Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정
#25

강사:Innova Lee(이 상훈)
학생: 김시윤

수업내용 복습

---adv non Idx---

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

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

typedef struct
{
	int count;
	char name[20];
	int score[0];
}FLEX;

FLEX *print(FLEX *p)
{
	int i;
	for(i=0;i<=10000;i++)
	{
	p->score[i]=i;
	printf("score[%d]=%d\n ",i,p->score[i]);
	}
	return p;
}

void prr(FLEX *p)
{
	 int i;
	for(i=0;i<=10000;i++)
	{
	p->score[i]=i;
	printf("score[%d]=%d\n ",i,p->score[i]);
	}
}

int main(void)
{
	FLEX *p=(FLEX *)malloc(4096);

//메모리를 4096 으로 사용하겠다. FLEX 사이즈는 카운트 4 바이트 네임 20 바이트 저것들은 주소값이다.
	//p->score[0];
//스코어는 0 이니까 문법상 없다는 얘기니까 사이즈오브하면 플렉스가 24 가나오는데 실체가 있어서 스코어 인덱스 추가하면 배열나옴.
	//p->score[1];
//malloc 을 많이하면 할당받는시간이랑 해제하는 시간이 길어서 나빴어. 뭐를 계속한다고 했을때 malloc 을 많이하면 속도가 느려진 그래서 한번에 크게 잡고 배열처럼 쓰라 했었음.
	print(p);
//지금 이게 한번에 크게잡고 배열처럼 씀. 서버에서 많이 씀. 자료구조 할때 저거로 하면 속도 빨라짐. 겟노드 할때마다 할당했었는데 엠얼록을 저렇게 셋팅하면 빠름 스코어는 이 구조체의 끝이 어딘가 를 의미 또 구조체 끝에서 새로운 시작은 어딘가를의미(구조체의 끝이자 새로운시작)
	prr(p);
	printf("%ld",sizeof(FLEX));
	return 0;
}
//100000 은 세그먼테이션 폴트 뜸
```

---

main 부터 확인해본다.

FLEX 라는 구조체를 포인터 변수 p 에게 4096 크기만큼 메모리를 동적할당시켜주었다 .

FLEX 라는 구조체에 count 와 name[20]의 크기는 24 바이트 이고 score 는 실체는 있는데 값이 존재하지 않으므로 0 이라 사용가능한 메모리는 4072 이다.여기서 score 는 실체는 있는데 값이 없어서 인덱스를 추가해서 값을 넣어주면 스코어에 값이 들어간다.

count	name[20]	

Score(구조체의 끝이자 새로운시작)

---adv_queue---

학원에서 남아서 공부도하고 물어도보고 집에서도 공부를 하다가 시간이 너무 늦어서 안자면 내일 수업을 못들을거같아서 큐는 못봤습니다..
학원 수업시간에 소스코드를 짜긴 짰었는데 .. 저는 구조체에 int datasize, int link 를 넣어 링크에 데이터를 집어넣는 방식으로 큐를 구현하였습니다.
이부분은 내일 복습 해야할거같습니다.

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

typedef struct
{
	int data;
	int idx;
}
queue;
//link 대신 index

typedef struct
{
	int full_num;
	int free_num;
	int total;
	int cur_idx;
	// free idx
	int free[1024];
	int total_free;
	queue head[0];
}
manager;
//풀넘은 배열의 최대치 값이 들어올때마다 하나씩 줄어들어
//total 은 최대값
//비어있는거의 개수 프리넘
//컬인덱스 현재 내가 어떤 인덱스를 가리키나
//프리라는 배열은 별도로 관리할라는 배열
//토탈프리는 별도로 관리한개 몇개인지.
//큐헤드가 총 배열

bool is_dup(int *arr, int cur_idx)
{
	int i, tmp = arr[cur_idx];

	for(i = 0; i < cur_idx; i++)
		if(tmp == arr[i])
			return true;

	return false;
}//중복을 허용하지 않는

void init_data(int *data, int size)
{
	int i;

	for(i = 0; i < size; i++)
	{
redo:
		data[i] = rand() % 100 + 1;

		if(is_dup(data, i))
		{
			printf("%d dup! redo rand()\n", data[i]);
			goto redo;
		}
	}
}
//초기화하는함수

void print_arr(int *arr, int size)
{
	int i;

	for(i = 0; i < size; i++)
```

```
                printf("arr[%d] = %d\n", i, arr[i]);
}

void init_manager(manager *m, int alloc_size)
{
        m->full_num = 0;
//할당한게 없어서 0 이다
        // 12: full_num, free_num, cur_idx
        // 8: data, idx
        m->free_num = (alloc_size / sizeof(int) - 1029) / 2;//4096+20 을 뺴야함
        m->total = (alloc_size / sizeof(int) - 1029) / 2;
//사이즈오브인트로 나눴기 떄문에 나누기 2 만 하면됨 숫자를 할당할수 있는 갯수가 1533 개 있다는 뜻. 처음에 토탈이랑 프리넘이랑 같은이유 아무것도 할당 안되있어서
        m->cur_idx = 0;
//아무것도 할당 안되있어서 초기값 0 이
}

void print_manager_info(manager *m)
{
        int i;

        printf("m->full_num = %d\n", m->full_num);//
        printf("m->free_num = %d\n", m->free_num);//
        printf("m->total = %d\n", m->total);//이거만 값이 나와야함
        printf("m->cur_idx = %d\n", m->cur_idx);//
        printf("m->total_free = %d\n", m->total_free);

        for(i = 0; i < m->total_free; i++)//맨처음에 토탈 프리가 없다. 디큐할때만 활성화됨
                printf("m->free = %d\t", m->free[i]);

        printf("\n");
}

void enqueue(manager *m, int data)//동작할당한 배열에 난수로 만든 데이터를 집어넣는다.
{
        m->head[m->cur_idx].data = data;//컬인덱스 0 넘어온 데이터값 셋팅
        m->head[m->cur_idx++].idx = m->cur_idx;//컬인덱스를 증가시켰어 따라서 1 이된다 ++시켜서
        m->free_num--;//1532 가 될거다 하나 들어오면 하나 마이너스 되서
        m->full_num++;//이건 1 이될거다.
}

void dequeue(manager *m, int data)
{
        int i;

        for(i = 0; i < m->full_num; i++)//풀넘의 갯수만큼 루플를 돌고있다
        {
                if(m->head[i].data == data)//데이터 일치하면 데이트 0 으로만들어
                {
                        m->head[i].data = 0;
                        m->head[i - 1].idx = m->head[i].idx;
                        m->free_num++;//해제 하나증가
                        m->full_num--;//해제 하나감소
                        m->free[m->total_free++] = i;//디큐한갯수 증가시
                }
        }
}

void print_queue(manager *m)
{
        int i = 0;
        int flag = 0;
        int tmp = i; // m->head[i].idx;

        printf("print_queue\n");

#if 0
        for(; !(m->head[tmp].data);)
                tmp = m->head[tmp].idx;
```

```
#endif

	while(m->head[tmp].data)
	{//헤드에 데이터가 있다면
		printf("data = %d, cur_idx = %d\n", m->head[tmp].data, tmp);
		printf("idx = %d\n", m->head[tmp].idx);

		for(; !(m->head[tmp].data);)//중간에 0 이 있으면 재껴서 없을때까지 템프인덱스 갱신
		{
			tmp = m->head[tmp].idx;
			flag = 1;
		}

		if(!flag)
			tmp = m->head[tmp].idx;

		flag = 0;
	}
}

bool is_it_full(manager *m)
{
	if(m->total < m->cur_idx)//컬인덱스가 풀넘보다 크면 넘어갔다는소리.
		return true;

	return false;
}

void enqueue_with_free(manager *m, int data)
{
	/*
				m->head[i].data = 0;
				m->head[i - 1].idx = m->head[i].idx;
				m->free_num++;
				m->full_num--;
				m->free[m->total_free++] = i;
	*/

	m->head[m->cur_idx - 1].idx = m->free[m->total_free – 1];
//초과됫기때문에 1 을 빼서 끝에를 가리킴
	m->total_free--;
	m->head[m->free[m->total_free]].data = data;
	m->head[m->free[m->total_free]].idx = m->free[m->total_free - 1];

	if(!(m->total_free - 1 < 0))
		m->head[m->free[m->total_free]].idx = m->free[m->total_free - 1];
	else
		printf("Need more memory\n");

	m->free_num--;
	m->full_num++;
}

int main(void)
{
	int i;
	bool is_full;
	int alloc_size = 1 << 12;
	int data[10] = {0};
	int size = sizeof(data) / sizeof(int);

	srand(time(NULL));
	init_data(data, size);
	print_arr(data, size);
//큐로 집어넣을 배열 생성

	manager *m = (manager *)malloc(alloc_size*4);
//복잡한 구조체 매니저 4096 메모리 할당
	init_manager(m, alloc_size);

	printf("Before Enqueue\n");
	print_manager_info(m);//매니저인포 디버다
```

```
	for(i = 0; i < size; i++)
		enqueue(m, data[i]);

	printf("After Enqueue\n");
	print_queue(m);

	dequeue(m, data[1]);

	printf("After Dequeue\n");
	print_queue(m);

	enqueue(m, 777);
	print_manager_info(m);
	print_queue(m);

	dequeue(m, data[4]);
	dequeue(m, data[5]);
	dequeue(m, data[6]);
	enqueue(m, 333);
	print_manager_info(m);
	print_queue(m);

#if 1
	// 강제로 꽉찼다 가정하고 free 공간을 활용 해보자!
	is_full = true;
#endif

	//if(is_it_full(m))
	if(is_full)
		enqueue_with_free(m, 3333);

	print_manager_info(m);
	print_queue(m);

	return 0;
}
```

---sem---

Sem.c

```
#include "sem.h"

int main(void)
{
	int sid;//세마포어 아이디
//세마포어 아이디 변수 선언
	sid = CreateSEM(0x777);
//세마포어 아이디를 0x777 로 크리에이트 함수로 넘겨준다

	printf("before\n");

	p(sid);//p 를 통해 세마코어 1 을 증가시킴

	printf("Enter Critical Sectiorn\n");

	getchar();

	v(sid);

	printf("after\n");

	return 0;
}
```

---

semlib.c

```
#include "sem.h"

int CreateSEM(key_t semkey) //여기서 semkey 는 sid 0x777
{
```

```
	int status =0, semid; //semid 상태를 0 으로 초기화시킨다
	if((semid = semget(semkey,1,SEMPERM | IPC_CREAT | IPC_EXCL))
== -1)//777 로 세마포어  아이디 얻고
// 키값이 0x777 인 아무것도 없을때 생성하고 1 개의 세마포어가 존재하면 생성하지 않는
세마포어의 식별값을 semid 반환한다.
//아이피시 포로세스들끼리 통신 아이피시 생성 해당 키값으로 세마포어가 있으면 씹어라
	{
	if(errno == EEXIST){//세마포어가 존재한다면
		semid = semget(semkey, 1 , 0);//현재 있는세마포어를 갖고온다.
//이미 세마포어가 존재할 경우에 식별자만 얻는다.
		}
	}
	else{//존재하지 않는다면
		status = semctl(semid,0,SETVAL,2);//현재 세마포어 아이디를 0 으
로 셋팅한다.//현재 세마포어를 생성한 프로세스만 그 값을 초기화 시킨다
	}

	if(semid == -1 || status == -1)//
	{
		return -1;}

	return semid;//아이디값을 리턴함
}

int p(int semid)
{
	struct sembuf p_buf = {0,-1,SEM_UNDO};
/*p_buf.sem_num = 0; /* 0 번째 세마포어 */
p_buf.sem_op = -1; /* sv + (-1) */
p_buf.sem_flg = SEM_UNDO; */

//세마포어 건드는 프로세스 종료되면 원래값으로 초기화 즉 0 으로 되돌려 SETUNDO 가
프로세스가 종료될때 초기화 시킨다.
	if(semop(semid,&p_buf,1) == -1){세마포어배열의개수
//senop 는 세마포어를 실질적으로 동작시키는 역활
```

```
세마포어의 값을 위에서 1 로 초기화 하면, semop 함수는 수행하면서 이 값
에 sem_op ('-1')을 더하게 되어 결론적으로 0 으로 만들게 되며, 다른 프로세스의
임계영역 진입을 막게 됩니다.
//세마포어를 1 증가시켜라 이 두줄이 패턴
#1 을 감소시켜 0 으로 만들어서 락을 걸어라.
		return -1;//연산실패하면 -1 리턴
	}
	return 0;
}

int v(int semid)
{
	struct sembuf p_buf = {0,1,SEM_UNDO};//피버프가 마이너스 1 빼기할때
적용 1 을더해서 락을 풀어라
	if(semop(semid, &p_buf,1) == -1)//세마포어 있는 씨엔티가 0
	{	return -1;//에러상황을 리턴하라 -1 리턴
	}
	return 0;
}

/*OS 에서 Lock 메커니즘이라 하면 크게 두개가있다
1. Semaphore -> 대기열이 있다.//프로세스 여러개 적용가능
2. Spinlock - > CPU 를 지속적으로 잡고있다 (폴링)//태스크에 여러개 불가능

//화장실을 예로든다 화장실에서 자리가 있다 급하면 들어간다 들어가서 문을 잠근다 이게 락
이다.
화장실이왔는데 다 차있어서 위에층이나 아래층 가는게 세마포어
getsem 락을 획득하는거 다른놈은 락이 풀릴때까지 접근 불가능
기다린다는거는 wait queue 로 간다는거 웨이트로 간다는건 컨테스트 스위칭 한다는거 컨테
스트 스위칭 비용은 크다 하드웨어 레지스터 싹다 느려터진 메모리로 다 옮겼다가 복원해야
한다.
저장하고 복원하는 과정속에서 클럭손실이 있다.
락을 걸어야하는게 대용량이면 세마포어가 좋다. 씨피유 리소스를 잡아먹어 다른일을 할수
없다.
```

대규모에서는 데이터를 전부 안정적으로 처리할라면 세마포어 사용해야한다.

화장실에서 락이 표시가 없을때 계속 한칸 두들기는게 스핀락
락을 얻어야하는게 단순 간단한게 스핀락이 좋다.
더하기 곱셈 뺄셈 1 콜록이라 1 콜록만 빨리하고 빠지는게좋다
*/

/*세마포어 하면 빠지지않는거 크리티컬 섹션
임계 역역 (쉽게 치명타가 터지는 구간)
fork() -> 프로세스를 만든다 가상메모리가 독립적 종속관계가 없다.

종속관계를 가지는 부모자식 관계가 있다 포크말고
쓰레드(thread) 도 쓰레드 당 태스크언다바 스터럭트를 만든다.
프로세스-독립
쓰레드-종속
*/

SEM_UNDO 도 보통 많이 사용하는 데, 이것
은 시스템이 현재 프로세스의 세마포어에 대한 동작을 추적하여 프로세스가 비정상
적인 종료를 수행하였을 경우에 세마포어의 값이 비정상적으로 되는 것을 막기위해
지금까지 수행한 프로세스의 작업을 모두 취소시키는 역할을 합니다.
가령 프로세스가 세마포어의 값을 0 으로 설정하고 난 후에 임계영역을 수행하던중,
어떠한 이유로 비정상적으로 중단되었을 경우에, 이 프로세스가 수행한 이전의 작
업을 원상복구하여 세마포어의 값을 1 로 다시 설정되지 않으면, 다른 프로세스들이
영원히 임계영역에 들어오지 못하는 수가 생길 수 있습니다. 이럴 경우를 대비해서
SEM_UNDO 를 사용합니다.

---

sem.h

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
```

```
#define SEMPERM 0777

int CreateSEM(key_t semkey);
int p(int semid);
int v(int semid);
```

---shared memory---

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);
```

---

send.c

```
#include "shm.h"
//page 공유 page frame 물리메모리 공유

int main(void)
{
        int mid;//memory id
```

```
	SHM_t *p;
//공유하는 메모리 번지가 p 가된다

	mid = OpenSHM(0x888);
//OpenSHM 함수에서 //key = 0x88 길이 = 24 바이트 ,0 을전달받음
//특정한 지정의 페이지프레임 접근에대한 아디값을 줌

	p=GetPtrSHM(mid);
//쉐어드 메모리의 포인터를 얻어와라 아이디값 전달하는 이유 개나소나 전달하면 안됨

	getchar();
//엔터가 입력되면
	strcpy(p->name,"아무개");//pname 에 아무개를 저장함
	p->score = 93;//스코어에 93 넣음

	FreePtrSHM(p);
//쉐어드 메모리 다썼으니 해제 한다. 해제하는데 시간이 걸림


	return 0;
}
send 는 write 한다
```

```
//쉐어드 메모리로 모터정보를 넘김. IPC 쓰는이유임 프로세스간에 정보를 공유하기 위해
/*차량을 만들면 메인 메모리 am5727
zybo,cortaxr5 가 연결됨
모터의 출력을 높여라 하면 코어택스에 신호가 전달됨 또는 이더넷
자이보도 또같음 영상처리 하는 프로세스 에이가 있으면
모터한테 명령 내리는 통신프로세스 비가 있다
영성처리한 결과를 비한테 보내거나 통신을 통해서 모터의 상황을 에이한테 보내야한다.
이정보가 서로 왔다갔다 해야지 영상처리를 하면서 모터 제어가 된다 이럴때 아이피시 쉐어
드 메모리
같은게 필요하다.*/
```

---

recv.c

```
#include "shm.h"

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

	mid = CreateSHM(0x888);
//shared 메모리 생성 키값 만들고 리시브하는거 사이즈값메모리 아이디값 얻음

	p = GetPtrSHM(mid);//물리메모리 주소 얻음

	getchar();
	printf("이름 : [%s], 점수 : [%d]\n",p→name,p→score);
//출력하면 아무개랑 93 나옴

	FreePtrSHM(p);
	return 0;
}//recv 는 read 한다
```

---

shmlib.c

```
#include "shm.h"

int CreateSHM(long key)
{
	return shmget(key,sizeof(SHM_t),IPC_CREAT | 0777);
//리시브는 물리메모리 리드해감 센드는 밑에서 물리메모리 라이트함
}

int OpenSHM(long key)
{
	return shmget(key, sizeof(SHM_t),0);
//key 값 셋팅 포인터 길이전달 0 을 전달
//key = 0x88 길이 = 24 바이트 ,0 을전달
```

```
}
//물리메모리에 페이지프레임을 얻음
SHM_t *GetPtrSHM(int shmid)
{
	return (SHM_t *)shmat(shmid, (char *)0,0);
//id 가지고 주소를 찾는거 0 번지 부터 주소를 찾는다. 공유메모리의 물리주소를 얻어옴
}

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