OS 과제 보고서

[Producer consumer problem]

12141579 윤찬미
010-9287-1679
cmmcme0604@gmail.com

I. 개요

• 구현의 목적
– Semaphore와 Mutex를 이용하여 Producer consumer problem 을 해결하고, Semaphore와 Mutexd에 대해 알 수 있다.

• 요구 사항
– <semaphore.h> 를 include 하여 POSIX 함수를 이용한다.
– 3개의 semaphore empty, full, mutex를 사용한다.
– 넘겨받는 매개변수를 통해 main함수의 대기시간과 producer,consumer thread의 생성 갯수를 정한다.
– empty가 0이면 procer이 block되고, full일 때 consumer가 block 된다.
– mutex_lock,mutex_unlock 함수를 사용하여 mutual exclusion을 보장한다.
– Error이 발생하면 error문을 출력한다.
– monitoring thread를 사용하여 producer나 consumer가 될 때 마다 버퍼 안의 item의 갯수를 기록해주고 실행된 thread를 블락시킨 후, monitoring thread로 갯수를 출력한다.

II.함수 명세

```
void *monitoring(void *para)
{
    while(1)
    {
        sem_wait(&mo2);
        printf("Acknowledge no : %d -> count==:%d\n",++all,cnt);
        sem_post(&mo1);
    }
}
```

monitoring 함수

지금까지 실행 한 연산의 횟수(all) 와 현재 buffer에 들어가 있는 item의 갯수를 출력한다.

이때, 세마포어를 사용하여 insert나 remove 가 진행 중일 때는 블락하고, insert,remove 가 끝나면 블락에 벗어나며 insert,remove 를 블락시킴

```
int insert_item(buffer_item item)
{

    if(sem_wait(&empty)!=0)
        return -1;

    pthread_mutex_lock(&mutex);

    buffer[insertnum]=item;
    insertnum=(insertnum+1)%BUFFER_SIZE;

    printf("insert : %d\n",++cnt);

    sem_post(&mo2);
    sem_wait(&mo1);

    pthread_mutex_unlock(&mutex);
    sem_post(&full);



    return 0;
}
```

insert 함수

만약 버퍼가 가득 차있다면 함수를 종료 시킨다.

그렇지 않으면 Critical Section에 다른 함수가 접근 하지 못하도록 mutex_lock를 걸어준다

랜덤으로 입력받은 item을 버퍼에 넣어 준 후, insert하는 위치를 한칸 증가 시킨다.
교재의 Producer,Consumer problem을 보면 환형 큐 형식으로 저장 되어 있기 때문에 'BUFFER_SIZE'로 나누어준다.

insert를 했다고 알려주는 출력문을 출력한 후,

monitoring_thread가 monitoring 함수에 접근 할 수 있도록 세마포어 값을 조정해준다.
insert와 monitoring이 실행 된 후 Critical Section 영역의 mutex_lock을 해지하고

버퍼에 한 item이 들어갔다는 것을 알리기 위해 세마포어 값을 증가시킨다.

```
int remove_item(buffer_item *item)
{

    if(sem_wait(&full)!=0)
        return -1;

    pthread_mutex_lock(&mutex);


    *item=buffer[removenum];
    buffer[removenum]=0;
    removenum=(removenum+1)%BUFFER_SIZE;
    printf("remove : %d\n",--cnt);

    sem_post(&mo2);
    sem_wait(&mo1);

    pthread_mutex_unlock(&mutex);
    sem_post(&empty);


}
```

remove 함수

만약 버퍼가 비어있다면 함수를 종료 시킨다.

그렇지 않으면 Critical Section에 다른 함수가 접근 하지 못하도록 mutex_lock를 걸어준다
환형 큐 형식으로 저장된 BUFFER에서 FIFO 형식으로 item을 삭제 해준다.
remove를 했다고 알려주는 출력문을 출력한 후,
insert함수와 같이 세마포어 값을 조정하여 monitoring 함수를 실행 한 후, Critical Section 영역의 mutex_lock을 해지하고

버퍼에 한 item이 들어갔다는 것을 알리기 위해 세마포어 값을 증가시킨다.

```c
void *producer(void *para)
{
    buffer_item item;
    while(1)
    {
        sleep(rand()%5);
        item=rand();
        if(insert_item(item))
            printf("report error condition");
        else
            printf("producer produced %d\n", item);

    }
}
```

producer 함수

random하게 5초 이내로 함수를 실행 시킨다. insert를 하지 못하면 에러문을 출력하고

insert를 성공하면 어떤 값이 insert 됐는지 출력한다.

```c
void *consumer(void *para)
{
    buffer_item item;
    while(1)
    {
        sleep(rand()%3);
        if(remove_item(&item))
            printf("report error condition");
        else
            printf("consumer consumed %d\n", item);
    }
}
```

Consumer 함수

random하게 3초 이내로 함수를 실행 시킨다. remove를 하지 못하면 에러문을 출력하고

remove를 성공하면 어떤 값이 remove 됐는지 출력한다.

```c
int main(int argc,char *argv[])
{
    int produce_num=atoi(argv[2]);
    int consume_num=atoi(argv[3]);
    int i=0;

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex,NULL);
    sem_init(&mo1, 0, 0);
    sem_init(&mo2, 0, 0);

    pthread_t producer_thread;
    pthread_t consumer_thread;
    pthread_t monitoring_thread;


    for(i=0;i<produce_num;i++)
        pthread_create(&producer_thread,NULL,producer,NULL);

    pthread_create(&monitoring_thread,NULL,monitoring,NULL);

    for(i=0;i<consume_num;i++)
        pthread_create(&consumer_thread,NULL,consumer,NULL);



    sleep(atoi(argv[1]));
    return 0;
}
```

main 함수

producer와 consumer thread를 몇개 생산하는지 입력받는다.

semaphore값과 mutex값을 초기화 시킨다.

3개의 thread_id를 만든다.

입력받은 값 만큼 producer,consumer thread를 생성하고
monitoring_thread는 한번 생성한 후 monitoring 함수에서 반복문을 사용하여 계속 실행시킬 수 있게 만든다.

입력받은 시간이 지나면 함수를 종료시킨다.

III.실행 화면

1)

```
cmmcme:~/workspace $ ./sema 20 5 3
insert : 1
Acknowledge no : 1 -> count==:1
insert : 2
Acknowledge no : 2 -> count==:2
remove : 1
Acknowledge no : 3 -> count==:1
remove : 0
Acknowledge no : 4 -> count==:0
insert : 1
Acknowledge no : 5 -> count==:1
remove : 0
Acknowledge no : 6 -> count==:0
insert : 1
Acknowledge no : 7 -> count==:1
remove : 0
Acknowledge no : 8 -> count==:0
insert : 1
Acknowledge no : 9 -> count==:1
remove : 0
Acknowledge no : 10 -> count==:0
insert : 1
Acknowledge no : 11 -> count==:1
remove : 0
Acknowledge no : 12 -> count==:0
insert : 1
Acknowledge no : 13 -> count==:1
insert : 2
Acknowledge no : 14 -> count==:2
insert : 3
Acknowledge no : 15 -> count==:3
remove : 2
Acknowledge no : 16 -> count==:2
remove : 1
Acknowledge no : 17 -> count==:1
remove : 0
Acknowledge no : 18 -> count==:0
```

2)

```
cmmcme:~/workspace $ ./sema 10 1 1
insert : 1
Acknowledge no : 1 -> count==:1
insert : 2
Acknowledge no : 2 -> count==:2
insert : 3
Acknowledge no : 3 -> count==:3
remove : 2
Acknowledge no : 4 -> count==:2
remove : 1
Acknowledge no : 5 -> count==:1
remove : 0
Acknowledge no : 6 -> count==:0
insert : 1
Acknowledge no : 7 -> count==:1
insert : 2
Acknowledge no : 8 -> count==:2
remove : 1
Acknowledge no : 9 -> count==:1
remove : 0
Acknowledge no : 10 -> count==:0
insert : 1
Acknowledge no : 11 -> count==:1
remove : 0
Acknowledge no : 12 -> count==:0
```

IV.Semaphore,Mutex의 사용

– Producer
wait(&empty)

– empty 세마포어의 값을 1 감소 시킨다.
– 만약에 empty가 0 –> 배열이 가득 차있으면 Block된다.

lock(&mutex)

– Producer, Consumer이 공통으로 가지는 critical section 부분을 다른 곳에서 접근 할 수 없게 mutex의 값을 0으로 만든다. unlock을 통해 mutex의 값을 1로 만들 때 까지 기다린다

post(&full)

– 한 item을 생성하여 insert 했기에 full을 증가한다.

unlock(&mutex)

– critical section 부분을 unlock 함으로서 다른 Thread가 해당 부분에 접근 할 수 있도록 mutex 값을 1로 만든다.

– Consumer

wait(&full)

– full이 0이면 배열이 비어있다는 것이다. 이때 배열이 비어있으면 Consumer가 remove를 할 수 없기 때문에 producer에서 item을 insert하고 full을 1 증가 시길 때 까지 block된다.

lock(&mutex)

– Producer, Consumer이 공통으로 가지는 critical section 부분을 다른 곳에서 접근 할 수 없게 mutex의 값을 0으로 만든다. unlock을 통해 mutex의 값을 1로 만들 때 까지 기다린다

post(&empty)

– item을 remove 했기 때문에 empty semaphore 값을 1 증가 시킨다.

unlock(&mutex)

– critical section 부분을 unlock 함으로서 다른 Thread가 해당 부분에 접근 할 수 있도록 mutex 값을 1로 만든다.

V.평가 및 개선 방향

-> 수업시간에 말씀하신 semaphore를 사용하여 monitoring_thread가 수행 할 때, insert,remove 함수를 block을 하였다.
-> 환형큐 형태로 insert와 remove의 위치를 조정하여 삽입하였다.
-> busy waiting을 사용하지 않고 구현 함으로서 계속 수행중이 아니기 때문에 cpu를 잡아먹지 않는다.

VI.과제 수행시 알게된 점

-> semaphore와 busy waiting의 차이점을 알게 되었으며, 과제를 해결하기 위해 다양한 세마포어 POSIX 함수를 찾아서 사용해보았다. multithread_programming 에서 semaphore와 mutex의 사용의 중요성을 알게 되었다.