파이썬 - HW5

임베디드스쿨1기

Lv1과정

2020. 08. 24

김인겸

1. 복습

1. inline함수 :

함수호출x(파이프라인안깨짐, rbp값과 복귀주소를 스택메모리에 쌓는과정이 사라짐)
컴파일러가 텍스트를 그대로 가져와서 실행 → 속도 up

텍스트가 길어지므로 실행파일의 크기가 커지는 단점이 있다.

마이크로소프트 비주얼 C++과 GCC와 같은 주류의 C++ 컴파일러들은 인라인 함수로 표시되지 않은 것이 있다고 할지라도 적절한 함수를 자동으로 인라인하는 옵션을 지원한다. - 위키백과

1. 복습

+ 메크로 함수 :
inline함수는 컴파일러가 처리하는 반면 메크로함수는 전처리기가 처리한다.
타입 구분이 없다.
단순 복제만 하는 기능이므로 괄호에 유의해야 한다.

```c
#include <stdio.h>

#define f(x) x*x+2*x+1

int main(void)
{
    printf("%d\n", f(2+1));

    return 0;
}
```

```
11
```

잘못된
사용

```c
#include <stdio.h>

#define f(x) ((x)*(x)+2*(x)+1)

int main(void)
{
    printf("%d\n", f(2+1));

    return 0;
}
```

```
16
```

올바른
사용

objdump활용 (objdump -d a.out)

명령어들도 결국엔 기계어 이다.

<+4>, <+5>의 의미 :
어셈블리어 명령어의 기계어 크기

16진수 → 2진수로 처리됨

컴파일러가 관리하는 섹션오프셋은
다이나믹링킹을 통해
가상메모리에 올라가고
맵핑을 통해
물리메모리에 올라간다.

```
0x000055555555515f <+0>:     endbr64
0x0000555555555163 <+4>:     push   %rbp
0x0000555555555164 <+5>:     mov    %rsp,%rbp
0x0000555555555167 <+8>:     sub    $0x10,%rsp
0x000055555555516b <+12>:    movl   $0x4,-0x8(%rbp)
0x0000555555555172 <+19>:    mov    -0x8(%rbp),%eax
0x0000555555555175 <+22>:    mov    %eax,%edi
0x0000555555555177 <+24>:    callq  0x555555555149 <test>
```

```
0000000001000 <_init>:
1000:       f3 0f 1e fa                 endbr64
1004:       48 83 ec 08                 sub    $0x8,%rsp
1008:       48 8b 05 d9 2f 00 00        mov    0x2fd9(%ri
100f:       48 85 c0                    test   %rax,%rax
1012:       74 02                       je     1016 <_ini
1014:       ff d0                       callq  *%rax
1016:       48 83 c4 08                 add    $0x8,%rsp
101a:       c3                          retq
```

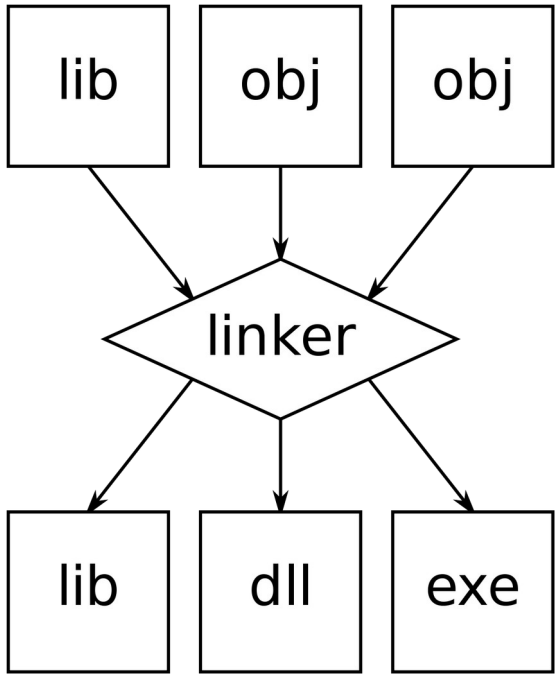
Static Linking 과 Dynamic Linking

1. 링커 : 컴파일러에 의해서 만들어진 1개 이상의 목적파일(obj)와 라이브러리에 포함된 코드들을 하나로 합쳐서 실행파일을 만들어줌
obj파일의 섹션오프셋을 가상메모리에 맵핑.(??)

2. Static Linking(정적 링킹) : 링킹할 때 외부 라이브러리 모듈을 실행파일에 직접 복사해서 넣는 방식
3개의 프로그램이 동일한 외부함수를 사용한다고 할 때 각각의 실행파일마다 똑같은 라이브러리가 포함됨.(중복된다는 의미)
(빠르지만 메모리를 잡아먹는다)

3. Dynamic Linking(동적 링킹) : 외부 라이브러리 모듈을 복사하는 방식이 아니라 주소만 가져오는 방식.
3개의 프로그램이 동일한 외부함수를 사용한다고 할 때 각각의 실행파일마다 주소만 가져오면 사용가능하므로 중복되지 않음
(매번 주소를 참조하므로 느리지만 메모리를 덜 잡아먹는다)

불일치에 대한 문제 : 만약 외부 라이브러리 모듈의 내용이 변경되거나 삭제될 시에 실행파일이 그 문제를 인식하지 못하는 경우가 발생한다.



paging(페이징)

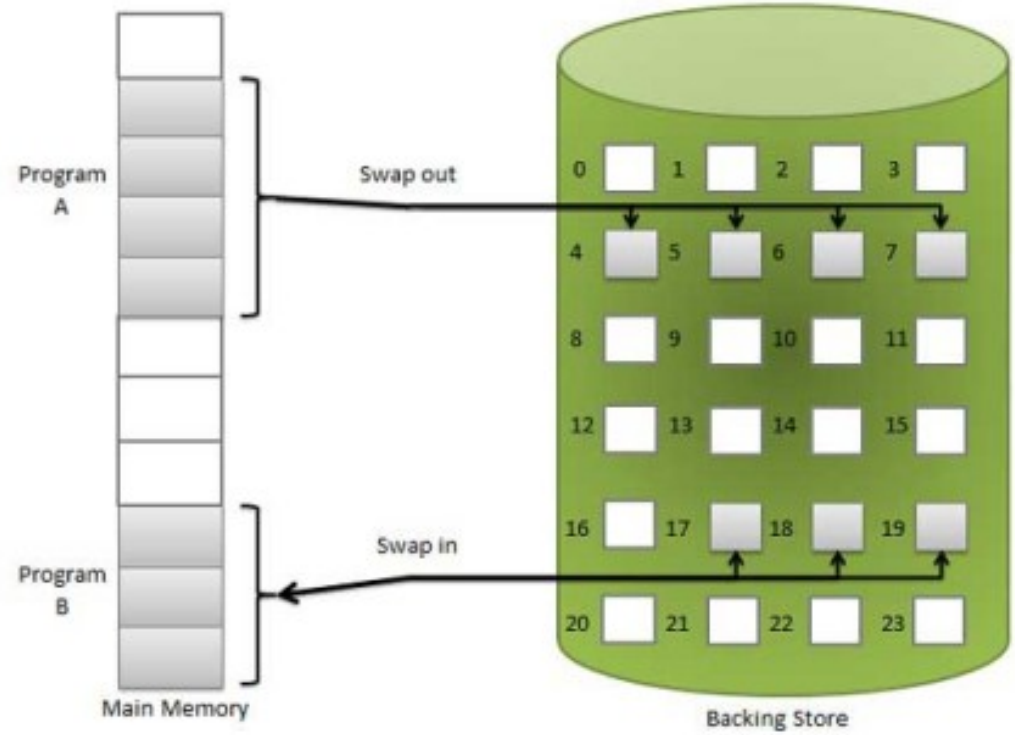
페이징 이란 : 가상기억장치(보조기억장치)에 저장된 프로그램을 주가억장치에 맵핑해서 사용하는 기법이다. 프로그램의 모든 내용을 주기억장치에 올리는 것이 아니라 필요한 일부분만 올렸다가 교체되는 방식이므로 작은 메모리로 큰 프로그램을 다룰 수 있고 여러 개의 프로그램을 같이 실행 시킬 수 있는 장점이 있다.(FIFO 방식)

Swapping : 페이지를 메모리에 swap in 하거나
swap out 하는 것

페이지 hit : 사용하려는 페이지가 주기억장치에 있음
페이지 폴트 : 사용하려는 페이지가 주기억장치에 없어서
가상기억장치로부터 페이지를 가져와야 됨

페이지 테이블 : 페이지가 물리메모리에 맵핑된 정보를 (페이지사상표) 포함(크기, 주소 등)

내부단편화 : 58KB 프로그램을 4K페이지로 나눴을 때
나머지 2K 부분을 내부 단편화 라고 한다.



스래싱 : 페이지폴트가 빈번해서 페이지를 swapping하는데 시간이 많이 걸려 성능이 떨어지는 현상.

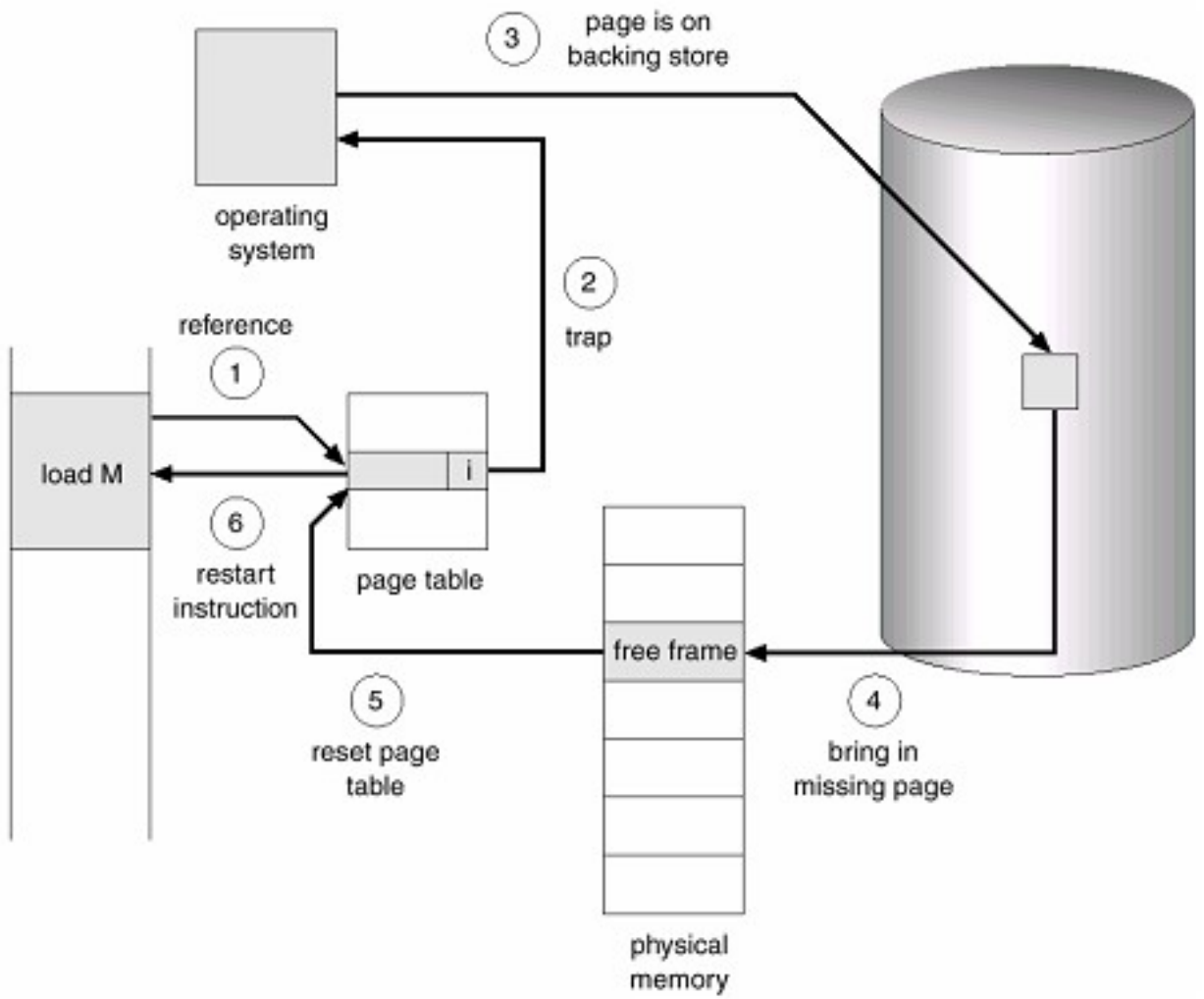
paging(페이징)

페이지폴트 처리과정

1. cpu가 명령을 함

2~4. 운영체제가 페이지테이블을 참초해서
물리메모리에 페이지를 맵핑

5. 페이지테이블에 변경내용을 기록



inline함수의 과도한 사용 → 프로그램의 크기 증가 → 페이지 개수 증가 → 페이지폴트 현상 빈번해짐
→ 스래싱 → 성능저하.

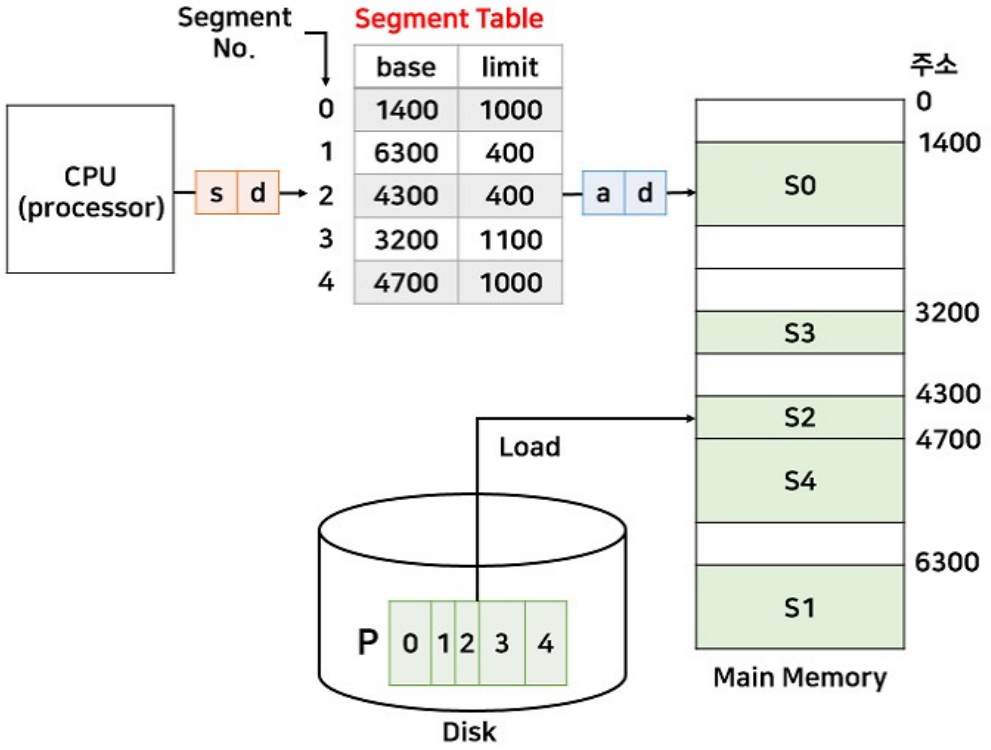
segametation(세그멘테이션)

페이징 기법은 크기가 고정이지만 세그멘테이션은 크기가 가변이다.

프로그램을 일정한크기로 자르는 것이 아니라 기능별로 잘라서 물리메모리에 프레임없이 순차적으로 쌓아가는 방식.

세그먼테이션을 또 페이지로 나누는 기법도 존재한다.

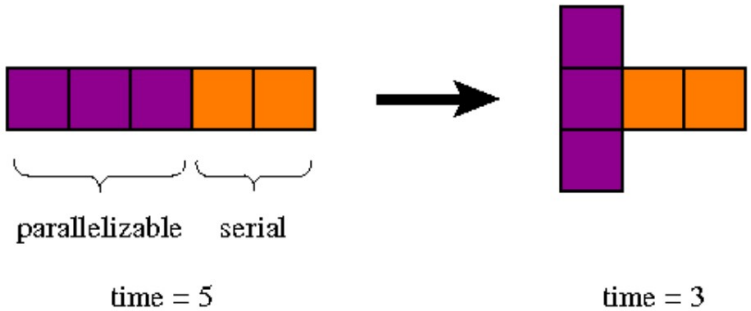
일단 알아만 두자~



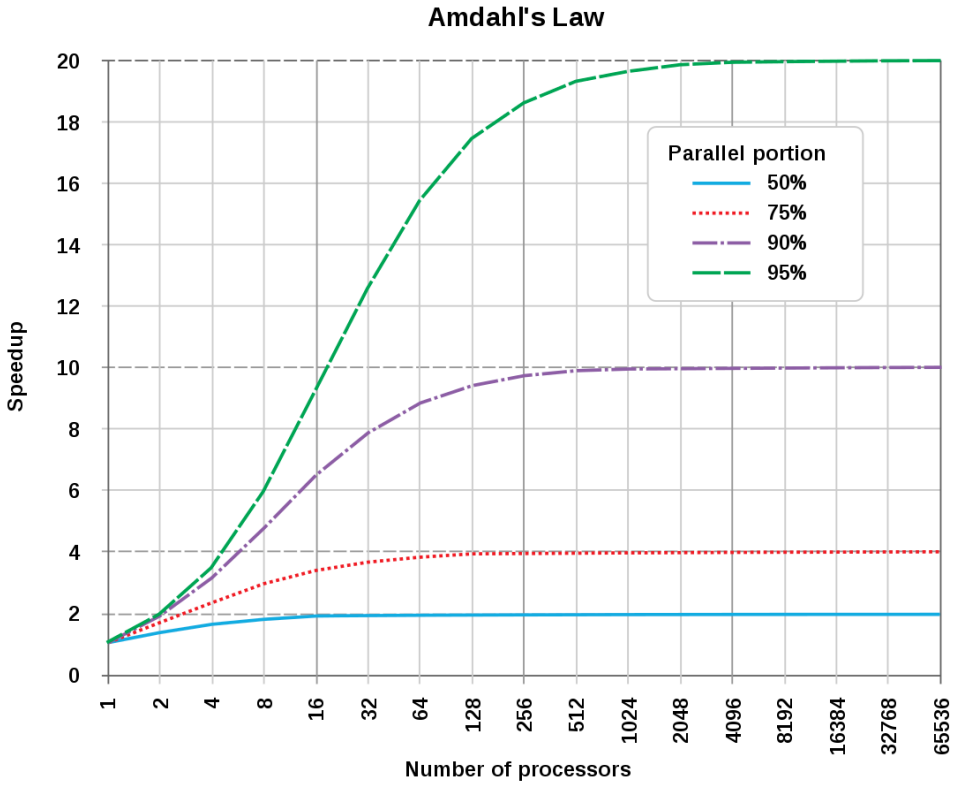
병렬 컴퓨팅

병렬 컴퓨팅이란 ? : 컴퓨터의 성능을 높이기 위해 cpu를 2개(듀얼코어), 4개(쿼드코어), 8개(옥타코어)를 이용하는 것

암달의 법칙 : 데이터를 처리하는데 순차적인 부분이 많으면 아무리 코어가 많아져도 성능을 향상시키는데 한계가 생긴다.



암달의 법칙



경쟁상태 : 둘 이상의 입력 또는 조작의 타이밍이나 순서 등이 결과값에 영향을 줄 수 있는 상태를 말한다.
예를들어) 2*2 + 2*3 계산에서 2*2와 2+2를 동시에 처리히 전혀다른 결과가 나오는것.

5. 범용성 있는 문제 만들기

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <stdio_ext.h>

int random_num(void);
int input(int count);
void game(int user_num, int answer, int count);

int main(void)
{
    int user_num;
    int count = 0;
    srand((unsigned) time(NULL));

    int answer = random_num();

    //게임 시작
    while(count != 20){
        user_num = input(count);
        count ++;
        game(user_num, answer, count);
    }

    return 0;
}
```

```c
int random_num(void)
{
    return rand() % 1000001;
}

int input(int count)
{
    int user_num;

    printf("숫자를 입력하세요(남은횟수 : %d회) : ", 20 - count);

    while(1){

        if(scanf("%d",&user_num) == 0){
            printf("잘못입력하셨습니다. 숫자를 입력해주세요\n");
            __fpurge(stdin);
        }
        else if(user_num > 1000000 || user_num < 0)
            printf("0~1,000,000까지의 숫자를 입력해주세요\n");
        else
            break;

    }
    return user_num;
}
```

5. 범용성 있는 문제 만들기

```
5  void game(int user_num, int answer, int count)
6  {
7      if(user_num == answer){
8          printf("정답입니다\n");
9      }
10     else if(count != 19){
11         printf("틀렸습니다\n");
12         if(user_num > answer){
13             printf("DOWN\n");
14         }
15         else{
16             printf("UP\n");
17         }
18     }
19     else{
20         printf("20번의 기회를 모두 소진하였습니다 게임을 종료합니다.\n정답은 %d 였습니다", answer);
21     }
22 }
```

스무고개 문제를 함수로 쪼개서 만들어봤습니다.

scanf함수 사용법과 예외처리

scanf함수의 반환값

```
 4 int main(void)
 5 {
 6     int num1, num2;
 7
 8     int result = scanf("%d %d", &num1, &num2);
 9
10     printf("%d\n", result);
11
12     return 0;
13 }
```

```
20 20
2
(base)
20 a
1
(base)
a a
0
```

형식지정지 %d와 맞는 정수를 입력할 경우 scanf함수는 1를 반환값으로 내보낸다.
(올바르게 처리되었다는 뜻)

형식지정자 %d와 맞지 않는 것을 입력할 경우 scanf함수는 0을 반환값으로 내보낸다.
(올바르게 처리되지 못했다는 뜻)
표준입력버퍼에는 잘못 입력된 a가 그대로 남아있기 때문에 반복문을 사용해서 scanf함수를 다시 사용하려면 표준입력버퍼에 남아 있는 값을 지워주는 과정이 필요한다
→ fflush(stdin), 리눅스환경에서는 __fpurge(stdin); 사용 (stdio_ext.h 라이브러리 포함해야함)

기타 내용 정리

문자열을 int형 정수로 반환해주는 함수 : atoi() (stdlib.h포함)

문자열이 숫자인지 문자인지 판단하는 함수 : isdigit(), isalpha(), isalnum() (ctype.h포함)

질문

1. 페이징과 요구페이징은 다른 건가요? 따로 검색을 해도 똑같은 내용만 나오길래 두 개를 그냥 혼용해서 사용하는 건지 궁금합니다.

2. objdump해서 보는 내용들이 obj파일인 건가요?

```
000000001000 <_init>:
1000:       f3 0f 1e fa             endbr64
1004:       48 83 ec 08             sub    $0x8,%rsp
1008:       48 8b 05 d9 2f 00 00    mov    0x2fd9(%ri
100f:       48 85 c0                test   %rax,%rax
1012:       74 02                   je     1016 <_ini
1014:       ff d0                   callq  *%rax
1016:       48 83 c4 08             add    $0x8,%rsp
101a:       c3                      retq
```

3. 여러 개의 c파일을 1개의 실행파일로 만

질문

완벽하게 예외처리 하는게 어렵습니다.

1. scanf의 반환값이 1인지 0인지에 따라서 구현해봤는데 한계가 있는 것 같습니다
예를 들어 1abc를 입력하면 scanf가 1만 먼저 인식하고 다음에 다시 실행됐을 때 abc를 인식하는 것 같습니다.

2. 문자열을 입력받고 isdigit()이 참이면 atoi()를 이용해서 int값을 반환값으로 내보내는 함수를 만들었는데 컴파일은 됐지만 입력할때마다 '세그멘테이션 오류'가 뜹니다.. 이유를 잘 모르겠습니다

```c
int input(int count)
{
    char user_num[10];

    printf("숫자를 입력하세요(남은횟수 : %d회) : ", 20 - count);

    while(1){
        __fpurge(stdin);
        scanf("%s",user_num);

        if(isdigit(user_num)){
            break;
        }
        else if(atoi(user_num) > 1000000 || atoi(user_num) < 0)
            printf("숫자의 범위를 벗어났습니다\n");
        else{
            printf("숫자를 입력해주세요\n");
        }
```

```
(base) ingyeomkim@ingyeomkim-Inspiron-759
숫자를 입력하세요(남은횟수 : 20회) : 20
```