OS

OS 란?

컴퓨터의 사용자와 하드웨어 사이에서 가교 역할을 하는 프로그램

사용자 인터페이스와 자원관리를 위한 프로그램의 집합

컴퓨터는 운영체제가 업어도 작동하는가?

=〉 **컴퓨터는 운영체제가 없어도 작동하지만 기능에 제약이 따른다.**

운영체제가 있는 기계와 없는 기계는 어떤 차이인가?

=〉 **운영체제가 있는 기계: 다양한 응용 프로그램을 설치해 사용 가능, 성능 향상을 위한 새로운 기능 쉽게 추가 가능**

운영체제는 성능을 향상하는 데만 필요한가?

=〉 **운영체제는 컴퓨터의 성능을 향상할 뿐 아니라 자원 관리, 사용자에게 편리한 인터페이스 환경 제공**

운영체제는 자원을 어떻게 관리하는가?

=〉 **운영체제는 사용자가 직접 자원에 접근하는 것을 막음으로써 컴퓨터 자원을 보호**

사용자는 숨어있는 자원을 어떻게 이용할 수 있는가?

=〉 **운영체제가 제공하는 사용자 인터페이스와 하드웨어 인터페이스를 이용하여 자원에 접근**

운영체제

- 사용자에게 편리한 인터페이스 환경을 제공하고 컴퓨터 시스템의 자원을 효율적으로 관리하는 소프트웨어
- 사용자로부터 자원 보호와 동시에 자원 관리, 인터페이스를 제공한다.

운영체제의 목표

운영체제의 역할	운영체제의 목표
자원 관리	효율성
자원 보호	안정성
하드웨어 인터페이스 제공	확장성
사용자 인터페이스 제공	편리성

OS 역사

1 세대 - 진공관

- IBM 701 부터 운영체제 1 세대로 본다 (일괄처리 시스템의 등장)

2 세대 - 트랜지스터

- CPU 를 할당 받을 다수개의 작업(Multi program)이 같이 주기억장치에 있다 (다중 프로그래밍 시스템)
- 여러 개의 처리장치(Multi processor)를 장착하여 동시에 여러 작업을 병렬로 처리 (다중 처리 시스템)

3 세대 - 직접회로

- 일괄처리, 시분할, 실시간작업 모두 지원 (Multi-mode)
- TCP/IP 표준 발표, 근거리 통신망(LAN)이 이 시기 즈음 탄생
- 1969 년 UNIX 출현

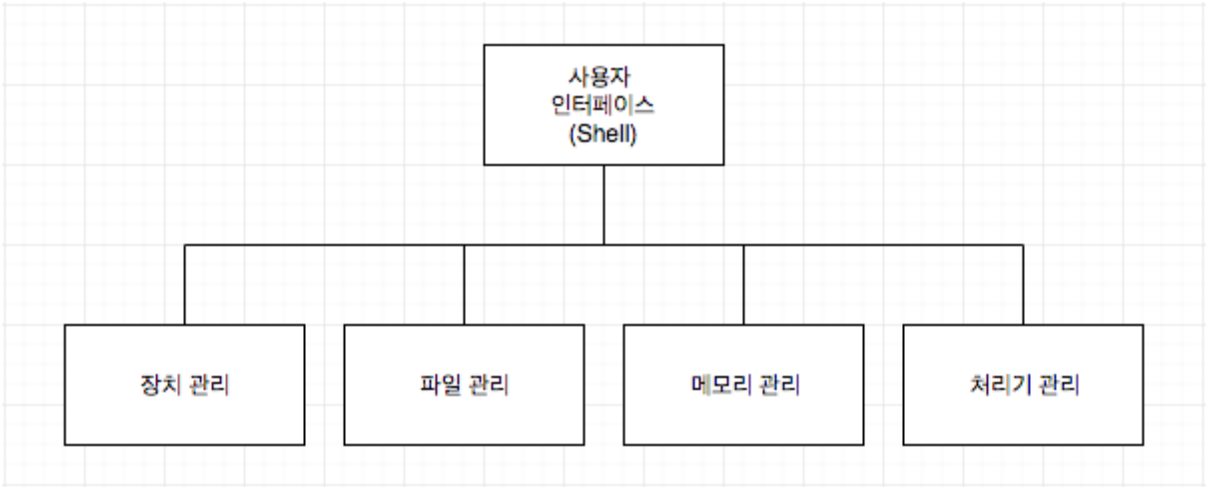
4 세대 - 고밀도 직접회로

- 컴퓨터 사용의 범용화
- 입출력 장치의 다양화
- 저장장치의 대용량화
- 데이터 통신의 발전
- 정보산업 출현
- 마이크로 프로세서의 등장 -〉 가격의 범용화
- 마이크로 프로세서: 하나의 칩에 연산, 제어, 레지스터들을 넣은 것

구분	시기	주요 기술	특징
0 기	1940's	-	진공관(0, 1) 에니악
1 기	1950's	천공 카드 리더 라인 프린터	일괄 작업 시스템 운영체제의 등장
2 기	Early 1960's	키보드 모니터	대화형 시스템
3 기	Late 1960's	C 언어	다중 프로그래밍 기술 개발 시분할 시스템
4 기	1970's	PC(Apple 2) 인터넷	개인용 컴퓨터의 등장 분산 시스템
5 기	1990's	웹 리눅스	클라이언트/서버 시스템
6 기	2000's	스마트폰	P2P 시스템(메신저, 파일 공유) 그리드 컴퓨팅 클라우드 컴퓨팅 사물 인터넷(IoT)

운영체제의 5 가지 구성요소

크게 Kernel 과 Utility-Program 으로 나눈다.



Kernel

운영체제의 핵심

컴퓨터가 처음 부팅될 때 주 기억 장치에 적재되어 시스템 종료(Shutdown) 되기 전까지 계속 주 기억장치에 적재되어 있는 프로그램

빈번하게 실행되는 프로그램을 디스크에 둘 경우 주 기억장치와 디스크 간의 입출력이 너무 빈번하게 일어나기 때문에 성능 저하가 있다. 즉 필수적으로 자주 쓰이는 프로그램을 메모리에 상주시키는데 이 프로그램을 Kernel 이라고 한다.

Kernel 보다 더 빠른 실행이 필요하고 높은 수준의 보호가 필요한 건 마이크로 프로그래밍하여 ROM 이나 PLA 같은 칩으로 만들기도 하며 이를 펌웨어(생긴 건 하드웨어 내부적으론 프로그램)라고 한다.

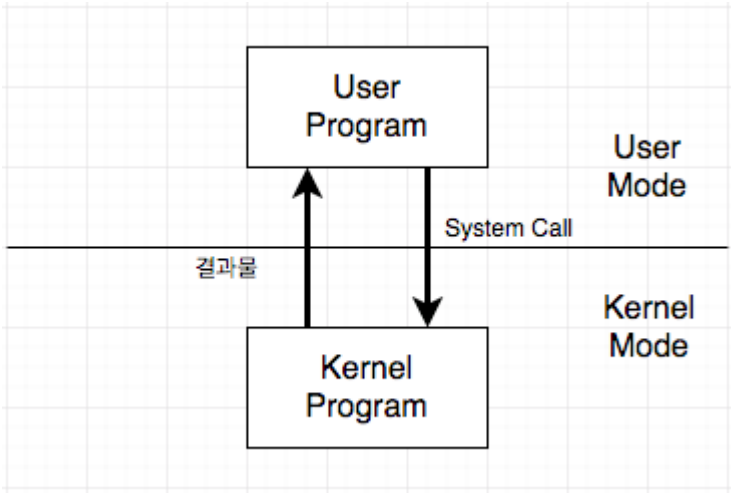
커널은 프로세스 관리, 메모리 관리, 저장장치 관리와 같은 운영체제의 핵심적인 기능을 모아 놓은 것. 운영체제의 성능은 커널이 좌우한다.

단일형 구조 커널: 커널의 핵심 기능을 구현하는 모듈들이 구분 없이 하나로 구성되어 있다.

계층형 구조 커널: 비슷한 기능을 가진 모듈을 묶어서 하나의 계층으로 만들고 계층 간의 통신을 통해 운영체제를 구현하는 방식.

마이크로 구조 커널: 운영체제가 프로세스 관리, 메모리 관리, 프로세스 간 통신 관리 등 가장 기본적인 기능만 제공

System call 이란?



Kernel 영역에 있는 프로그램만 할 수 있는 것들을(Disk I/O, Memory access) 유저 수준에서 사용하길 원할 때 System call 을 하여 원하는 결과물을 얻을 수 있다.

시스템 호출은 커널이 자신을 보호하기 위해 만든 인터페이스. 커널은 사용자나 응용 프로그램으로부터 컴퓨터 자원을 보호하기 위해 자원에 직접 접근하는 것을 차단. 자원을 이용하려면 시스템 호출이라는 인터페이스를 이용해야 한다.

- 시스템 호출은 커널이 제공하는 시스템 차원의 사용과 관련된 함수
- 응용 프로그램이 하드웨어 자원에 접근하거나 운영체제가 제공하는 서비스를 이용하려 할 때 사용
- 운영체제는 커널이 제공하는 서비스를 시스템 호출로 제한하고 다른 방법으로 커널에 들어오지 못하게 막음으로써 컴퓨터 자원 보호
- 시스템 호출은 커널이 제공하는 서비스를 이용하기 위한 인터페이스, 사용자가 자발적으로 커널 영역에 진입할 수 있는 유일한 수단

OS 의 목적

사용자의 편리성과 자원의 효율적 사용

부팅

전원 ON -〉 커널이 메모리에 올라옴(Bootstrap Loader, 부트 프로그램 등) -〉 장치준비, 레지스터 초기화 -〉 사용자 입력 대기

컴퓨터를 실행할 때 운영체제를 메모리에 올리는 과정

사용자가 컴퓨터의 전원을 켜면 롬에 저장된 바이오스 실행되어 하드웨어 점검, 이상 없을 시 메모리에 부트스트랩 코드 올려 실행. 부트스트랩 코드는 하드디스크에 저장된 운영체제를 메모리로 가져와 실행함으로써 부팅을 마무리.

레지스터

CPU 는 여러 개의 레지스터를 가지고 메모리보다 빠르지만 용량이 작다.

시스템과 사용 목적에 따라 8 비트, 16 비트, 32 비트등의 크기를 가진다.

CPU 는 PSW(Program Status Word)라는 현재상태를 저장하는 레지스터가 있다.

레지스터: CPU 내에 데이터를 임시로 보관하는 곳

사용자 가시 레지스터
- 데이터 레지스터(DR, Data register): 메모리에서 가져온 데이터를 임시로 보관할 때 사용. CPU 에 있는 대부분의 레지스터가 데이터 레지스터 =〉 일반 혹은 범용 레지스터라고

부른다.
- **주소 레지스터(AR, Address register)**: 데이터 또는 명령어가 저장된 메모리의 주소를 저장

사용자 불가시 레지스터(특수 레지스터)
- **프로그램 카운터(명령어 포인터)**: 다음에 실행할 명령어의 위치 정보(코드의 행 번호, 메모리 주소)를 저장
- **명령어 레지스터**: 현재 실행 중인 명령어 저장
- **메모리 주소 레지스터**: 메모리 관리자가 접근해야 할 메모리의 주소 저장
- **메모리 버퍼 레지스터**: 메모리 관리자가 메모리에서 가져온 데이터 임시 저장
- **프로그램 상태 레지스터**: 연산 결과(양수, 음수 등)을 저장

명령어 처리

명령어를 읽어 처리기에 있는 레지스터로 가져오는 것을 말함 (Fetch)

인터럽트(Interrupt)

운영체제가 자원을 효율적으로 관리하기위해 각 자원의 상황을 알아야 하는데 이를 매번 조사할 수 없으니 인터럽트라는 것을 이용한다.

각 자원들은 인터럽트를 통해 자신의 상태변화를 CPU 에 알려줘 CPU 는 폴링 방식처럼 주기적으로 시간을 들이지 않아도 각 자원의 상황을 알 수 있다.

즉 CPU 가 처리해야 될 일들이 있을 때 하드웨어 및 소프트웨어는 인터럽트를 통해 CPU 에게 알릴 수 있고 처리된다.인터럽트는 크게 2 가지로 하드웨어 인터럽트와 소프트웨어 인터럽트(트랩)으로 나눈다.

- 하드웨어 인터럽트 -〉 하드웨어 수준에서 일어나는 인터럽트로 CPU 외부의 디스크나 마우스, 키보드의 입출력 등 CPU 의 처리가 필요한 상황일 때 발생한다.
- 트랩 -〉 소프트웨어 수준에서 일어나는 인터럽트로 프로그램의 예기치 못한 종료나, CPU 의 처리가 필요한 상황일 때 발생한다. (e.g. system call)

인터럽트의 처리

CPU 는 인터럽트가 들어오면 실행 중이던 프로그램을 잠시 메모리에 돌려놓고 인터럽트 처리 루틴을 실행한다.

인터럽트를 처리하는 과정에서 실행 중이던 프로그램의 값을 잃어버릴 수 있으므로 인터럽트 처리 전에 PSW, PC 레지스터의 값 등을 시스템 스택에 저장한다.

인터럽트의 처리가 끝나면 PC 나 PSW 정보를 CPU 에 되돌리고 작업중이던 프로그램을 이어서 처리할 수 있다.

이러한 일련의 과정을 Context Switching 이라고 한다.

중첩된 인터럽트의 처리

하나의 인터럽트가 끝난 뒤 다음 인터럽트를 처리하는 순차적 처리 혹은 Context Switching 을 중첩하여 처리할 수 있다.

우선순위가 더 높은 인터럽트가 들어왔을 때 중첩하여 처리하기도 한다.

CPU 가 입출력 관리자에게 입출력 명령 보냄
-〉 입출력 관리자는 명령 받은 데이터를 메모리에 가져다 놓거나 메모리에 있는 데이터를 저장장치로 옮김
-〉 데이터 전송이 완료되면 입출력 관리자는 완료 신호를 CPU 에 보냄

인터럽트 방식은 CPU 의 작업과 저장장치의 데이터 이동을 독립적으로 운영함으로써 시스템의 효율을 높임

기억 장치의 계층적 구조

Access Time, 용량, 가격(bit 당 단가)의 차이로 분류 가능

속도가 높으면서 가격이 비싼 게 있고, 속도가 느리고 가격이 싸며 용량이 높은 것들이 있으므로 용도에 맞게 저장장치를 계층적으로 잘 구성해야 한다.

저장장치의 계층 구조: 속도가 빠르고 값이 비싼 저장장치를 CPU 가까운 쪽에 두고, 값이 싸고 용량이 큰 장치를 반대쪽에 배치해 적당한 가격으로 빠른 속도와 큰 용량을 동시에 얻는 방법

I/O 방식

Programmed I/O

CPU 가 입력을 지시 후 컨트롤러 버퍼를 계속 확인한다.

인터럽트가 필요 없는 대신 다른 CPU 작업을 못한다.

- Interrupt-driven I/O
    - Programmed I/O 와 다르게 CPU 를 다른 작업에 활용 가능
    - 잦은 Interrupt 가 단점이다 (잦은 Context Switching 이 발생)
- Direct Memory Access(DMA)
    - CPU 대신 입출력 작업을 해줄 Channel 이라는 Satellite processor 를 이용하는 방식
    - CPU 는 입출력 할 데이터의 시작 주소와 크기 등을 Channel 에 알려준다.
    - 한번의 입출력(Block)단위로 CPU 에게 인터럽트 하므로 Interrupt-driven I/O 보다 인터럽트가 적다.

Hardware 구성에 따른 입출력

Isolated I/O(독립적 입출력)

- 입출력 장치들이 입출력 버스(I/O Bus)를 통해 CPU 와 연결되어 있는 경우
- I/O Bus 를 통해 해당장치의 지정, 데이터, 입출력을 구분해주는 제어 값이 전달됨
- 입출력 명령어가 Instruction set 에 추가되어 제어로직이 복잡해지고, I/O Bus 를 장착하는데 추가 비용이 있다.

Memory-mapped I/O(메모리 주소지정 입출력)

- 입출력 장치들이 메모리와 함께 Memory Bus 에 연결됨
- 입출력 명령어가 따로 없고 메모리 명령어를 사용한다.
- 메모리에 추가적인 공간을 차지하는 게 단점이다.

https://velog.io/@sangmin7648/%EC%98%A4%EB%8A%98%EC%9D%98-%EB%B0%B0%EC%9B%80-019-%EC%BB%B4%ED%93%A8%ED%84%B0%EA%B5%AC%EC%A1%B0-IO%EC%99%80-Bus