TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018.02.27
5 일차
강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – 신민철
akrn33@naver.com

fib , fact 재귀호출 없이 사용하는법
배열의 선언방법
배열은 []안에 길이 값이 들어간다 선언해주지 않으면 {}안에 들어있는 값의 길이만큼 들어간다.
길이가 7 인데 {}안의 값이 3 개밖에 없으면 나머지는 0 으로 초기화 해준다.

string 은 한번 써주면 바꾸지 못하는데
문자배열은 바꿀수있음

문자배열은 생성해줄때 글자수보다 2 개정도 넣어주는게 좋음 끝에 \0 이 들어가야되기때문에
3d 메모리
mux 는 1 개의 입력을 받아서 여러개의 값을 출력하는것이고
demux 는 여러개의 입력을 받아서 1 개의 값을 출력하는 것.
게임을 만들게 되면 4 4 행렬을 많이 쓴다 단위행렬
잘못된 메모리에 접근할때 나온다 segmetation default 특히 커널영역에 접근할 때.
배열에 접근할때 j 에 값이 이상하

배열의이름은 메인주소
%p 주소값 뿌릴때 쓴다.

배열의이름은 주소와 같다. 따라서
int arr[3] = {1, 2, 3};
int *p =arr;하면
arr 의 이름을 *p 에 대입해도 arr 의 주소가 주소를 저장할수있는 변수인 *p 에 주소가 저장된다.
그렇다면 p[0]은 arr 의 첫번째 값을 가리키고있는것이다.
2 차원배열은 1 차원 배열이 배열 형태로 모인 것.

배열이 순차적으로 배치되어있음을 입증하시오
그런문제에 이런것들을 코딩하시면 됩니다.

-->>>
#include<stdio.h>

int main(void)

```
{
	int arr[3][4];

	printf("arr address = %p\n",arr);
	printf("arr[0] address = %p\n",arr[0]);
	printf("arr[1] address = %p\n",arr[1]);
	printf("arr[2] address = %p\n",arr[2]);
	return 0;
}
```

==> arr[0]에는 arr[0][0], arr[0][1], arr[0][2], arr[0][3]4 개가 들어가있기때문에
int 형 배열이므로 4*4 는 16 바이트 이다.
그래서 출력결과가 1 씩 늘어날 때 마다 16 바이트씩 증가한다.

출력결과>>
arr address = 0x7ffc46dceb30
arr[0] address = 0x7ffc46dceb30
arr[1] address = 0x7ffc46dceb40
arr[2] address = 0x7ffc46dceb50

```
#include<stdio.h>

int main(void)
{
	int arr[3][4];

	printf("arr address = %lu\n",sizeof(arr));
	printf("arr[0] size = %lu\n",sizeof(arr[0][0]));
	printf("arr[1] size = %lu\n",sizeof(arr[1]));
	printf("arr[2] size = %lu\n",sizeof(arr[2]));

	return 0;
}
```

2 중배열의 크기를 출력하는 코드인데, arr[0][0] 은 작은단위 이므로 크기가 4 바이트가 출력이되는데
arr[1]은 1,0 1,1 1,2 1,3 을 다 포함하고있기 때문에 16 바이트로 나온다

주소값을 알고있으면, 경계가 사라진다
함수어디서든 변수에대한 제어를 실행할 수 있기 때문에 포인터를 사용한다.

```
#include<stdio.h>
void add_arr(int * arr)
{
	int i;
	for(i = 0; i < 3; i++)
	{
		arr[i] +=7;
	}
}

void print_arr(int * arr)
{
	int i;

	for(i = 0; i < 3; i++)
	{
		printf("arr[%d] = %d\n", i, arr[i]);
	}
}

int main(void)
{
	int arr[3] = {1, 2, 3};
	add_arr(arr);
	print_arr(arr);

	return 0;
}
```

주소
조금 더 엄밀하게 주소를 저장할 수 있는 변수
무언가를 가르키는 녀석
Pointer 의 크기는 HW 가 몇 bit 를 지원하느냐에 따른

그런데 왜 hw 에 따라 달라질까?

최상위 메모리 주소 = 0xffffffff 32bit

배열의 이름 ==주소

포인터는 어떻게 선언하는가?
가르키고 싶은 녀석의 자료형을 적음
주소값을 저장할 변수명(이름)을 적음
변수명 앞에 '*'을 붙임
자료형 없이 변수명 앞에 '*'만 붙은 경우
해당 변수가 가르키는 것의 값을 의미함

rtos 이란 realtimeoperatingsystem 의 약자다
실시간운영체제?

segmentation fault 가 나는 이유?

우리가 기계어를 보면서 살펴봤던 주소값들이
사실은 전부 가짜 주소라고 말했었다.
이 주소값은 엄미라게 가상 메모리 주소에 해당하고
운영체제의 Paging 메커니즘을 통해서
실제 물리 메모리의 주소로 변환된다.
(윈도우도 가상메모리 개념을 베껴서 사용한다)
그렇다면 당연히 맥(유닉스)도 쓴다는 것을 알 수 있을 것이다.

가상 메모리는 리눅스의 경우
32 비트 버전과 64 비트 버전이 나뉜다

32 비트 시스템은 2^32 = 4GB 의 가상 메모리 공간을 가짐
여기서 1:3 으로 1 을 커널이 3 을 유저가 가져간다.
1 은 시스템(HW, CPU, SW 각종 주요 자원들)에 관련된 중요한 함수 루틴과 정보들을 관리하게 된다.

3 은 사용자들이 사용하는 정보들로
문제가 생겨도 그다지 치명적이지 않은 정보들로 구성됨
64 비트 시스템은 1:1 로 2^63 승에 해당하는 가상메모리를 각각 가진다.
문제는 변수를 초기화 하지 않았을 경우 가지게 되는 쓰레기 값이 0xCCCCCCCC...CC 로 구성됨이다.
32 비트의 경우에도 1:3 경계인 0xC0000000000 을 넘어가게됨
64 비트의 경우엔 시작이 C 이므로
이미 1:1 경계를 한참 넘어감

그러므로 접근하면 안되는 메모리 영역에 접근하였기에
엄밀하게는 Page Fault(물리 메모리 할당되지 않음)가
발생하게 되고 원래는 Interrupt 가 발생해서
Kernel 이 Page Handler(페이지 제어기)가 동작해서
가상 메모리에 대한 Paging 처리를 해주고
실제 물리 메모리를 할당해주는데 Kernel 쪽에서 강제로 기각해버리면서
Segmentation Fault 가 발생하는 것이다.
실제 Kernel 쪽에서 들어온 요청일 경우에는
위의 메커니즘에 따라서 물리 메모리를 할당해주게 된다.

```
#include<stdio.h>

int main(void)
{
	int i =0;
	int * ptr;// = &i;

	printf("ptr = %p\n",ptr);
	printf("ptr value = %d\n",*ptr);

	*ptr = 27;

	printf("ptr value = %d\n",*ptr);
	return 0;
}

#include<stdio.h>
```

```
int main(void)
{
	int * ptr = NULL;

	printf("ptr = %p\n",ptr);
	printf("ptr value = %d\n",*ptr);

	return 0;
}
```

==>segmentation default 가 나는 코드들이다.

실행시 이러한 결과가 나온다.

```
ptr = (nil)
Segmentation fault (core dumped)
```

포인터 변수를 쓰지 않고 가리키는 방법
==>

```
#include<stdio.h>

int main(void)
{
	int num = 3;
	*(&num) +=30;
	printf("num = %d\n",num);
	return 0;
}
```

Pointer to Pointer

Pointer 에 대한 Pointer 는
Pointer 도 Stack 에 할당되는 지역변수
즉, Pointer 에 대한 주소값도 존재함
얻는법은 동일함('&'를 이용)
그 주소값은 Pointer 에 대한 Pointer 로'*'
코드를 그림으로 바꾸는거 그림을 코드를 바꾸는법을 할수있어야한다.

별하나당 주소하나.
별 두개가 붙으면 주소의 주소를 받겠다.
그래서

이중포인터를 활용하면 재귀호출을 안하고 트리(자료구조)를 구현할 수있음.

```
#include<stdio.h>
int main(void)
{
	int num1 = 3, num2 = 7;// 변수를 선언하고 값을 초기화.
	int *temp = NULL;	//*temp 를 아무것도 가리키지 않게 초기화.
	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);

	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);
	return 0;
}
```

배열포인터와 포인터배열

```
#include<stdio.h>
int main(void){
	int i,j,n1,n2,n3;
	int a[2][3] = {{10,20,25},{30,40,45}};
	int *arr_ptr[3] = {&n1, &n2, &n3};
	int (*p)[4] = a;
	 //포인터변수 p 의 4 번째 인덱스에 a 의 4 번째 인덱스값을 넣음
```

```
        for(i = 0; i < 3; i++)
        *arr_ptr[i] = i;

        for(i = 0; i< 3; i++)
        printf("n%d = %d\n",i,*arr_ptr[i]);

        for(i = 0; i<2; i++)
        printf("p[%d] = %d\n",i,*p[i]);

        return 0;
}
```

==>포인터와 배열을 선언해서 값을 호출하는 문장

행렬의 덧셈 뺄셈 곱셈 나눗셈

(5 3) + (2 1) = (7 4)
(4 2) (4 7) (8 9) → 덧셈

(5 3) - (2 1) = (3 2)
(4 2) (4 7) (0-5) → 뺄셈

A = (a11 a12) B = (b11 b12)
(a21 a22), (b21 b22) 이면
AB = (a11*b11+a12b21 a11b12+a12b22)
(a21b11+a22b21 a21b12+a22b22)
이다. 적용하면
(5 3) * (2 1) = (22 26)
(4 2) (4 7) = (16 18) → 곱셈

A = (a1 a2)
(a3 a4)

A-1= 1 (d -b)
a1a4-a2a3 (-c a)

(5 3) 역행렬 = 1 (2 -3) =(-1 1.5)
(4 2) - 2 * (-4 5) (2 -2.5)→ 나눗셈(역행렬)