

Introduction

System Programming - USG

Jaeho Kim

jaeho.kim@gnu.ac.kr

목차

- 시스템 프로그래밍이란
- 유닉스/리눅스 시스템의 역사
- 시스템 호출과 라이브러리 함수의 비교

컴퓨터 프로그래밍?

컴퓨터 프로그래밍을 하는 이유?

- 내가 원하는 일을 컴퓨터에게 시키기 위함



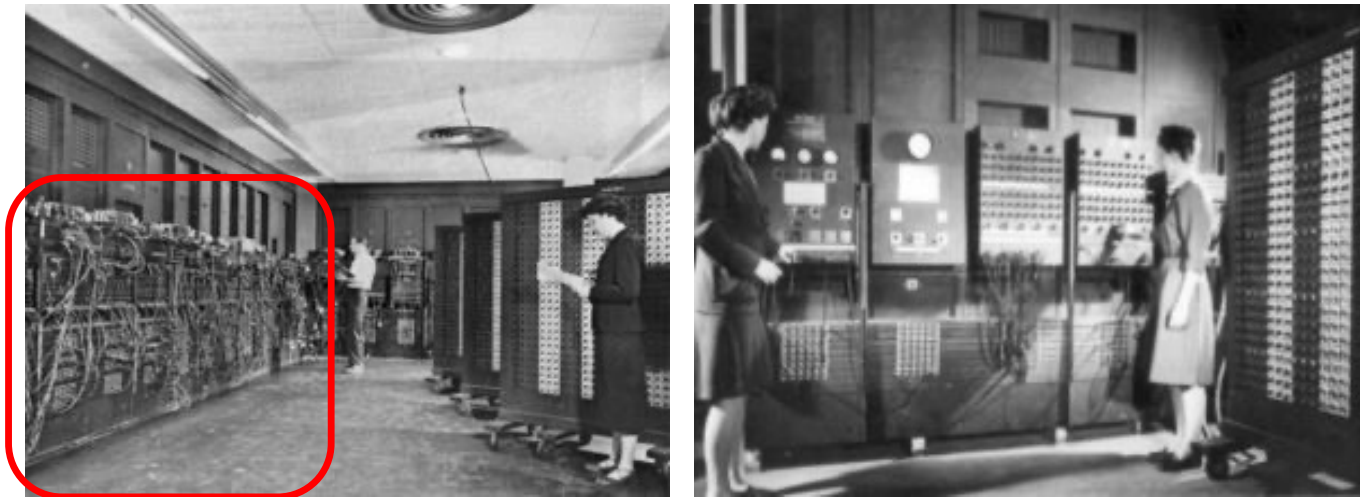
자연 언어
(natural language)



프로그래밍 언어
(programming language)

초기 컴퓨터의 프로그래밍

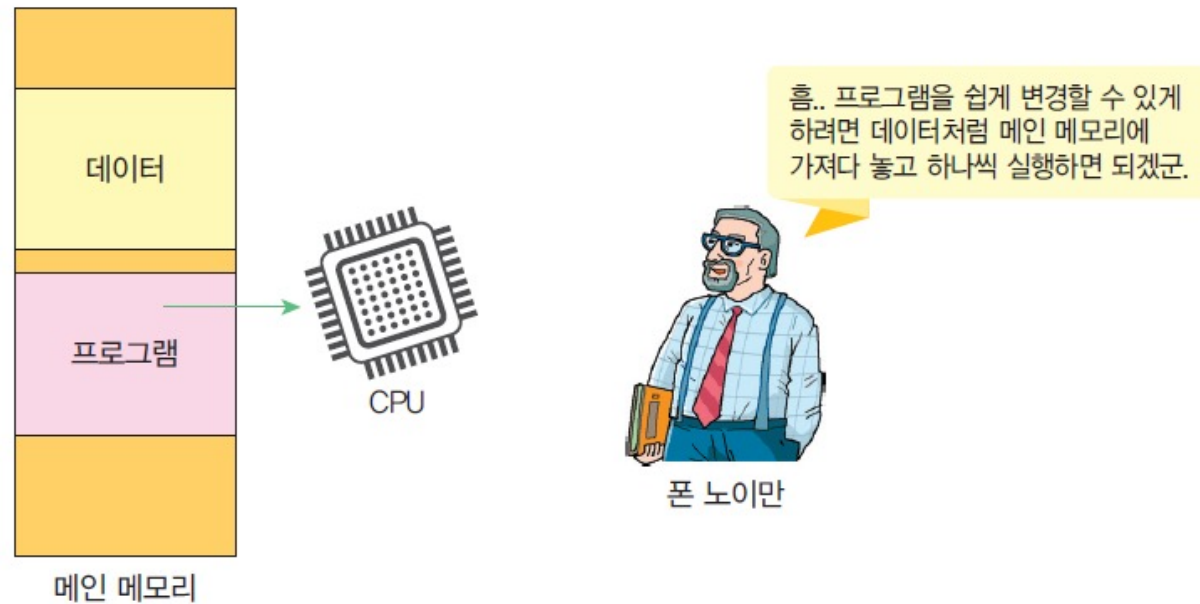
- 초기 컴퓨터인 ENIAC의 프로그램은 스위치에 의하여 기억되었고 프로그램을 변경할 때마다 그 많은 스위치들을 처음부터 다시 연결하여야 했다.



* 1943년 탄도 궤적을 계산할 목적으로 ENIAC이 개발이 시작됨

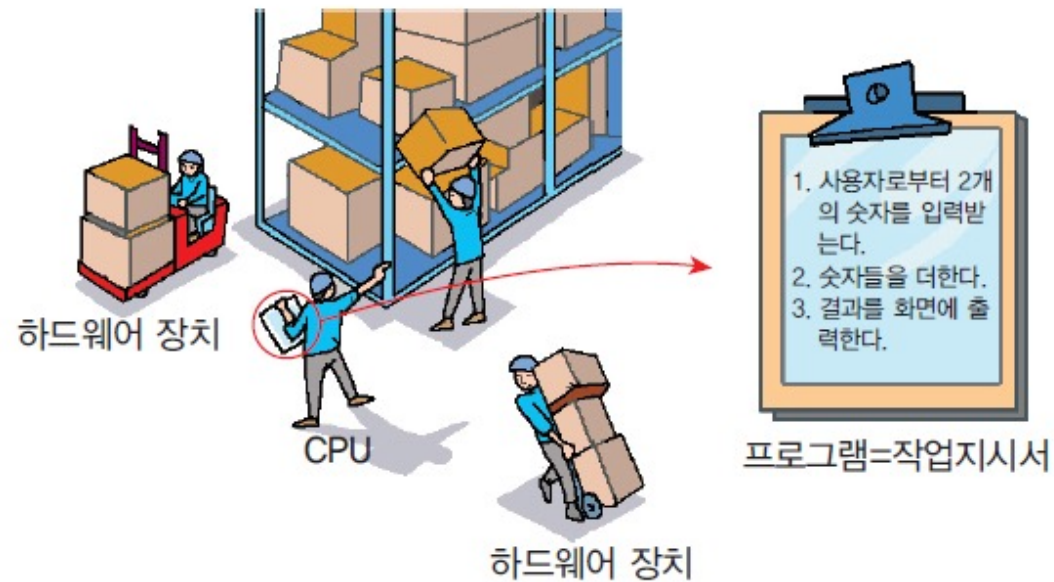
폰노이만 구조

- 프로그램은 **메인 메모리에 저장**된다.
- 메인 메모리에 저장된 프로그램에서 명령어들을 순차적으로 가져와서 실행한다.

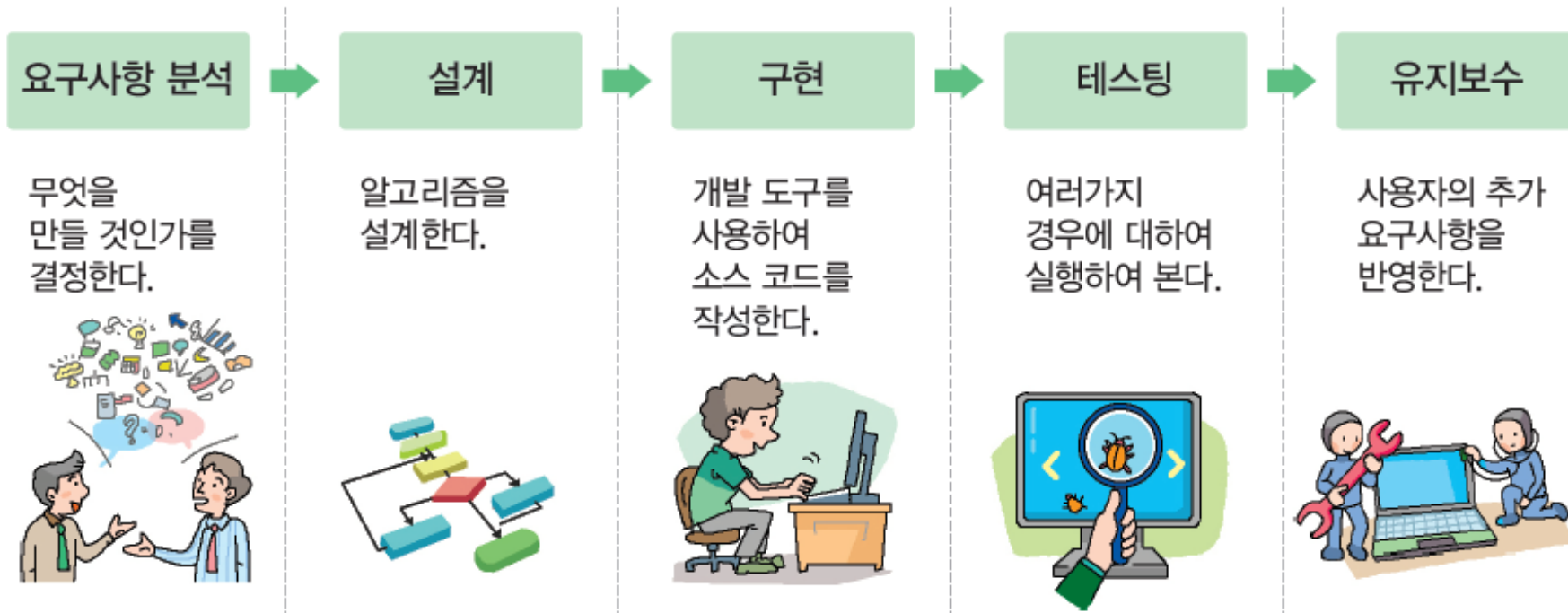


프로그램==작업지시서

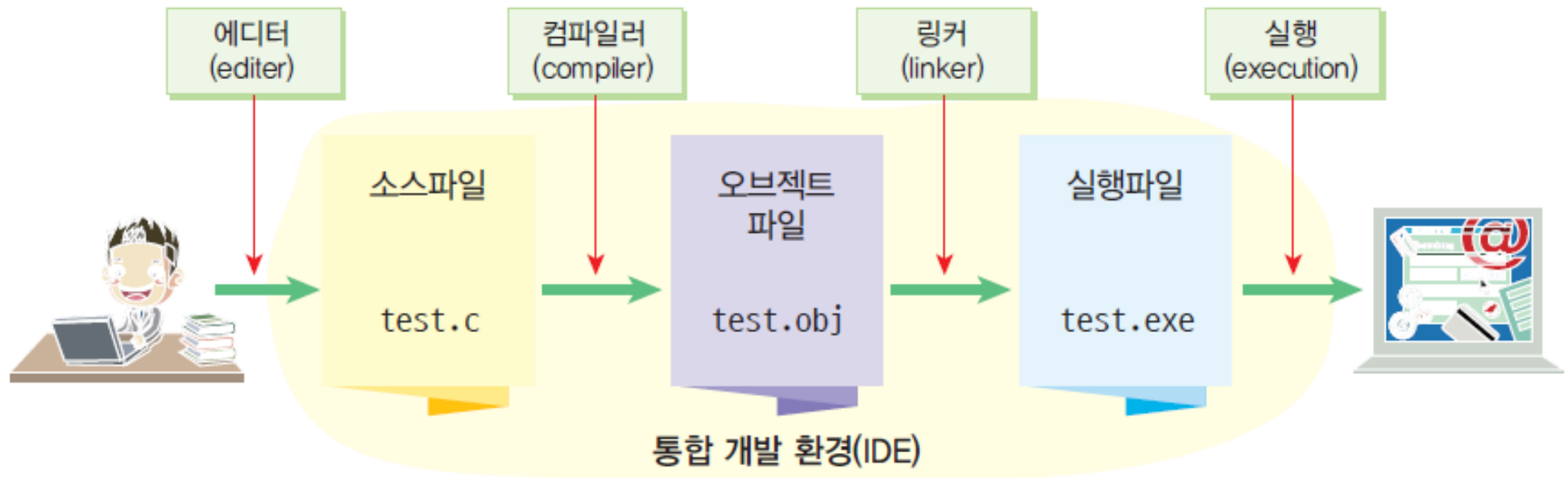
- 프로그램: 컴퓨터에게 해야 할 작업의 내용을 알려주는 문서



프로그램 개발 과정

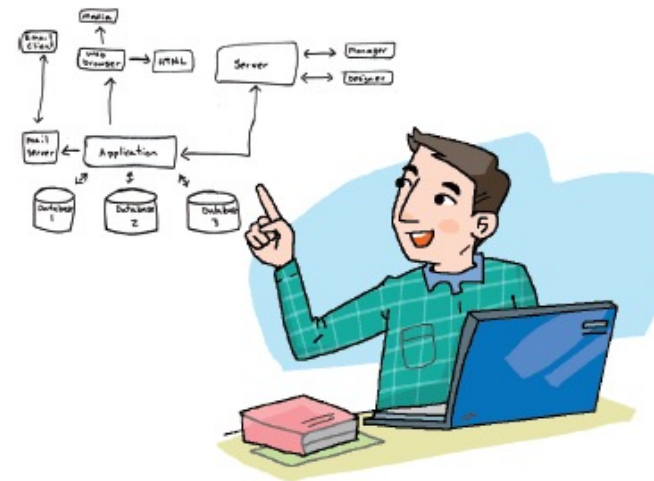


프로그램 구현 과정



설계

- 문제를 해결하는 알고리즘을 개발하는 단계
- 자연어, 순서도 또는 의사 코드를 도구로 사용
- 알고리즘은 프로그래밍 언어와는 무관
- 알고리즘은 원하는 결과를 얻기 위하여 밟아야 하는 단계에 집중적으로 초점을 맞추는 것



소스 작성

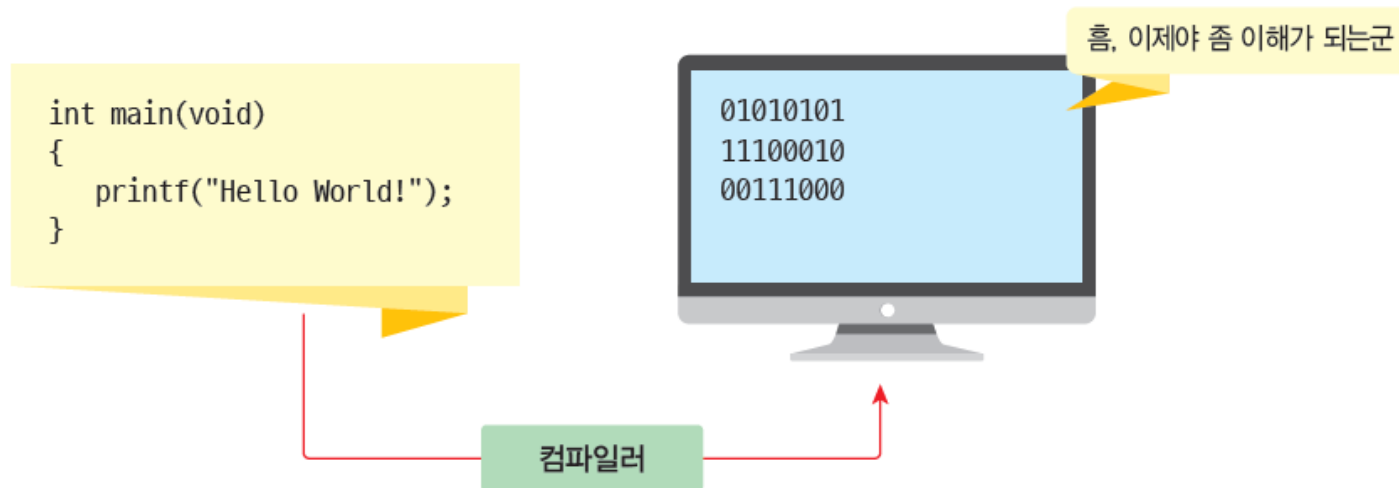
- 알고리즘의 각 단계를 프로그래밍 언어를 이용하여 기술
- 알고리즘을 프로그래밍 언어의 문법에 맞추어 기술한 것을 *소스 프로그램(source program)*
- 소스 프로그램은 주로 텍스트 에디터나 통합 개발 환경을 이용하여 작성
- 소스 파일 이름: (예) test.c



```
int main(void)
{
    printf("Hello World!");
}
```

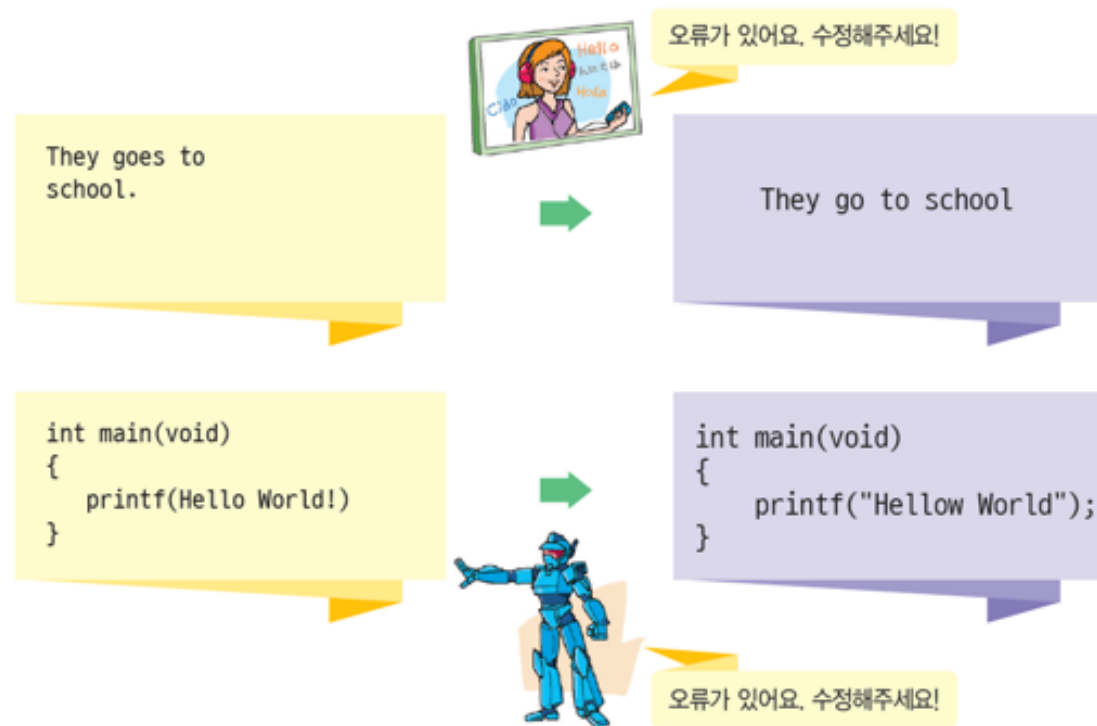
컴파일

- 소스 프로그램을 오브젝트 파일로 변환하는 작업
- 오브젝트 파일 이름: (예) test.obj



컴파일 오류

- 컴파일 오류(compile error): 문법 오류
 - (예) He go to school;



링크

- 컴파일된 목적(object) 프로그램을 라이브러리와 연결하여 실행 프로그램을 작성하는 것
 - 실행 파일 이름: (예) test.exe
- *라이브러리(library)*: 프로그래머들이 많이 사용되는 기능을 미리 작성해 놓은 것
 - (예) 입출력 기능, 파일 처리, 수학 함수 계산
- 링크를 수행하는 프로그램을 *링커(linker)*라고 한다.

오브젝트 파일

```
#include <stdio.h>

int main(void)
{
    printf("Hello World!");
    return 0;
}
```

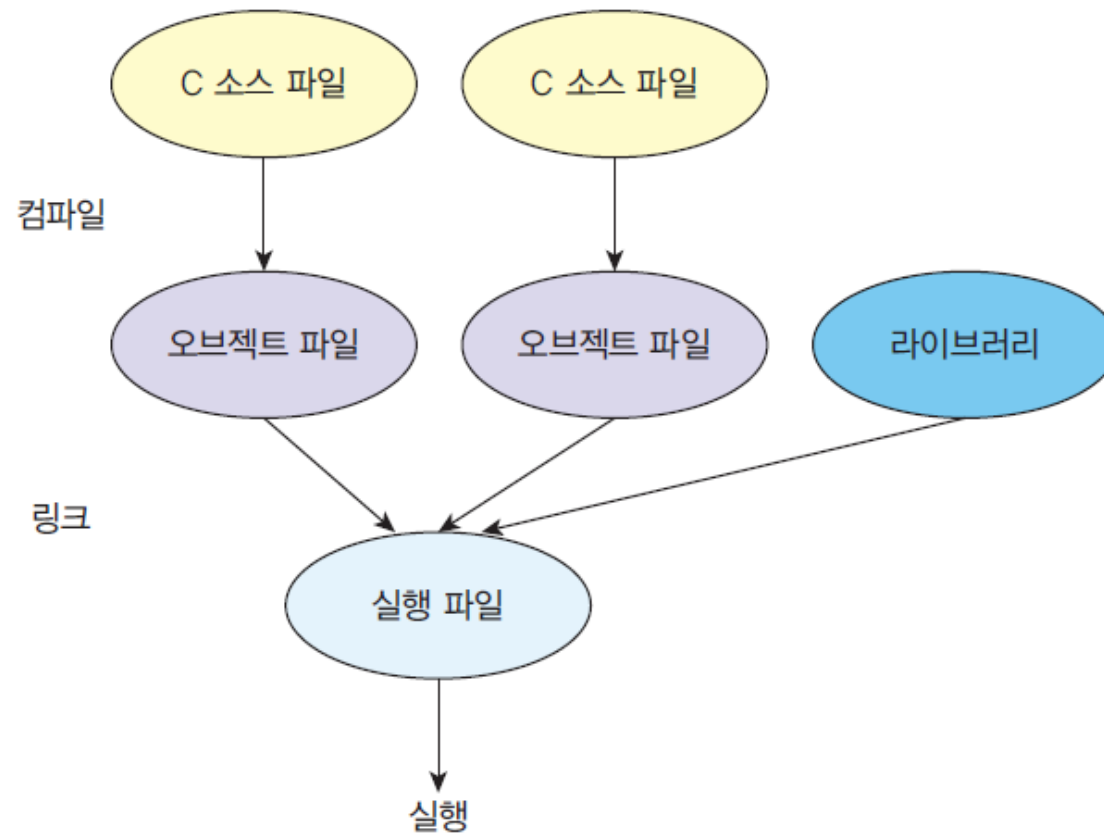
소스 파일



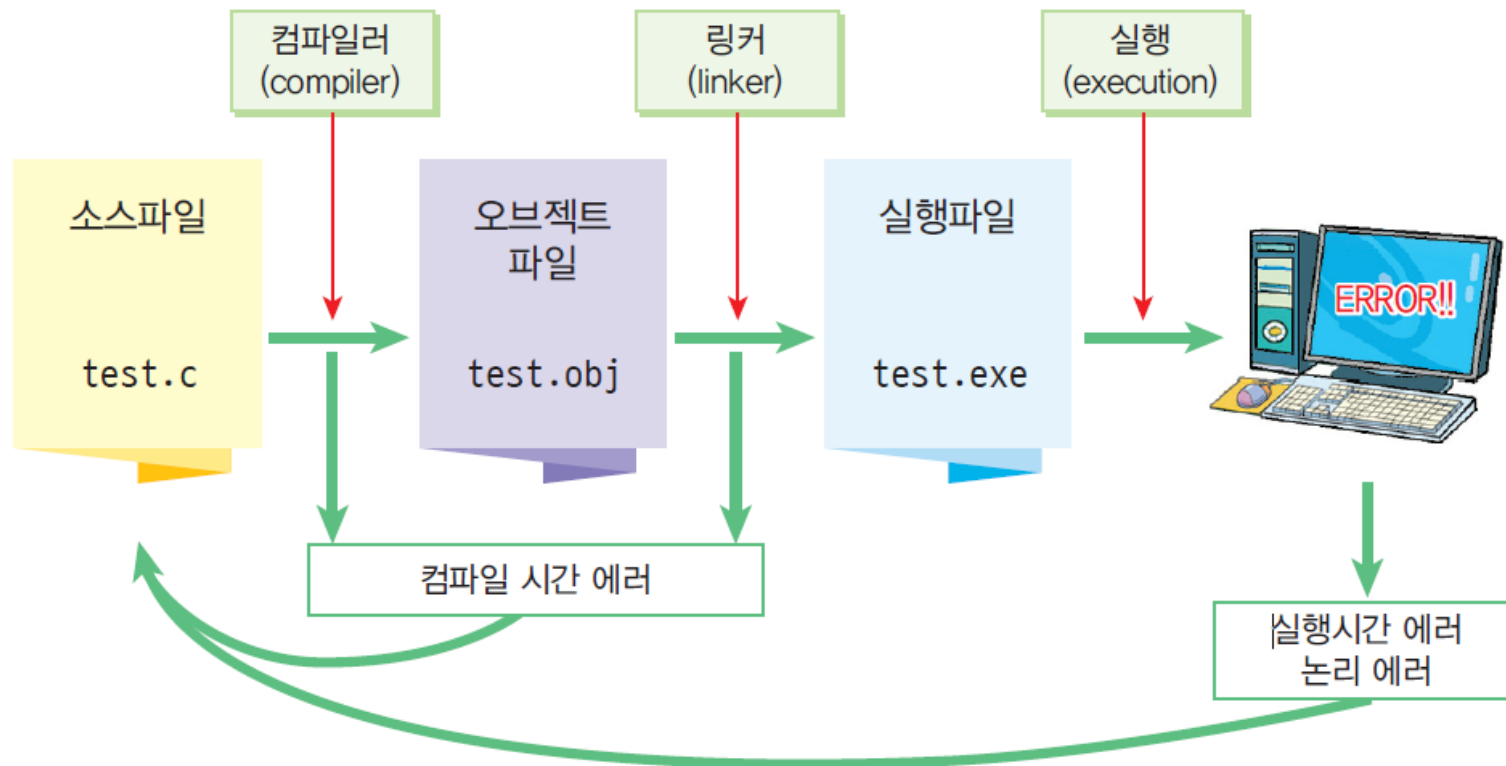
```
010101000001111101010
101010100000111110101
010101010000011111010
101010101000001111101
010101010100000111110
101001010100000111110
10101...
```

오브젝트 파일

링크



실행 및 디버깅



실행 및 디버깅

- 실행 시간 오류(run time error):
 - (예) 0으로 나누는 것
 - 잘못된 메모리 주소에 접근하는 것
- 논리 오류(logical error):
 - 문법은 틀리지 않았으나 논리적으로 정확하지 않는 것
 - (예)

- ① 그릇1과 그릇2를 준비한다.
- ② 그릇1에 밀가루, 우유, 계란을 넣고 잘 섞는다.
- ③ 그릇2를 오븐에 넣고 30분 동안 350도로 굽는다.

실수로 빈그릇을 오븐에 넣는다면
논리적인 오류입니다.



디버깅

- 소스에 존재하는 오류를 잡는 것



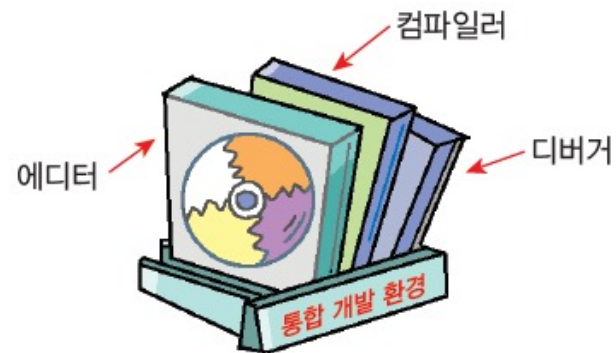
소프트웨어의 유지 보수

- 소프트웨어의 유지 보수가 필요한 이유
 - 디버깅 후에도 버그가 남아 있을 수 있기 때문
 - 소프트웨어가 개발된 다음에 사용자의 요구가 추가될 수 있기 때문
- 유지 보수 비용이 전체 비용의 50% 이상을 차지



통합 개발 환경

- 통합 개발 환경(IDE: integrated development environment)
= 에디터 + 컴파일러 + 디버거



통합 개발 환경의 예

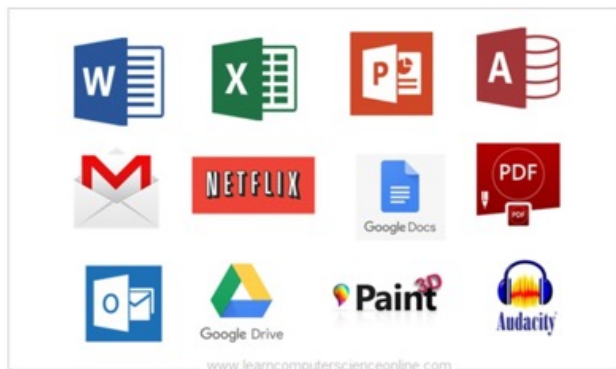
- Visual Studio: 마이크로소프트
- Visual Studio Code: 마이크로소프트
- 이클립스(eclipse): 오픈 소스 프로젝트
- Dev-C++: 오픈 소스 프로젝트
- Gcc, gdb, etc.: 리눅스 환경 개발환경



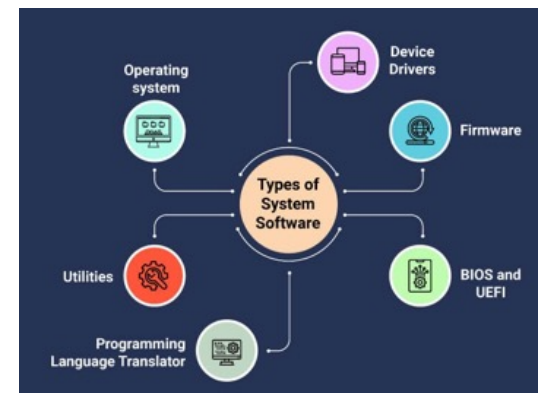
시스템 프로그래밍?

Software categories

- 소프트웨어 사용과 기능에 따른 분류
 - Application software
 - Designed to carry out a specific task
 - System software
 - Designed for a platform to help other software's execution



Application software



System software

What is system programming?

- Activity of programming system software
 - 운영체제 커널과 시스템 라이브러리를 직접 호출하는 소프트웨어
- Example of system software
 - Shell, text editor, compiler, debugger, core utilities, system daemons, servers, database, etc.
- 컴퓨터 시스템 자원을 활용하는 소프트웨어 개발
 - 컴퓨터 시스템 자원은 하드웨어와 소프트웨어로 구성됨
- 운영체제(리눅스)의 자체 기능을 활용하는 프로그래밍

Computer System



Computer System

Input device



Output Device

Main memory



CPU

Secondary storage



Communication device



시(poem)

- 컴퓨터 프로그램 생애

C와 프로세스

내가 작성한 C 프로그램은,
컴파일되어,
링크 과정을 거쳐,
“수행 가능한 프로그램 파일”로 만들어져서,
어느 디렉토리 위치하게 되리니,
내가 셸을 통해 이를 수행시키면,
결국 하나의 프로세스가 되어 수행 되리라....



프로그램 생애

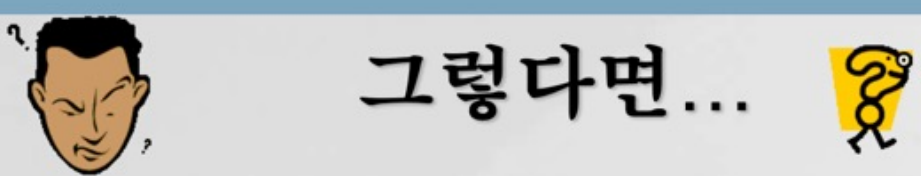
- 태아: 프로그램
- 출생: 프로세스

프로세스

- 프로그램: 하드디스크에 저장되어 있는 수행가능한 파일
 - 때로는 원시코드(C/C++)를 프로그램이라고 부르기도 하나 시스템 프로그래밍 관점에서는 프로그램은 컴파일/링크가 끝난 executable 파일을 뜻함
- 프로세스: CPU에 의하여 수행되고 있는 프로그램
 - 셸: 사용자와의 대화를 위한 프로세스
 - ls 등 명령어: 사용자가 사용하는 명령어. 셸에 의하여 프로세스로 새롭게 생성됨
 - a.out이나 사용자 프로그램도 프로세스로 수행됨

프로그램 구성

- 내가 작성한 코드
 - 사용자 프로그램코드
- 내가 작성하지 않은 코드
 - 라이브러리 함수
 - 시스템 호출



그렇다면...

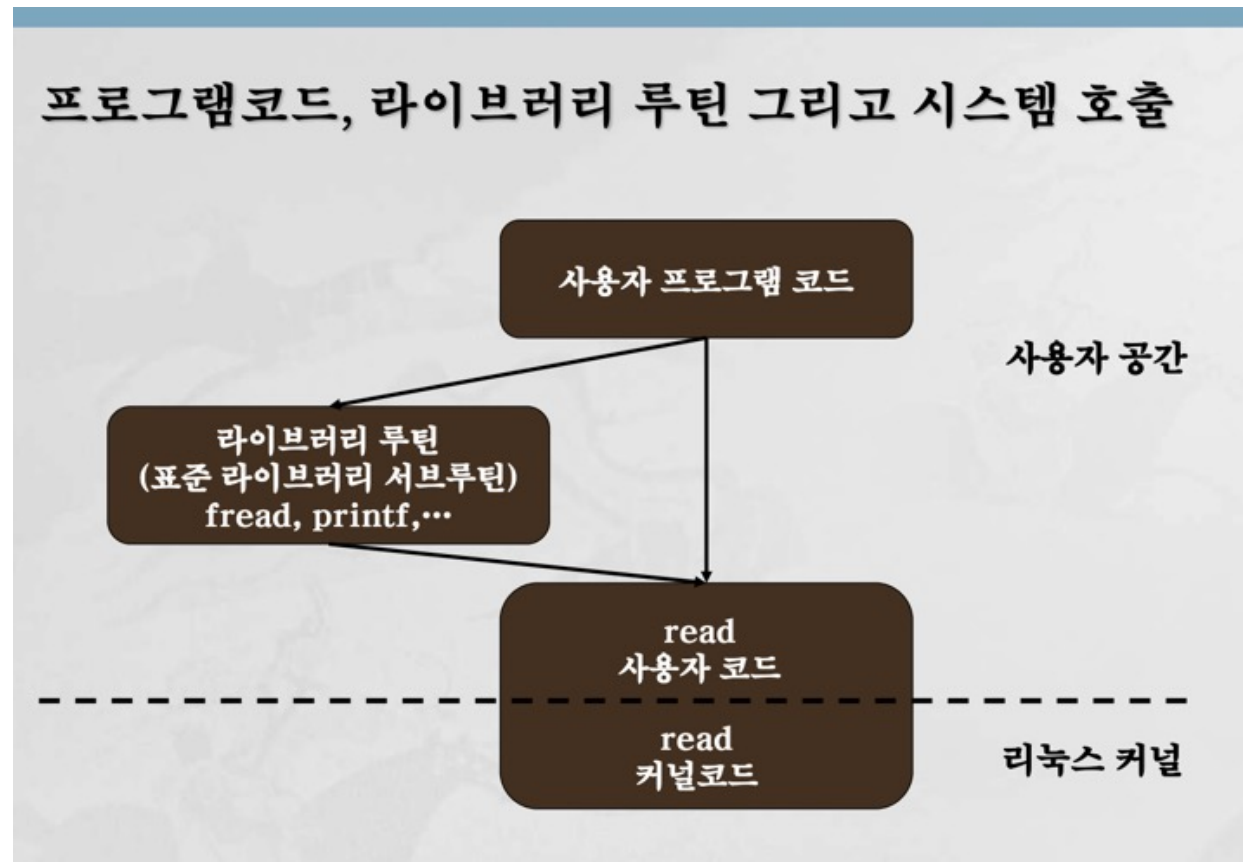
- 내가 만든 프로그램 중에
- 내가 짤것도 아닌 것이 많이 들어 있는데...
그것은 누가 짤것이고...누가 수행해주고
있는 것인가?

프로그램 실행 관계

- 내가 작성한 코드
 - 사용자 프로그램 코드
- 내가 작성하지 않은 코드
 - 라이브러리 루틴(함수)
 - 시스템 호출(운영체제 커널 함수)

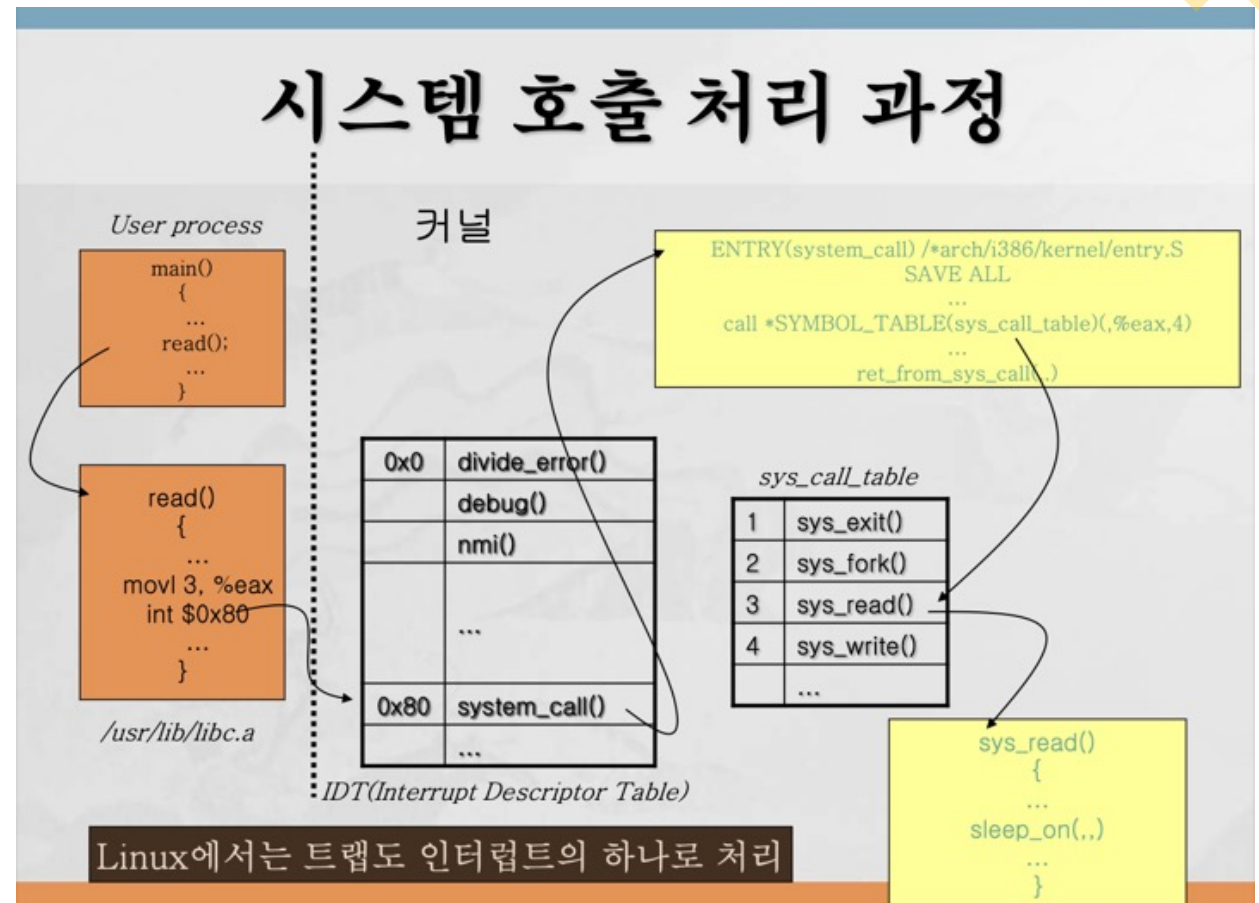
```
1  #include <stdio.h>
2
3  int main()
4  {
5      int sum;
6      sum = 1 + 2;
7      printf("sum: %d\n", sum);
8      return 0;
9  }
```

Source: OLC Linux system programming – 김두현 교수님



사용자 프로세스가 시스템 호출(system call) 요청

- 사용자 프로세스가 라이브러리 함수를 통해 시스템 호출을 요청



시스템 호출 종류별 분류

- 이 과목에서는 다양한 시스템 호출을 이용하여 프로그램을 작성 함

시스템 호출

● 5가지 부류로 나눌 수 있음

프로세스 제어	- end, abort, load, execute, create, terminate - get/set process attribute	- wait for time, wait for event, wait for signal - allocate/free memory
파일 조작	-Create/delete file -Open/close	-Read, write, reposition -Get/set file attribute
주변장치 조작	-Request/release device -Read, write, reposition	-Get/set device attribute -Logically attach/detach
정보관리	-Get/set time or date -Get/set system data	- Get/set process, file, device attribute
통신	-create, delete connection -Send, receive message	-Transfer status info. -Attach/detach remote device

유닉스/리눅스 시스템

Unix 시스템의 역사

- Multics (Multiplexed Information and Computer Service)
 - 1960년대 말
 - Bell Lab, MIT, GE 공동 프로젝트
 - 복잡한 기능으로 실패했지만 현대 대부분의 운영체제에 영향을 끼침

- Unix

- Ken Thompson & Dennis Ritchie in Bell Lab in 1969
- 초기 버전은 어셈블리 언어로 작성
 - 단일 사용자 시스템
 - 간단한 파일 시스템 + 명령어 해석기(shell) + Utility
- Thompson은 BCPL(Basic Combined Programming Language) 언어를 확장하여 B 언어를 개발하고 유틸리티를 작성
- Ritchie는 B 언어의 단점을 보완한 C 언어 개발
 - CLP → BCLP → B → C → C++
- 1973년 C언어로 4번째 버전의 Unix를 다시 작성
- 이후 BSD와 상용 유닉스 (System V) 계열로 분리되어 발전



Ken Thompson Dennis Ritchie

Unix 시스템의 역사

- BSD (Berkeley Standard Distribution)
 - 1974년 12월 Unix License를 처음으로 획득
 - 1977년 PDP-11용 최초의 BSD 버전 발표
 - BSD 4.0 (1980), BSD 4.1 (1981), [BSD 4.2 \(1983\)](#)
 - BSD 4.3 (1986), BSD 4.4 (1993)
 - 많은 기술적 진보를 이룸
 - 가상 메모리 (Virtual Memory)
 - TCP/IP와 IPC 통합
 - FFS (Fast File System)
 - Socket 소개

The FreeBSD Project

FreeBSD is an operating system used to power modern servers, desktops, and embedded [platforms](#). A large [community](#) has continually developed it for more than thirty years. Its advanced networking, security, and storage features have made FreeBSD the platform of choice for many of the [busiest web sites](#) and most pervasive embedded networking and storage devices.

» [Learn More](#)



**Download
FreeBSD**

Supported Releases

- » Production: [12.1](#), [11.4](#), [11.3](#)
- » Upcoming: [12.2](#)
- » Upcoming: [13.0](#)
- » [Support Lifecycle](#)

Unix의 발전

- 1991년에 등장한 리눅스는 오픈 소스로 공개되어 지속적으로 발전
- 리눅스는 같은 커널을 기반으로 하는 다양한 형태의 배포판을 사용

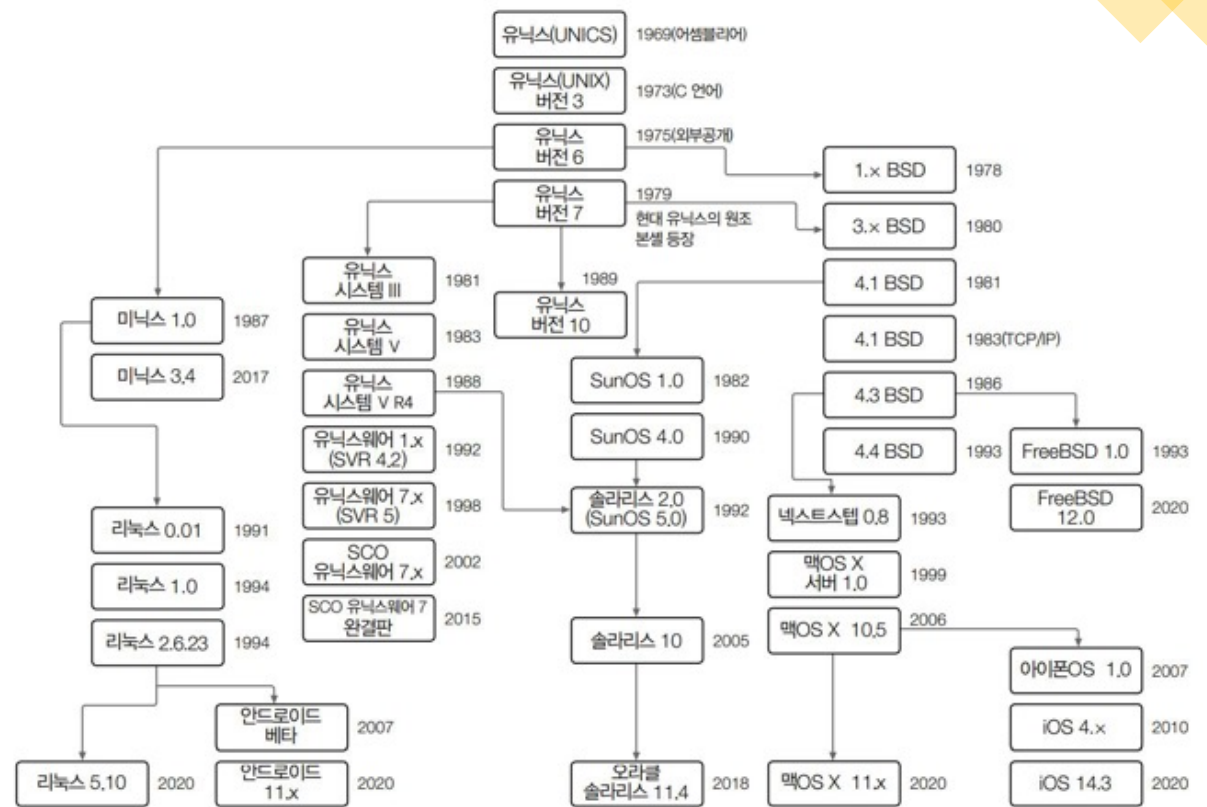
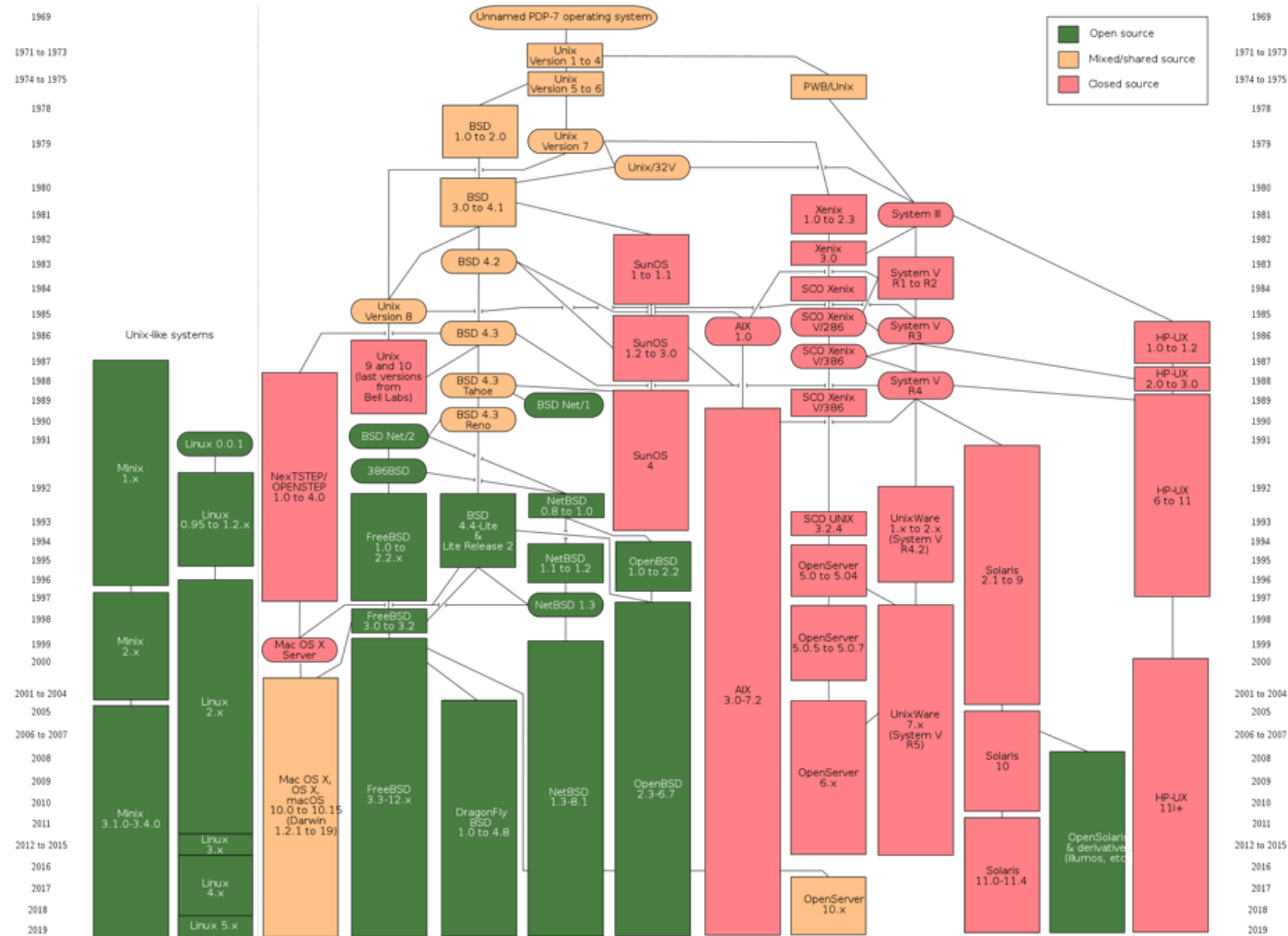


그림 1-1 유닉스와 리눅스의 발전 과정

Unix 시스템의 역사



Movie

- Hidden Figures, 2016



유닉스 시스템의 구성요소

- 커널 (Kernel)
 - 운영체제의 핵심으로 컴퓨터 시스템 자원을 관리
 - 프로세스 스케줄링, 메모리 관리, 입출력 관리, 파일관리
- 셸 (Shell)
 - 명령어 해석기 (Command Interpreter)
 - 사용자가 입력한 명령을 커널이 실행할 수 있는 명령으로 번역
 - 커널의 실행 결과를 다시 셸에게 전달
 - 여러 유틸리티 프로그램을 실행시켜 그 명령을 수행하도록 함
- 유틸리티 (Utility) 및 응용 프로그램
 - 사용자 편의를 위한 다양한 프로그램 및 응용 프로그램
 - 편집기, 문서처리 도구, 전자우편, 파일관리자
 - 사용자가 작성한 여러 응용 프로그램

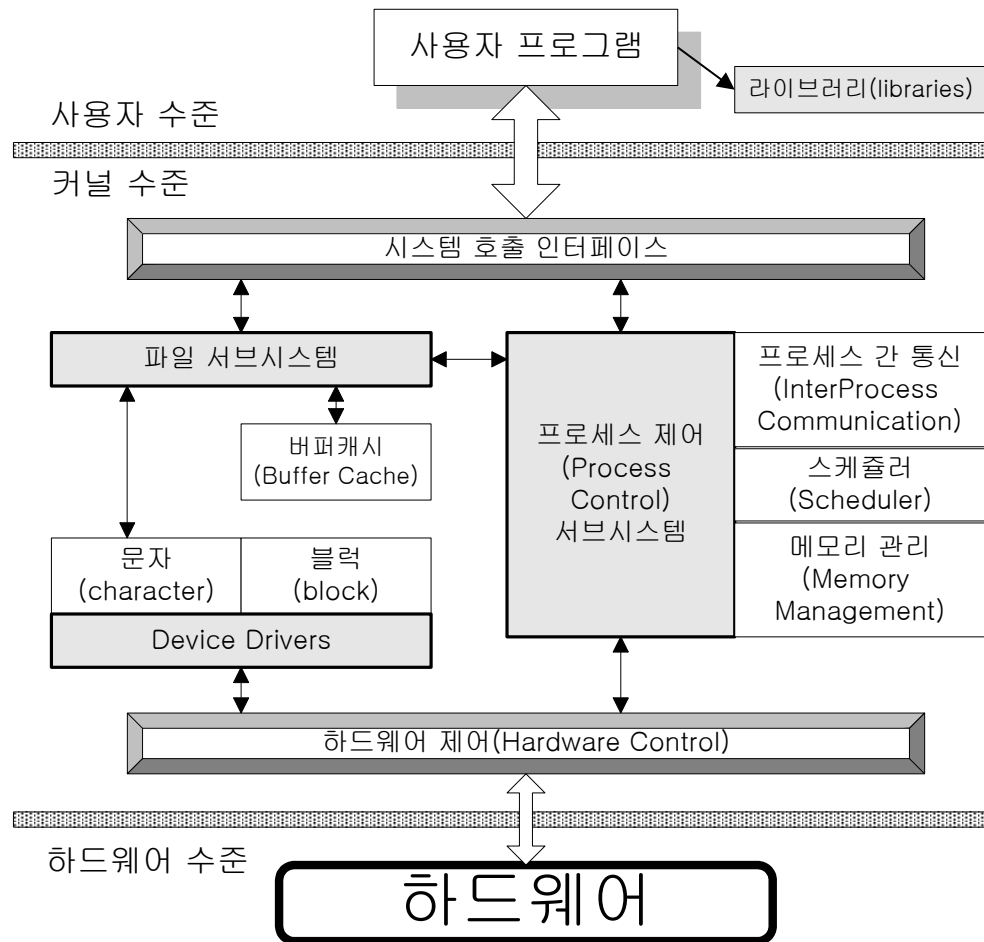
UNIX 시스템 프로그래밍이란

- 유닉스시스템 프로그래밍의 정의
 - 유닉스에서 제공하는 시스템 호출을 사용해 프로그램을 작성하는 것을 의미
- 시스템 호출
 - 응용프로그래밍이 유닉스 시스템의 서비스(기능)를 사용하고자 할 경우 제공되는 프로그래밍 인터페이스
 - 기본적인 형태는 c 언어의 함수 형태로 제공

리턴값 = 시스템호출명(인자, ...);

- 라이브러리 함수
 - 라이브러리 : 미리 컴파일된 함수들을 묶어서 제공하는 특수한 형태의 파일
 - 자주 사용하는 기능을 독립적으로 분리하여 구현해 둬으로써 프로그램의 개발과 디버깅을 쉽게 하고 컴파일을 좀 더 빠르게 할 수 있다
 - /lib, /usr/lib에 위치하며 lib*.a 또는 lib*.so 형태로 제공

시스템 호출(System Call)과 라이브러리(Library) 함수



시스템 호출과 Library 함수

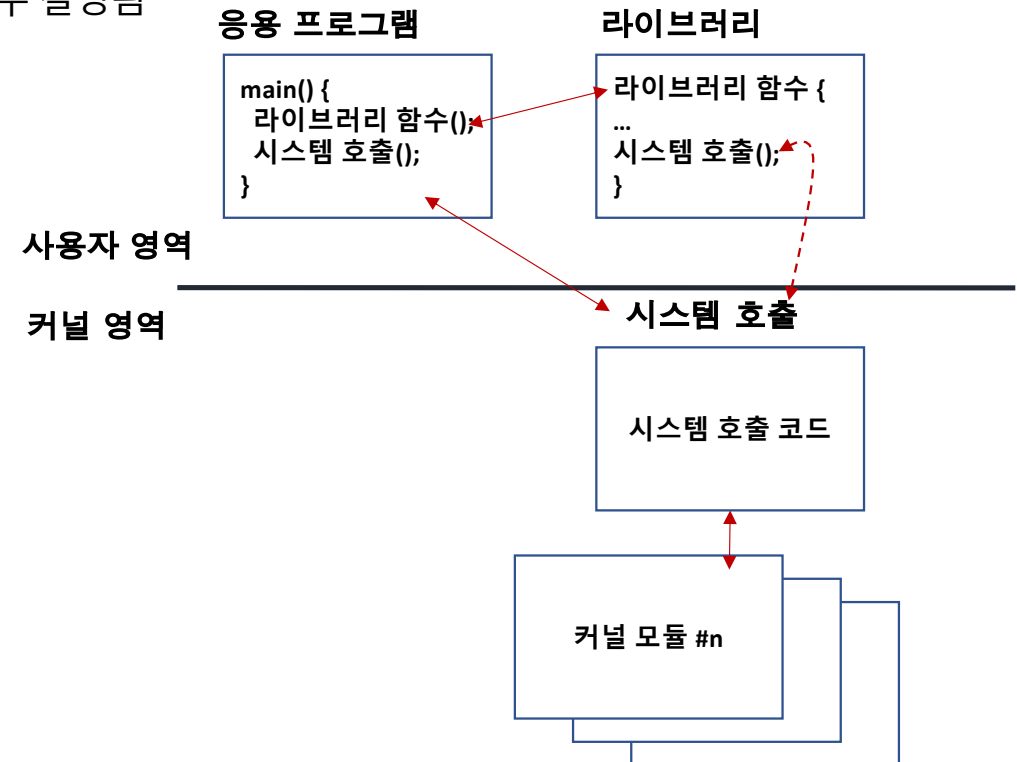
- 시스템 호출 (system call)
 - 운영체제에서 제공되는 시스템 수준의 함수
 - 시스템 호출에 의해 수행되는 대부분의 코드는 커널의 일부분으로 실행됨
 - 커널에서 제공되는 직접적인 함수를 이용
 - s/w 인터럽트 기법 (trap)을 사용
 - 파일관리: open(), close(), read(), write(), stat() 등
 - 프로세스관리: fork(), exec(), exit(), wait(), signal() 등
- Library 함수
 - 한 개 혹은 그 이상의 시스템 호출을 이용하여 작성된 사용자 수준의 함수
 - Library 함수를 호출할 수 있으나 그 자체가 커널의 진입점이 아님
 - printf()는 시스템 호출인 write() 함수를 호출하는 라이브러리

시스템 호출 vs. 라이브러리 함수

- 시스템 호출
 - 커널의 해당 서비스 모듈을 직접 호출하여 작업하고 결과를 리턴
- 라이브러리 함수
 - 라이브러리 함수의 기능에 따라 시스템 호출 사용 유무 결정됨

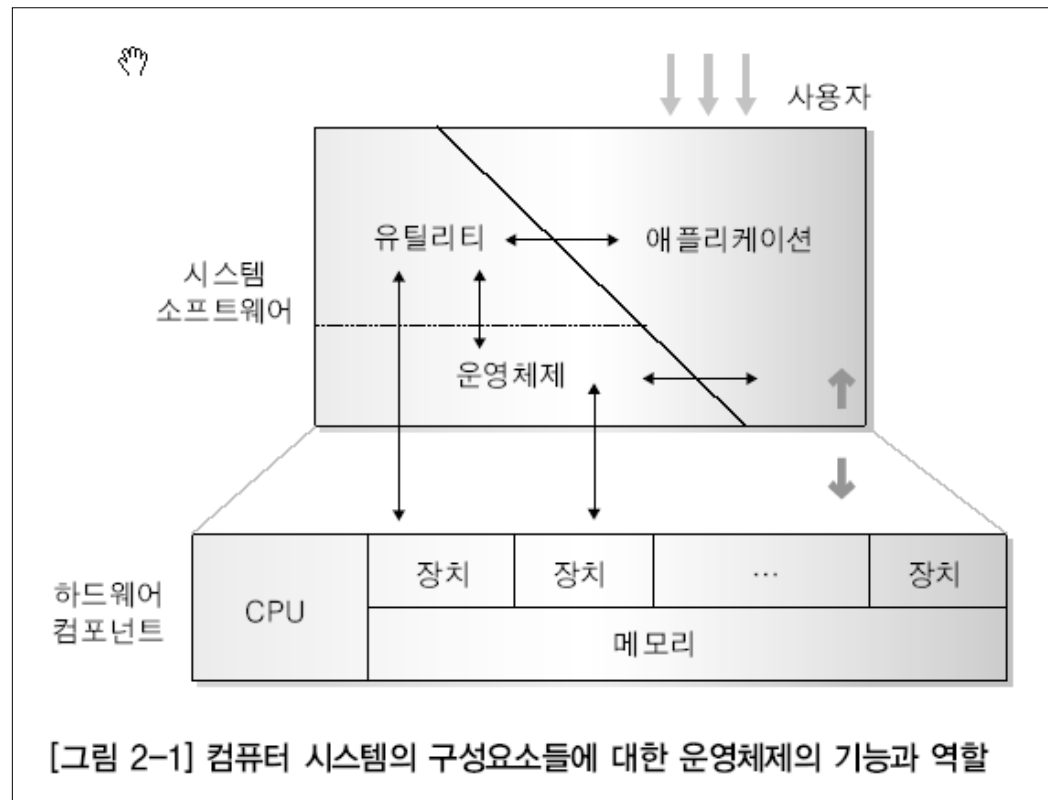


일상생활에서 시스템 호출과 유사한 예시(은행)



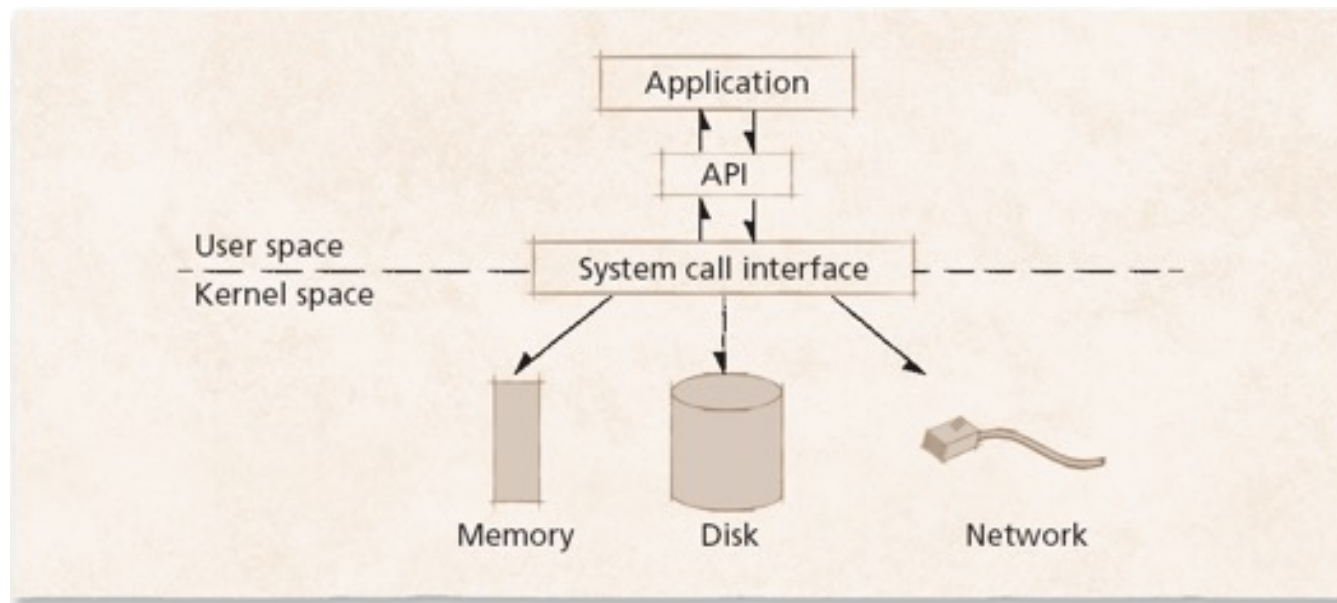
운영체제 개념 (참고)

- 일반적인 관점에서 본 운영체제의 기능과 역할



운영체제 서비스: API (참고)

- 사용자는 User Mode에서 User Space 만 사용 가능
- Kernel Space에 있는 OS의 기능을 사용하기 위해서는 OS가 제공하는 API를 호출하여 system call을 통해 OS에 접근
- Application Programming Interface
 - 프로그래머가 운영체제의 서비스를 사용하기 위하여 호출하는 루틴



QnA