TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 - 이상훈
gcccompil3r@gmail.com
학생 – 이우석
colre99@naver.com

[3/28(수) - 25일차]

IPC (Inter process Communication)

예를들어 A프로세스 와 B프로세스가 각각 존재한다고 했을때, A프로세스와 B프로세스는 원래 서로 공유를 할 순 없다. 하지만, IPC를 사용하면 shared memory를 통하여 프로세스간에 정보들을 공유할 수 있게된다. 개념으로 말하자면,메모리 상에 특정 공간을 잡아놓고 해당공간에 서로 접근할 수 있는 권한을 만든다. 그리고 그 공간에 정보들을 넣으면 공유가 되어 접근 권한을 가진 프로세스들이 메모리에 접근 가능하며 읽고 쓰고 할 수 있게 된다.

단, 여기에는 그로인해 문제가 발생될 수 있는 영역이 생긴다.
임계영역 (Critical Section) 이 생긴다. 이 공간은 공유되는 공간에 한꺼번에 오게되면 값을 제대로 읽지 못하게 되는 프로세스 또는 그러한 값을 제대로 쓰지 못하는 문제가 발생될 수 공간이다. 이 공간을 해결하기 위해 세마포어 라는게 있다.

* 공유되는 구간이 될때 그 구간을 Lock 을 걸어야 데이터가 안전하게 된다.

-세마포어 (semaphore) = (OS 에서 Lock 메커니즘)

세마포어는 우선 코드를 대규모로 처리해야 하는 것을 예약을 걸어두고
안정적으로 처리할 수 있게 해준다. 단점은 성능을 좀 희생해야 한다는 것.

-스핀락 (spinlock) = (이 또한 Lock 메커니즘) (polling)

스핀락은 CPU 를 지속적으로 잡고있는다. (= 단일작업 = 프로세스 여러개에
적용 불가능). 그렇기 때문에 코드가 적다면 스핀락을 써주는게 성능이
더욱 최적화 된다.

두개의 차이점은

상황에 따라 다르겠지만 상황에 맞게 사용한다면 데이터도 안전하게

사용할 수 있으며, 성능도 효율적을 사용할 수 있다. 정리하자면
이 두가지를 사용하는 이유가 IPC 에서 공유되는 공간이 생겨 치명적인 공간인 임계영역(Critical Section)에 피해를 줄이기 위해.
즉, Context Switching 의 발생을 최소화 하기 위함이다.

IPC 에 대한 예로,

ws91@air: ~/my_proj/Homework/sanghoonlee/lec/lsp

```c
#include <sys/types.h>
#include <sys/ipc.h>  //messages, semaphore, shared.
#include <sys/sem.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>  // 에러 찾는 헤더파일

#define SEMPERM 0777

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

/* IPC (=Inter process Communication)
사용하는 이유는 A프로세스와 B프로세스가 각각 존재
한다고 가정했을때 A와 B가 서로 가지고있는 정보를
공유할 수 없기 때문이다. 다시말해, IPC를 사용하면,
프로세스간에 정보를 공유할 수 있도록 만들어준다.

*/
```

:wq

ws91@air: ~/my_proj/Homework/sanghoonlee/lec/lsp

```c
#include "sem.h"

int main(void)
{
   int sid;

   sid = CreateSEM(0x777); //sema의 권한코드
         //세마 공간만들며, 전용아이디 생성.

   printf("before\n");

   p(sid); // 더하기 패턴 ,문을 잠그는 역할 함수.

   printf("Enter Critical Section\n");
   // 이공간 자체가 여러프로세스들이 동시에 씀.

   getchar(); // 텍스트를 입력받아 한글자만 출력.
    //크리티컬 영역에 진입, + 빠져나갈수 있게 해준다.

   v(sid);  // 빼기 패턴, 다른애가 들어가서
            //다시 쓸수 있게 세팅해주는 함수.

   printf("after\n"); // 출력하고 종료.

   return 0;
```

ws91@air: ~/my_proj/Homework/sanghoonlee/lec/lsp

```c
#include "sem.h"

int CreateSEM(key_t semkey)
{
   int status = 0, semid;           //semacore에 권한을 준다
      //상태 = 0.          SEM - operation permission structure.
   if((semid = semget(semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == 1)
   { //위에는 샘겟 시스템콜 세마포 만들기 (interp procesor prosection)
                                    //해당 키로 씹으라는 소리
      if(errno == EEXIST)  //세마가  혹시이미 존재하면 아이디값을 얻어라.
      {
            semid = semget(semkey, 1, 0);
      }           //현재 있는 세마를 가져오겠다는것.
   }
      else //세마가 존재하지않을때 else로 온다
      status = semctl(semid, 0, SETVAL, 2); //세마퍼에 지정될 값을 지정.
            // 현재 세마의 아이디를 가운데로.. 값은 0.
      //setval은 집합안의 세마퍼 값을 union의 val멤버의 값으로 지정.
      if(semid == -1 || status == -1)
      {             //semctl 은 시스템호출. 통제동작 수행.
        return -1;  // -1인경우는 에러. (=세마 아이디값을 리턴)
      }

   return semid;  //세마 아이디값 얻음.

}
                     //sem undo가 프로세가 종료될때 초기화
int p(int semid)     // 더하기 패턴.
{                    //프로세스 종료할때, undo쓴다. 다시
   struct sembuf p_buf = {0, -1, SEM_UNDO}; //sem undo는
   if(semop(semid, &p_buf, 1) == -1) //종료되면 원래값으로 초기화 시켜라
   {     //즉, 프로세스가 종료되면 시스템에서 세마의 설정을 원상태로.
      return -1; //semop는 세마코어값을 1 증가시켜라.
   }             //연산 실패하면 연산값은 -1로 된다.
                 //semop를 통해 피버프의 연산 수행하고 사용중으로 만든다.


   return 0; //정상적이면 리턴 0.
}

int v(int semid)   // 빼기 패턴.
{
   struct sembuf p_buf = {0, 1, SEM_UNDO};
   if(semop(semid, &p_buf, 1) == -1)
   //semop를 통해 피버프를 연산 수행.
   {
      return -1; // 사용중에서 사용 가능으로 만듬.
   }
```

1,1 Top

semaphore 의 헤더파일, 내용파일, 라이브러리 가 있다.

[Sem]

```c
#include "sem.h"

int main(void)
{
   int sid;

   sid = CreateSEM(0x777); //sema의 권한코드
         //세마 공간만들며, 전용아이디 생성.

   printf("before\n");

   p(sid); // 더하기 패턴 ,문을 잠그는 역할 함수.

   printf("Enter Critical Section\n");
   // 이공간 자체가 여러프로세스들이 동시에 씀.

   getchar(); // 텍스트를 입력받아 한글자만 출력.
    //크리티컬 영역에 진입, + 빠져나갈수 있게 해준다.

   v(sid);  // 빼기 패턴, 다른애가 들어가서
            //다시 쓸수 있게 세팅해주는 함수.

   printf("after\n"); // 출력하고 종료.

   return 0;
}


/*gets() : getchar()에서 받은 첫글잘 빼고, 나머지를
//키 버퍼에 저장한다.
gets()를 또 사용하면, 모든 문자열을 키버퍼에 저장.
왜냐하면 첫번째 사용하고 버퍼가 비어있기 때문.
결국, gets()함수는 키버퍼가 비어 있으면 문자열 받음.
*/
```

[Semlib]

```c
#include "sem.h"

int CreateSEM(key_t semkey)
{
   int status = 0, semid;            //semacore에 권한을 준다
      //상태 = 0.          SEM - operation permission structure.
   if((semid = semget(semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == 1)
   { //위에는 샘겟 시스템콜 세마포 만들기 (interp procesor prosection)
                                          //해당 키로 씹으라는 소리
      if(errno == EEXIST)  //세마가  혹시이미 존재하면 아이디값을 얻어라.
      {
            semid = semget(semkey, 1, 0);
      }            //현재 있는 세마를 가져오겠다는것.
   }
      else //세마가 존재하지않을때 else로 온다
      status = semctl(semid, 0, SETVAL, 2); //세마퍼에 지정될 값을 지정.
            // 현재 세마의 아이디를 가운데로.. 값은 0.
      //setval은 집합안의 세마퍼 값을 union의 val멤버의 값으로 지정.
      if(semid == -1 || status == -1)
      {              //semctl 은 시스템호출. 통제동작 수행.
         return -1;  // -1인경우는 에러. (=세마 아이디값을 리턴)
      }

   return semid;  //세마 아이디값 얻음.

}
                     //sem undo가 프로세가 종료될때 초기화
int p(int semid)     // 더하기 패턴.
{                    //프로세스 종료할때, undo쓴다. 다시
   struct sembuf p_buf = {0, -1, SEM_UNDO}; //sem undo는
   if(semop(semid, &p_buf, 1) == -1) //종료되면 원래값으로 초기화 시켜라
   {     //즉, 프로세스가 종료되면 시스템에서 세마의 설정을 원상태로.
      return -1; //semop는 세마코어값을 1 증가시켜라.
   }              //연산 실패하면 연산값은 -1로 된다.
                  //semop를 통해 피버프의 연산 수행하고 사용중으로 만든다.


   return 0; //정상적이면 리턴 0.
}

```

```
41 int v(int semid)   // 빼기 패턴.
42 {
43    struct sembuf p_buf = {0, 1, SEM_UNDO};
44    if(semop(semid, &p_buf, 1) == -1)
45    //semop를 통해 피버프를 연산 수행.
46    {
47       return -1; // 사용중에서 사용 가능으로 만듬.
48    }
49
50 return 0; // => 구현.
51
52 }
```

[semheader]

```
 1 #include <sys/types.h>
 2 #include <sys/ipc.h>  //messages, semaphore, shared.
 3 #include <sys/sem.h>
 4 #include <stdio.h>
 5 #include <string.h>
 6 #include <stdlib.h>
 7 #include <errno.h>  // 에러 찾는 헤더파일
 8
 9 #define SEMPERM 0777
10
11 int CreateSEM(key_t semkey);
12 int p(int semid);
13 int v(int semid);
14
15 /* IPC (=Inter process Communication)
16 사용하는 이유는 A프로세스와 B프로세스가 각각 존재
17 한다고 가정했을때 A와 B가 서로 가지고있는 정보를
18 공유할 수 없기 때문이다. 다시말해, IPC를 사용하면,
19 프로세스간에 정보를 공유할 수 있도록 만들어준다.
20
21 */
```