**2018-02-21 (1회차)**

**[1] 리눅스에서 자주 사용하는 명령어**

| | |
|---|---|
| 터미널 켜기 | Ctrl + Alt + T |
| 현재 디렉토리가 어디인지 알려준다 | pwd |
| 현재 디렉토리에 뭐가 있는지 보여준다 | ls |
| C프로그램 컴파일 | gcc test.c |
| 디렉토리 만들기 | mkdir lecture |
| 프로그램 디버깅 | gdb |
| 디렉토리 이동 | cd |

1. 절대경로: '/' 최상위 root에서 가고 싶은 위치까지 지정

ex) cd /home/id/lecture/test

2. 상대경로: 현재 위치를 기준

'..' 의 경우에는 상위 디렉토리

'.' 은 현재 디렉토리

ex) cd ../result 상위 디렉토리 이동 후 result로 이동

cd../../test 상위 디렉토리 두 번 이동 후 result로 이동

| | |
|---|---|
| clear | 화면지우기 |
| cd ~ | home/ID 계정의 시작 디렉토리로 이동 |
| ls.. | 상위 디렉터리를 본다 |
| rm -rf result | result 디렉토리를 지운다 |
| rm -rf a.out test | a.out 과 test 지운다 |
| rm -rf / | 주의! C드라이브(/)를 지운다 |
| vi print_message.c (코드명) | print_message.c 코드를 수정한다 |
| cat print_message.c | print_message.c 내용보기 |
| | (cat: 리눅스 파일내의 전체정보를 화면에 출력) |
| gcc print_message.c | 컴파일 -> a.out 실행파일 생성된다 |
| ./a.out | 현재위치(./)의 a.out을 실행 |
| gcc -o print_msg(바뀐 파일명) print_message.c (바꿀파일) | a.out 실행파일 이름 바꾸기 |
| cp print_message.c print_str.c | 내용동일 다른이름으로 복사 |

**[2] 항공기나 우주선에 고성능 아키텍쳐를 사용할 수 없는 이유**

정답: 우주방사선 때문에 안정성 위협

그래서 Radiation Hardened(방사선보호 처리된 장치)를 쓴다

+ 어떤 물체든 전자장비가 많이 들어가면 방사선 방출, 치사량이 아닐 뿐

**[3] 우주방사선이 위험한 이유**

정답: 맥스웰 방정식으로 설명가능

전하를 띤 입자가 이동하면 전기장 발생

전기장의 변화는 자기장을, 자기장의 변화는 전기장을 발생
여기서, 전기장이 존재한다는 것은 전하가 이동한다 즉 전류가 흐른다
따라서 강력한 우주방사선에 의해 강한 전류가 유도 될 경우 장비 오작동을 일으킨다
항공분야의 안정성을 해결한 솔루션에는 'Radiation Hardened' 가 붙는다

**[4] C언어 프로그래밍 기본**

```
#include <stdio.h>
int main(void) {                    // main함수 반환형 void 주지 말 것
// your code goes here
        return 0;                    // 0 반환 시 정상종료
}
```

**[5] 리눅스 기본설정**

| | |
|---|---|
| sudo apt-get update | sw 업데이트 |
| sudo apt-get install vim | vim 설치 |

*. Vim
- vi 에디터에 설정을 더하여 편리하게 사용할 수 있도록 해주는 에디터
- 명령, 입력/편집모드가 있다
lang support 설치

**[6] vim 에디터 사용하기**

**1. 기본설정**

| | |
|---|---|
| set nu | line번호 띄우기 |
| set hl search | 찾는 문자 형광펜 설정 |
| sudo apt-get install build-essential | gcc프로그램 설치 |

**2. 코드 에디팅**
vi print_message.c (코드명)

**[명령모드]**
- a, i, A, I 입력시 입력모드 전환
    u: 명령취소, 되돌리기 (ctrl+z의 기능)
    R: 앞으로 가기 **?**
    dnd: n줄 지우기 **?**
    4x: 4글자 지우기
    yy: (1줄)복사
    yny: 커서 밑으로 n줄 복사 **?**
    p: (1줄)붙여넣기 - 여러번 반복 가능

%s/바꿀것/변경후/g: 치환명령

Ex) %s/Nihao/Hi/g: nihao를 hi로 치환

/test(찾는문자): "test"문자검색

n 아래로 / N 위로 찾기

$: 맨끝으로 이동하기

:숫자 (숫자)번째 행(컬럼)으로 이동

ctrl f 페이지 다운 / ctrl b 페이지 업

wq : write quit 저장하고 나가기

**[텍스트 편집/입력모드]**

- Insert표시로 알 수 있다
- ESC: [명령모드] 로 전환

A: 커서가 위치한 행의 맨 뒤에 입력

I: 맨 앞에

a: 현재 커서 다음 자리에 입력

i: 현재 커서 자리에 입력

**[7] 소스 디버깅** (-g 디버깅 옵션)

**1. 디버깅 심볼**

- 확장자는 pdb
- 디버깅에 필요한 정보들이 들어가 있다.

(소스코드 레벨에서 디버깅이 가능하도록 소스코드의 파일명이나 라인넘버 등이 들어갈 수 있음)

**2. -g옵션 있을 때와 없을 때 비교**

Ex) gcc -g -o print_msg print_message.c

gcc -o print print_message.c

gdb print (-g옵션 없을때)

no debugging symbol

gdb print_msg (-g옵션 있을때)

debugging

**[8] github 연동하고 디버깅하기**

sudo apt-get install git

git pull origin master (homework 디렉토리에서)

gcc -g -o debug func1.c

ls

gdb debug 프로그램 디버깅(심볼을 읽는다)

b main main의 break point

r Run 구동한다

| | |
|---|---|
| disas | 기계어 보기 (화살표 부분까지 프로그램 실행된 것) |
| b*(첫번째 줄) | 화살표를 첫번째 줄로 이동하여 여기에 Break Point를 걸어준다 |
| r | |
| disas | 첫번째 줄로 이동된 화살표 확인 가능 |

**[9] C언어의 메모리 구조(아키텍쳐, 레이아웃)**

1. 스택 지역변수
2. 힙 동적할당 malloc calloc
3. 데이터 전역변수, static 선언한 것 위치(초기화되지 않은 모든 것은 0으로 저장)
4. 텍스트 머신코드 위치

운영체제가 프로그램 실행 시 필요한 메모리 공간(지역변수, 전역변수 선언을 위해) 할당하는 메모리 공간은 크게 스택(Stack), 힙(Heap), 데이터(Data)영역이 있다.
프로그램 실행시마다 메인 메모리(RAM)에 할당한다.

**[10] 지역변수, 전역변수**

지역변수는 그 지역변수를 가진 함수만의 것이고, 함수 밖에서는 사용할 수 없다.
반면에 전역변수는 프로그램 어디에서든 사용할 수 있다. 단, 함수 안에서 전역변수 값을 읽는 것은 가능하나 수정하는 것은 금지된다
지역변수는 함수가 실행될 때마다 새로 만들어지고, 함수의 실행이 종료되면 삭제된다.
함수의 입력 통로와 출력 통로를 매개변수와 return 문으로 제한한다

**[11] 변수와 포인터**

변수 - 메모리에 존재하는 정보를 저장하는 공간
포인터 - 주소를 저장하는 공간

**1. 데이터 타입 (자료형)**

int, short, char, float, double, long double

**2. 변수선언 : 데이터타입 이름= 숫자;**

```
Int main(void){
int num1 = 3, num2 = 7;
float num3 = 7.7;
double num4 = 3.3;
printf("num1=%d, num3=%f, num4=%lf₩n", num1, num3, num4);

int res1;
double res2;
res1=num1*num3;          // float으로 계산 23.1, 저장은 int로 23
res2=num1*num2;          // 21
```

```
printf("res1=%d, res2=%lf₩n");
return 0;
}
```

**[12] printf() 함수**

```
printf("%s₩n",test); "문장, 문자열"
printf("%d₩n",10);
printf("%f₩n",7.7); 7.700000
printf("%.1f₩n",7.7); 7.7 소수점 첫째자리까지
printf("안녕하세요₩n");
```

**[13] 함수안에서 함수 호출하기**

```
int myfunc(int num, int num2){
        return num << 1;      // 비트연산자
                               // return num*2;
                              // return num+3;
        //반환형 float으로 한다 - retrun num+3.3 일때
}

float test(int a, int b) {
         printf("test함수 안에서 myfunc함수를 호출했다₩n");
         return myfunc(a,b);
}
```