TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 문한나

mhn97@naver.com

예제 1)

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

typedef struct{

        int score;
        char name[20];

}ST;

typedef struct{

        int count;
        char name[20];
        int score[0]; //형체가 없음 이 구조체의 끝이 어딘가...다시 시작점이다 구조체의 끝이자 시작

}FLEX;

int main(void){

        FLEX *p = (FLEX *)malloc(4096); //메모리를 4096 바이트로 사용하겠다
        printf("%d\n",sizeof(FLEX));
        int i;
        for(i=1;i<=100;i++){
        p->score[i] = i;
        printf("%d\n",p->score[i]);
}
        return 0;

}

//인덱스 포인터 0 을 배열처럼
//메모리에 malloc 많이 하면 안좋음 할당받고 해제하는 시간이 오래걸려서 속도가 느려짐
//한번에 크게잡고 그것을 배열처럼 쓰는 것이 좋음
//서버에서 많이 씀 자료구조도 이걸로하면 속도 더 빨라진다
//한번에 저렇게 할당하면 매번 할당안받아도된다
```

```
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
mhn@mhn-Z20NH-AS51B5U:~/linux/26$
```

예제 2)

<sem.c>

```
#include"sem.h"

int main(void){

	int sid;
	sid = CreateSEM(0x777); //권한

	printf("before\n");

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

	getchar();

	v(sid);

	printf("after\n");

	return 0;
}
```

<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) //777 로 sema 락 걸꺼야 sema 권한주고,프로세스간통신(ipc) 해당 키 값으로 sema 있으면 씹어라
		if(errno == EEXIST)
			semid = semget(semkey, 1, 0); //1 로 세팅
	else
		status = semctl(semid, 0, SETVAL, 2);

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

	return semid;
}

int p(int semid){

	struct sembuf p_buf = {0, -1, SEM_UNDO}; //다른숫자신경 x 프로세스 종료할 때  sem_undo sema 값 원래값으로 되돌려라(0)
	if(semop(semid, &p_buf, 1) == -1) //sama 값 1 증가시켜라
		return -1; //연산실패하면 -1 리턴
	return 0; //정상적으로 처리가 되었다면 0 리턴
}


int v(int semid){

	struct sembuf p_buf = {0, 1, SEM_UNDO}; //걍 패턴임 뺄셈할 때
	if(semop(semid, &p_buf, 1) == -1) //0
```

```
                return -1;
        return 0;

}
```

//os -> lock 매커니즘은 2 개 semaphore,spinlock
//spinlock 은 cup 를 지속적으로 잡고있음 polling
//semaphore 대기열이 존재
//2 개의 차이점? semaphore 는 프로세스여러개적용가능
//getsem->lock 획득 lock 이 풀릴때까지 다른놈 접근 x
//spinlock 한번 lock 잡으면 안놓아줌
//누가 더 좋은지는 상황에 따라 다름
//대기열->기다림->waitqueue => 컨텍스트스위칭(비용이 큼->하드웨어 레지스터 메모리로,다시 복원(클록손실)) cpu 리소스잡아먹음 이거 하는동안 다른일못함
//대규모 -> semaphore => 들어온 것 다 처리해줘야함 하나 잡고있으면 뒤에꺼 다 못함 성능을 희생하더라도 데이터를 지키겠다
//단순,간단 -> spinlock => 그냥 빨리 끝내는 것이 이득임
//결국 프로그램을 만들게 되면 둘 다 씀 연산이 단순해진다면 spinlock 쓰자 하나의 프로그램에도 둘 다 쓸 수 있음
//semaphore -> critical section(임계영역) 치명타 터지는 구간
//포크는...가상메모리가 독립적(종속관계 x) 프로세스
//쓰레드 -> task_struct 만듬 얘는 종속임 완전히 메모리를 공유함
//임계영역 -> 여러 task 가 동시에 접근해서 정보가 꼬일 수 있는 구간
//전역변수라 해서 전부 크리티컬섹션이 아님
//코드구현이 어셈블러로 되어있음(락 두개 다)

<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);
```

```
mhn@mhn-Z20NH-AS51B5U:~/linux/26$ gcc semlib.c sem.c
mhn@mhn-Z20NH-AS51B5U:~/linux/26$ ./a.out
before
Enter Critical Section

after
mhn@mhn-Z20NH-AS51B5U:~/linux/26$ vi semlib.c
```

세마포어란?
세마포어는 OS 의 lock 메커니즘의 하나로써 프로세스간 데이타를 동기화 하고 보호하는데 그목적이 있다.
세마포어는 여러개의 프로세스에 의해서 공유되는 자원의 접근제어를 위한 도구라고 할 수 있다.

작동원리?
세마포어는 상호 배제 알고리즘으로 임계 영역을 만들어서 자원을 보호한다. 차단을 원하는 자원에 대해서 세마포어를 생성하면 해당 자원을 가리키는 세마포어 값이 할당된다. 이 세마포어값을 검사해서 임계영역에 접근할 수 있는지를 결정하게 된다.
세마포어 값이 0 이면 이 자원에 접근할 수 없으며,0 보다 큰 정수면 해당 정수의 크기만큼의 프로세스가 자원에 접근할 수 있다라는 뜻이 된다. 즉 세마포어 값이 0 이면 자원을 사용할 수 있을때까지 기다리고, 0 보다 크면 접근하게 된다. 자원에 접근하면 세마포어 값을 1 감소해서 0 으로 만들고 다른 프로세스가 자원에 접근할 수 없도록 한다. 사용이 끝나면 값을 다시 1 증가시켜서 다른 프로세스가 자원을 사용할 수 있도록 만들어주면 된다.

Semget
세마포어의 생성 혹은 접근을 위해 semget 을 제공한다.
int semget(key_t key, int nsems, int semflg);

인수

key_t key 시스템에서 세머포어를 식별하는 집합 번호
int nsems 세마포어 집합 내의 세마포어 개수로 접급 제한하려는 자원의 개수
int semflg동작 옵션

semflg	옵션 내용
IPC_CREATE	key 에 해당하는 공유 세머포어가 없다면 새로 생성한다. 만약있다면 무시하며 생성을 위해 접근 권한을 지정해 주어야 한다.
IPC_EXCL	세머포어가 이미 있다면 실패로 반환하며 세머포어에 접근하지 못한다. 이 옵션이 없어야 기존 세마포어를 사용할 수 있다.

반환

-1	실패
-1 이외	새로 만들어진 세마포어 식별자 또는 key 와 일치하는 세마포어 식별자

첫번째 매개변수는 세마포어의 유일함을 보장하기 위해서 사용하는 키값이다.

두번째 매개변수 nsems 는 세마포어 셋 즉 배열의 크기다. 이값은 최초 세마포어를 생성하는 생성자의 경우에 크기가 필요하다(보통 1). 그외에 세마포어에 접근해서 사용하는 소비자의 경우에는 세마포어를 만들지 않고 단지 접근만 할뿐임으로 크기는 0 이 된다.

새로 생성하거나 접근하는 것은 semflg 를 통해 제어한다
semflg 의 값으로 IPC_CREAT 는 만약 커널에 해당 key 값으로 존재하는 세마포어가 없다면, 새로 생성 한다
IPC_EXCL 는 IPC_CREAT 와 함께 사용하며, 해당 key 값으로 세마포어가 이미 존재한다면 실패값을 리턴한다.
만약 IPC_CREAT 만 사용할경우 해당 key 값으로 존재하는 세마포어가 없다면, 새로 생성하고, 이미 존재한다면 존재하는 세마포어의 id 를 넘겨준다. IPC_EXCL 을 사용하면 key 값으로 존재하는 세마포어가 없을경우 새로 생성되고, 이미 존재한다면 존재하는 id 값을 돌려주지 않고 실패값(-1)을 되돌려주고, errno 를 설정한다.

semget 은 성공할경우 int 형의 세마포어 식별자를 되돌려주며, 모든 세마포어에 대한 접근은 이 세마포어 실별자를 사용한다

semop
semop 함수로 접근제어를 할 수 있다. 접근제어는 세마포어를 얻거나 되돌려 주는 방식으로 이루어진다.
int semop(int semid, struct sembuf *sops, size_t nsops);

첫번째 매개변수는 semget 을 통해서 얻은 세마포어 식별자이다.
두번째 매개변수는 struct sembuf 로써, 어떤 연산을 이루어지게 할런지 결정하기 위해서 사용된다.
구조체의 내용은 다음과 같으며, sys/sem.h 에 선언되어 있다.

```
struct sembuf
{
```

```
    short sem_num;    // 세마포어의수
    short sem_op;     // 세마포어 연산지정
    short sem_flg;    // 연산옵션(flag)
}
```

sem_flg	sem_flg 내용
IPC_NOWAIT	호출 즉시 실행하지 못했을 경우 기다리지 않고 실패로 바로 복귀합니다.
SEM_UNDO	프로세스가 종료되면 시스템에서 세마포어 설정을 원래 상태로 되돌립니다. 그러므로 보통 이 옵션을 사용합니다.

세번째 인자는 변경하려는 세마포어 개수로 변경하려는 세마포어 개수가 여러 개일 때 사용합니다.

semctl()
semctl() 함수로도 세마포어를 제어할 수 있다.
int semctl (int semid, int semnum, int cmd, union semun arg)

int semid 시스템에서 세머포어를 식별하는 집합 번호
int semnum 세마포어 집합 내에서의 세마포어 위치
int cmd 제어 명령

cmd	cmd 내용
GETVAL	세마포어의 현재 값을 구한다.
GETPID	세마포어에 가장 최근에 접근했던 프로세스의 프로세스 ID 를 구한다.
GETNCNT	세마포어 값이 증가하기를 기다리는 프로세스의 개수
GETZCNT	세마포어 값이 0 이 되기를 기다리는 프로세스의 개수
GETALL	세마포어 집합의 모든 세마포어 값을 구한다.
SETVAL	세마포어 값을 설정
SETALL	세마퍼어 집합의 모든 세마포어 값을 설정
IPC_STAT	세마포어의 정보를 구한다.
IPC_SET	세마포어의 소유권과 접근 허가를 설정
IPC_RMID	세마포어 집합을 삭제

union semun arg CMD 에 따라 달라지며, 설정 또는 값을 구하는 변수

```
union semun{
   int                  val;
   struct   semid_ds   *buf;
   unsigned short int  *arrary;
}
```

반환
0 <= 성공
-1 실패

예제 3)

<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);
```

<**shmlib.h**>

```
#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); //0 번지부터 id 값 찾겠다

}

int FreePtrSHM(SHM_t *shmptr){

return shmdt((char *)shmptr);

}
```

<send.c>

```
//쉐어드메모리? 메모리 공유
//어떤 메모리 공유? 페이지를 공유(물리메모리 공유)

#include "shm.h"

int main(void){

	int mid;
	SHM_t *p; //공유하고자 하는 메모리의 번지

	mid = OpenSHM(0x888); //오픈 쉐어드메모리

	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);

	p=GetPtrSHM(mid);

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

	FreePtrSHM(p);

	return 0;

}
```

shmget()

shmget() 함수는 공유 메모리를 생성합니다.

공유 메모리는 단어 뜻에서 알 수 있듯이 하나의 프로세스에서가 아니라 여러 프로세스가 함께 사용하는 메모리를 말합니다. 이 공유 메모리를 이용하면 프로세스끼리 통신을 할 수 있으며, 같은 데이터를 공유할 수 있습니다. 이렇게 같은 메모리 영역을 공유하기 위해서는 공유 메모리를 생성한 후에 프로세스의 자신의 영역에 첨부를 한 후에 마치 자신의 메모리를 사용하듯 사용합니다.

즉, 공유 메모리를 사용하기 위해서는 공유 메모리를 생성한 후에, 이 메모리가 필요한 프로세스는 필요할 때 마다 자신의 프로세스에 첨부를 한 후에 다른 메모리를 사용하듯 사용하면 되겠습니다.

int shmget(key_t key, int size, int shmflg);

key_t key 공유 메모리를 구별하는 식별 번호
int size 공유 메모리 크기
int shmflg동작 옵션

shmflg	옵션 내용
IPC_CREATE	key 에 해당하는 공유 메모리가 없다면 새로 생성한다. 만약있다면 무시하며 생성을 위해 접근 권한을 지정해 주어야 한다.
IPC_EXCL	공유 메모리가 이미 있다면 실패로 반환하며 공유 메모리에 접근하지 못한다. 이 옵션이 없어야 기존 공유 메모리에 접근할 수 있다.

반환

-1 실패
-1 이외 공유 메모리 생성 성공, 공유 메모리 식별자

shmat()
shmat() 함수는 공유 메모리를 마치 프로세스의 몸 안으로 첨부합니다.

공유 메모리는 단어 뜻에서 알 수 있듯이 하나의 프로세스에서가 아니라 여러 프로세스가 함께 사용하는 메모리를 말합니다. 이 공유 메모리를 이용하면 프로세스끼리 통신을 할 수 있으며, 같은 데이터를 공유할 수 있습니다.

이렇게 같은 메모리 영역을 공유하기 위해서는 **공유 메모리를 생성한 후에 프로세스의 자신의 영역에 첨부를 한 후에 마치 자신의 메모리를 사용하듯 사용**합니다.

즉, 공유 메모리를 사용하기 위해서는 공유 메모리를 생성한 후에, 이 메모리가 필요한 프로세스는 필요할 때 마다 자신의 프로세스에 첨부를 한 후에 다른 메모리를 사용하듯 사용하면 되겠습니다.

void *shmat(int shmid, const void *shmaddr, int shmflg);

int shmid 공유 메모리를 구별하는 식별 번호
void *shmaddr첨부되는 어드레스 주소. 일반적으로 NULL 을 지정
int shmflg 동작 옵션

shmflg	옵션 내용
SHM_RDONLY	공유 메모리를 읽기 전용으로
SHM_RND	shmaddr 이 NULL 이 아닌 경우일 때만 사용되며, shmaddr 을 반올림하여 메모리 페이지 경계에 맞춘다.

반환
(void *) -1실패
이외 프로세스에 첨부된 프로세스에서의 공유 메모리 주소

shmdt()

shmdt() 함수는 프로세스에 첨부된 공유 메모리를 프로세스에서 분리합니다.

공유 메모리는 단어 뜻에서 알 수 있듯이 하나의 프로세스에서가 아니라 여러 프로세스가 함께 사용하는 메모리를 말합니다. 이 공유 메모리를 이용하면 프로세스끼리 통신을 할 수 있으며, 같은 데이터를 공유할 수 있습니다.

이렇게 같은 메모리 영역을 공유하기 위해서는 **공유 메모리를 생성한 후에 프로세스의 자신의 영역에 첨부를 한 후에 마치 자신의 메모리를 사용하듯 사용**합니다.

즉, 공유 메모리를 사용하기 위해서는 공유 메모리를 생성한 후에, 이 메모리가 필요한 프로세스는 필요할 때 마다 자신의 프로세스에 첨부를 한 후에 다른 메모리를 사용하듯 사용하면 되겠습니다. 이후에 **공유 메모리 사용이 필요 없어지면 프로세스에서 제거합니다**.

int shmdt(const void *shmaddr);

인수
void *shmaddr 분리할 공유 메모리 주소

반환
-1 실패
0 공유 메모리 분리 성공