운영체제(Operation System) (=가상머신, 자원관리자)

- 프로그램 수행을 위해선 프로세서가 메모리로부터 명령어를 차례로 패치하고 디코딩, 수행. (한 번에 여러 개의 프로그램을 수행하기도 함)
- 이런 프로그램 수행을 위해 사용되는 메모리를 공유, 디바이스 장치와 상호작용 등의 작업들을 정확하고 효율적으로 수행할 수 있도록 관장하는 역할
- '가상화' 로 물리적인 자원(메모리, 프로세서, 디스크) 할당을 가상의 형태로 진행
- 사용자가 프로그램을 실행하고 메모리나 디바이스, 파일 등을 활용할 수 있도록 시스템콜이라는 API 라이브러리 제공

CPU 가상화

- 특정 문자열을 1초에 한 번씩 출력하는 프로그램을 수행했을 때, & 를 사용해 같은 프로그램 여러 개 동시에 수행했을 때, 여러 문자열이 출력되는 것을 볼 수 있음
- 한 프로세스에 한 CPU가 할당되면 한 프로그램이 ^C 를 통해 종료가 되어야 다른 프로그램이 실행될 수 있을 것 같지만 그렇지 않음.
- 마치 CPU가 여러 개 있어서 여러 프로세스에 CPU가 각각 할당되고 그래서 여러 프로그램을 동시에 수행할 수 있는 것처럼 보이는 환상을 제공 => CPU Virtualization
- 여러 프로그램이 동시에 수행될 때 그 중 어떤 프로그램이 먼저 수행될지는 OS 정책에 달림

메모리 가상화

- 메모리(물리 자원으로서)는 byte들의 배열로, 데이터나 명령어를 읽고 쓰기 위해서 주소를 사용
- 프로그램에 사용되는 모든 데이터와 명령어도 메모리에 저장되어 있고, 이 메모리에 접근하여 명령어/데이터를 가져오거나 저장하는 등의 작업 진행
- 정수형 포인터 변수를 생성하고 1초에 1씩 증가하여 출력하도록 하는 프로그램을 &를 사용해 2개를 동시에 수행할 때, 각 프로그램에 대한 다른 프로세스가 생성되지만 포인터 변수의 주소는 같음.
- 주소가 같기 때문에 한 프로그램에서 1 증가해 1이 출력됐다면 그 값이 다른 프로그램에도 반영되어 2가 출력될 것 같지만 그렇지 않고 1이 출력 됨
- 마치 메모리가 각 프로세스마다 존재하여 주소는 같아도 서로 다른 메모리를 사용하기

때문에 이렇게 독립적으로 값이 변하는 것 같은 환상을 제공 => Memory Virtualization

(실제로는 같은 물리 메모리의 다른 주소 공간에 맵핑)

병행성(동시성)

- 동일한 프로그램에서 같은 데이터를 사용하여 여러 작업을 수행할 때 병행성이 지켜지지 않으면 문제가 발생할 수 있음
- 하나의 프로그램에서 스레드를 여러 개 만들면 프로세스에서 fork()를 통해 부모, 자식 프로세스와 만드는 것처럼 원래 수행 중이던 프로그램이 복제됨. 하지만 프로세스 모델과는 조금 다르게 다중 스레드는 같은 메모리를 공유하게 되고 하나의 스레드의 수행 결과가 다른 스레드에 영향을 미침
- 총 N번의 loop를 돌며 정수형 변수를 1씩 증가시키는 프로그램이 2개의 스레드로 수행되면 2번째로 수행되는 스레드의 변수는 2N이 될 것 같지만 N이 커지면 제대로 된 결과가 출력되지 않음
- 2개의 스레드가 같은 정수형 변수를 공유하면서 하나의 스레드가 수행되면 다른 스레드로부터 독립적으로 수행을 완료할 수 있어야 하는 '원자성'을 만족하지 못해서 병행성 문제 발생
- 실제로는 여러 프로세스가 자원을 놓고 경쟁하고 있지만 마치 협력적으로 자원에 접근하려는 것 같아 보이는 환상을 제공 => Concurrency

영속성

- DRAM 메모리는 전원이 꺼지면 데이터를 잃어버리기(휘발성) 때문에 데이터를 영구적으로 저장할 수 있는 비휘발성 저장소 필요 ex) 하드드라이버, SSD
- 운영체제에서 소프트웨어는 데이터를 영속적으로 저장하기 위해 디스크를 파일 시스템으로 추상화해 다루고 시스템은 사용자가 생성한 모든 파일들을 안정적이고 효율적인 방식으로 저장
- 파일에 있는 데이터들은 서로 공유될 수 있기 때문에 CPU, 메모리처럼 각 프로그램/프로세스에게 개별적인 가상 디스크를 제공하는 것은 아님
  - ⇨ 프로그램을 작성하는 과정을 생각해보면 편집기로 코드 작성하고 컴파일 하면 소스파일, 목적 파일, 수행가능 파일 등 여러 파일이 생김
  - ⇨ 이 과정에서 디바이스는 복잡한 과정을 거쳐 데이터를 쓰고 저장하는 작업을 수행하겠지만 사용자는 open(), write(), close() 같은 시스템콜을 사용하여 마치 함수를 이용하듯이 쉽게 파일 시스템을 다뤄 데이터를 읽고 쓰고 저장할 수 있음

운영체제의 설계 및 구현 목표

운영체제(Operation System)

: 병행성 및 영속성을 고려하며 CPU, Memory, Disk, 그리고 이것들을 가상화 한 자원을 효율적으로 관리하고 안전하게 저장

1. 추상화
   - ⇨ 시스템을 사용하기 편리하고 이해하기 쉽도록 만드는 추상을 제공
   - ⇨ 하드웨어적인 요소들에 대해서 많이 알지 못하더라도 프로그램 작성, 어셈블리 코드 이해, 프로세서 생성 등의 일들을 할 수 있어야 함
2. 높은 성능을 제공
   - ⇨ 처리시간이나 메모리 사용의 최소화를 위한 끊임없는 전략 탐색 필요
   - ⇨ 가상화처럼 사용이 편리하면서도 비용이 들지 않는 high-performance 전략 중요
3. 응용프로그램 간 혹은 운영체제와 응용 프로그램 간의 보호
   - ⇨ 동시에 많은 프로그램을 수행하기 위해서는 한 프로그램이 다른 프로그램에 안 좋은 영향을 끼치지 못하도록 **독립성**을 보장받아야 함
   - ⇨ 운영체제가 수행을 멈추면 운영체제가 관리하던 모든 응용 프로그램이 멈추기 때문에 운영체제의 **안정성**이 매우 중요
4. 에너지 효율성
5. 보안
6. 유동성/이동성
   - ⇨ 운영체제가 스마트폰처럼 더 작고 휴대할 수 있는 장치에서 사용됨에 따라 이동성이 중요해지고 시스템이 어떻게 사용되는지에 따라 운영체제도 다른 목표를 가지고 동작하도록 설계하는 것이 중요

역사

1세대 OS: 라이브러리들의 집합

- 낮은 수준의 입출력 함수
- Batch Processing -> 수행할 작업들이 일정 이상 모이면 한 번에 수행

2세대 OS : 라이브러리를 넘어선 보호의 개념

- **시스템콜을 사용하여 운영체제를 컨트롤**
- User 모드와 kernel 모드로 구분하여 운영체제를 보호
- Trap 시스템콜을 사용하여 user 모드에서 kernel 모드로 전환할 수 있고, 모든 시스템 하드웨어에 접근할 수 있는 권한이 주어지며 입출력 요청을 초기화 하거나 프로그램에 더 많은 메모리를 사용 가능하게 하는 등의 작업을 수행할 수 있음. 작업이 끝난 후에는 다시 user모드로 돌아옴

3세대 OS : 멀티프로그래밍(멀티 프로세싱)

- CPU 활용도가 높아지면서 여러 프로그램을 마치 동시에 수행하는 효과를 얻을 수 있음
- 하나의 메모리에 여러 개의 프로세스가 올라갈 수 있음(메모리의 보호와 병행성이 중요)
-

4세대 OS : PC 뿐만 아니라 스마트폰, IOT 기기등 다양한 시스템에서 사용되는 OS

이번 학기 배우고 싶은 목표

1. 운영체제 수업에서 배우는 내용과 더불어 저번 학기 시스템 프로그래밍 수업에서 확실히 이해하지 못했던 내용까지도 이번엔 확실히 이해하고 넘어가기
2. 스스로 더 많이 실습해보고 싶고, 배운 것을 복습하여 추가 질문도 많이 해보기
3. 스레드나 동기화, 백그라운드 실행에 대해 확실히 공부하고 개인적으로 하고 있는 안드로이드 앱 개발에서 활용하여 성능 향상과 관련된 이슈를 연구해보기

BONUS

cpu.c

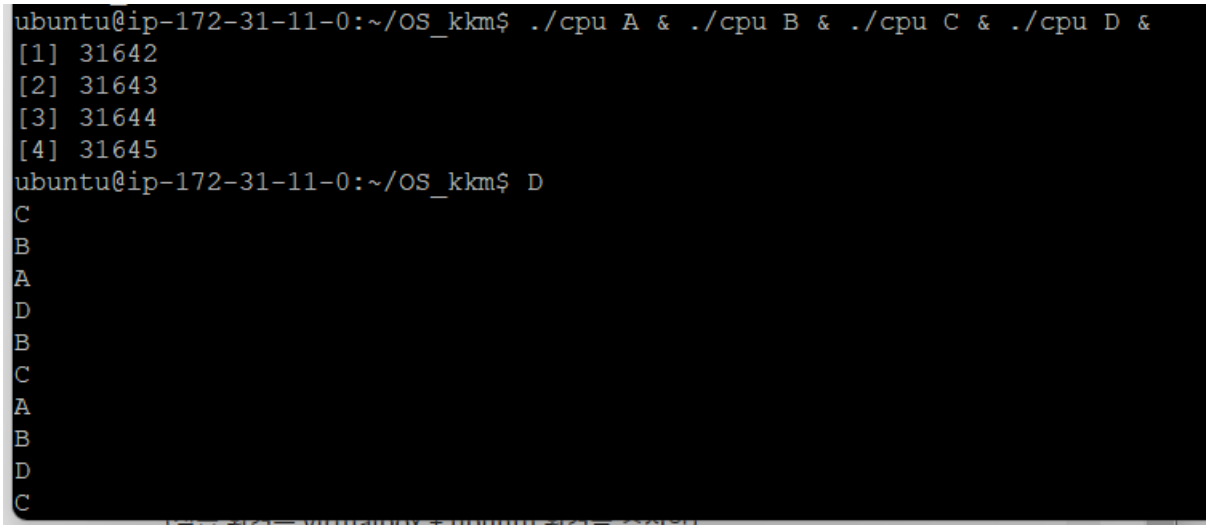
1)

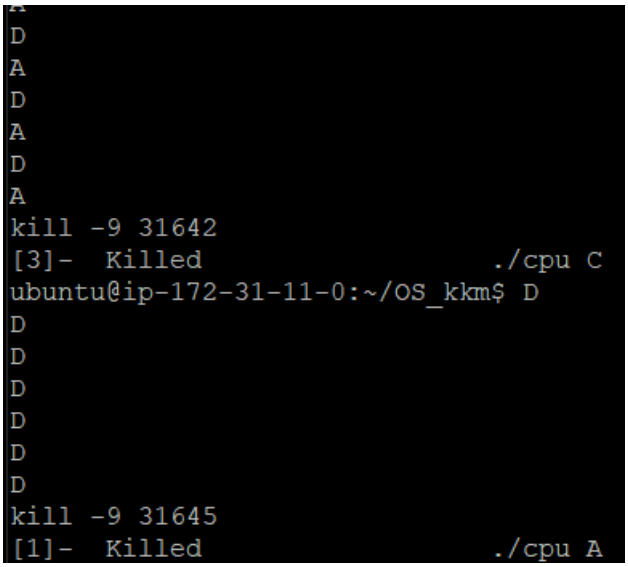
```
ubuntu@ip-172-31-11-0:~/OS_kkm$  ./cpu A
A
A
A
A
A
^C
```

2) 백그라운드로 여러 개 수행

스케줄링 정책에 따라 순서는 달라진다.

Kill -9 PID 로 프로세스를 종료해야 하는데 계속 수행 결과가 출력되어 명령어 입력이 힘들었다.





mem.c

```
ubuntu@ip-172-31-11-0:~/OS_kkm$ ./mem 1
(30479) addr pointed to by p: 0x558595435260
(30479) value of p: 2
(30479) value of p: 3
(30479) value of p: 4
(30479) value of p: 5
(30479) value of p: 6
(30479) value of p: 7
(30479) value of p: 8
^C
```

thread.c

1)

```
ubuntu@ip-172-31-11-0:~/OS_kkm$ ./threads 10000
Initial value : 0
Final value   : 20000
```

2) loop를 너무 많이 돌면서 병행성이 지켜지지 않은 사례

```
ubuntu@ip-172-31-11-0:~/OS_kkm$ ./threads 10000000
Initial value : 0
Final value   : 14350451
```

io.c

```
ubuntu@ip-172-31-11-0:~/OS_kkm$ ./io
ubuntu@ip-172-31-11-0:~/OS_kkm$ ls
common.h          cpu    file.txt  io.c      mem    threads
common_threads.h  cpu.c  io        makefile  mem.c  threads.c
ubuntu@ip-172-31-11-0:~/OS_kkm$ vi file.txt
hello world
~
```