# *Xilinx Zynq FPGA, TI DSP,MCU기반의 프로그래밍 및 회로 설계 전문가 과정*


**강사 - Innov (이상훈)**

**gcccompil3r@gmail.com**

**학생 - 이유성**

**dbtjd1102@naver.com**

index[0]배열

```
#include<stdio.h>
#include<stdlib.h>

typedef struct
{
	int score;
	char name[20];
}ST;

typedef struct
{
	int count;
	char name[20];
	int score[0];  // 인덱스 0 ->실제로 배열처럼 사용할 수 있다.  ..주소값만 있다.(포인터가
맞다)
		// 형체가 없다.(존재 x)..그런데 값을 넣고있다.-> 실체가 있다 ..

}FLEX;

int main(void)
{
	int i;
	FLEX *p = (FLEX *)malloc(4096); // 메모리를 4096으로 사용하겠다  메모리에 malloc을
					//많이하면
		//왜냐면 해제하는 시간 할당받는 시간 길어진다.(속도느려짐)->한번에 크게
		//잡고 배열처럼 써라.
	// 서버한번에 크게 받아놓으면 터널로 진입하는 시간이 없어,할당 받는시간 줄어듬.
	//p-> score[0]=100;
	//p-> score[1]=200;

	for(i= 0 ; i<100 ; i++){     //10000까진 오류안뜨는데 10만은 오류뜸.(sigmentation fault)
				// 다른 메모리 영역 과 겹쳐서 오류가 뜰 수 있다.(주소)
				// - 이 구조체의 끝은 어딘가?(score는 구조체의 끝이자 시작)
	p->score[i] = i+1;


	printf("%d \n",p->score[i]);

	printf("%p\n",&p->score[i]);
	}
	printf("%lu\n",sizeof(FLEX));       //플렉스의 사이즈는 24가 나옴
	return 0;
}
```

os lock메커니즘에서

-semaphore

대기열이 있어 프로세스 여러개 처리 가능.(대규모 코드)
(wait queue로 빠진다 -> context switching을 한다는 소리)-(메모리로 올리고 보관)
-> 비용이 많이 든다.
lock이 풀릴 때까지 다른 프로세스들은 접근이 불가능.(우회한다.)
프로세스들을 다 처리해야하니까 성능을 희생하더라도 데이터를 안정적
처리하겠다.

-semaphore하면 빠질 수 없는게
critical section(양계 입력)어디서 치명타 터질까?
:여러 task들이 동시에 data에 접근해서 값이 꼬일 수 있는 구간 =>lock걸어줌.

-spinlock

CPU를 지속적으로 잡고있다(Polling한다) (단순하고 간단한 코드)
여러개 처리 x , 굳이 context switching하는 것보다 빨리 끝내는게 낫다.
lock이 풀릴 때까지 뺑뺑이 돈다(한자리만 계속 똑똑똑똑..)

운영체제의 메모리 공간에 접근하는 것은 원천적으로 불가능

fork() ->독립적 o  종속적x

프로세스 - 독립 ->병렬적 실행은 contextswitching(제어권을 서로 넘겨줌 mov도중 제어권 넘어가면 순서대로 동작 x
프로세스->task_struct생성

thread(쓰레드) - 종속->완전히 메모리 공유 /stack,heap 공유해도 별 의미x
전역변수…

쓰레드->task_struct생성
=>프로세스가 지니는 제한(독립적인 메모리공간 , 통신하기위해 IPC가 필요)를 해결

semaphore 코드분석이 덜 되었습니다.

shared memory

send.c

//shared 메모리 : 메모리 공유. 어떤 메모리 공유? 페이지 공유 애는 진짜 물리메모리를 공유한다는 뜻. 물리단위 최소  페이징프레임  4kb

```
#include "shm.h"

int main(void)
{
	int mid;
	SHM_t *p;

	mid = OpenSHM(0x888);

	p = GetPtrSHM(mid);

	getchar();
	strcpy(p->name,"아무개");
	p->score = 93;

	FreePtrSHM(p);

	return 0;
}
```

recv.c

```
#include "shm.h"

int main(void)
{
	int mid;
	SHM_t *p;

	mid = CreateSHM(0x888);//쉐어드 메모리 생성

	p = GetPtrSHM(mid);//메모리 아이디값 얻어옴

	getchar();
	printf("이름 : [%s], 점수 : [%d]\n",p->name,p->score);

	FreePtrSHM(p);

	return 0;
}
```

shmlib.c

```
//shared memory

#include "shm.h"

int CreateSHM(long key)  //ipc를써야 공유 가능.
{
	return shmget(key,sizeof(SHM_t), IPC_CREAT | 0777);//키값 세팅 쉐어드 메모리를
얻습니다.
                                                //페이지 프레임자체를 얻는다.
}

int OpenSHM(long key)
{
	return shmget(key,sizeof(SHM_t),0);
}

SHM_t *GetPtrSHM(int shmid) // id값가지고 at장소 찾는다
{
	return (SHM_t *)shmat(shmid,(char *)0,0); // at장소 0번지부터 찾겠다는 소리공유메모리의
물
리 주소를 얻어옴
}

int FreePtrSHM(SHM_t *shmptr)
{
	return shmdt((char *)shmptr);
}


shm.h

#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>

typedef struct
{
	char name[20];
	int score;
}SHM_t;

int CreateSHM(long key);
int OpenSHM(long key);
SHM_t *GetPtrSHM(int shmid);
int FreePtrSHM(SHM_t *shmptr);
```

//컴파일 방법

// gcc -o send shmlib.c send.c
// gcc -o recv shmlib.c recv.c

//터미널 두 개 띄우고 ./send 이후에 ./recv

결과

yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/3.6st$ ./recv

이름 : [아무개], 점수 : [93]

-shared memory(공유메모리)

한 한개의 메모리를 여러 장치(프로세서)가 공동으로 사용하는 형태의 기억장치 또는 다중처리 시스템을 사용할 때 여러 프로세서가 하나의 기억장치를 공유하여 사용하는 것을 의미
실제 물리 메모리를 공유한다.

send에서는 물리 메모리에 write를 하는 것이고 recv에서는 물리 메모리에서 read를 하는 역할을 한다.
IPC통신을 위해 SHM은 중요함.
프로세스간 정보를 공유하기 위해 반드시 사용해야 한다
이 방식으로 페이지프레임 아이디 값을 가져와 물리 메모리 주소에 직접 값을 넣을 수 있다.

IPC
프로세스 간의 정보 공유를 하기 위해 사용.

A프로세스와 B프로세스가 있다고 가정,
A프로세스와 B프로세스는 서로 정보를 공유 할 수 없다
하지만 IPC를 사용하면 shared memory를 통하여 프로세스간의 정보들을 공유할 수 있게 된다.
메커니즘으론 메모리 상에 특정 공간을 잡아 놓고 해당공간에 서로 접근할 수 있는 권한을 만든다.
그리고 그 공간에 정보들을 넣으면 공유가 되어 접근 권한을 가진 프로세스들이 메모리에 접근 가능하며 읽고 쓰고 할 수 있게 된다.
(shared memory는 총무님 참고..)