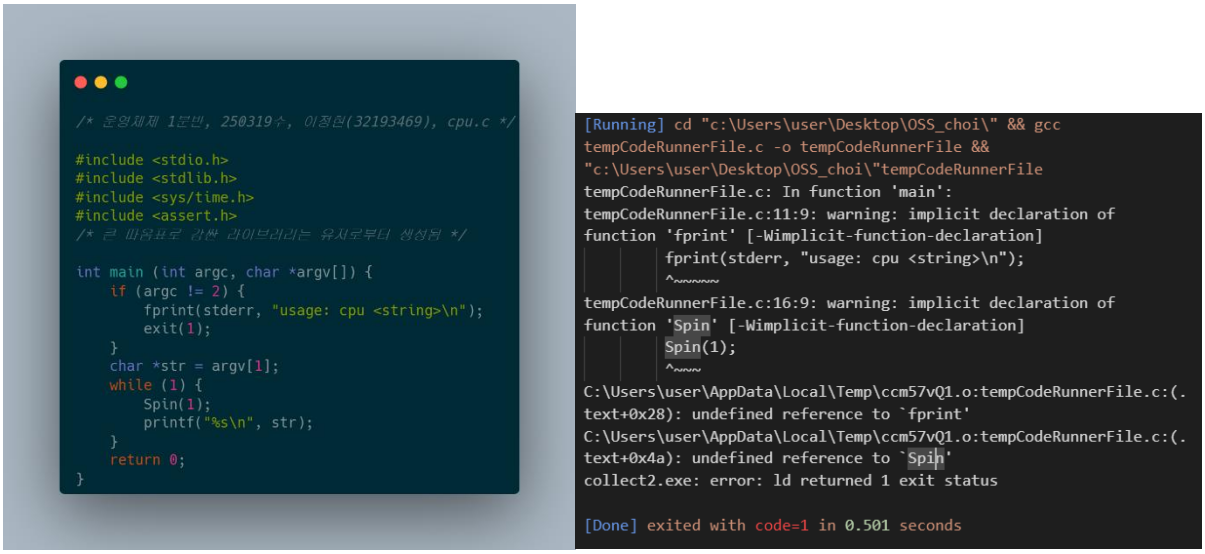
REPORT

운영체제 리포트 #1

- OSTEP 2장 요약 및 분석 -

주 제 : OSTEP 2장 요약 및 분석
교 수 : 최종무 교수님
학 과 : 소프트웨어학과
학 번 : 32193469
이 름 : 이정현
제 출 : 2025. 4. 17.(목)
이메일 : 32193469@dankook.ac.kr
전 화 : 010-7518-3803

2.1 Virtualizing CPU (CPU 가상화)



```c
/* 운영체제 1분반, 250319수, 이정현(32193469), cpu.c */

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
/* 큰 따옴표로 감싼 라이브러리는 유저로부터 생성됨 */

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

```
[Running] cd "c:\Users\user\Desktop\OSS_choi\" && gcc
tempCodeRunnerFile.c -o tempCodeRunnerFile &&
"c:\Users\user\Desktop\OSS_choi\"tempCodeRunnerFile
tempCodeRunnerFile.c: In function 'main':
tempCodeRunnerFile.c:11:9: warning: implicit declaration of
function 'fprint' [-Wimplicit-function-declaration]
        fprint(stderr, "usage: cpu <string>\n");
        ^~~~~~
tempCodeRunnerFile.c:16:9: warning: implicit declaration of
function 'Spin' [-Wimplicit-function-declaration]
        Spin(1);
        ^~~~
C:\Users\user\AppData\Local\Temp\ccm57vQ1.o:tempCodeRunnerFile.c:(.
text+0x28): undefined reference to `fprint'
C:\Users\user\AppData\Local\Temp\ccm57vQ1.o:tempCodeRunnerFile.c:(.
text+0x4a): undefined reference to `Spin'
collect2.exe: error: ld returned 1 exit status

[Done] exited with code=1 in 0.501 seconds
```

Figure 1: [그림 2.1] cpu.c 출력 코드 및 실행 결과

표준 C언어로 작성된 그림 2.1은 Spin()이라는 이름의 [1]Lock을 주기 위한 함수을 호출하는 코드입니다. 여기서 Spin()의 역할은 해당 구문을 1초 동안 실행한 후, 리턴하는 함수입니다. 스핀이 종료되면 사용자가 printf()를 통해 전달한 문자열을 출력합니다.

출력 결과는 프로그램 수행 후 1초가 지나면 사용자가 전달한 입력 문자열을 출력합니다. 참고로 반복 실행되는 이 프로그램을 중단하기 위해서는 Ctrl + C 단축키가 필요하다. *컴파일 옵션은 다음과 같다:* ***gcc -o cpu cpu.c -Wall***

컴파일 옵션 중 -o를 생략한다면 기본 파일명인 a.c라는 파일명으로 생성된다. 15p의 parallel 수행한 코드 결과를 보면, 컴파일 시에 프로세스가 1개였지만 프로그램을 4개 모두 동시에 실행한 것처럼 보인다. 그 이유는 운영체제가 시스템 내부에 여러 개의 가상 CPU가 존재하는 것처럼 가짜로 만들어 낸 것이다. 이처럼 하나의 CPU를 이용해 여러 개의 CPU가 동시에 존재하는 것처럼 보이게 하여 여러 개의 프로그램을 한 번에 동시에 수행할 수 있게 해주는 개념이 바로 **CPU 가상화 (Virtualizing CPU)**라고 한다.

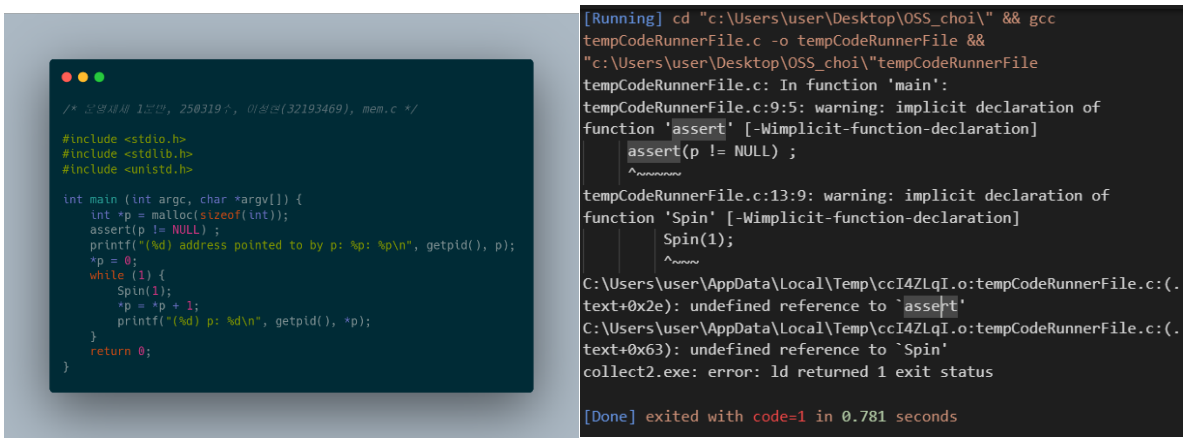
▶ [1] 새로운 프로그램을 실행하는 방법 → 프로세스 (process)

▶ [2] 다음 순서에 실행할 프로세스를 선택하는 방법 → 스케줄링 (scheduling)

---

[1] Spin(): 특정 임계 영역에 진입할 때까지 루프를 돌면서 재시도를 반복하는 방식으로 구현된 Lock 연산의 일종.

2.2 Lab 1: CPU Virtualization



```
/* 운영체제 1분반, 250319수, 이정현(32193469), mem.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char *argv[]) {
    int *p = malloc(sizeof(int));
    assert(p != NULL) ;
    printf("(%d) address pointed to by p: %p: %p\n", getpid(), p);
    *p = 0;
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p);
    }
    return 0;
}
```

```
[Running] cd "c:\Users\user\Desktop\OSS_choi\" && gcc
tempCodeRunnerFile.c -o tempCodeRunnerFile &&
"c:\Users\user\Desktop\OSS_choi\"tempCodeRunnerFile
tempCodeRunnerFile.c: In function 'main':
tempCodeRunnerFile.c:9:5: warning: implicit declaration of
function 'assert' [-Wimplicit-function-declaration]
    assert(p != NULL) ;
    ^~~~~~
tempCodeRunnerFile.c:13:9: warning: implicit declaration of
function 'Spin' [-Wimplicit-function-declaration]
        Spin(1);
        ^~~~
C:\Users\user\AppData\Local\Temp\ccI4ZLqI.o:tempCodeRunnerFile.c:(.
text+0x2e): undefined reference to `assert'
C:\Users\user\AppData\Local\Temp\ccI4ZLqI.o:tempCodeRunnerFile.c:(.
text+0x63): undefined reference to `Spin'
collect2.exe: error: ld returned 1 exit status

[Done] exited with code=1 in 0.781 seconds
```

[그림 2.2] mem.c 출력 코드 및 실행 결과

표준 C언어로 작성된 그림 2.2은 malloc() 함수를 호출하여 특정 포인터 변수에 메모리를 할당하는 코드입니다.

※ Scheduling (스케줄링 기법)

▶ 1) Round-Robin(RR)

▶ 2) Multi Level Feedback Queue(MLFQ)

▶ 3) Lottery

▶ 4) Stride

sched.cpp 이 cpp 파일의 코드 126줄을 한 줄 한 줄 모두 읽고, 어떤 프로그램인지 개조식으로 설명해줘. 추가적으로 이 cpp 파일에 RR 클래스, MLFQ 클래스, Lottery 클래스 , Stride 클래스 등 4가지 스케줄러 클래스를 구현하는 방법을 알려주고 직접 RR 클래스, MLFQ 클래스, Lottery 클래스 , Stride 클래스 스케줄러를 구현하는 cpp 코드를 첨부파일로 생성해줘.

위 언급한 4가지 스케줄러 코드를 생성할 때는 "아래와 같은 규칙을 지켜줘.

규칙:

- schd.cpp의 RR, FeedBack, Lottery, Stride 클래스를 구현한다

• 각 클래스의 1) 생성자, 2) int run() 함수를 수정 및 작성하여 구현한다

• 각 클래스의 위 2가시 함수의 선언은 수정할 수 없다

• 생성자는 부모 생성자를 호출하고, name을 초기화 해야한다 (기존 내용을 수정하지 만 것)

• 각 클래스의 멤버 변수/함수는 자유롭게 추가 가능하다

• 반드시 C++로 구현하야 하고 라이브러리는 C++ STL만 사용 가능하다

• sched.cpp 외 다른 파일은 수정, 추가, 제출이 불가하다 - RR과 FeedBack은 time quantum이 다르더라도, 동일한 class로 작성한다

• 생성자를 통해 time quantum을 전달 받는다 - FeedBack의 큐 개수는 4개, Boosting 정책은 없다

※ 목차

<퍼플렉>

https://www.perplexity.ai/search/sched-cpp-i-cpp-pailyi-kodeu-1-c1VnjeNAS..vhpEPbjnVCA#0

<지피티>

https://chatgpt.com/canvas/shared/6800d7c5243c81918c1fcf3cf5575f5b

https://chatgpt.com/canvas/shared/6800d7c5243c81918c1fcf3cf5575f5b

※ Virtualizing Memory 혼동 주의 (같은 주소이지만, 분리된 공간)

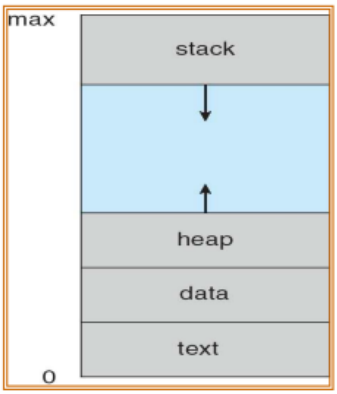
프로세스 A와 프로세스 B 내 포함된 변수 C가 저장된 주소가 모두 0x20000이라고 해서, 프로세서 A 내 포함된 변수 C와 프로세서 B 내 변수 C가 항상 같은 변수인 것은 아니다. 변수 C가 저장된 변수의 주소가 동일하기 때문에 두 프로세스가 변수 C를 공유하는 것처럼 보인다. **하지만 두 프로세스에 저장된 변수 C가 저장된 물리적 공간은 다르다**

사실 하나의 프로세스는 다른 프로세스와 자료 공유 (data sharing)를 하지 않는다. 따라서 프로세서 A에 포함된 변수 C와 프로세스 B에 포함된 변수 B는 서로 아무런 관련이 없는 전혀 다른 변수이다. 예시를 들자면, 지구 행성의 정현이와 화성 행성의 정현이는 다르다는 것과 비슷하다.

하나의 프로그램이 수행하는 여러 메모리 연산은 다른 프로그램의 주소 공간에 영향을 주지 않는다. 현재 실행 중인 프로세스의 입장에서는, 자기 자신만의 물리 메모리를 갖는 셈이다. 실제로는 물리 메모리는 공유 자원이고, 운영체제에 의해 관리된다.

▶ [1] 프로세스의 주소 공간을 관리하는 방법 → *주소 공간 (address space)

▶ [2] 프로세스들 사이에 메모리를 보호하는 방법 → 가상 메모리 (virtual memory)



[그림 2.2] 주소 공간의 4가지 구성*

👊 스택 (Stack) 영역: 지역변수, 매개변수, 리턴값 등 잠시 사용되었다가 **사라지는 임시 데이터**를 저장하는 영역 함수 호출 시 생성되고, 함수가 끝나면 시스템에 반환된다.

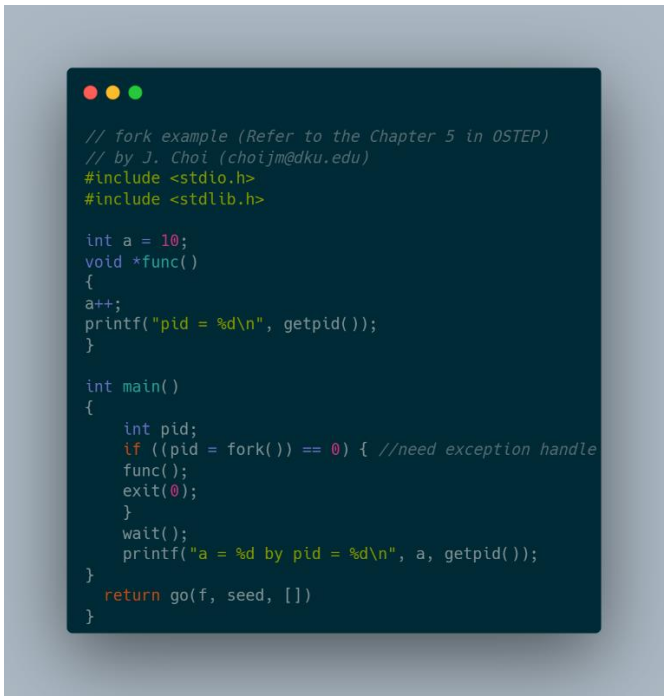
👊 힙 (Heap) 영역: malloc(), new() 같은 **동적인 메모리**를 할당할 때 위치하는 메모리 영역

👊 데이터 (Data) 영역: **전역변수, 정적변수, 배열, 구조체** 등이 저장되는 영역

👊 텍스트 (Text, Code) 영역: **프로그램 명령 코드 자체**를 저장하는 메모리 영역으로, Hex(16진수) 파일이나 BIN(2진수) 파일이 저장되는 영역

2.3 Concurrency (동시성)

- 프로세스 vs 스레드 차이점: data sharing



```c
// fork example (Refer to the Chapter 5 in OSTEP)
// by J. Choi (choijm@dku.edu)
#include <stdio.h>
#include <stdlib.h>

int a = 10;
void *func()
{
a++;
printf("pid = %d\n", getpid());
}

int main()
{
    int pid;
    if ((pid = fork()) == 0) { //need exception handle
    func();
    exit(0);
    }
    wait();
    printf("a = %d by pid = %d\n", a, getpid());
}
  return go(f, seed, [])
}
```

```
[Running] cd "c:\Users\user\Desktop\OSS_choi\" && gcc
tempCodeRunnerFile.c -o tempCodeRunnerFile &&
"c:\Users\user\Desktop\OSS_choi\"tempCodeRunnerFile
tempCodeRunnerFile.c: In function 'func':
tempCodeRunnerFile.c:10:22: warning: implicit declaration of
function 'getpid' [-Wimplicit-function-declaration]
 printf("pid = %d\n", getpid());
                      ^~~~~~
tempCodeRunnerFile.c: In function 'main':
tempCodeRunnerFile.c:16:16: warning: implicit declaration of
function 'fork' [-Wimplicit-function-declaration]
     if ((pid = fork()) == 0) { //need exception handle
              ^~~~
tempCodeRunnerFile.c:20:5: warning: implicit declaration of
function 'wait' [-Wimplicit-function-declaration]
     wait();
     ^~~~
C:\Users\user\AppData\Local\Temp\cchaZSfc.o:tempCodeRunnerFile.c:(.
text+0x3a): undefined reference to `fork'
C:\Users\user\AppData\Local\Temp\cchaZSfc.o:tempCodeRunnerFile.c:(.
text+0x5b): undefined reference to `wait'
```

[그림 2.3] process.c / thread.c 출력 코드 및 실행 결과

이 코드의 실행 결과를 예상하기 위해서는 '부모 프로세스는 자식 프로세스와 다르게return 값이 항상 0보다 크다'라는 배경지식이 필요하다. 부모 프로세스는 자식 프로세스의 pid 값을 리턴 받는데, 이 리턴 값이 항상 0보다 크기 때문이다.

두 번째로 이 코드에서 의미하는 *wait()* 는 부모 프로세스가 자식 프로세스의 수행이 끝날 때까지 강제로 기다리게 하는 트랜잭션을 의미한다. 그렇다면 이 코드의 출력 결과를 예측할 수 있을 것이다. 결과값은 11일까? 10일까? 출력 결과 예측에 가장 필요한 지식은 **'같은 값을 가진 전역변수라도 다른 프로세서의 변수에 접근할 수 없다'**라는 것이다. A 프로세스 내부의 변수를 B 프로세스에서 확인하거나 변경할 수 없다. 따라서 이 코드의 출력 결과는 11이 된다.

이러한 동시성 문제는 주로 'multi-thread' 환경의 프로그램 수행 시 자주 발생한다. 동시성 문제를 해결하는 가장 쉬운 방법은 코드마다 실행되는 순서를 다르게 하는 우선순위를 다르게 하는 것이다. 특정 줄의 코드 실행 앞에 lock을 걸고, 해당 코드 종료 후에 lock을 풀어 지연성을 주면 동시성 문제를 쉽게 해결할 수 있다. Ex) 은행 송금

2.4 Persistence (영속성)

영속성을 이해하기 위해서는 메모리와 디스크를 비교하면 도움이 된다. 아래 표를 기반으로 두 저장장치를 비교해보았다. 가장 큰 차이점은 데이터에 접근하는 방식이 다르다는 것이다. 디스크는 HDD 내부의 플래터가 물리적으로 회전하며 필요한 데이터를 찾는 방식이고, DRAM은 트랜지스터에 0과 1의 전기적 신호를 통해 데이터를 불러온다.

파일 시스템은 사용자가 생성한 모든 파일을 효율적이고 안정적으로 저장하는 역할을 한다. CPU, 메모리 가상화와 달리 **운영체제는 각 어플리케이션에 대해 가상화된 개인 디스크를 생성하지 않는다.**

▶ [1] 사용자는 데이터를 영구적으로 유지해 관리한다 → 내구성 (Durability)

▶ [2] DRAM은 전압을 필요로 하고, DISK, SSD는 명시적인 데이터 읽기/쓰기가 필요

메모리 (DRAM)	vs	디스크 (HDD)
4GB ~	용량	500GB ~
10 ~ 50ns (3,200~6,400MT/s)	속도	30~150MBps
휘발성	전원 방식	비휘발성
₩ 48,000 ~ (4GB 기준)	가격	₩ 109,000 ~ (1TB 기준)
Byte 단위	접근 방식 (단위)	Sector 단위
PC, 스마트폰, 디지털기기	사용 분야	하드디스크(HDD), SSD(Solid State Drive)

Table 1: 메모리와 디스크 간 비교

2.5 Design Goals (설계 목표)

- 1) 추상화(Abstraction): 추상화는 자세한 세부 사항은 숨기고 직접적인 문제에 집중하는 것이다. 우리가 **자동차 운전을 할 때, 엑셀과 브레이크의 동작 원리를 모르더라도 간단한 조작만으로 운전을 할 수 있는 것** 또한 추상화 기법의 일종이다.

구현 측면에서 설명하면, 추상화는 논리 게이트를 고려하지 않아도 어셈블리 코드를 작성할 수 있게 한다. 또한 트랜지스터에 대한 배경지식이 없어도 게이트를 이용해 우리가 프로세스를 만들 수 있도록 도와준다.

- 2) 성능 (Performance): 운영체제 설계에 두 번째로 중요한 목표는 성능이다. OS 성능에서 가장 중요한 지표는 '오버헤드 최소화'이다. 이는 운영체제의 수많은 기능을 '오버헤

드 없이 제공'해야 하는 것을 의미한다. 오버헤드의 시간은 '명령어'를 의미하고, 오버헤드의 공간은 '메모리나 디스크'를 의미한다. **시간과 공간을 가능한 최소한으로 사용하여 OS의 필수 기능을 제공하는 것이 OS의 설계 목표이다.**

- 3) 보호 (Protection): 보호는 운영체제의 원칙 중 '고립(isolation)' 원칙의 기본이다. 현재 **수행 중인 프로세스를 다른 프로세스로부터 방해를 받지 않도록 고립시키는 일**은 보호 원칙의 기본이며, OS의 역할의 뿌리가 될 수 있다.

- 4) 신뢰성 (Reliability): **운영체제는 운영체제가 포함된 장치가 24/7 언제든 프로세스나 유저의 요청에 올바른 응답을 해야 할 필요가 있다.** 그래서 늘 운영체제에는 높은 수준의 신뢰성이 요구된다. 우리가 빠른 은행 업무를 위해 언제든 ATM을 이용하는 것처럼 장치 내 응용 프로그램과 유저도 동작에 필요한 연산을 연산 순서를 OS에게 일부 요구한다.

  ▶ [1] 정책: '어떤' 작업을 수행할 것인가? (What) → 다음에 어떤 프로세스를 실행?

  ▶ [2] 매커니즘: '어떻게' 작업을 수행할 것인가? (How) → 다중 프로세스 관리 방법

2.6 Some history (역사 훑어보기)

- 1) 멀티프로그래밍 (Multiprogramming): 효율적으로 컴퓨터 자원을 활용하기 위해 멀티프로그래밍 기법이 사용되고 있다. 한 번에 여러 개의 프로그램을 수행하는 방식보단, **한 번에 하나의 프로그램만 실행시키면서 OS는 여러 작업을 메모리에 탑재해 여러 작업을 빠르게 '번갈아' 수행하는 방식이 CPU 사용률을 향상시켰다.** 한 번에 여러 개의 프로그램을 번갈아 실행하면서 프로세스를 전환하는 '문맥 교환' 능력이 OS에게 많이 필요해지기 시작했다. 추가적으로 메모리 보호, 병행성에 대한 처리능력 또한 OS에게 요구되고 있는 현실이다.

- 2) 유닉스 (UNIX): 우리가 현재 사용하는 모든 PC / 모바일기기의 시작은 유닉스 OS이다. 켄 톰슨과 데니스 리치에 의해 개발되었으며 크게는 윈도우/리눅스 등 포직스 문법을 사용하는 모든 유닉스 기반의 OS를 모아 '포직스 OS'라고 칭한다.

요즘은 세탁기부터 자동차까지 OS가 모두 포함된다. 이러한 기기에도 OS를 사용하는 가장 큰 이유는 '유지 보수 편이'가 크다. 차량에 새로운 기능이 추가될 때마다 카센터에 방문하는 것이 번거로우니, 차량 주인이 직접 차량의 신규 기능을 다운 받으면 편리할 것이다. **새 기능을 포함된 버전을 다운받기 위해서는 네트워크 통신이 필요하며 네트워크 통신 또한 OS에서 지원하는 기능이다.**

2.7 Summary (요약)

- 1) 운영체제란? 자원 관리자이면서, 자원을 효율적으로 관리하도록 돕는다.

- 2) 자원은 (물리 자원 vs 가상 자원)으로 나뉘며, 가상 자원에는 프로세스, 스레드, 가상 메모리가 있으며, 가상 자원을 구현한 물리 자원에는 CPU, DRAM, 디스크 등이 있다.

- 3) OSTEP 교재 범위: 가상화, 동시성, 영속성

3. References (참고 문헌)

- 1) Remzi H.Arpaci.,『 Operating Systems Three Easy Pieces』, 홍릉과학출판사, 2020, 23쪽

4. Goal of your OS study (OS 수업 목표)

- 1) 링크사업단 장기 현장실습으로 한국정보통신기술협회 (TTA)의 SW시험인증연구소 인턴 근무 당시, CentOS, Ubuntu, Rocky Linux 등 서버용 PC에 리눅스 OS를 설치하고 해당 CPU와 메모리 용량을 측정해야 했으나, 기본적인 리눅스 명령어 (free -h, kill pid, reboot, top etc.)에 익숙하지 않아 매번 리눅스 명령어를 찾아보기 힘들었고, **한 번 제대로 공부해야 하는 갈증을 느꼈습니다.**

- 2) 학점을 잘 받기보다는 나중에 실무에서 개발자로서 최소한의 OS 지식과 매일같이 PC의 전원을 키고 몇 시간씩 사용하는 **컴퓨터가 켜졌을 때부터 크롬 같은 어플리케이션 SW이 실행되기까지 과정을 사수에게 자신 있게 설명해보고 싶어** 큰 맘을 먹고 재수강을 신청했습니다.

- 3) 결과적으로 이번 OS 수업의 목표는 **1) 운영체제에서 스레드의 역할, 2) 로드와 패치 과정이 무엇인지 3) 문맥교환, 교착상태, 세마포어의 정의**, 다음 3가지 개념을 확실하게 이해하여 컴퓨터공학 전공자로서 C언어 기초 코드를 쓰더라도 시간과 공간 오버헤드를 최소한으로, CPU 성능은 최대 효율로 동작하는 코드를 작성할 수 있는 OS 기본 지식을 갖춘 소프트웨어 테스터로 성장해보고 싶은 원대한 OS 수업 목표가 있습니다.