TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 최대성
c3d4s19@naver.com

2018.03.28. – 025일차 수업

구조체 내부에서 길이 제한 없는 배열 선언

```
typedef struct s {
        int data;
        int arr[0];
}S

S* newQueue = (S*)malloc(SIZE);
```

이 경우 arr는 남은 공간의 시작지점을 가리키는 포인터가 되고 배열 arr[]는 할당받은 구조체의 남은 공간을 모두 사용할 수 있다.

사용 예제 (배열로 Queue 구현)

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

#define SIZE 4096

typedef struct _queue{
        int first;
        int last;
        int totalData;
        int data[0];
}Queue;

Queue* getQueue(){
        Queue* newQueue = (Queue*)malloc(SIZE);
        newQueue->first = 0;
        newQueue->last = 0;
        newQueue->totalData = 0;
        return newQueue;
}

void enQueue(Queue* Q, int data){
        //큐 이미 가득찬 상태면 함수 종료
        if(++(Q->totalData) > SIZE/4 - 3 - 1){
                printf("Queue is full!₩n");
                (Q->totalData)--;
                return;
        }
        //data 넣기
        Q->data[Q->last] = data;
        //index가 배열 공간을 넘어가면 0으로
초기화
        if(++(Q->last) > SIZE/4 - 3 - 1){
                Q->last = 0;
        }
}

int deQueue(Queue* Q, int data){
        int popData = -1;
        int cur = Q->first;
        //큐 안에 아무것도 없는 상태면 -1 리턴
        if(Q->totalData == 0){
                printf("Queue is empty!₩n");
                return -1;
        }

        while(cur != Q->last){
                //데이터 찾은 경우
                if(Q->data[cur] == data){
                        popData = Q->data[cur];
                        Q->data[cur] = -1;
                        if(--(Q->totalData) == 0)
                                break;
                }
                //cur++ 배열공간 넘어가면 0으로
초기화
                if(++cur > SIZE/4 - 3 - 1){
                        cur = 0;
                }
        }
        return popData;
}

void printQueue(Queue* Q){
        int cur = Q->first;
        int remain = Q->totalData;
        if(Q->totalData == 0){
                printf("empty!₩n");
        }
        //남아있는 자료가 있는 경우
        while(remain > 0){
                //데이터값이 -1이 아닌 경우
                if(Q->data[cur] != -1){
                        remain--;
                        printf("%d ₩n",Q-
>data[cur]);
                        if(++cur > SIZE/4 -3 -1)

                                cur = 0;
                }
                else
                        cur++;
        }
}

int main(){
        Queue* Q1 = getQueue();
        //deQueue(Q1);
        enQueue(Q1,11);
        enQueue(Q1,11);
        enQueue(Q1,22);
        enQueue(Q1,33);
        enQueue(Q1,37);
        enQueue(Q1,44);
        enQueue(Q1,48);
        printQueue(Q1);
        printf("pop: %d₩n",deQueue(Q1, 11));
        printf("pop: %d₩n",deQueue(Q1, 33));
        printQueue(Q1);
        return 0;
}
```

---

* IPC(Inter Process Communication)

-> 프로세스간 정보공유

물리메모리 공유 방법 (Shared Memory)

<sys/shm.h> <sys/ipc.h> 헤더 추가

---

int shmget(key_t [key], size_t [size], int[shmflg])

-> 주어진 [key]를 접근번호로 하는 [size]크기의 공유메모리 공간 할당을 커널에 요청하고 할당 성공시 공유메모리를 가르키는 포인터(식별자)를 반환, 실패하면 -1을 반환

[shmflg]

IPC_CREAT -> 새로운 영역 할당

IPC_EXCL -> 공유 메모리가 이미 있으면 에러 리턴

---

void* shmat(int [shmid], const void* [shmaddr], int [shmflag])

-> 사용할 공유 메모리(물리 메모리) 주소 반환, 실패시 -1

ex) (struct s*)shmat([shmid], NULL, 0);

-> 사용하지 않는 적당한 메모리 영역 반환(바로 위의 경우)

(malloc()처럼 반환받을 형식 포인터 설정 가능)

---

shmdt([분리할 공유 메모리 주소])

-> 프로세스에 첨부된 공유메모리를 프로세스에서 분리

메모리 공유(Shared Memory) 예제

<shm.h 파일>

```
/*

컴파일 방법
gcc -o send shmlib.c send.c
gcc -o send shmlib.c recv.c
터미널 두 개 띄우고 ./send 이후에 ./recv

*/

#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);
```

<shmlib.c 파일>

```
#include "shm.h"

int CreateSHM(long key){
        return shmget(key,
sizeof(SHM_t) ,IPC_CREAT | 0777);
}
int OpenSHM(long key){
        return shmget(key, sizeof(SHM_t), 0);
}
SHM_t* GetPtrSHM(int shmid){
        return (SHM_t*)shmat(shmid, (char*)0,0);
}
int FreePtrSHM(SHM_t* shmptr){
        return shmdt((char*)shmptr);
}
```

<send.c 파일>

```
#include "shm.h"

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

        mid = OpenSHM(0x888);

        p = GetPtrSHM(mid);

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

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

<recv.c 파일>

```
#include "shm.h"

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

        mid = CreateSHM(0x888);

        p = GetPtrSHM(mid);

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

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

OS Lock 메커니즘

① Semaphore

-> 프로세스 대기열 생성 후 Context Switching으로 하나의 메모리를 여러 프로세스들이 번갈아가며 동시 사용 가능

② Spinlock

-> 하나의 프로세스가 끝날 때 까지 CPU를 지속적으로 잡아서(Polling 방식) 다른 프로세스는 접근 불가

Critical Section (임계영역) – 가상메모리 DATA영역(전역변수)

: 여러 Task들이 동시에 접근해서 값이 꼬일 수 있는 구간

-> 안전하게 사용하기 위해 위의 방법으로 Lock을 걸어야 한다.

---

int semget(key_t [key], int [nsems], [semflag])

-> [key]와 일치하는 [semid] (세마포어 식별자)반환

[nsems]

-> 세마포어 집합 내의 세마포어 개수

== 접근 제한하려는 자원(프로세스 등) 개수

---

int semctl(int [semid], int [semnum], int [cmd], [arg])

-> 컨트롤 성공시 0 이상의 값, 실패시 -1 반환

[semid]

-> 세마포어 식별자

[semnum]

-> 세마포어 집합 내에서의 세마포어 위치(0부터 시작)

[cmd]가 SETVAL 인 경우

-> 세마포어 값을 설정

[arg]

-> [cmd]에 따라 다르며, 설정 또는 값을 구하는 변수

---

int semop(int [semid], struct sembuf* [sops], unsigned [snops])

-> 성공시 0, 실패시 -1 반환

[semid]

-> 세마포어 식별자

[sops]

-> 아래 구조체의 포인터

```
struct sembuf {
        short sem_num;    //세마포어 번호
        short sem_op;     //세마포어 증감값
        short sem_flg;    //옵션
}
```

[snops]이 SEM_UNDO인 경우

-> 프로세스 종료시 세마포어 설정을 원래상태로 되돌린다

세마포어(Semaphore) 예제

<sem.h 파일>

```
#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 파일>

```
#include "sem.h"

int CreateSEM(key_t semkey){
        int status = 0, semid;
        if((semid = semget(semkey, 1, SEMPERM |
IPC_CREAT | IPC_EXCL)) == -1){ printf("semget₩n");
                if(errno ==
EEXIST){ printf("errno₩n");
                        semid =
semget(semkey,1,0);
                }
        }
        else{
                status = semctl(semid, 0, SETVAL,
2);
                printf("status = %d₩n",status);
        }
        if(semid == -1 || status == -1)
                return -1;
        return semid;
}

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

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

<sem.c 파일>

```
#include "sem.h"

int main(){
        int sid;
        sid = CreateSEM(0x777);

        printf("before₩n");
        printf("sid = %d₩n",sid);
        p(sid);

        printf("Enter Critical Section₩n");

        getchar();
        v(sid);

        printf("after₩n");
        return 0;
}
```