2018.02.27 - 5일차 수업

---

* sagmentation Fault가 나는 이유

우리가 기계어를 보면서 살펴봤던 주소값들이 실제로는 전부 가짜 주소이다.

이 주소값은 엄밀하게 가상 메모리 주소에 해당하고

운영체제의 paging메커니즘을 통해서 실제 물리 메모리의 주소로 변환된다. (윈도우, 맥(유닉스)도 가상 메모리 개념을 사용한다)

가상 메모리는 리눅스의 경우

32비트 버전과 64비트 버전으로 나뉜다.

32비트 시스템은 2^32 = 4GB 의 가상 메모리 공간을 가지고 여기서 1:3으로 1을 커널이 3을 유저가 가져간다.

1의 커널은 시스템(HW, CPU, SW 각종 주요 자원들)에

관련된 중요한 함수 루틴과 정보들을 관리하게 된다.

3은 사용자들이 사용하는 정보들로 문제가 생겨도

그다지 치명적이지 않은 정보들로 구성된다.

64비트 시스템은 1:1로 2^63승에 해당하는 가상메모리를 각각 가진다.

문제는 변수를 초기화하지 않았을 경우 가지게 되는 쓰레기값이 0xCCCCCCCCCCCC...CCC로 구성된다.

32비트의 경우에도 1:3 경계인 0xC000000000을 넘어가게된다. 64비트의 경우엔 시작이 C이므로 이미 1:1경계를 한참 넘어간다

그러므로 접근하면 안되는 메모리 영역에 접근하였기에

엄밀하게는 Page Fault(물리 메모리 할당되지 않음)가

발생하게 되고 원래는 Interrupt가 발생해서

Kernel이 Page Handler(페이지 제어기)가 동작해서

가상 메모리에 대한 Paging처리를 해주고

실제 물리 메모리를 할당해주는데

문제는 이것이 User쪽에서 들어온 요청이므로

Kernel 쪽에서 강제로 기각해버리면서

Segmentation Fault가 발생하는 것이다.

실제 Kernel 쪽에서 들어온 요청일 경우에는

위의 메커니즘에 따라서 물리 메모리를 할당해주게 된다.

---

* 배열 포인터

int (*p)[2] -> int형의 [2]배 만큼의 크기씩 이동함 (int (*)[2] p 의 의미)

* 이중포인터 예제

```
#include <stdio.h>
int main()
{
        int num1 = 3, num2 = 7;
        int *temp = NULL;
        int *num1_p = &num1;
        int *num2_p = &num2;
        int **num_p_p = &num1_p;

        printf("*num1_p = %d₩n", *num1_p);
        printf("*num2_p = %d₩n", *num2_p);
```

변수	num1	num2	
데이터	3	7	
포인터	num1_p	num2_p	temp
데이터	&num1	&num2	NULL
이중포인터	num_p_p		
데이터	&num1_p		

```
	temp = *num_p_p;
	*num_p_p = num2_p;
	num2_p = temp;

	printf("*num1_p = %d₩n", *num1_p);
	printf("*num2_p = %d₩n", *num2_p);
```

변수	num1	num2	
데이터	3	7	
포인터	num1_p	num2_p	temp
데이터	&num2	&num1	&num1
이중포인터	num_p_p		
데이터	&num1_p		

```
	return 0;
}
```

배열에 문자열을 입력 받고, 각 배열 요소가 짝수인 경우만을 출력하는 함수를 작성하라

```
#include <stdio.h>
#define STR_MAX_LEN 100
int main()
{
	char arr[STR_MAX_LEN];
	printf("문자열 입력(%d자 이내): ",
STR_MAX_LEN);
	scanf("%s", arr);
	int strLen = 0;
	while (arr[strLen] != NULL && strLen < 100)
		strLen++;
	for (int i = 0; i < (strLen + 1) / 2; i++)
		printf("%c", arr[i * 2]);
	return 0;
}
```

아래와 같은 숫자들이 배열에 들어 있다고 가정한다.

	3, 77, 10, 7, 4, 9, 1, 8, 21, 33

이 요소들을 배열에 거꾸로 집어넣어보자

```
#include <stdio.h>
void change(int* A, int* B)
{
	int temp = *A;
	*A = *B;
	*B = temp;
}
int main()
{
	int arr[] = { 3, 77, 10, 7, 4, 9, 1, 8, 21, 33};
	int arrLen = sizeof(arr) / sizeof(int);
	for (int i = 0; i < (arrLen+1) / 2; i++)
	{
		change(&arr[i], &arr[(arrLen - 1) - i]);
	}
	for (int i = 0; i < arrLen; i++)
		printf("arr[%d] = %d₩n", i, arr[i]);
	return 0;
}
```

위의 숫자 	3, 77, 10, 7, 4, 9, 1, 8, 21, 33

에서 홀수 번째 요소의 합과 짝수 번째 요소의 합을 곱하시오

```
#include <stdio.h>
int main()
{
	int arr[] = { 3, 77, 10, 7, 4, 9, 1, 8, 21, 33};
	int arrLen = sizeof(arr) / sizeof(int);
	int sum1 = 0, sum2 = 0;
	for (int i = 0; i < arrLen; i++)
	{
		if (i % 2 == 0)
			sum1 += arr[i];
		else
			sum2 += arr[i];
	}
	int result = sum1*sum2;
	printf("result = %d₩n", result);
	return 0;
}
```

행렬의 곱셈, 덧셈, 나눗셈, 뺄셈에 대해 조사하시오

숫자를 예로 들어서 계산도 해보시오

기본적으로 행렬은 크기(행과 열의 수, size)가 같은 행렬끼리 덧셈과 뺄셈이 가능하고 같은 위치의 성분끼리 연산을 한다. 곱셈은 곱하는 앞 행렬의 행 수와 뒤 행렬의 열의 수가 같아야 곱셈이 가능하다. 나눗셈의 경우 비슷한 개념으로 역행렬이 있는데 역행렬은 특정 조건에서만 존재한다. 먼저 역행렬이 존재하기 위해서는 정방행렬이어야 하고 행렬식 값이 0이 아니어야 한다.

- 행렬 사칙연산 관련 정리 조사자료 링크

http://contents.kocw.net/KOCW/document/2015/hankyong/kwaknotae/7.pdf

---

우리는 예제에서 주소값을 교환하여 값을 변경하는 것을 해보았다.

그렇다면 변수 3 개를 놓고, 이것에 대해서 무한 Loop 를 돌면서 저글링을 해보자!

```
#include <stdio.h>
void change(int* A, int* B, int* C)
{
	int temp = *A;
	*A = *B;
	*B = *C;
	*C = temp;
}
int main()
{
	int a = 1, b = 2, c = 3;
	for (int i = 0; i < 20; i++)
	{
		change(&a, &b, &c);
		printf("%d  %d  %d₩n", a, b, c);
	}
	return 0;
}
```

총 7개의 통장을 만들어서 100만원 단위로 최대 500만원까지 입금하였다.

이자율이 연 4%라고 할 때, 3년 후 각각의 총액을 구하시오

```
#include <stdio.h>
int main()
{
	double bankbook[7] =
{ 100,200,300,400,500 };
	int year = 3;
	for (int i = 0; i < year; i++)
	{
		printf("₩n₩n %d년 후₩n", i+1);
		for (int j = 0; j < 7; j++)
		{
			bankbook[j] *= 1.04;
			printf("bankbook[%d]
= %.2lf₩n", j, bankbook[j]);
		}
	}
	return 0;
}
```

---

char str1[] = “devil”, char str2 = “angel”일 때 2개의 문자열을 서로 바꿔보라

```
#include <stdio.h>
int main()
{
	char str1[] = "devil";
	char str2[] = "angel";
	char temp;
	for (int i = 0; i < 5; i++)
	{
		temp = str1[i];
		str1[i] = str2[i];
		str2[i] = temp;
	}
	printf(" str1 = %s ₩n str2 = %s ₩n", str1,
str2);
	return 0;
}
```

---

3 by 3 행렬의 곱셈을 계산할 수 있는 프로그램을

만드시오.

```
#include <stdio.h>
int get_3x3(int N[3][3]);
void main()
{
	int x, y, i, j;
	int A[3][3];
	get_3x3(A);

	int B[3][3];
	get_3x3(B);

	int C[3][3];
	for (x = 0; x<3; x++)
		for (y = 0; y < 3; y++)
		{
			C[x][y] = 0;
			for (i = 0; i < 3; i++)
			{
			    C[x][y] += A[x][i] * B[i][y];
			}
		}
	printf("행렬A*B의 결과값₩n");
	for (i = 0; i < 3; i++)
	{
		for (j = 0; j < 3; j++)
			printf("%d₩t", C[i][j]);
		printf("₩n");
	}
}
int get_3x3(int N[3][3])
{
	int i, j, n;
	printf("3x3행렬 입력₩n");
	for (i = 0; i<3; i++)
		for (j = 0; j < 3; j++)
		{
		    printf("%d행 %d열 입력₩n", i, j);
		    scanf_s("%d", &n);
		    N[i][j] = n;
		}
	printf("입력한 행렬값₩n");
	for (i = 0; i < 3; i++)
	{
		for (j = 0; j < 3; j++)
			printf("%d₩t", N[i][j]);
		printf("₩n");
	}
	printf("₩n");
	return 0;
}
```

---

int arr[3][3] = {{1, 13, 2}, {7, 3, 5}, {2, 9, 11}}에서 최대값을 찾는 함수를 만드시오

```
#include <stdio.h>
int main()
{
	int arr[3][3] = { { 1, 13, 2 },{ 7, 3, 5 },{ 2, 9, 11 } };
	int max = 0;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			if (max < arr[i][j])
				max = arr[i][j];
		}
	}
	printf("max = %d₩n", max);
	return 0;
}
```

삼각형의 넓이 구하는 문제

case 1) 밑변, 높이

case 2) 밑변, 밑변과 다른 변이 이루는 각도

```
#include <stdio.h>
#include <math.h>
double calTriArea1(double width, double hight)
{
	return width*hight / 2;
}
double calTriArea2(double length1, double length2, double deg)
{
```

```
        return length1*length2*sin(deg*3.14 / 180) / 2;
}
int main()
{
        printf("area1 = %lf₩n", calTriArea1(10, 10));
        printf("area2 = %lf₩n", calTriArea2(10, 10, 90));
          return 0;
}
```