TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 하성용
accept0108@naver.com

25 일차
리눅스 프로그래밍 8 일차

실습
adv_tech4.c
값을 넣어서 출력되게하라(ex.10000)

```
#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;

int main(void)
{
	FLEX *p=(FLEX *)malloc(4096);
	p->score[0];
	p->score[1];
	return 0;
}
```

답

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

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

typedef struct
{
	int count;
	char name[20];
	int score[0]; //0 짜리배열을 실제배열처럼 사용가능하다는게 핵심
} FLEX;

int main(void)
{
	FLEX *p=(FLEX *)malloc(4096);
			//4096 을 잡아놓음, 메모리를 4096 으로 사용하겠다는뜻
	int i;
	for(i=0; i<10000; i++)
	{
		p->score[i]=i+1;
		printf("%d\n", p->score[i]);
	p->score[0];
	p->score[1];
```

```
	}
	return 0;
}
```

malloc 의 단점은 할당받는시간과 해제하는시간이 김

그걸 커버하기위해서 한번에 크게잡고 그것을 배열처럼 쓰라는것
커널로 진입하는시간이 없어지므로 할당받는속도가 빨라짐

score 가 의미하는것은
score 의 size 를 찍으면 24 가 나오는데 이 24 가 구조체의 끝이자 새로운시작

실습
이 할당기법을 사용하여 큐를 구현하라

adv_queue.c

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

typedef struct
{
	int data;
	int idx;
}
queue;

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;

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;
}

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;
			}
		}
}

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;
		// 12: full_num, free_num, cur_idx//1024 빼야함
		// 8: data, idx
		m->free_num = (alloc_size / sizeof(int) - 12) / 2;
//몇개가 필요한지, 4096x4=16384-4096-20=1533 이 사실상 이용할수잇는,
//할당할수잇는 갯수가 1533 개
		m->total = (alloc_size / sizeof(int) - 12) / 8;
//처음 토탈프리넘이 같은이유는 아무것도 할당한게 없기때문
		m->cur_idx = 0;
}

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);
		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++)
			printf("m->free = %d\t", m->free[i]);

		printf("\n");
}

void enqueue(manager *m, int data)
{
		m->head[m->cur_idx].data = data; //컬 인덱스에 넘어온데이타를 넣고잇음
		m->head[m->cur_idx++].idx = m->cur_idx;
					//++에서 증가시켯기때문에 1 이 들어감
		m->free_num--; //1522
		m->full_num++; //1
}

void dequeue(manager *m, int data)
{
```

```
	int i;

	for(i = 0; i < m->full_num; i++)
	{
		if(m->head[i].data == data)
		{
			m->head[i].data = 0;
			m->head[i - 1].idx = m->head[i].idx;
				//해제된 것들만 -1
			m->free_num++;
			m->full_num--;
			m->free[m->total_free++] = i;
		}
	}
}

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)
	{
		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);)
		{
			tmp = m->head[tmp].idx;
			flag = 1;
		}

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

		flag = 0;
	}
}

bool is_it_full(manager *m)
{
	if(m->full_num < m->cur_idx)
//cur_idx 가 full_num 보다 크다는건 넘어갔다는것
//즉, 꽉찼다
		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];
        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);

        manager *m = (manager *)malloc(alloc_size);
        init_manager(m, alloc_size);
        printf("Before Enqueue\n");
        print_manager_info(m);

        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
		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.c

```
#include"sem.h"

int main(void){

		int sid;
		sid = CreateSEM(0x777);

		printf("before\n");

		p(sid);
		printf("Enter Critical Section\n");

		getchar(); //잠깐대기하기위해 문자하나받음

		v(sid); //키가 하나랑비슷한데 p

		printf("after\n");

		return 0; //정상적종료
}
```

---

sem.h

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

#define SEMPERM 0777

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

semlib.c

```
#include"sem.h"

int CreateSEM(key_t semkey){

        int status = 0, semid;
        if(( semid = semget(semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == 1)
                                                        //IPC:프로세스간 통신//
                if(errno == EEXIST) //세마코어가 존재한다면
                        semid = semget(semkey, 1, 0);
                                        //세마코어있는거 가져온다는거
        else                            //세마코어가 존재하지 않는다면
                status = semctl(semid, 0, SETVAL, 2);
//셈씨티엘이란걸 하게되는데
//셈씨티엘은 셋벨류를 하고있는데 가운데 있는걸로 지정해준다는거 즉 0

        if(semid == -1 || status == -1)
                return -1;

        return semid; //
}

int p(int semid){

        struct sembuf p_buf = {0, -1, SEM_UNDO};
                                        //UNDO 는 프로세스종료하면원래값으로 초기화시켜라
        if(semop(semid, &p_buf, 1) == -1)
                                        //세마코어값을 1 증가시켜라
                return -1;              //세마코어값이 1 이되고 연산실패하면 -1 됨
        return 0;                       //정상실행되면
}


int v(int semid){

        struct sembuf p_buf = {0, 1, SEM_UNDO};
        if(semop(semid, &p_buf, 1) == -1)
                return -1;
        return 0;

}
```

OS Lock 메커니즘
1. Semaphore → 대기열
ex)예약
세마코어는 성능을 희생하고 데이터를 가져감
대규모 작업에 적합

2. Spinlock
cpu 를 지속적으로 잡고 있음
(즉 polling)
ex)화장실에있는데 3 초마다 문 두드림
스핀락은 빨리 끝낼수있음
단순간단한 작업에 적합

Critical section(임계 영역)

쉐어드메모리
쉐어드 메모리는 페이지를 공유함(물리메모리 공유)

컴파일 방법

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

mid = 메모리 아이디

send.c

```
#include "shm.h"

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

	mid=OpenSHM(0x888);//메모리 아이디값을 얻음


	p=GetPrtSHM(mid);//메모리주소를 얻어옴

	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);
```

```
	p=GetPtrSHM(mid);

	get_char();
	printf("이름:[%s][%d]\n",p->name, p->score);
				//출력하면 아무개와 스코어 93 이 나옴

	FreePtrSHM(p);
	return 0;
}
```

---

shmlib.c

```
#include "shm.h"

int CreateSHM(long key)
{
	return shmget(key,sizeof(SHM_t) IPC_CREAT|0777);
				//권한주는거
				//공유메모리를 두고 한쪽은 쓰고 한쪽은 읽어가고있음
				//샌드는 물리메모리에 롸이트 리시브는 물리메모리에서 리드해감
				//그래서 IPC_CREAT 가 붙어있는거
}
int OpenSHM(long key)
{
	return shmget(key,sizeof(SHM_t),0);
	//0 을줘서 이포인터를 쉐어드메모리로 지정할것이다하고 쉐어드메모리를 얻습니다
	//공유된페이지주소를 얻게됨, 페이지프레임 물리메모리 자체를 얻게된다는거
}

SHM_t *GetPtrSHM(int shmid)
{
	return (SHM_t *)shmat(shmid, (char *)0,0);
}

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 *GetPtSHM(int shmid);
int FreePtrSHM(SHM_t *shmptr);
```