussed in the lecture, but there exist other structure types.

- **Monolithic kernel**: The entire OS is in kernel mode/space[1], including *device driver, memory management, file systems*, etc. The representative example is Linux.
  - Pros: good isolation, good security guarantee, better performance
  - Cons: inflexibility, reliability, larger size in kernel space
- **Microkernel**: Only the most fundamental services required, e.g., memory management, scheduling, etc, are implemented in kernel mode/space, and other system functions or services, like device drivers, are implemented in user space.
  - Pros: reliability, security, flexibility, smaller size in kernel space
  - Cons: complexity, communication overhead
- **Hybrid kernel**: A hybrid kernel strives to combine monolithic kernel with microkernel. Compared to a microkernel, a hybrid kernel can have some services, functions, or device drivers implemented in the kernel space, which thus differentiates it from a monolithic kernel. As a result, hybrid kernel can be considered as a tradeoff between monolithic kernel and microkernel.
  - Pros and Cons: It inherits the pros and cons from both monolithic kernels and microkernels.

1.2 Learning Objectives

Questions you should be able to answer after this chapter

- What are the three OS structures?
- What are the pros and cons of each OS structure?
- Which OS structure do modern OSs use? E.g., Windows, Linux.
- What are the design goals for OSs?

[1] mode and space can be used interchangeably in this context

2 Process

Chapter	Name
4	The Abstraction: The Process
5	Interlude: Process API
6	Mechanism: Limited Direct Execution

Table 1: Reading chapters

2.1 Notes

In general, a **process** in general is a running program that provides a CPU abstraction for program execution.

A process is a dynamic object, whereas a program is a static object.

Executing a command within a Linux terminal is a representative example of using fork(), exec(), and wait(). Terminal/Shell is a program, so a process is generated when Terminal is launched. The following snippet is from shell.c of xv6. Terminal is always waiting for a command, and then some pre-processing will be conducted to determine what kind of command it is receiving. fork1() creates a new process, where the return value is used to check whether the executing one is the child process. In the child process, runcmd() replaces the content of the parent process with the new command. wait() ensures that the child process finishes before the parent process, i.e., Terminal.

```
while(getcmd(buf, sizeof(buf)) >= 0){
    if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
      // Clumsy but will have to do for now.
      // Chdir has no effect on the parent if run in the child.
      buf[strlen(buf)-1] = 0;  // chop \n
      if(chdir(buf+3) < 0)
        fprintf(stderr, "cannot cd %s\n", buf+3);
      continue;
    }
    if(fork1() == 0)
      runcmd(parsecmd(buf));
    wait(&r);
}
```

2.2 Learning Objectives

Questions you should be able to answer after this chapter

- What is a process? and what is the main difference between a program and a process?
- What do fork(), exec(), and wait() do?
  - Given a code snippet with these functions, you should know how to analyze the code and decide what is the output.
- What are the two modes of OSes? Why does an OS need the two modes? How does a process use the two modes?
- What are the two approaches OS can use to switch processes?

3 Scheduling

Chapter	Name
7	Scheduling: Introduction
8	Scheduling: The Multi-Level Feedback Queue
9	Scheduling: Proportional Share

Table 2: Reading chapters

3.1 Learning Objectives

Questions you should be able to answer after this chapter

- What are *turnaround time*, *waiting time*, and *response time*?
- What is starvation in scheduling algorithms?
- What are FIFO, SJF, STCF, RR, MLFQ? What are pros and cons of different scheduling algorithms?
  - Given a job set with all needed information, such as arrival time, execution time, etc, you should be able to compute *average turnaround time* and *average response time/waiting time* for different scheduling algorithms.
- What is the default scheduling algorithm of Linux? How does it work?

4 Memory

Chapter	Name
13	The Abstraction: Address Spaces
15	Mechanism: Address Translation
16	Segmentation
17	Free-Space Management

Table 3: Reading chapters

4.1 Notes

An **address space** is the memory abstraction for a process, providing better isolation between processes and an effective way to manage the memory.

A **memory management unit (MMU)** is a special hardware to facilitate the translations of virtual addresses to physical addresses.

4.2 Learning Objectives

Questions you should be able to answer after this chapter

- What is the address space of a process? What does an address space of a process usually consist of?
- What are virtual address and physical address?
- What are static relocation and dynamic relocation? How do they work respectively?
- What are the two memory allocation methods when allocating contiguous memory space? What are their respective pros and cons? (This is not in the textbook, but you can find this part from the lecture slides.)
- How do different allocation algorithms, such as FF, BF, and WF, work when allocating memory space?
  - Given a set of memory partitions and a set of processes with different memory requirements, you should know what would be the final allocation using a certain algorithm?
- What is segmentation? How does it work?

5 Page

Chapter	Name
18	Paging: Introduction
19	Paging: Faster Translations (TLBs)
20	Paging: Smaller Tables
21	Beyond Physical Memory: Mechanisms
22	Beyond Physical Memory: Policies

Table 4: Reading chapters

5.1 Notes

- **Paging/page mechanism**: the most used virtual memory management mechanism in modern OSs. It divides both virtual memory and physical memory into blocks of the same size.
- **Page:** In paging, blocks of virtual memory are called *pages*.
- **Page frame**: In paging, blocks of physical memory are called *page frames*, or simply *frames*.
- **Page table**: Page table is used to store the mapping of each page to its corresponding page frame. Usually, page table is a per-process structure, i.e., every process has a page table for its address space. For instance, the `proc` data structure has a data structure `pagetable_t`.

Since both pages and page frames have the same size, OS can benefit from such design in terms of simplicity and flexibility. paging is a fixed-parition method, so it may suffer from internal fragmentation.

Most OSs deploy a per-process page table, where each process maintains its own page table. Such method can provide better memory isolation between processes. But the inverted page table implements a system-wide page table that stores translations for all page frames (physical memory).

5.2 Learning Objectives

Questions you should be able to answer after this chapter

- What is page in OS? How does paging work in OS?
- How to use a virtual address to conduct page translation?
  - Given a virtual address space and page size, how many bits do we need for page number and offsets, respectively?
  - Given a memory system with all paging information, you should be able to translate some given virtual addresses to corresponding physical addresses.
- What is page table?
- What are the two problems of paging system without hardware assistance and solutions for the two problems?
- What is TLB? Why do we need TLB in paging system?
- What is page replacement? What are page hit and page miss?
- How do different page replacement policies/algorithms work, e.g., OPTIMAL, FIFO, LRU, etc?
  - Given a page replacement policies and a page access sequence, you should be able to calculate the average memory access time.

6 Concurrency

Chapter	Name
26	Concurrency: An Introduction
27	Interlude: Thread API

Table 5: Reading chapters

6.1 Learning Objectives

Questions you should be able to answer after this chapter

- What is thread? What are the differences between thread and process?
- What is POSIX? How do some common thread APIs from POSIX work, like pthread_create()?
- What is the concurrency issue?
  - Given a code snippet with threads, you should be able to analyze the output and identify the possible issue.
- What is the critical-section problem? What should a solution for the critical section problem contain?

7 Locks

Chapter	Name
28	Locks

Table 6: Reading chapters

7.1 Learning Objectives

Questions you should be able to answer after this chapter

- What is lock in OS?
- Why do locks need hardware instructions, like Test-And-Set() and Compare-And-Swap(), to implement locks?
- What is spinlock? What are pros and cons of spinlocks? What is the method to address the performance issue of spinlocks?
- What are ticket lock and queue lock?

8 Condition Variables

Chapter	Name
30	Condition Variables

Table 7: Reading chapters

8.1 Notes

Conditional variables are often used in conjunction with locks in order to prevent race conditions.

8.2 Learning Objectives

Questions you should be able to answer after this chapter

- What is condition variable? Why do we need condition variable?
- What is producer/consumer problem? How to implement a solution for P/C problem using condition variable?
  - Given a code snippet, you should be able to analyze it.

9 Semaphores

Chapter	Name
31	Semaphores
32	Common Concurrency Problems

Table 8: Reading chapters

9.1 Learning Objectives

Questions you should be able to answer after this chapter

- What is semaphore? What are the operations semaphores have?
- How to use semaphore to implement solutions for mutual exclusion, ordering, and P/C problem?
  - Given a code snippet, you should be able to analyze it.
- What is deadlock? What are the four conditions for deadlocks?

10 I/O Devices

Chapter	Name
36	I/O Devices
37	Hard Disk Drives
44	Flash-Based SSDs

Table 9: Reading chapters

10.1 Learning Objectives

Questions you should be able to answer after this chapter

- Why do I/O systems deploy a hierarchical structure?
- What are port, bus, controller, and device driver?
- What are I/O interrupt and polling? Pros and cons.
- What are I/O instructions and memory-mapped I/O?
- What are programmed I/O and DMA?
- What are the key components of hard disk drives?
  - Given a hard disk specification, you should be able to calculate average access time.
- What are SSDs?
  - How is an SSD composed? Different components.
  - What are the operations of SSDs? Read, erase, programming.

11 File Systems

Chapter	Name
39	Interlude: Files and Directories
40	File System Implementation

Table 10: Reading chapters

11.1 Notes

11.1.1 File and File System

File system: A method and structure to manage storage devices, like HDDs and SSDs, and external storage devices (USB flash). It faciliate the management, access, retrieval of files and directories.

11.1.2 File Naming

Three types of naming for files:

- ***inode***: A data structure to store metadata of a file. It has a unique number which is like process ID, and every file in the disk should have an inode. We can index a file using its inode number.
  - Cons: A modern computing system may have thousands or millions of files, so it is difficult for users to remember all inode numbers and use these numbers to manipulate files.
- ***path***: Path is a human-readable identifier frequently used to access files, e.g., /user/test.txt in Unix-like systems or C:\Windows\test.txt in Windows.
  - A file's path is associated with an inode, allowing the OS to index the file using the path name.
  - The path-inode link/mapping is stored in the directory which owns the file.
- ***file descriptor***: A file descriptor is a run-time identifier (unsigned integer) used by processes to reference an open file.
  - Every process maintains a file descriptor table to store all open file descriptors in use.
  - Every process has three file descriptors by default, 0 for STANDIN, 1 for STANDOUT, and 2 for STANDERR

11.2 Learning Objectives

Questions you should be able to answer after this chapter

- What are file and file system?
- What are the three names OS uses for a file?
- What do the common file operations, e.g., open(), read(), etc, perform? What is the *offset* of a file and how do different file operations change the *offset*?
  - Given a code snippet, you should be able to analyze it.
- How is the on-disk structure organized?
- What is directory?
- How do different file operations access the on-disk structure?

Bibliography

Abraham, Silberschatz, Gagne Greg et al. (2014). *Operating System Concepts Essentials.-2nd*.

Arpaci-Dusseau, Remzi H and Andrea C Arpaci-Dusseau (2021). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books, LLC.

To-Do List

- Add learning notes to each section. (In progress)
- Add related file names and data structures from XV6 to each section.