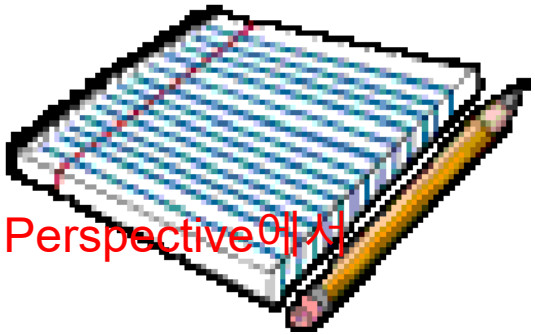
시스템 프로그래밍 (2016년 2학기)

장경선

충남대학교 컴퓨터공학과



과목운영에 관해 꼭 알아둘 사항!

수업 진행 형태

- 강의 주당 2시간, 실습 2시간,
- 강의 시간: Flip Learning 형태로 이루어짐. 미리예습을 해올 수 있어야 함

- 실습시간: 주어진 실습과제를 수행하는 시간
  -실습시간 시간중에 해야 하는 과제와
  -실습시간 후에 수행해야 할 과제로 구성

강의, flip learning 방법(1/2)

1. 수업에 들어오기전에, 해당 주의 강의동영상을 미리 시청하고
2. 동영상을 시청한 후에, 질문 사항을 적어서 개인 과제로 제출함. (수업시간 하루전까지)
3. 강의 시간 중에는 팀단위(4-6명)로 질문을 모아 팀내에서 해결하거나, 교수에게 질문함.
   질문에 대한 답은 교수/튜터나, 다른 팀에서 할 수 있음.
   --> 팀별 질문 목록과 질문 해소 결과는 팀별로 교수/튜터에게 제출함. (팀별 작업결과1)

강의, flip learning 방법(2/2)

4. 질문 해소가 어느정도 된 후에는, 교수가 내어준 문제를 팀 단위로 푸는 시간을 가지며, 이시간에는 다른 팀과 이야기하거나 할 수 없음.
→ 문제 해결 결과를 교수/튜터에게 제출함. (팀별 작업 결과 2)
5. 팀 별로 문제를 다 풀었으면, 그 결과를 제출한 후에, 팀별로(또는 몇몇 팀의) 문제 해결 방법에 대한 발표 시간과 팀간에 질문하는 시간을 가짐.
6. 교수/튜터는 마지막으로 문제에 대한 답/해결책을 정리해서 제시해 줌.

** 팀 별 작업 결과1과 2는, 팀 별 활동 점수에 반영되며, 학기말에 팀내 공헌도평가를 통해 팀 내에서 구성원들의 참여도를 상호 평가함.

성적 평가 구성비

- 중간시험 : 20%
- 기말시험 : 20%
- 실습 활동 및 실습 과제물: 40%
- 예습 및 강의시간중 팀별활동: 20%

출석과 성적, 부정 행위에 대해서

- 전체 출석 시간의 1/4(15시간) 이상 결석할 경우(16시간부터) F학점 처리됨.
- 불가항력적인 일이 발생한 것을 증명할 수 있는 경우를 제외하고는 병원진료등의 개인사유로 결석하는 것은 인정하지 않음.
- 첫주(1주)에 강의변경등의 사유로 늦게 변경하더라도 출석하지 않은 경우에는 출석으로 인정하지 않음.
- 부정행위에 대해서... 엄중 처벌됨.

부정 행위에 대해서

- 부정행위란?
  - 코드 공유: 복사, 재입력, 들여다 보기, 제공하
  - 설명하기: 다른 사람에게 말로 코드 관련
  - 코칭: 다른 친구가 한줄 한줄 코드를 써가도록 코칭
  - 웹에서 해결책 검색
  - 이전 코스나 온라인 코스 등에서 코드 복사
    - 주어진 코드만 사용할 수 있음.
- 부정행위가 아닌 것?
  - 시스템이나 도구 사용법을 설명해주는 것
  - 상위 수준, 추상적인 수준에서 다른 사람 돕는 것

부정행위의 처벌, 적발

- 부정행위 처벌
  - 예외없이 F학점 처리됨. 수업에서 제외됨.
  - 상담 기록 형태로 남김.
- 부정행위 적발
  - 고도의 도구 사용해서 적발함.
  - 의심되는 경우 직접 면담/구술을 통해 검증함
- 부정행위 하지 말고,
  - 일찍 과제를 시작하고,
  - 어렵거나, 곤란하면, 조교/튜터/교수에게 문의할것

교재, 홈피, 선수

- 교재 :
  - Computer systems : A programmer's perspective, 3rd Edition(3판), by Randal E. Bryant and David O'Hallaron, Prentice Hall
- 참고문헌
  - 리눅스, 유닉스 관련 책들 ….
- 교과목 홈페이지
  - http://e-learn.cnu.ac.kr (각자 통합정보시스템 id로 로그인함)
  - 개인과제 올리기, 강의/실습 자료
  - 공지 사항등을 보지 않아서 생기는 불이익이 없길…
- 선수지식
  - C 언어 프로그래밍

시스템 프로그래밍

강의 1. 과목 소개

이 과목의 주제

- 컴퓨터 프로그래밍은 추상화(Abstraction)
  - 추상화는 사용자를 위한 것….
- 컴퓨터 전공의 추상화
  - 자료구조, 어셈블리어, 고급언어, …, 함수, 객체/클래스, …
  - 가상메모리 ?
  - 아이폰의 앱들 – 카카오톡, 페이스북
- 개발자는 내부를 알아야!!!
  - 버그가 있을 때
  - 시스템 내부의 동작에 대한 이해가 필요할 때
- 과목의 목표
  - 효율적인 프로그래머를 만들자, 오픈 소스에 대한 개념도 갖자!!!
    - 기괴한 버그를 효과적으로 퇴치
    - 내가 작성한 프로그램이 무슨 일을 하고 있는지에 대한 심오한 이해제공
    - 성능을 고려한 프로그래밍
  - 향후 컴퓨터공학 전공 과목들을 위한 준비

컴퓨터에서 수학법칙이?

- 예
  - $x^2 \geq 0$? (한 수의 제곱은 0 또는 양수다!! ??)
    - 실수의 경우 : 성립!
    - 정수의 경우는?:
      - 40000 * 40000 --> 1600000000
      - 50000 * 50000 --> ??
  - (x + y) + z = x + (y + z)? 결합법칙은 항상 성립?
    - 정수의 경우 - Unsigned & Signed Int's: Yes!
    - 실수의 경우도?:
      - (1e20 + -1e20) + 3.14 --> 3.14 b.c 참고
      - 1e20 + (-1e20 + 3.14) --> ??

컴퓨터 내에서의 연산

- 의미 없는 랜덤 값을 만들지는 않는다
  - 수식연산은 중요한 수학법칙을 갖는다
- 그러나 컴퓨터에서는 일반적인 수학 법칙을 가정할 수 없다
  - 데이터 표현의 유한성 때문에
- 관찰(Observation)
  - 어떤 추상화가 어떤 상황에 적용된 것인지 이해하는 것이 필요
  - 컴파일러 개발자나 고급 프로그래머들에게는 중요한 주제

어셈블리어도 알아야?

- 대개의 경우, 어셈블리 프로그램을 직접 작성할 가능성은 ..... zero !
  - 컴파일러가 훨씬 더 우수하고 참을성이 있다
- 어셈블리어를 이해하는 것은 하드웨어 수준의 실행모델을 이해하는데 매우 중요
  - 버그가 있을 때 프로그램의 동작
    - 고차원 언어 모델로는 이해할 수 없다
  - 프로그램의 성능을 튜닝할 때
    - 프로그램의 효율성을 이해하기 위해
  - 시스템 소프트웨어 구현시
    - 컴파일러는 기계어 코드가 목적코드이다
    - 운영체제는 프로세스의 상태를 관리해야 한다
  - malware를 만들거나 malware와 싸우기 위해
    - x86 어셈블리를 배운다

C 코드와 어셈블리어(a.c → a.s)

a.c

```
#include <stdio.h>

main(int argc, char *argv)
{
int i;
for(i=0; i < 20; i++)
printf("i = %d\n",i);

}
```

어셈블리 코드 예제(a.s)

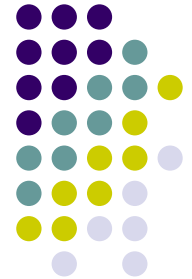
a.c - assembled – a.s (page1/2)

```
  .file   "a.c"
        .section        .rodata.str1.1,"aMS",
@progbits,1
.LC0:
        .string "i = %d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB24:
        .cfi_startproc
        pushq   %rbx
        .cfi_def_cfa_offset 16
        .cfi_offset 3, -16
        movl    $0, %ebx
 .L2:
        movl    %ebx, %edx
        movl    $.LC0, %esi
```

(page 2/2)

```
        movl    $1, %edi
        movl    $0, %eax
        call    __printf_chk
        addl    $1, %ebx
        cmpl    $20, %ebx
        jne     .L2
        popq    %rbx
        .cfi_def_cfa_offset 8
        ret
        .cfi_endproc
.LFE24:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 4.9.2-
10ubuntu13) 4.9.2"
        .section        .note.GNU-
stack,"",@progbits
```

메모리는 유한하고, 중요하다!!



- 메모리크기는 무한하지 않다
  - 메모리는 할당해주고 관리해야 한다
  - 많은 응용프로그램들은 메모리에 영향 받는다
- 메모리 참조 버그는 치명적이다(특히C언어)
  - 버그의 결과가 시공간적으로 무연관성을 보인다
- 메모리 성능은 일정하지 않다
  - 캐시와 가상메모리 효과는 성능에 비선형적 영향을 미친다
  - 프로그램들을 메모리 시스템의 특성에 최적화 시키면 성능이 대폭 개선된다

메모리 참조 버그 (fun.c)

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

```
fun(0)  ->    3.14
fun(1)  ->    3.14
fun(2)  ->    3.1399998664856
fun(3)  ->    2.0000061035156
fun(4)  ->    3.14, then segmentation fault
```

```
sun@sun-virtualubuntu:~/Dropbox/2015-SysProg/2016-lecture/sysp-01-sr
0- 3.140000
1- 3.140000
2- 3.140000
*** stack smashing detected ***: ./fun terminated
Aborted (core dumped)
```

메모리 참조오류

- **C와 C++는 메모리 보호 기능이 없음**
  - 배열의 인덱스 범위 오류, 포인터 값(주소)의 오류
  - malloc()과 free()의 남용 …
- **골치 아픈 버그로 시달리게 됨**
  - 시스템과 컴파일에 따라 달라짐
  - 잘못을 야기한 곳과 잘못된 곳(오염위치)는 무관
  - 잘못된 참조는 바로 오류를 야기하지 않을 수있음.
- **대책?**
  - 자바,Python 등으로 프로그램?
  - 오류 원인 분석? 도구 사용 ?

메모리 시스템 성능 (loop1.c, loop2.c)

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

59,393,288 clock cycles

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

1,277,877,876 clock cycles

(Measured on 2GHz Intel Pentium 4)

21.5 **배 더 느리다**!

- 계층적 메모리 구조를 이해하는 사람과 그렇지 못한 사람
- 성능은 메모리 접근 방식에 영향을 받는다
  - 다중 배열에서 어떻게 인덱싱 하는가에 따라 달라진다

컴퓨터는 여러 일을 함!!

- *컴퓨터는 프로그램 실행 외에도...*
- 데이터의 입출력이 필요하다
  - I/O 시스템은 프로그램의 신뢰성과 성능에 매우 중요하다
- 프로그램이 순차적으로 실행되는 것은 아니다
  - 예외적인 순서로(돌발!) 진행될 수 있어 컴퓨터다

작업관리자 ??

이 과목의 목표

- 프로그램의 하드웨어에서 동작 이해.
  - 어셈블리 프로그래밍을 이용해서
- 리눅스의 프로그래밍 환경
  - VI, GDB, GCC 등 개발 도구에 익숙해짐.
- 컴퓨터 시스템을 활용하는 프로그래밍 기법을 학습한다
  - Process control
  - Signal
  - Memory management

컴퓨터 시스템에 관한 깊은 이해를 제공한다

강의 일정(변경될 수 있음!!!!)

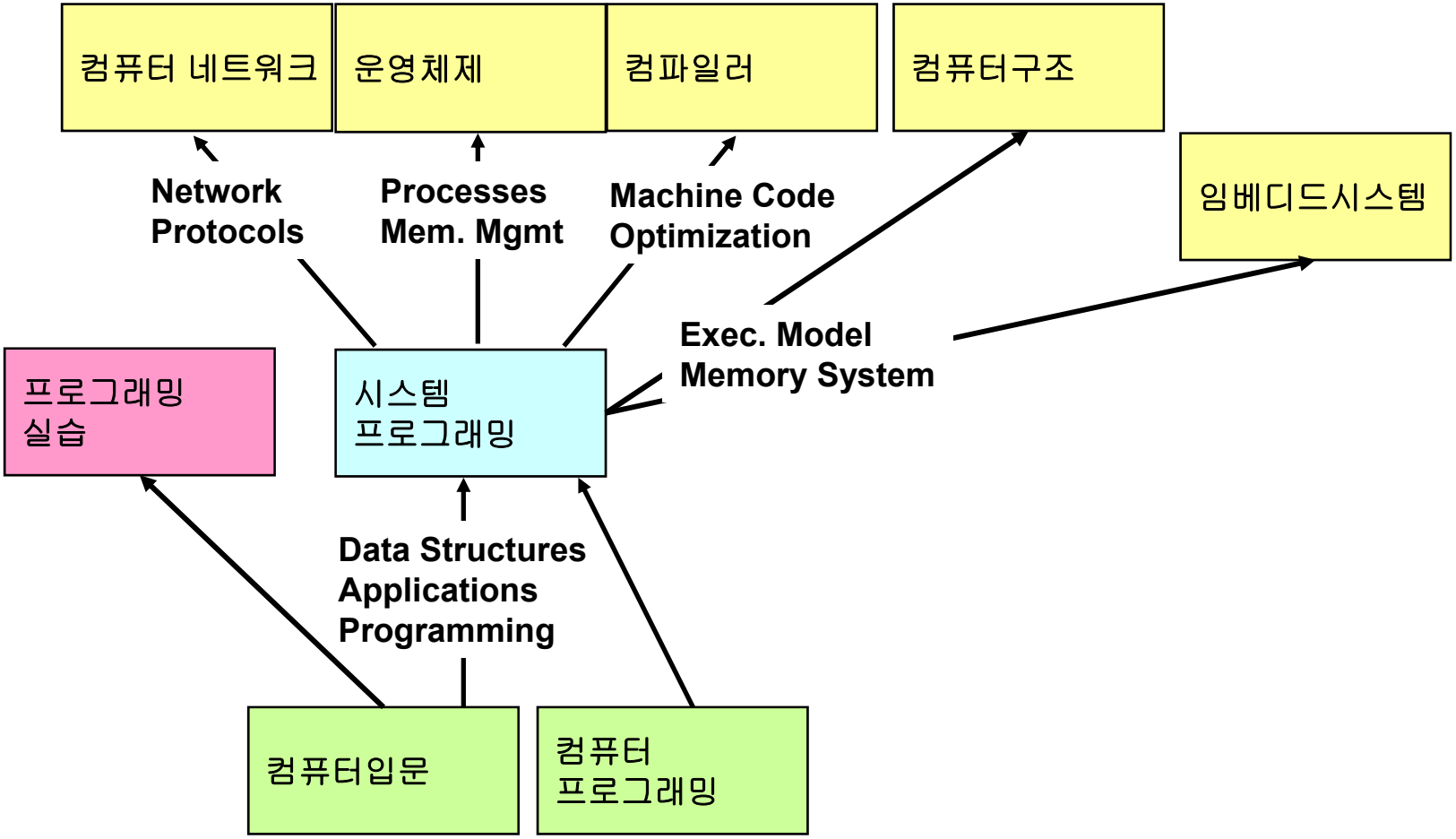
주	날짜	강의실	날짜	실습실
1	9월 1일(목)	**소개 강의**	9월 6일(화)	**리눅스 개발환경 익히기 (VI, 쉘 기본명령어들)**
2	9월 8일(목)	**정수 표현 방법**	9월 13일(화)	GCC & Make, shell script
3	9월 15일(목)	**추석 휴강**	9월 20일(화)	**C 복습과 GDB 사용하기 1 (소스 수준 디버깅)**
4	9월 22일(목)	**실수 표현 방법**	9월 27일(화)	**Data lab (GDB활용)**
5	9월 29일(목)	**어셈1 - 데이터이동**	10월 4일(화)	**어셈1 – move(실습),**
6	10월 6일(목)	**어셈2 – 제어문**	10월 11일(화)	**어셈2- 제어문 (실습)**
7	10월 13일(목)	**어셈3 – 프로시저**	10월 18일(화)	**어셈3-프로시저(실습)**
8	10월 20일(목)	**어셈보충/중간시험**	10월 25일(화)	**GDB 사용하기2(어셈수준)**
9	10월 27일(목)	**보안**(buffer overflow)	11월 1일(화)	Binary bomb 1 (GDB활용)
10	11월 3일(목)	**프로세스 1**	11월 8일(화)	Binary bomb 2 (GDB활용)
11	11월 10일(목)	**프로세스 2**	11월 15일(화)	Tiny shell 1
12	11월 17일(목)	**시그널**	11월 22일(화)	Tiny shell 2
13	11월 24일(목)	**동적메모리 1**	11월 29일(화)	Malloc lab1
14	12월 1일(목)	**동적메모리 2**	12월 6일(화)	Malloc lab2
15	12월 8일(목)	**기말시험**	12월 13일(화)	Malloc lab3

이 수업이 끝날 때 여러분은,

- 리눅스에서 C 프로그래밍 하는 기술을 습득한다
- IA-32 어셈블리어를 사용할 수 있게 된다.
- 다른 프로세서의 어셈블리어를 쉽게 습득할 수 있게 된다.
- 저 차원의 디버깅 기술을 사용할 수 있게 된다.
- 컴퓨터 내부에서의 숫자의 표현방법을 이해하게 된다
- 고급언어가 어떻게 번역되고, 컴퓨터에서 처리되는지 이해하게 된다.
- 운영체제의 예외처리 과정을 이해한다
- 프로세스를 제어하는 프로그램을 작성할 수 있다
- 운영체제의 메모리 관리 원리를 이해하게 된다
- Open source 개발자의 기초를 다지게 됨…

막강한 준-프로 시스템 프로그래머가 된다!

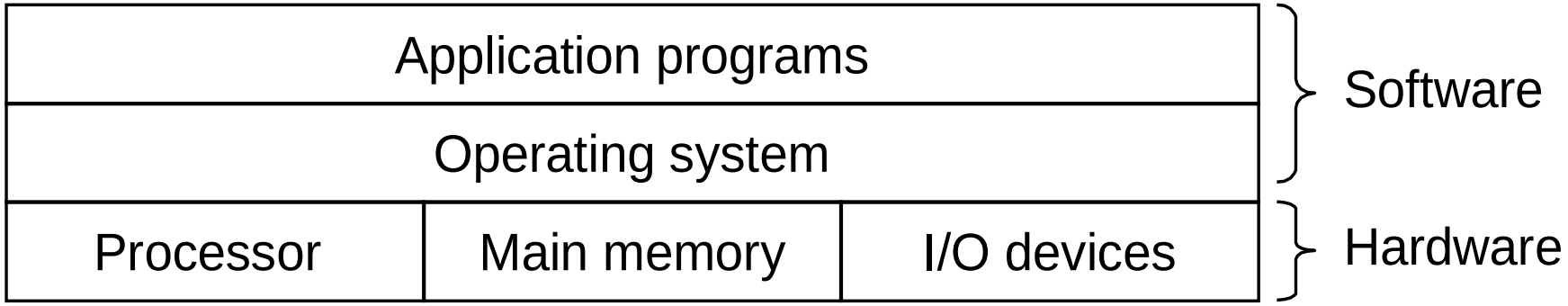
컴퓨터 전공과목과의 결정적 관계



제 1장. 컴퓨터 시스템 개관

- 컴퓨터 시스템의 본질을 찾아서...

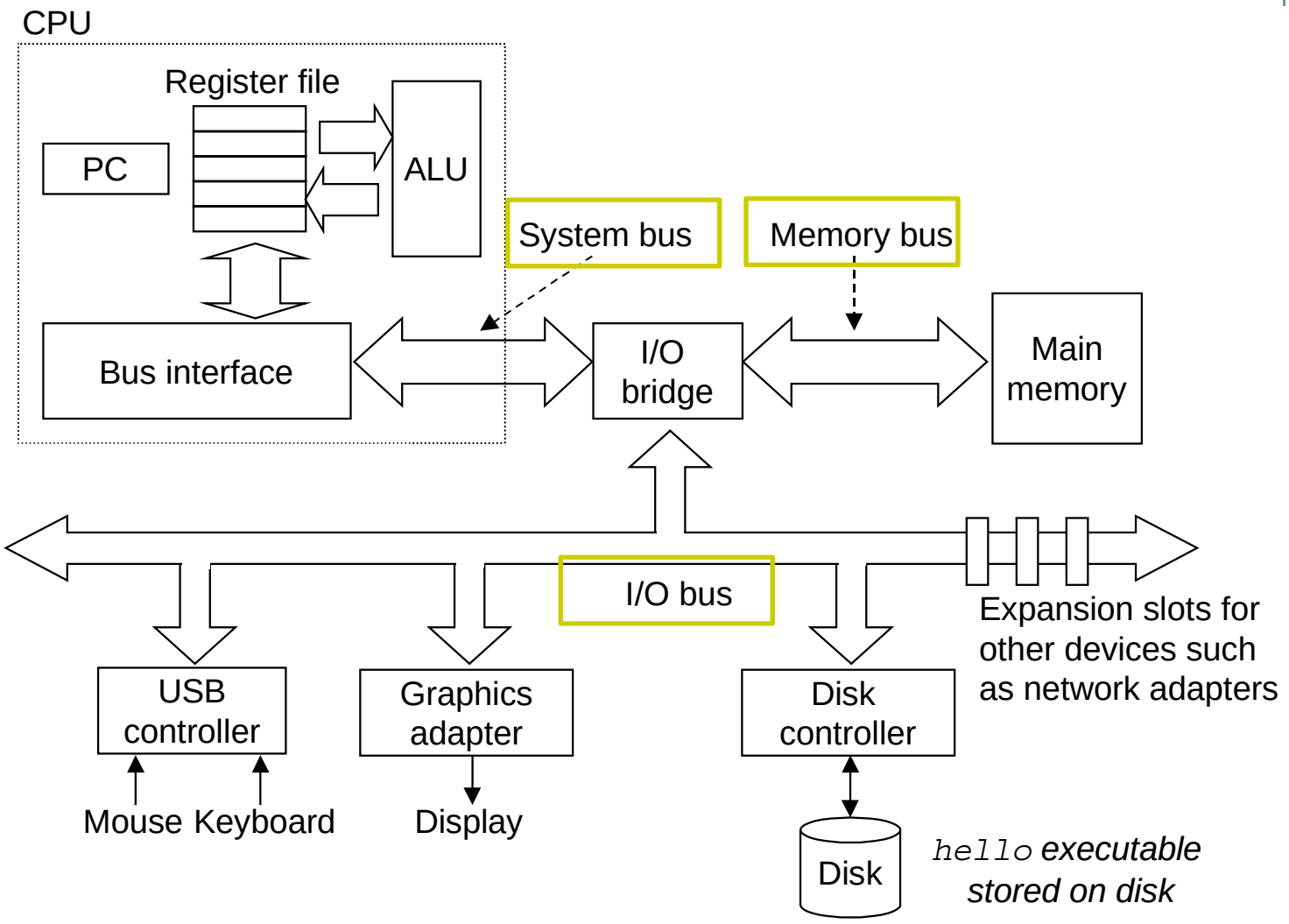
컴퓨터 시스템의 계층도


Application programs
Operating system
Processor
Main memory
I/O devices
Software
Hardware

컴퓨터 하드웨어의 구조

- 하드웨어 구성요소
  - 버스 buses
  - 입출력 장치, I/O devices
  - 주기억장치, main memory
  - 캐쉬, cache memory
  - 중앙처리장치, CPU (Central Processing Unit)
- 버스
  - 구성요소간의 정보교환 통로
  - 종류 : 입출력버스, 시스템 버스

컴퓨터의 구조


CPU
Register file
PC
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Display
Disk
hello executable stored on disk

프로세서

- 중앙처리장치 Central Processing Unit : CPU
  - 산술논리연산장치 : arithmetic and logic unit, ALU
  - 제어장치 : Control unit
  - 레지스터
    - 프로그램 카운터 PC
    - 상태 레지스터
    - 메모리 주소 레지스터
    - 메모리 데이터 레지스터
    - 명령 레지스터, instruction pointer, IP
    - 범용 레지스터, general-purpose register

CPU

- 주요기능
  - 기억장치로부터 명령/데이터 호출 및 저장
  - 명령의 해석 : 제어장치
  - 명령의 실행 : ALU
  - 입출력 연산 : I/O
- 명령 주기 Instruction cycle
  - 인출 fetch : PC에 의거하여 명령어를 가져옴
  - 해독 decode : 명령어를 해석하여 제어 신호 생성
  - 실행 execution : 가져온 명령어를 실행
  - 결과저장 store : 실행 결과를 저장

운영체제 Operating System

- 운영체제란?
  - 응용프로그램이 하드웨어를 효율적으로 사용할 수 있도록 도와주는 프로그램
- 목적
  - 하드웨어 추상화 Hardware Abstraction
  - 시스템 인터페이스 추상화
- 운영체제의 관리대상
  - 프로세스
  - 메모리
  - 입출력장치
  - 소프트웨어 자원
    - 파일시스템, 라이브러리, 유틸리티

Hello world ?!!

- 위대한 프로그램 hello.c

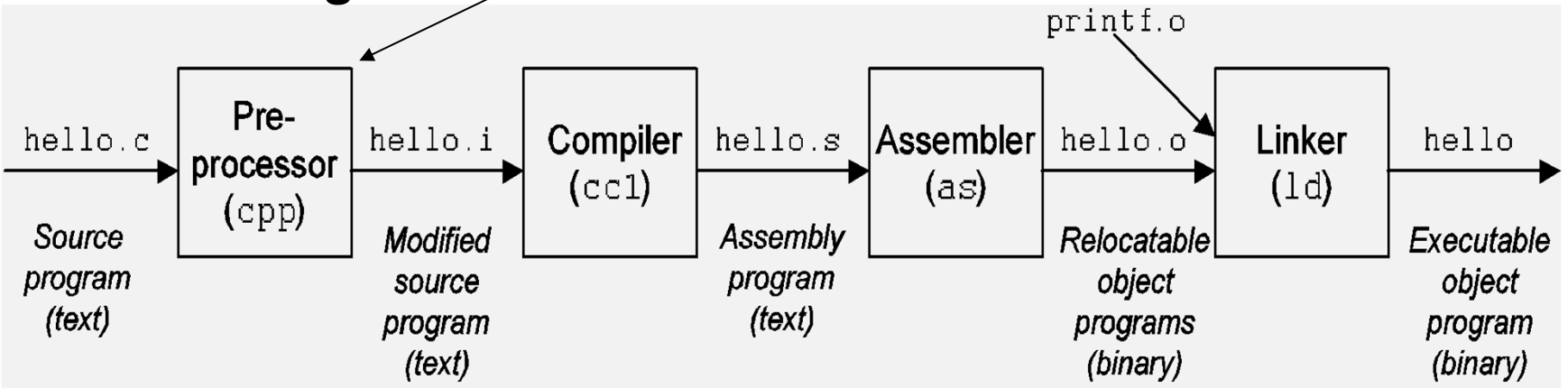
```
#include <stdio.h>

int main()
{
        printf("hello, world\n");
}
```

- 본질 ! 컴퓨터의 모든 정보는 비트, 바이트로 저장된다

프로그램은 변신한다

#include <stdio.h>
#define MAX 1024

- hello.c 컴파일 과정
  - unix> gcc –o hello hello.c



- 본질 ! 컴파일러의 뒷면에는 이렇게 복잡한 일이 일어나고 있다

어셈블리 프로그램 ? 실행 목적 프로그램 ? 기계어 프로그램 ?

컴파일 과정을 이해하는 것은 매우 중요하다

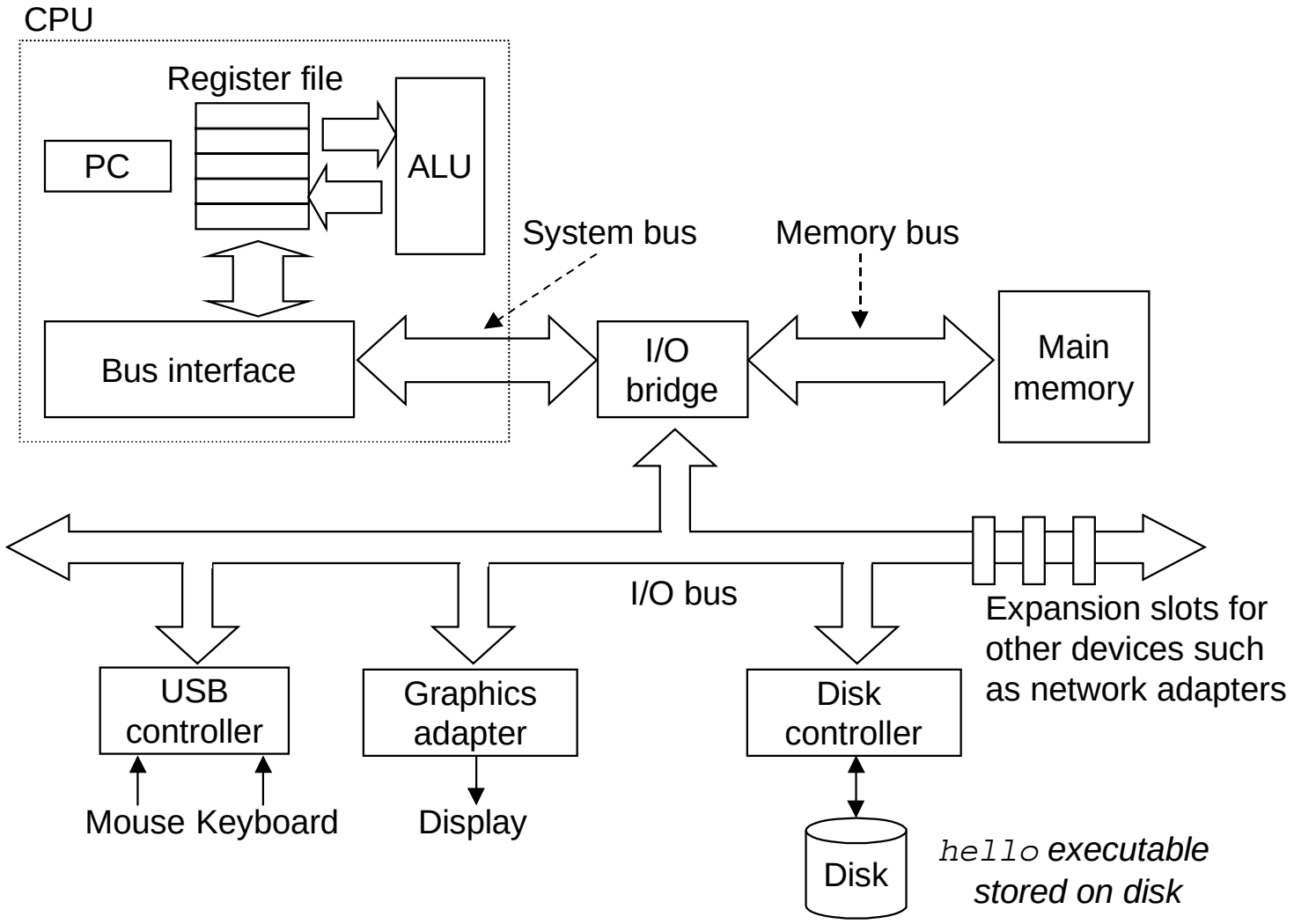
- 프로그램이 컴퓨터 내부에서 어떻게 처리되는지 이해할 수 있다
- 대규모 프로그램 개발 과정을 이해할 수 있다
- 프로그램 성능을 개선할 수 있다
- 링크 시에 발생하는 에러를 이해할 수 있다

변환된 프로그램의 실행

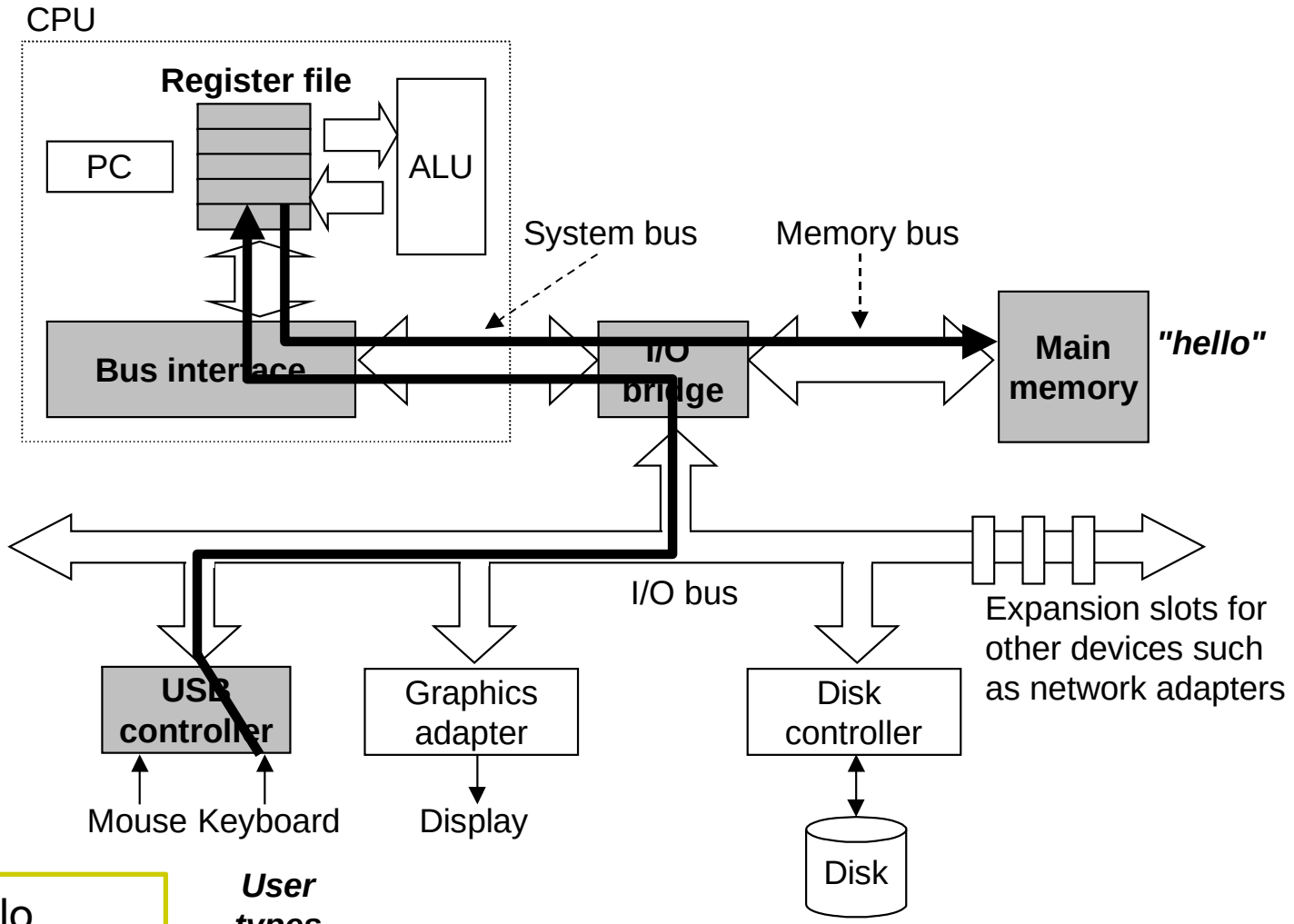
```
cesl-linux> ./hello
hello, world
cesl-linux>
```

- 리눅스 쉘에서 실행파일 이름을 치면 실행됨
  - 쉘이 여러분이 입력한 파일 이름을 찾아서, 그것이 실행파일이면 프로그램을 로드하여 실행함
  - hello 프로그램이 실행되면서 결과가 화면에 출력됨.
  - 쉘이 다음 명령을 기다리고 있음
- 본질?

컴퓨터의 구조

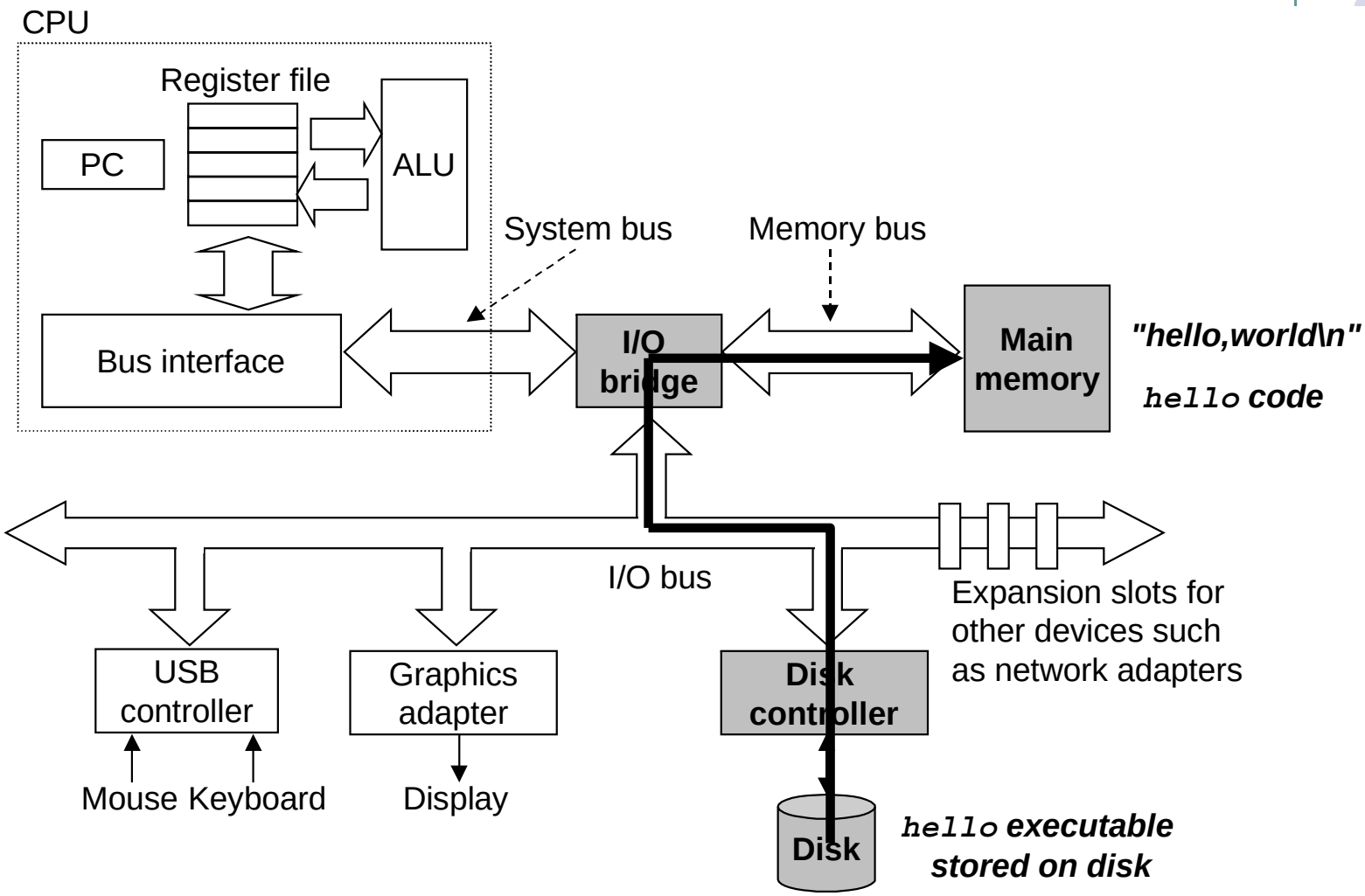

CPU
Register file
PC
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Display
Disk
hello executable stored on disk

실행 순서1: hello 명령의 인식

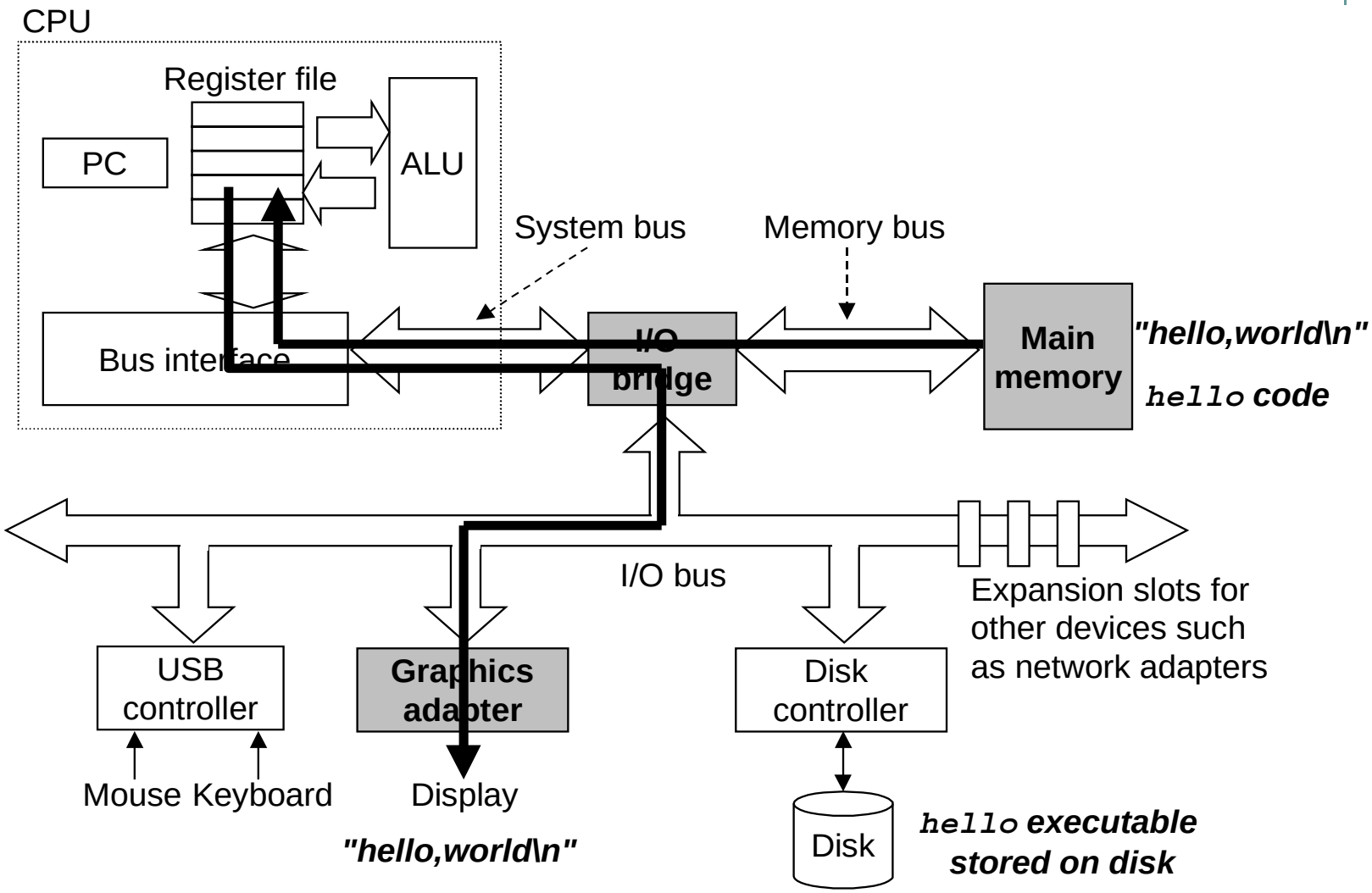



```
$ ./hello
hello, world
```

실행 순서2 : hello 프로그램의 로딩



실행 순서3 : hello 프로그램의 실행



요약

- 컴퓨터 시스템은 하드웨어, 시스템 소프트웨어, 응용프로그램으로 구성된다.
- 프로그램은 번역되어 실행된다.
- 프로그램은 시스템 프로그램의 도움으로 하드웨어에서 실행된다.

다음주를 위한 준비

- 실습 1 (화요일) – 리눅스 배우기: VI 편집기와 쉘 기본 명령어들 배우기

- 다음주 동영상 강의 예습: 정수표현방법
- 예습 질문은 개인과제로 올림
- 강의자료 내려 받기. 동영상 강의 시청. 예습 질문 과제로 올리기는 ➔
  - e-learn.cnu.ac.kr