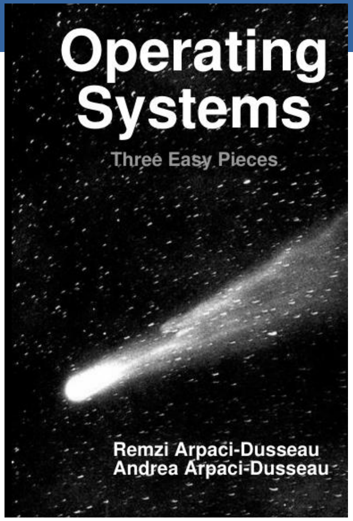
Operating System (OS)

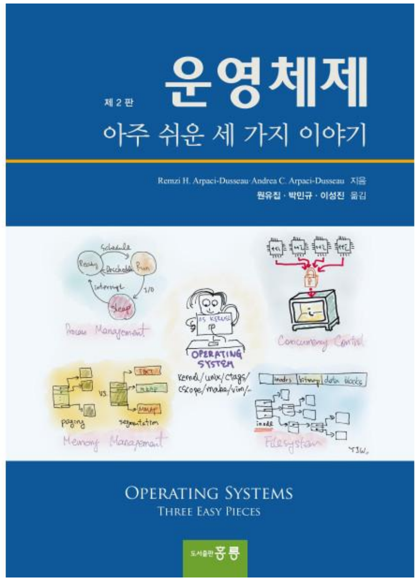
안인규

교재: Operating Systems: Three Easy Pieces

- **Operating Systems: Three Easy Pieces**
  - **http://www.ostep.org**
  - Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau (University of Wisconsin-Madison)
  - NEW: Security Chapters by Peter Reiher (UCLA)



- 운영체제-아주 쉬운 세가지 이야기 (홍릉과학출판사)


제 2 판
운영체제
아주 쉬운 세 가지 이야기
Remzi H. Arpaci-Dusseau·Andrea C. Arpaci-Dusseau 지음
원유집 · 박민규 · 이성진 옮김
Operating Systems
Three Easy Pieces
홍릉

수업계획

Virtualization		Concurrency	Persistence
3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*
★4 Processes	★13 Address Spaces code	★26 Concurrency and Threads code	36 I/O Devices
★5 Process API code	★14 Memory API	★27 Thread API code	★37 Hard Disk Drives
★6 Direct Execution	★15 Address Translation	★28 Locks code	38 Redundant Disk Arrays (RAID)
★7 CPU Scheduling	★16 Segmentation	★29 Locked Data Structures	★39 Files and Directories
★8 Multi-level Feedback	★17 Free Space Management	★30 Condition Variables code	★40 File System Implementation
★9 Lottery Scheduling code	★18 Introduction to Paging	★31 Semaphores code	★41 Fast File System (FFS)
10 Multi-CPU Scheduling	★19 Translation Lookaside Buffe	★32 Concurrency Bugs	★42 FSCK and Journaling
11 *Summary*	★20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)
	★21 Swapping: Mechanisms	34 *Summary*	44 Flash-based SSDs
	★22 Swapping: Policies		45 Data Integrity and Protection
	23 Complete VM Systems		46 *Summary*
	24 *Summary*		47 *Dialogue*
			48 Distributed Systems
			49 Network File System (NFS)
			50 Andrew File System (AFS)
			51 *Summary*

수업계획

- 강의 진행:
  - PPT 강의교재 (ecampus.koomin.ac.kr 에 업로드)
  - 실습과제 - (15%)
  - 출석 - (5%)
  - 중간 (40%), 기말고사 (40%)

- 질문:
  - Inkyu.an@kookmin.ac.kr

수업계획

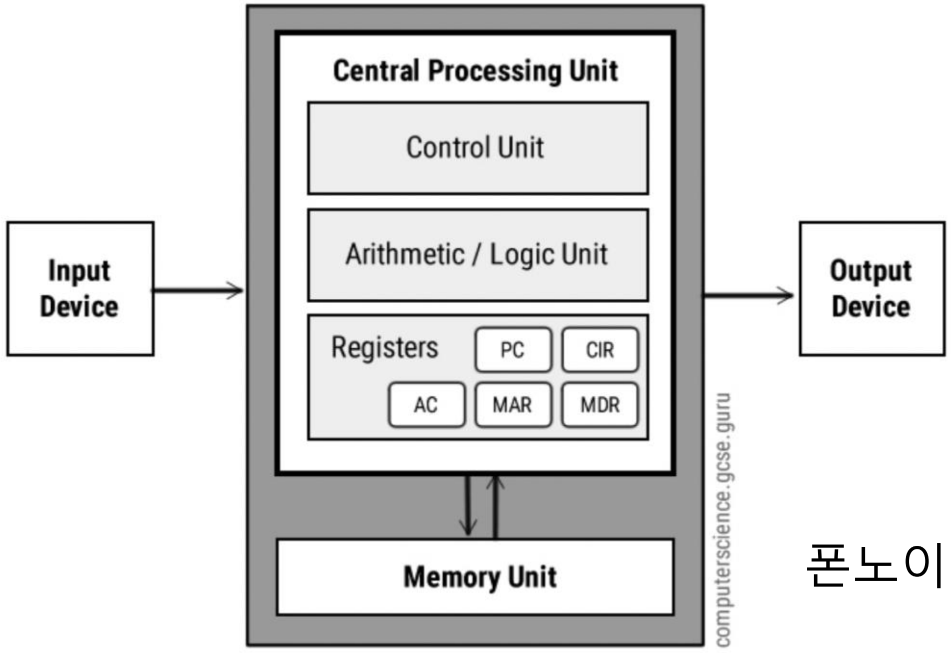
- 평가 (성취기반평가):
  - 1단계: 기본 등급 결정
    - 기본등급 결정 기준: 중간 고사 성적 분포에 따름
    - 중간고사, 기말고사, Quiz 성적의 절대값에 따라 분류
  - 2단계: 등급 조정 효과
    - 지속적 노력이 필요한 퀴즈, 숙제, 출석 등을 기반으로 함
    - 개인의 성취에 따라 등급이 상향 조정되는 효과가 있음
    - 난이도의 차이로 기말고사 평균 성적에 영향이 있을 등급 결정 기준을 조정함

Introduction

Operating System (OS)
안인규

What happens when a program runs?

- A running program executes instructions
  1. The processor fetches an instruction from memory
  2. The processor decodes it (i.e., figures out which instruction this is)
  3. The processor execute it
  4. The processor moves on to the next instruction, until the program finally completes



폰노이만 구조

Operating System (OS)

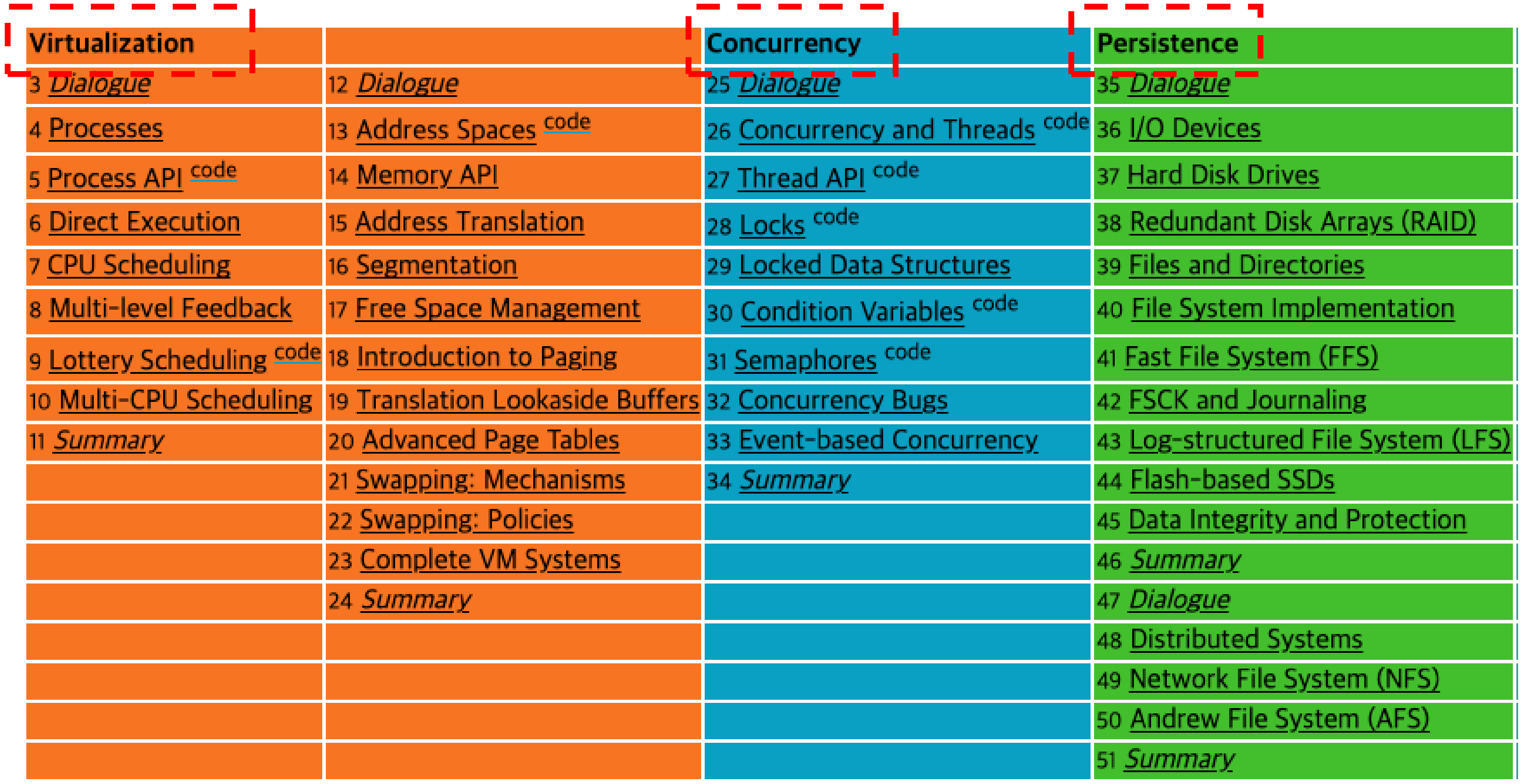
- Operating System (OS)
  - Make it easy to run programs
  - Allow programs to share memory
  - Enable programs to interact with devices

→ **OS is in charge of making sure the system operates <u>correctly</u> and <u>efficiently</u>**

Operating System (OS)

OS is in charge of making sure the system operates correctly and efficiently

가상화 동시성 영속성



Virtualization		Concurrency	Persistence
3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*
4 Processes	13 Address Spaces code	26 Concurrency and Threads code	36 I/O Devices
5 Process API code	14 Memory API	27 Thread API code	37 Hard Disk Drives
6 Direct Execution	15 Address Translation	28 Locks code	38 Redundant Disk Arrays (RAID)
7 CPU Scheduling	16 Segmentation	29 Locked Data Structures	39 Files and Directories
8 Multi-level Feedback	17 Free Space Management	30 Condition Variables code	40 File System Implementation
9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores code	41 Fast File System (FFS)
10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK and Journaling
11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)
	21 Swapping: Mechanisms	34 *Summary*	44 Flash-based SSDs
	22 Swapping: Policies		45 Data Integrity and Protection
	23 Complete VM Systems		46 *Summary*
	24 *Summary*		47 *Dialogue*
			48 Distributed Systems
			49 Network File System (NFS)
			50 Andrew File System (AFS)
			51 *Summary*

Virtualization (가상화)

- OS takes a physical resource (e.g., the processor, memory, or a disk)
- Transform a physical resource into a more general, powerful, and easy-to-use virtual form of itself

- System calls
  - OS provides some interfaces (APIs)
    → Running a program, allocating memory, or accessing a file

Virtualization

- The OS manages resources (CPU, memory, and disk)
- The OS allows
  - many programs to run (thus sharing the CPU)
  - many programs to concurrently access their own instructions and data (thus sharing memory)
  - many programs to access devices (thus sharing disks)

Virtualization (CPU)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: cpu <string>\n");
                exit(1);
        }
        char *str = argv[1];
        while (1) {
                Spin(1);
                printf("%s\n", str);
        }
        return 0;
}
```

<cpu.c>

Virtualization (CPU)

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
A
ˆC
prompt>
```

“Control + c” → You can halt the program

Virtualization (CPU)

- Running many programs at once …

```
prompt> ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A …
```

Virtualizing the CPU!
→ The illusion that the system has a very large number of virtual CPUS

Virtualization (Memory)

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: mem <value>\n");
                exit(1);
        }
        int *p = malloc(sizeof(int));  // a1: allocate some memory
        assert(p != NULL);
        printf("(%d) address of p: %08x\n", getpid(), (unsigned) p);
                // a2: print out the address of the memory
        *p = atoi(argv[1]); // assign value to addr stored in p
        while (1) {
                Spin(1);
                *p = *p + 1;
                printf("(%d) p: %d\n", getpid(), *p); // a4
        }
        return 0;
}
```

<mem.c>

Virtualization (Memory)

```
prompt> ./mem 1
(2134) address pointed to by p: 0x200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

변수 p가 할당된 메모리 주소: 0x200000

Virtualization (Memory)

- Running the program (mem.c) multiple times

```
prompt> ./mem 1 & ./mem 5 &
[1] 24113
[2] 24114
(24113) address pointed to by p: 0x200000
(24114) address pointed to by p: 0x200000
(24113) p: 1
(24114) p: 6
(24114) p: 2
(24113) p: 7
(24113) p: 3
(24114) p: 8
(24113) p: 4
(24114) p: 9
...
```

Virtualization (Memory)

- Running the program (mem.c) multiple times

```
prompt> ./mem 1 & ./mem 5 &
[1] 24113
[2] 24114
(24113) address pointed to by p: 0x200000
(24114) address pointed to by p: 0x200000
(24113) p: 1
(24114) p: 6
(24114) p: 2
(24113) p: 7
(24113) p: 3
(24114) p: 8
(24113) p: 4
(24114) p: 9
...
```

Virtualizing memory!

- 프로그램 (./mem)들이 같은 physical memory를 공유하는 것이 아닌, 각각 private memory를 가지고 있는 것처럼 보인다

Virtualization (Memory)

- Each process accesses its own private virtual address space (Virtualizing memory)
  - The OS maps address space onto the physical memory
  - A memory reference within one running program does not affect the address space of other processes
  - Physical memory is a shared resource, managed by the OS

Concurrency (동시성)

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
        int i;
        for (i = 0; i < loops; i++) {
                counter++;
        }
        return NULL;
}

int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr,
               "usage: threads <value>\n");
               exit(1);
        }
        loops = atoi(argv[1]);
        pthread_t p1, p2;
        printf("Initial value :
               %d\n", counter);

        Pthread_create(&p1, NULL,
                       worker, NULL);
        Pthread_create(&p2, NULL,
                       worker, NULL);
        Pthread_join(p1, NULL);
        Pthread_join(p2, NULL);
        printf("Final value :
               %d\n", counter);
        return 0;
}
```

<thread.c>

- **Thread: a function running within the same memory space.**

Concurrency

- *loops* determines how many times each of the two workers will increment the shared counter in a loop
- *loops*: 1000

```
prompt> gcc -o thread thread.c -Wall -pthread
prompt> ./thread 1000
Initial value : 0
Final value : 2000
```

- *loops*: 2000

```
prompt> ./thread 100000
Initial value : 0
Final value : 143012 // huh??
prompt> ./thread 100000
Initial value : 0
Final value : 137298 // what the??
```

Concurrency

- Increment a shared counter → take **three instructions**
    1. Load the value of the counter from memory into register.
    2. Increment it
    3. Store it back into memory

- Three instructions이 한번에 (atomically) 실행되지 않고, 개별적으로 실행된다 → 즉, 중간 중간 무엇인가에 방해를 받는다!

- Problem of **concurrency** happen

Persistence

- Devices such as DRAM store values in a volatile (휘발성 물질).
- *Hardware* and *software* are needed to store data persistently (영구적으로).
  - **Hardware**: I/O device such as a hard drive, solid-state drives(SSDs)
  - **Software**:
    - File system manages the disk.
    - File system is responsible for storing any files the user creates.

Persistence

- Create a file `(/tmp/file`) that contains the string “hello world”
  - `open()`, `write()`, and `close()` system calls are routed to the part of OS called the file system, which handles the requests

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>

int
main(int argc, char *argv[])
{
        int fd = open("/tmp/file", O_WRONLY | O_CREAT
                            | O_TRUNC, S_IRWXU);
        assert(fd > -1);
        int rc = write(fd, "hello world\n", 13);
        assert(rc == 13);
        close(fd);
        return 0;
}
```

Persistence

- What OS does in order to write to disk?
  - Figure out **where** on disk this new data will reside
  - **Issue I/O requests** to the underlying storage device (기본 저장 장치)

- File system handles system crashes during write.
  - **Journaling** or **copy-on-write**
  - Carefully <u>ordering</u> writes to disk

Design Goals

- Build up **abstraction (=virtualization)**
  - Make the system convenient and easy to use.

- Provide high **performance (성능)**
  - Minimize the overhead of the OS.
  - OS must strive to provide virtualization without excessive overhead.

- **Protection** between applications
  - Isolation: Bad behavior of one does not harm other and the OS itself.

Design Goals

- High degree of **reliability (신뢰성, 안정성)**
  - The OS must also run non-stop.
- Other issues
  - Energy-efficiency
  - Security
  - Mobility