REPORT

운영체제



**과목명 :운영체제**

**교수명 :최종무**

**학 과 : 소프트웨어학과**

**학 번 : 32162566**

**이 름 : 안찬웅**

**2022년 3월 25일**

EMB0000356c3453

<목 차>

I. 클라우드 컴퓨팅 기술 3

2.0 introduction to Operating Systems 3

2.1 Virtualizing CPU 3

2.2 Virtualizing Memory 3

2.3. Concurrency 3

2.4 Persistence 4

2.5 Design Goals 4

2.6 Some History 5

4. 데이터 암호와 알고리즘 소스 코드 6

6. 클라우드의 응용 사례 8

III. 제안사항 및 후기 9

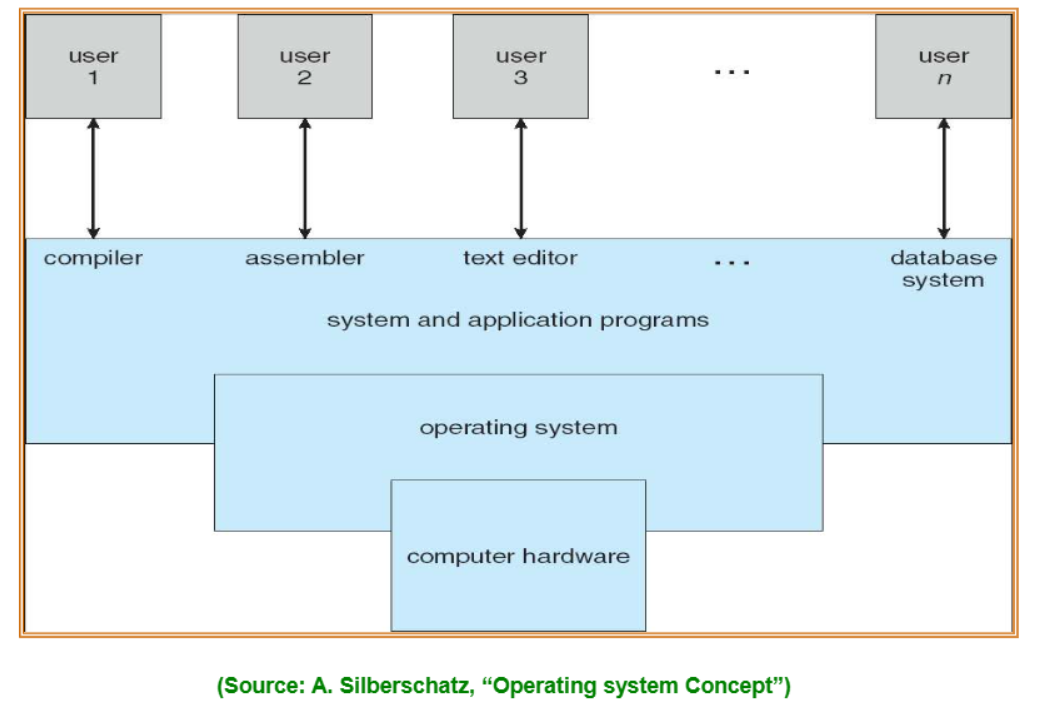
1. 클라우드 컴퓨팅에 대한 독창적인 생각과 활용 방안 9

2. 멀티미디어 교과목 수강 후기 10

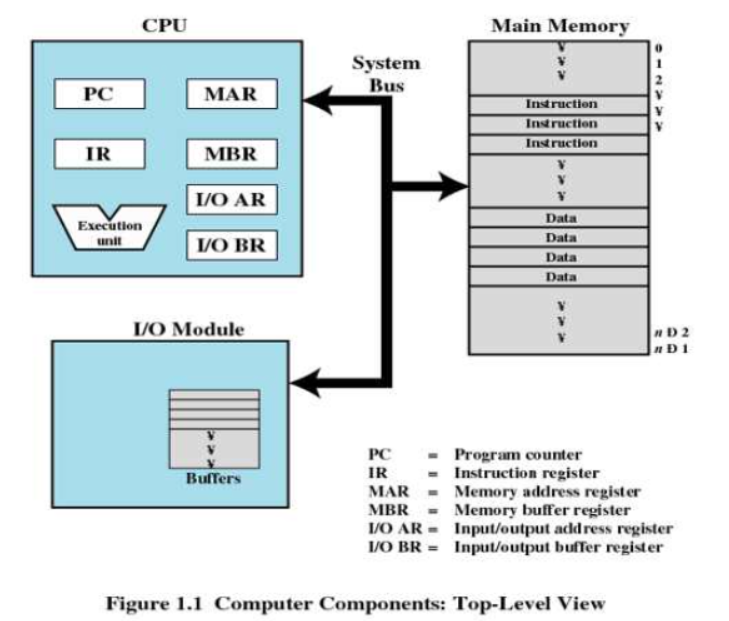
IV. 참고자료 11

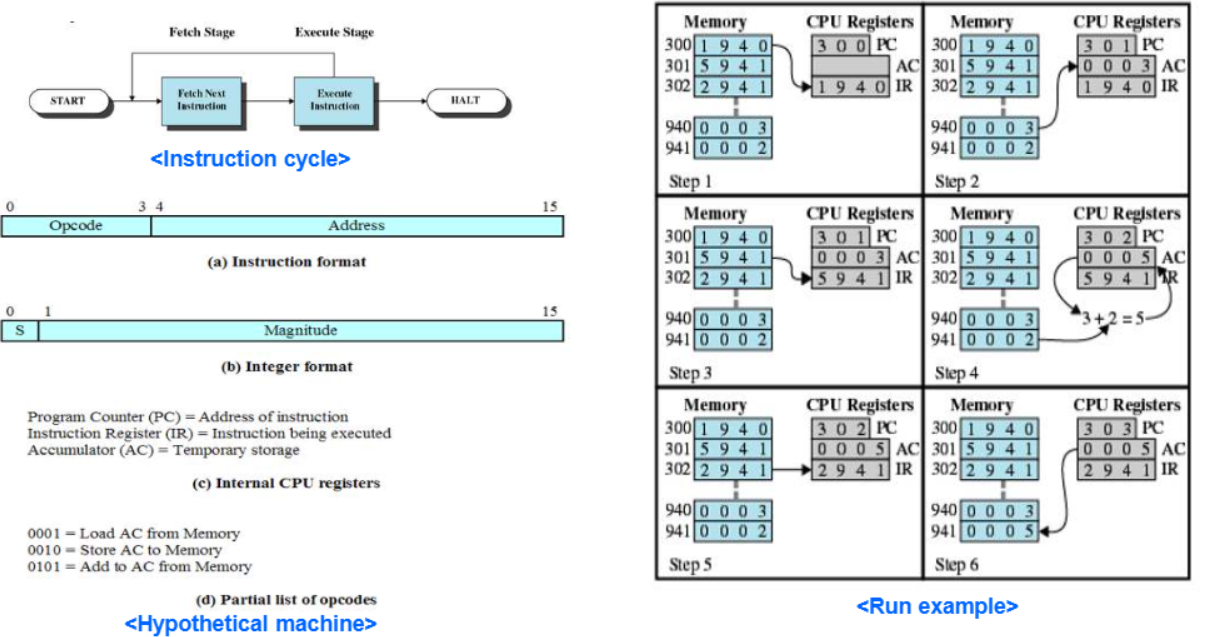
**2.0 컴퓨터를 바라보는 여러가지 관점**

**2.0 - 1 컴퓨터시스템의 층 구조**



1. **– 2 프로그램이 실행될 때 무엇이 발생하는가?**
2. simple view about running a program



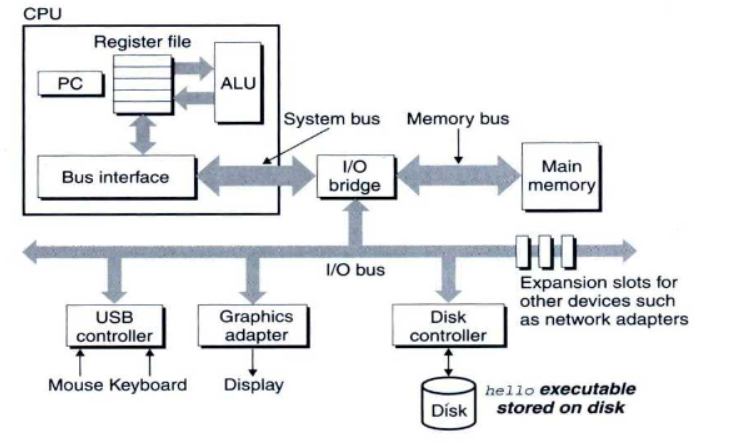


프로그램이 수행되는 과정은 fetch 와 Execute의 반복이다.

1. A lot of stuff for running a program

* Loading memory management, scheduling, context switching, I/O prcessing, file management, IPC, …
* Operating System: 1) 쉽게 프로그램 작동 2) 정확하고 효과적으로 작동

요약) 어떤 프로그램이 수행될 때 이 프로그램이 쉬운 환경을 제공해주고, 자원을 관리하는 역할을 한다.



**OS정의**

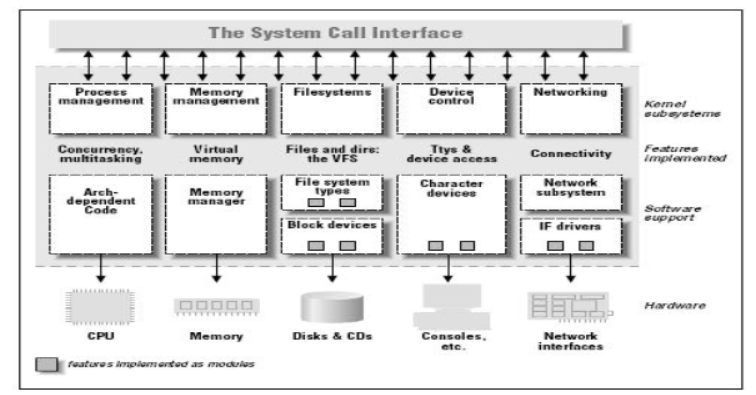
자원관리자

* Physical resources: CPU(core), DRAM, Disk, Flash, KBD, Network, …
* Virtual resources: process, Thread, Virtual memory, Page, File, Directory, Driver, Protocol, Access control, Security, …

**가상화**

* 물리적인 자원들을 논리적인 자원으로 변환

<OS가 관리하는 자원>



OS는 자원을 관리하고 System call을 제공하는 시스템 소프트웨어이다.

**System call**

* OS가 제공하는 인터페이스
* 운영체제가 사용자 프로그램에게 제공하는 잘 정의된 인터페이스

텍스트이(가) 표시된 사진

자동 생성된 설명

**System call**

* Standard (e.g.. POSIX, Win32, …)
* Mode switch (user mode, kernel mode)
* 일반 프로그램들은 유저모드에서 동작하고, OS는 커널모드에서 동작한다.
* User mode와 kernel mode로 나누는 이유는 소프트웨어를 보호하기 위해서이다.
  1. **CPU 가상화**
* 더 많은 CPU를 가지고 있는 것처럼 보이게 하는 것.

**lssues for Virtualizing CPU**

* 새로운 프로그램 실행방법 -> process
* 새로운 프로세스 만드는법 -> fork()
* 프로세스 종료 -> exit()
* 새로운 프로그램 수행 -> exec()
* 프로세스 멈추기 -> sleep(), pause(), lock()
* 다음 수행할 프로세스 선택 -> scheduling
* 다중프로세스 실행방법 -> context switch
* 다중코어(CPU) 실행방법 -> multi-processor scheduling, cashe affinity, load balancing
* 프로세스간 커뮤니케이션 방법 -> IPC, Socket
* 프로세스에 이벤트를 알리는 방법 -> signal (e.g ^C)

각각 프로세스들이 자신마다 cpu를 가지고 있는 것 처럼 환상을 제공하는 것 => cpu가상화

**2.2 Virtualizing Memory**

**Memory**

* Can be considered as an array of bytes

**Another program example**

* Allocate a portion of memory and access it

**lssues for Virtualizing Memory**

* How to manage the address space of a process? -> text, data, stack, heap
* How to allocate memory to a process? -> malloc(), calloc(), brk()
* How to deallocate memory form a process? -> free()
* How to manage free space? -> buddy, slab
* How to protect memory among processes? -> virtual memory
* How to implement virtual memory? -> page, segment
* How to reduce the overhead of virtual memory? -> TLB
* How to share memory among processes -> shared memory
* How to exploit memory to hide the storage latency? -> page cache, buffer cache
* How to manage NUMA? -> local/remote memory

메모리 가상화 - 제한되어 있는 메모리를 여러 프로세스들이 공유하고 있는데, 프로세스가 독립적으로 사용하고 무한적으로 큰 메모리를 가지고 있는 것처럼 환상을 제공하는 것

**2.3 Concurrency**

Background: how to create a new scheduling entity?

* Two programming model: process (task) and thread
* Key difference : data sharing

fork() -> process model

create-> thread model

**정의**

* 같은 데이터를 동시에 접근하려고 할 때 발생하는 문제들

**Issues for Concurrency**

* How to support concurrency correctly? -> lock(), semaphore()
* How to implement atomicity in hardware? -> test\_and\_set(), swap()
* What is the semaphore?
* What is the monitor?
* How to solve the traditional concurrent problems such as producer-consumer, readers- writers and dining philosohpers?
* What is a deadlock?
* How to deal with the deadlock?
* How to handle the timing bug?
* What tis the asynchronous I/Os?

자원을 놓고 경쟁하는 상황에 공유자원이 협력적으로 접근 할 수 있는 것 같은 환상을 제공 하는것 -> 병행성

**2.4 Persistence (영속성)**

**Background: DRAM VS Disk**

* Capacity, Speed
* Access granularity: Byte vs Sector
* Durability: Volatile vs Non-volatile

**정의**

* Users want to maintain data permanently
* Requiring write data into storage (disk, SSD)

**Lssues for Persistence**

* How to access a file? -> open(), read(), write()
* How to manage a file? -> inode, FAT
* How to manipulate a directory?
* How to design a file system? -> UFS, LFS Ext2/3/4, FAT, F2F3, NFS, AFS
* How to find a data in a disk?
* How to improve performance in a file system? -> journaling, copy-on-write
* What is a role of a disk device driver?
* What are the internals of a disk and SSD?
* What is the RAID?

**2.5 Design Goals**

1. Abstraction

* Focusing on relevant issues only while hiding details

e.g.) Car, File System, Make a program without thinking of logic gates

* “Abstraction is fundamental to everything we do in computer science” by Remzi

2. Performance

* Minimize the overhead of the OS (both time and space)

3. Protection

* lsolate processes from one another
* Access control, security

4. Reliability

* Fault-tolerant

5. Others

* Depend on the area where OS is employed
* Real time, Energy-efficiency, Mobility, Load balancing, Autonomous

Separation of Policy and Mechanism

* Policy: which (or What) to do?

e.g. ) which process should run next?

* Mechansim: How to do?

e.g. ) Multiple processes are managed by a scheduling queue or RB-tree

**2.6 some History**

Early Operating Systems: Just libraries

* Commonly-used functions such as love-level l/Os (e.g. MS-DOS)
* Batch processing

A number of jobs were set up and the then run all together (Not interactive)

* Beyond Libraries: Protection

Require OS to be treated differently than user applications

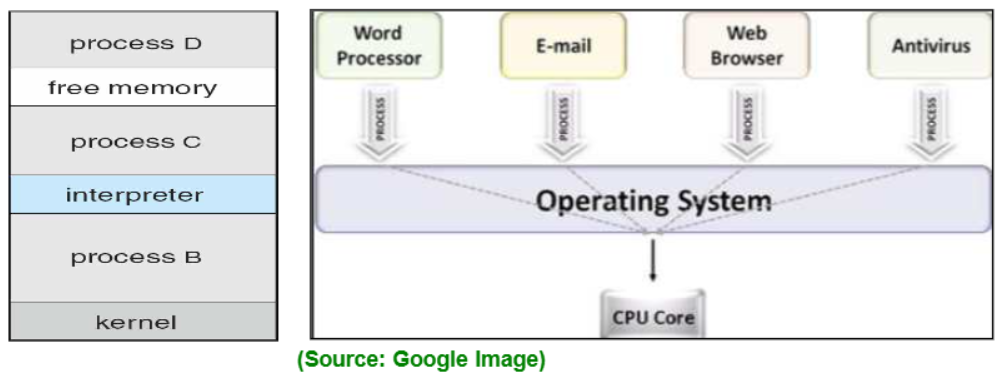
Separation user/kernel mode, system call

Use trap (special instruction, SW interrupt) to go into the kernel mode

Transfer control to a pre-specific trap handler (system \_call handler)

The Era of multiprogramming

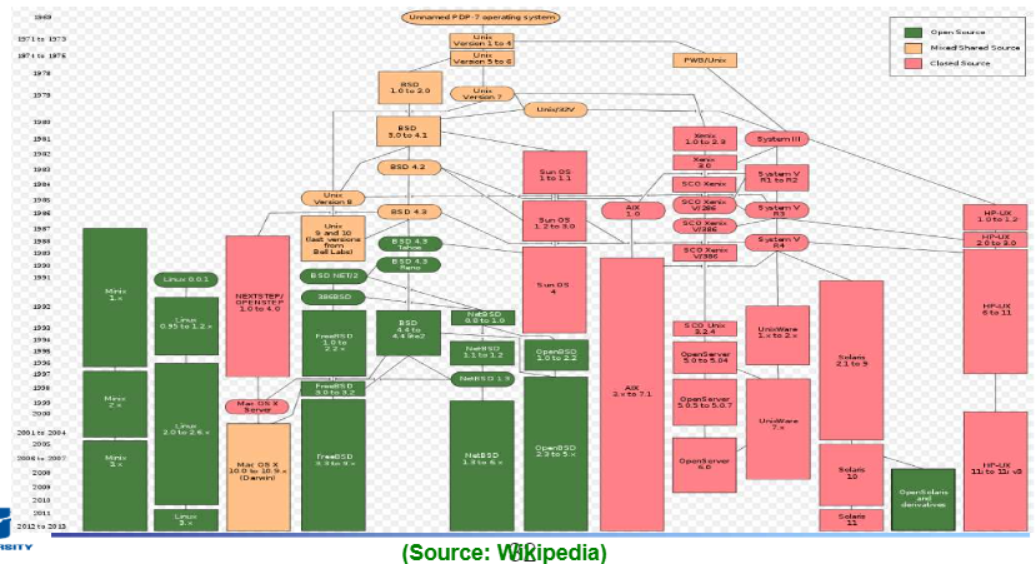
* Definition: OS load a number of applications into memory and switch them rapidly
* Reason: Advanced hardware -> Want to utilize machine resources better -> multiple users share a system (workstation, minicomputer) -> multiprogramming (and multitasking)
* Especially important due to the slow I/O devise -> while doing l/O, switch CPU to another process -> enhancing CPU utilization
* Memory protection and concurrency become quite important -> UNIX



The Era of multiprogramming

UNIX

* By ken Thompson and Dennis Ritche (Bell Labs), Influenced by Multics
* C language based, excellent features such as shell, pipe, inode, small, everything is a file
* Influence OSes such as BSD, SUNOS, AIX, HPUX, Nextstep and Linux



**The Modern Era**

PC

* MS Windows, Mac OS X, Linux

Smartphone

* Android, iOS, Windows Mobile

IoT

* What is the next?

**2.7 Summary**

OS

* Resource manager (Efficiency)
* Make Systems easy to use (convenience)

Cover in this book

* Virtualization, Concurrency, Persistence

Not being covered

* Network, Security, Graphics
* There are several excellent courses for them

운영체제 수업을 수강하면서 배우고 싶은 목표

운영체제 수업을 하면서 배우고 싶은건 크게 두가지로 나눌 수 있습니다. 몇 년후 개발자가 되기 위해 운영체제라는 과목을 통해 얻고 싶은 것 과 , 운영체제 수업안에서 얻고 싶은 것 인데 첫번째로 운영체제라는 수업을 통해서 안찬웅이라는 개발자가 얻고 싶은 것은 전반적인 소프트웨어적인 개념입니다. 처음 공부를 시작했을 때는 그냥 코딩만 잘하면 되는 것 아닌가? 라는 생각을 했습니다. 그러나 소프트웨어학과에서 공부를 하면서 그것은 좋은 개발자가 되기 위한 방법은 아니라는 것을 알았습니다. 전반적인 개념들을 알고 쓰는 것이 정말 중요하다는 것을 느꼈습니다. 그래서 좋은 개발자가 되기 위해서 배우고 싶습니다.

두번째, 운영체제 강의를 통해 하드웨어와 응용프로그램이 어떻게 연결되어 어떤 성능과 기능을 제공할 수 있는지 알고 싶고, 스케줄러, 커널, 교착상태 등과 같은 개념에 대해서 학습하고 싶습니다. 그렇게 공부를 한다면 좋은 개발자에 한걸음 나아갈 것 입니다.