REPORT

과 목 명 : 운영체제
담담교수 : 최종무 교수님
소　　속 : 소프트웨어학과
학　　번 : 32151671
이　　름 : 박민혁

단국대학교
Dankook University

1. 운영체제 기본 요약

운영체제는 가상화(virtualization)라고 불리는 기법을 사용한다. 운영체제는 프로세서, 메모리, 또는 디스크와 같은 물리적인 자원을 이용하여 일반적이고, 강력하고, 사용이 편리한 가상 형태의 자원을 생성한다. 때문에 운영체제를 때로는 가상 머신(virtual machine)이라고 부른다. 사용자의 프로그램의 프로그램 실행, 메모리 할당, 파일 접근과 같은 가상 머신과 관련된 기능들을 운영체제에게 요청할 수 있도록, 운영체제는 사용자에게 API를 제공한다. 보통 운영체제는 응용 프로그램이 사용 가능한 수백 개의 시스템 콜을 제공한다. 운영체제가 프로그램 실행, 메모리와 장치에 접근, 기타 이와 관련된 여러 작업을 진행하기 위해 이러한 시스템 콜을 제공하기 때문에, 우리는 운영체제가 표준 라이브러리(standard library)를 제공한다고 일컫기도 한다. 마지막으로, 가상화는 많은 프로그램들이 CPU를 공유하여, 동시에 실행될 수 있게 한다. 프로그램들이 각자 명령어와 데이터를 접근할 수 있게 한다. 프로그램들이 디스크등의 장치를 공유할 수 있게 한다. 이러한 이유로 운영체제는 자원 관리자(resource manager)라고도 불린다. CPU, 메모리 및 디스크는 시스템의 자원이다. 효율적으로, 공정하게, 이들 자원을 관리하는 것이 운영체제의 역할이다.

2. CPU 가상화

가상화 하는 핵심적인 이유는 시스템을 사용하기 편리하게 만들기 때문이다.

```
#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;
}
```

<그림 1> 간단한 예 : 반복해서 출력하는 코드 (cpu.c)

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

```
^C
prompt>
```

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

<그림 2> 동시에 많은 프로그램 실행시키기

프로세서가 하나밖에 없음에도 프로그램 4개 모두 동시에 실행되는 것처럼 보인다. 이것은 하드웨어의 도움을 받아 운영체제가 시스템에 매우 많은 수의 가상 CPU가 존재하는 듯한 환상을 만들어 낸 것이다. 하나의 CPU 또는 소규모 CPU 집합을 무한개의 CPU가 존재하는 것처럼 변환하여 동시에 많은 수의 프로그램을 실행시키는 것을 CPU 가상화(virtualizing the CPU)라 한다.

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include "common.h"
int main(int argc, char* argv[])
{
        int* p=malloc(sizeof(int)); // a1
        assert(p!=NULL);
        printf("(%d) memoryaddress of p: %08x₩n",
                getpid(), (unsigned)p); // a2
        *p=0; // a3
        while(1) {
                Spin(1);
                *p=*p+1;
                printf("(%d) p: %d₩n", getpid(), *p); // a4
        }
        return 0;
}
```

<그림 3> 메모리 접근 프로그램(mem.c)

3. 메모리 가상화

컴퓨터에서의 물리 메모리 모델은 바이트의 배열이다. 메모리를 읽기 위해서는 데이터 주

소를 명시해야 한다. 메모리는 프로그램이 실행되는 동안 항상 접근된다. 프로그램은 실행 중에 자신의 모든 자료 구조를 메모리에 유지하고 load와 store 또는 기타 메모리 접근을 위한 명령어를 통하여 자료 구조에 접근한다. 명령어 역시 메모리에 존재한다. malloc()을 호출하여 메모리를 할당하는 그림 3의 프로그램 출력은 다음과 같다.

```
prompt> ./mem
(2134) memory address of p: 00200000
(2134) p: 1
(2134) p: 2
...
^C
```

```
prompt> ./mem &; ./mem &
[1] 24113
[2] 24114
(24113) memory address of p:00200000
(24114) memory address of p:00200000
(24113) p:1
(24114) p:1
(24113) p:2
(24114) p:2
...
```

<그림 4> 메모리 프로그램 여러 번 실행하기

프로그램의 결과를 보면, 각 프로그램은 물리 메모리를 다른 프로그램과 공유하는 것이 아니라 각자 자신의 메모리를 가지고 있는 것처럼 보인다. 운영체제가 메모리 가상화를 하기 때문에 이런 현상이 생긴다. 각 프로세스는 자신만의 가상 주소 공간을 갖는다. 운영체제는 이 가상 주소 공간을 컴퓨터의 물리 메모리로 매핑(mapping)한다. 하나의 프로그램이 수행하는 각종 메모리 연산은 다른 프로그램의 주소 공간에 영향을 주지 않는다. 실행 중인 프로그램의 입장에서는, 자기 자신만의 물리 메모리를 갖는다. 실제로는 물리 메모리는 공유 자원이고, 운영체제에 의해 관리된다.

4. 병행성

병행성이란, 프로그램이 동시에 발생하는 그리고 반드시 해결해야 하는 문제들을 가리킬 때 이 용어를 사용한다. 병행성 문제는 우선 운영체제 자체에서 발생한다. 병행성 문제는 운영체제만의 문제는 아니다. 멀티 쓰레드 프로그램도 동일한 문제를 드러낸다.

```
#include<stdio.h>
#include<stdlib.h>
#include "common.h"
vclatile 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;
}
```

<그림 5> 멀티 쓰레드 프로그램 (threads.c)

```
prompt > ./thread 100000
Initial value : 0
Final value : 143012 // 어?
prompt > ./thread 100000
Initial value :0
Final value : 137298 // 뭐라고??
```

예상하지 못한 결과의 원인은 명령어가 한 번에 하나씩만 실행된다는 것과 관련 있다. 앞 프로그램의 핵심 부분인 counter를 증가시키는 부분은 세 개의 명령어로 이루어진다. counter 값을 메모리에서 레지스터로 탑재하는 명령어 하나, 레지스터를 1 증가시키는 명령어 하나, 레지스터의 값을 다시 메모리에 저장하는 명령어 하나 이렇게 3개의 명령어로 구성된다. 이세 개의 명령어가 원자적(atomically)으로 실행되지 않기 때문에 이상한 일이 발생할 수 있다.

5. 영속성

DRAM과 같은 장치는 데이터를 휘발성 방식으로 저장하기 때문에 메모리의 데이터는 쉽게 손실될 수 있다. 전원 공급이 끊어지거나 시스템이 갑자기 고장나면 메모리의 모든 데이터는 사라진다. 데이터를 영속적으로 저장할 수 있는 하드웨어와 소프트웨어가 필요하다. 저장 장치는 모든 시스템에 필수적이다.

```
#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);
close(fd);
return 0; }
```

<그림 6> 입출력을 수행하는 프로그램(io.c)

6. 설계 목표

가장 기본적인 목표는 시스템을 편리하고 사용하기 쉽게 만드는 데 필요한 개념들을 정의하는 것이다. 컴퓨터 과학에서 추상화는 모든 일에 근간이다.

운영체제의 설계와 구현에 중요한 목표는 성능이다. 다른 말로 표현하면 오버헤드를 최소화 하는 것이다. "가상화"와 "사용하기 쉬운 시스템을 만드는 것"은 의미가 있지만 반드시 해야 하는 것은 아니다. 가상화 및 다른 운영체제 기능을 과도한 오버헤드 없이 제공해야 한다. 또 다른 목표는 응용 프로그램 간의 보호, 그리고 운영체제와 응용 프로그램 간의 보호이다. 다수 프로그램들이 동시에 실행되기 때문에, 운영체제는 한 프로그램의 악의적인 또는 의도치 않은 행위가 다른 프로그램에게 피해를 주지 않는다는 것을 보장해야 한다. 그리고 운영체제는 계속 실행되어야 한다. 운영체제가 실패하면 그 위에서 실행되는 모든 응용 프로그램도 실패하게 된다. 이러한 종속성 때문에 운영체제는 높은 수준의 신뢰성을 제공해야 한다.

다른 중요한 목표들도 존재한다. 에너지-효율성(energy-efficiency)은 녹색 세상을 위해 중요하다. 악의적인 응용 프로그램에 대한 보안(security, 사실은 보호의 확장)은 현재와 같은 네트워크 환경에서 특히 중요하다. 이동성(mobility)은 운영체제가 작은 장치에서 사용될수록 중요해지고 있다. 이 책에서 다루는 많은 주제들은 다양한 기기에서 모두 유용하다.

7. 역사 약간

단순 라이브러리 -> 라이브러리를 넘어서 : 보호 -> 멀티프로그래밍 시대 -> 현대

운영체제 전성기에 개발된 좋은 아이디어들이 현재 시스템에서도 여전히 적용된다. 더욱이 이 아이디어들은 계속 발전하고 있다. 이러한 발전으로 운영체제에 많은 기능들이 추가되고 사용하기 좋은 시스템이 되고 있다.

8. 요약

오늘날 운영체제는 시스템을 사용하기 쉽게 만들었다. 거의 모든 운영체제가 이 책에서 논의하게 될 기술을 사용한다.

1. cpu.c

```
os-lecture@os-lecture:~/ostep-code/intro$ ./cpu "A"
A
A
A
A
^C
os-lecture@os-lecture:~/ostep-code/intro$ ./cpu A & ./cpu B & ./cpu C & ./cpu D
&
[1] 3281
[2] 3282
[3] 3283
[4] 3284
C
A
B
D
```

2. mem.c

```
os-lecture@os-lecture:~/ostep-code/intro$ vi mem.c
os-lecture@os-lecture:~/ostep-code/intro$ gcc -o mem mem.c -Wall
os-lecture@os-lecture:~/ostep-code/intro$ ./mem
(3432) address pointed to by p: 0x16a7010
(3432) p: 1
(3432) p: 2
(3432) p: 3
(3432) p: 4
(3432) p: 5
^C
os-lecture@os-lecture:~/ostep-code/intro$ ./mem & ./mem &
[1] 3433
[2] 3434
(3434) address pointed to by p: 0x1903010
(3433) address pointed to by p: 0x20ef010
os-lecture@os-lecture:~/ostep-code/intro$ (3434) p: 1
(3433) p: 1
(3434) p: 2
(3433) p: 2
```

3. threads.c

```
os-lecture@os-lecture:~/ostep-code/intro$ gcc -o threads threads.c -Wall -pthre
ad
os-lecture@os-lecture:~/ostep-code/intro$ ./threads 1000
Initial value : 0
Final value   : 2000
os-lecture@os-lecture:~/ostep-code/intro$ ./threads 100000
Initial value : 0
Final value   : 150380
os-lecture@os-lecture:~/ostep-code/intro$ ./threads 100000
Initial value : 0
Final value   : 148318
os-lecture@os-lecture:~/ostep-code/intro$
```

4. io.c

```
os-lecture@os-lecture:~/ostep-code/intro$ vi io.c
os-lecture@os-lecture:~/ostep-code/intro$ ls
Makefile   common.h          cpu    io    mem    threads    tmp.txt
README.md  common_threads.h  cpu.c  io.c  mem.c  threads.c
os-lecture@os-lecture:~/ostep-code/intro$ cat tmp.txt

os-lecture@os-lecture:~/ostep-code/intro$ ./io
os-lecture@os-lecture:~/ostep-code/intro$ cat tmp.txt
hello world
os-lecture@os-lecture:~/ostep-code/intro$
```

목표와 다짐

우선 운영체제 과목 강의를 듣기 전, 최종무 교수님께 시스템프로그래밍 강의를 들은 적이 있다. 시스템프로그래밍 과목에서, 나는 B+라는 점수를 받았다. 하지만 만족을 하지 못했다. 중간고사 시험에서 A를 맞을 수 있는 성적을 받았다. 그렇지만 기말고사를 잘 보지 못하였고, 또한 과제를 소홀히 하여 프로젝트 점수는 1등을 했지만, 개인과제에서 점수가 낮아 B+라는 점수를 맞았다. 사실 너무 아쉬웠다. 충분히 A를 맞을 수 있었기 때문이다. 그래서 이번 운영체제 과목에서 시스템프로그래밍 과목에서 공부한 것을 기반으로 수업을 듣기 때문에, 걱정이 많다. 시스템프로그래밍 과목 마지막 부분을 잘 모르고 넘어왔기 때문이다. 그래서 나는 시스템프로그래밍을 복습하면서, 운영체제를 공부할 예정이다. 물론 어렵고 바쁠 것이다. 공부 외에도, 학회장이란 직을 맡고 있기 때문에 현재 수업도 많이 빠진 상태이며, 일이 너무 바빠서, 수업도 제대로 듣지 못하고 있다. 그렇지만 그럼에도 불구하고 좋은 성적과 내가 하고 싶은 일을 하기 위해 한 걸음 한 걸음 다가 갈 수 있도록, 남들보다 더한 노력을 할 것이다. 사실 운영체제에서 뭐를 터득하고 갈 지는, 아직 잘 모른다. 그렇지만, 시스템프로그래밍 과목을 배우면서 조금 더 컴퓨터에 관한 것들을 알게 되었고, 운영체제 또한 수업이 끝나 갈 때쯤이면, 그때와 비슷한 느낌이 들지 않을까? 물론, 이번엔 더 열심히 하여 A라는 점수를 맡고 싶다. 또한, 이 어려운 과목을 학회장 일을 병행하면서도, 좋은 성적을 맞아서, 후배들에게 학회장이 학업에 많은 영향을 끼치지 않는 구나. 라는 것을 알려주고 싶다. 그러기 위해선, 첫 번째로 과제에 조금 더 노력을 할 것이다. 항상 대학 과제를 대충대충 하는 그런 습관이 있었다. 하지만 2018년도 2학기에 시스템프로그래밍뿐 아니라, 자료구조에서도 과제의 중요성을 깨닫게 되었다. 아무리 시험을 잘 봐도, 과제점수가 낮으면 성적이 낮아질 수밖에 없다. 그래서 나는 과제에 조금 더 노력을 쏟을 예정이다. 두 번째로, 친구들의 도움을 많이 받을 예정이다. 나는 항상 혼자 공부하려는 습관이 있었다. 최종무 교수님뿐 아니라 황두성 교수님도 이런 이야기를 한 적이 있으시다. '컴퓨터로 무언가를 하는 것은 혼자 할 수 있는 것이 많이 없어요.' 그것을 저번 학기에 많이 깨달았다. 그래서 이번엔 혼자 공부 하는 것이 아닌, 친구들과 같이 공부를 할 예정이다. 나보다 훨씬 잘하는 친구들이 주변에 많기 때문에 도움을 많이 요청할 예정이다. 마지막으로는, 예습은 어렵되, 복습 위주로 하여서 시험기간에 최대한 투자하는 시간이 적게 하도록 할 것이다. 이번엔 7전공을 듣는다. 그렇지만, 과목 하나하나에 복습을 하지 않고, 시험기간에 몰아서 하게 된다면, 이번 학기는 배워가는 것도 많이 없을 것이며, 성적도 잘 안 나올 것 같기 때문이다. 그래서 복습을 많이 할 예정이다. 비록 지금 많이 아는 것이 남들보다는 부족 하지만, 종강 할 때는 다음 학기에 나를 필요로 하는 사람이 꼭 나올 수 있는 그런 사람이 될 것이다. 운영체제뿐 아니라 모든 과목을 이런 자세로 임 할 것이다.