TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

1. 시스템 프로그래밍 – 7

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


typedef struct{

    int score;
    char name[20];
}ST;

typedef struct{

    int count;
    char name[20];
    int score[0]; // 구조체의 끝이자 새로운 시작을 가리킴.
}FLEX;


int main(void){

    int i;
    FLEX *p = (FLEX *)malloc(4096);  //동적할당 많이하면
                                     //  할당 받는 시간과 해제하는 시간이 걸린다

    printf("%d", sizeof(FLEX));
    for(i =0; i<10000; i++){
        p->score[i] = i+1;
        printf("p%d %d\n",i+1, p->score[i]);
    }

    return 0;
}

/* Q 구조체 내에 배열 인덱스를 굳이 설정하지 않아도 되는거 아닌가? 어짜피 동적이면?
*/
```

- int score[0]

위와 같은 선언을 하면 배열 내에 인덱스를 굳이 설정하지 않아도 원하는 만큼 동적으로 값을 넣을 수 있다.

단, 동적할당을 다른 곳에서 사용하고 있다면 메모리 사용하는 부분이 겹쳐 segmentation fault가 발생할 수 있으니 주의해서 사용해야한다.

✓ 동적할당은 많이 하면 할수록 할당받는 시간과 해제하는 시간이 오래 걸리므로 최소한으로 사용해야만 최적의 속도를 낼 수 있다.

1. 시스템 프로그래밍 – 7 (Semaphore & Spinlock)

- Semaphore

1. 프로세스 여러 개를 끝까지 실행시켜준다.
 -> os lock 메커니즘으로 대규모 코드에는 세마포어가 적합하다.

2. 대기 프로세스는 Wait queue에 들어가 있으며 context switching이 자주 발생한다.

3. 락이 풀릴 때 까지 다른 프로세스는 접근이 불가능하다.

- Spinlock

1. CPU를 지속적으로 잡으며 polling한다. (계속 문을 두들기는 것과 같다.)

2. 락이 풀릴 때까지 계속 기다린다.
-> 단순 간단한 코드에는 스핀락이 적합하다.

1. 시스템 프로그래밍 – 7 (Semaphore 구현)

```c
#include"sem.h"

int CreateSEM(key_t semkey){

    int status = 0, semid;
    // semkey  0x777이 와야 틀림 ipc 프로세스간 통신

    if(( semid = semget(semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == -1){
        if(errno == EEXIST){   // sem 있으면
            semid = semget(semkey, 1, 0);  // 존재하는 sem가져옴
        }
    }
    else
        status = semctl(semid, 0, SETVAL, 2);// set value를 0으로 설정

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

    return semid;
}

int p(int semid){

    struct sembuf p_buf = {0, -1, SEM_UNDO};  // 멋셈
    // sem_undo 세마포어 되면 이전 상태로 되돌려라

    if(semop(semid, &p_buf, 1) == -1) //  세마포어 값을 1 증가 시켜라
        return -1;
    return 0;
}


int v(int semid){

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

}
```

```c
#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);
```

```c
#include"sem.h"

int main(void){

    int sid;
    sid = CreateSEM(0x777);

    printf("before\n");

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

    getchar();

    v(sid);

    printf("after\n");

    return 0;
}
```

- CreateSEM

세마포어 생성을 위해 semkey로 0x777을 넘겨 semget으로 semid를 얻어 온다.

P는 세마포어 값을 1증가 시키고

V는 세마포어 값을 1감소 시킨다.

ipc 프로세스간 통신을 위해 semkey에는 0x7770이 와야한다. (물리적 주소 접근을 위함.)

1. 시스템 프로그래밍 – 7 (Shared Memory 구현)

```c
#include "shm.h"

int CreateSHM(long key){
    //shm 생성
    return shmget(key, sizeof(SHM_t), IPC_CREAT | 0777);
}

int OpenSHM(long key){
    return shmget(key,sizeof(SHM_t),0);  // 공유된 페이지 주소를 얻음.
}                                        //물리 메모리의 페이지 프레임을 얻음.

SHM_t *GetPtrSHM(int shmid){
    return (SHM_t *)shmat(shmid,(char *)0, 0); //at shm주소를 찾기 --물리주소
}

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

```c
#include"shm.h"
int main(void){ //  물리 메모리에서 read

    int mid;
    SHM_t *p;

    mid = CreateSHM(0x888);  // SHM 생성

    p = GetPtrSHM(mid);

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

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

```c
#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 FreePtreSHM(SHM_t *shmptr);
```

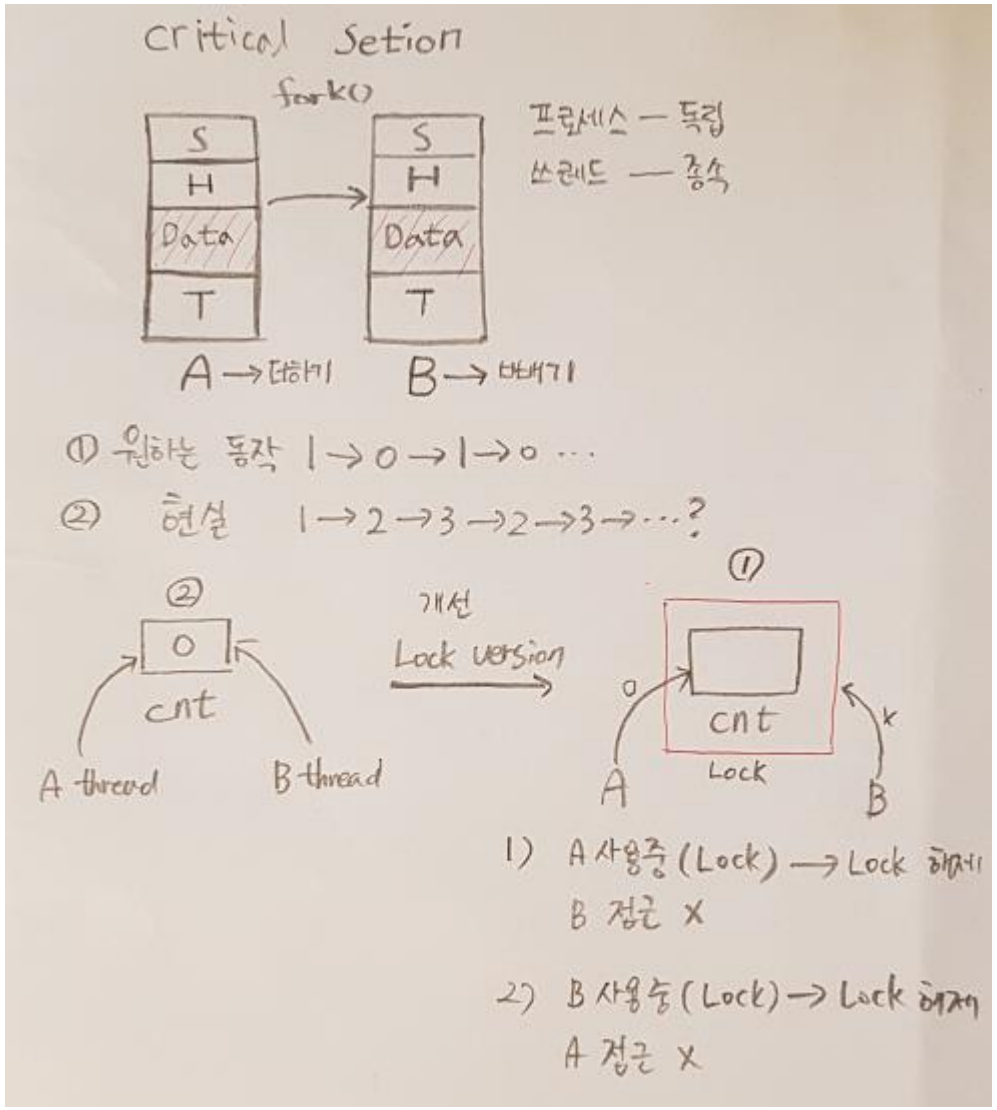
```c
#include"shm.h"

int main(void){ // 물리 메모리에 write

    int mid;   // memory id
    SHM_t *p;   // shared memory

    mid = OpenSHM(0x888);  // 페이지프레임 아이디 값

    p = GetPtrSHM(mid);  // 실제 주소를 얻어옴.

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

    FreePtrSHM(p);  // shm 해제. free 시간이 좀 걸림.

    return 0;
}
```

- Send에서는 물리 메모리에 write를 하는 것이고
- Recv에서는 물리 메모리에서 read를 하는 역할을 한다.

ipc 통신을 위해 SHM은 정말 중요하다.
프로세스간 정보를 공유하기 위해서 반드시 사용해야 한다.

이 방식으로 페이지프레임 아이디 값을 가져와 물리 메모리 주소에 직접 값을 넣을 수 있다.

1. 시스템 프로그래밍 – 7 (critical section 임계 영역)



- 여러 프로세스를 생성하는 것은 여러 개의 task_struct를 만든다는 것과 같다. 즉, 여러 task들이 동시에 data에 접근해서 정보가 꼬일 수 가 있다.

→ 이러한 구간을 critical section이라고 부른다.

- fork()는 독립적인 프로세스를 만들고, thread는 종속적인 프로세스를 만든다.