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

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

파일

I/O 제어,

프로세스 제어,

멀티 태스킹과 컨텍스트 스위칭,

signal 활용법,

IPC 기법

구조체는 길이 제한이 없다

```
#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)
{
	int i;
	FLEX *p = (FLEX *)malloc(4096);
	p-> score[0];
	p-> score[1];
	printf("%d\n",sizeof(FLEX));
	return 0;
}
```

```
#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)
{
	int i;
	FLEX *p = (FLEX *)malloc(4096);
	for(i=1;i<=100;i++)
	{
	p-> score[i]=i;
	printf("%d\n",p->score[i]);
	}

	for(i=1;i<=10000;i++)
	{
```

	p-> score[i]=i;
	// printf("%d\n",p->score[i]);
	}
	printf("%d\n",sizeof(FLEX));
	
	return 0;
	}

Score 는 인덱스 0의 배열이다. 이는 구조체의 끝이며, 이후 새로운 시작점이 될 수 있다. sizeof(FLEX)를 하면 24로 주소가 나온다. (앞의 두 크기)	

속도가 빨라짐.
score 가 의미하는것
이구조체의 끝은 새로운 시작시점이된다. #include <stdio.h>

IPC - 프로세스간 정보공유

쉐이드 메모리는

메모리를 공유한다는것

어떤 메모리를 공유하나?

페이지를 공유한다. 따라서 물리메모리를 공유한다는 것이다.

페이지 프레임과 페이지는 다르다.
실제 다루는 것은 페이지 프레임이며
페이지는 어떤것을 처리하는것이다. \

ipc 를 쓰는 이유 프로세스간 메모리를 공유하기위해 사용한다

→ IPC를 사용하면 shared memory를 통해 프로세스간 정보공유가 가능하다.

critical section (임계 영역)

여러 task들이 동시에 접근해서 정보가 꼬일수 있는 구간.

fork 함수만이 프로세스를 만들 수 있을까?
fork 는 종속관계가 아예 없었다.

프로세스 ---------- 독립
쓰레드(thread) ---- 종속
(쓰레드도 fork 함수처럼 task_struct 를 만들 수 있다.)

ex)
전역변수 data 가 있다 .
Thread a,b에서 각각 더하기, 빼기를 하고있다.
우리가원하는 것은 = 1,0,1,0,1,0,1 …. 이다
이대로 동작할 까?

A 프로세스가 동작할때 B프로세스가 동작하길 원한다. 이때 cpu는 context swiching을 하는데, 이 구간에서 A프로세스가 다시 동작 할 수있다.
그러므로 임계영역에서 lock 을 해주는 것이 필요하다.

세마포어 함수를 사용하여 공유자원을 동시에 여러프로세스가 사용하는 것을 막고, 오직 자원을 하나의 프로세스만 사용할 수 있도록 보장해줌.

Lock – 1. semaphore 2. spinlock

```
#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;

}
~
~
~
sem.c                                                   1,1           All
#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);
~
~
~
~
sem.h                                                   1,1           All
```

```
#include "sem.h"

int CreateSEM(key_t semkey)
{
        int status = 0;
        int semid;
        if((semid = semget(semkey,1,SEMPERM | IPC_CREAT | IPC_EXCL)) == 1)
                if(errno==EEXIST)
                {
                        semid = semget(semkey,1,0);
                }
                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};
        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;
}
~
~
~
~
~
~
~
semlib.c                                                16,11-18      All
```

lock 은 칸이 비어있을때 들어가 잠그는 것
1. semaphore → 대기열 → lock이 풀릴 때까지 기다림 , 즉 wait queue 로 간다. = context switching을 함. Push 을 통해 다시들어가야함.
2. spinlock → cpu를 지속적으로 잡고 있음. (polling) → 화장실에서 나올 때까지 두드린다.

무엇이 좋은 것인가?

대규모- 세마포어, 성능을 조금 포기하더라도 데이터를 정확히 저장하겠다.
단순 간단 - 스핀락

* semget() - 세마포어 집합을 생성하거나, 이미 생성된 세마포어에 접근

<table>
<tr><td>헤더</td><td>sys/types.h
sys/ipc.h
sys/sem.h</td></tr>
<tr><td>형태</td><td>int semget (key_t key, int nsems, int semflg)</td></tr>
<tr><td>인수</td><td>key_t key 시스템에서 세머포어를 식별하는 집합 번호
int nsems 세마포어 집합 내의 세마포어 개수로 접급 제한하려는 자원의 개수
int semflg동작 옵션
<table>
<tr><th>semflg</th><th>옵션 내용</th></tr>
<tr><td>IPC_CREATE</td><td>key에 해당하는 공유 세머포어가 없다면 새로 생성한다. 만약있다면 무시하며 생성을 위해 접근 권한을 지정해 주어야 한다.</td></tr>
<tr><td>IPC_EXCL</td><td>세머포어가 이미 있다면 실패로 반환하며 세머포어에 접근하지 못한다. 이 옵션이 없어야 기존 세마포어를 사용할 수 있다.</td></tr>
</table></td></tr>
<tr><td>반환</td><td>-1 실패
-1 이외새로 만들어진 세마포어 식별자 또는 key 와 일치하는 세마포어 식별자</td></tr>
</table>

1. 프로세스는 공유자원을 사용하기 전에 p()함수를 호출한다.

2. p()는 특정 변수값을 확인하여 공유 자원이 다른 프로세스가 사용 중인지의 여부를 판단하고, 다른 프로세스가 사용 중이면 그프로세스가 사용을 중지할 때 까지 대기한다. 사용 가능하다면 특정 변수값을 변경하고 복귀한다.

3.프로세스는 공유자원을 사용한다.

4. 프로세스는 공유 자원에 대한 작업을 완료했다면 s()함수를 호출하여 특정 변수 값을 원래의 값으로 복원한다.

*semctl() 함수 - 세미포어를 제어한다.

헤더

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

형태

```
int semctl ( int semid, int semnum, int cmd, union semun arg)
```

인수

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 ID를 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

*semop() - 세모포어 값을 변경한다. 세미 포어를 사용하기 위해 먼저 세마포어 값을 1감소시키고, 사용후에는 1증가시키는데, 이렇게 세마포어 값을 증감하는 것을 요청하는 함수이다.

헤더

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

형태

```
int semop ( int semid, struct sembuf *sops, unsigned nsops )
```

인수

int semid — 시스템에서 세머포어를 식별하는 집합 번호

struct sembuf *sops — 세마포어 값을 계사하기위한 설정 값

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

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

unsigned nsops — 변경하려는 세마포어 개수로 변경하려는 세마포어 개수가 여러 개일 때 사용합니다.
예를 들어 변경하려는 세마포어 개수가 한 개라면

```
struct sembuf sbuf = { 0, -1, 0};
semop( semmop, &sbuf, 1);
```

이렇게 값이 1이 되지만 한번에 2개 이상의 세마포어를 변경한다면,

```
struct sembuf pbuf[2]   = {
    { 0,  1, 0} ,   // 첫번째 세마포어 값을 1 증가
    { 1, -1, 0}     // 두번째 세마포어 값을 1 감소
  }
semop( semmop, pbuf, 2);
```

반환

0 성공
-1 실패

코드 – sem.c, semlib.c, sem.h

sem.c

```c
#include "sem.h"

int main(void)
{
        int sid;
        sid = CreateSEM(0x777);  //세마코어의 권한 코드
                                 // 세마공간을 만들고, 전용 아이디를 생성
        printf("before\n");

        p(sid);                  // 더하기 패턴, 문을 잠그는 역할을 하는 함수

        printf("Enter Critical Section\n"); //p를 통해서 세마코어를 증가시킨다.

        getchar(); //텍스트를 입력받아 한글자만 출력한다. 대기하기 위해 엔터치면 넘어간다.
                   //크리티컬 영역에 진입, + 빠져나갈 수 있게 해준다.
        v(sid);    //빼기 패턴

        printf("after\n");
        return 0;

}
~
~
```

semlib.c

```c
#include "sem.h"

int CreateSEM(key_t semkey)  // 777로 lock 권한 설정
{
    int status = 0;
    int semid;
    if((semid = semget(semkey,1,SEMPERM | IPC_CREAT | IPC_EXCL)) == 1)
        // 세마포어에 작업권한 주고, 세마포어의 개수는 1개, 없으면 만들고 | 있으면 무시
        if(errno==EEXIST) // 세마포어가 존재한다면
        {
            semid = semget(semkey,1,0); //현재 세미포어 값을 가져옴
            // 식별집합번호,세마포어 개수 접근 제한 자원의 개수, 동작옵션--> 0?? 현재 세미포어값
        }
        else
        {
            status = semctl(semid,0,SETVAL,2);  // semctl: 세마코어 제어.. 세마코어값을
            //   식별id,집합내의 세마포어위치, setval: 세마코어 값을 설정하겠다, 2로 설정함.
        }
    if(semid == -1 || status == -1) // 접근에 실패한다면
    {
        return -1;
    }
    return semid;  // 접근 성공하면
}

int p(int semid)  //더하기 패턴
{
    struct sembuf p_buf = {0,-1,SEM_UNDO}; //sem_undo :프로세스가 종료되면 원래값으로 초기화
   //  {식별번호, 세마포어 증감값,옵션:설정초기화}
    if(semop(semid, &p_buf,1) ==-1)       // 원래값은 세마의 설정을 원상태로 함.
    // 세마포어를 식별하는 집합번호, 세마포어 값을 설정하는 값, 변경하려는 세마포어 개수 --> 개수가 두개라면
    //  위의 struct  도 두개집합으로 만들어야함
    {
        return -1; // 실패
    }
    return 0;   // 성공
}

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

sem.h

```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);
```

컴파일 방법

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

터미널 두개 띄우고 ./send이후에 ./recv