SCSC 운영체제 및 실습 보고서

11주차 - 세마포어를 이용한 생산자와 소비자

일어일문학과
201000287
유다훈

생산자와 소비자를 어떻게 구현하였는지?

Mutex : 쓰레드들 간에 공유하는 데이터 영역을 보호하기 위해서 사용
Critical section을 만들고, 단 하나의 쓰레드만 Critical section에 접근할 수 있도록 강제

Semaphore : 쓰레드들 간에 공유하는 데이터 영역을 보호하기 위해서 사용하는 Mutex개념.
단 하나의 쓰레드만 Critical section에 접근할 수 있도록 강제

sem_itit(sem_t* sem, int pshared, unsigned int value)
critical section에 진입하기 위해 세마포어를 생성하는 함수
sem : 세마포어 참조값 저장을 위한 변수
pshaerd : 0 하나의 프로세스 / 그 외의 값은 둘 이상의 프로세스에서 접근 가능한 값
value : 생성되는 세마포어의 초기 값
몇개의 스레드가 세마포어를 사용할 것인지에 따라 value값을 정해주어야 한다.

sem_wait(sem_t* sem)
세마포어 값을 하나 감소시키는 함수.

sem_post(sem_t* sem)
세마포어의 값을 하나 증가시키는 함수

세마포어 초기값 설정

```
    /*세마포어 값 적당한 수로 초기화*/
        sem_init(&memory->sem, 1, 1);   //뮤텍스처럼 버퍼에 생산자 소비자 중
 하나만 접근하도록 제어하는 세마포어
        sem_init(&memory->empty, 1, 2); //버퍼가 비어있으면 실행 안되도록 제
어하는 세마포어
        sem_init(&memory->full, 1, 5);  //버퍼가 가득차있으면 실행 안되도록
제어하는 세마포어
```

sem_init(&memory->sem, 1, 1)

뮤텍스처럼 버퍼에 생산자 혹은 소비자 중 하나만 접근하도록 하는 세마포어
둘 중 하나 이며 세마포어가 0일때는 접근 할 수 없으므로 초기값 1 부여

sem_init(&memory->empty, 1, 2)

버퍼가 비어있으면 소비자가 실행이 안되도록 하는 세마포어
소비자는 둘 이므로, 둘 이상 못 들어 가게끔 초기값 2 부여

sem_init(&memory->full, 1, 5)

버퍼가 가득 차 있으면 실행 안되도록 제어하는 세마포어
생산자는 다섯 이므로 다섯 이상 못 들어가게끔 초기값 5 부여

기본 상황

```
/* 적절한 세마포어 사용*/
while(memory->CQ_count == 10){

}
sem_wait(&memory->full);
sem_wait(&memory->sem);

state = addQ(memory, input);
//공유 메모리 버퍼에 아이템 생산
printf("producer %d add Q %d\n", id, input);
sem_post(&memory->sem);
sem_post(&memory->empty);


/* 적절한 세마포어 사용*/
```

```
 /* 적절한 세마포어 사용*/
while(memory->CQ_count == 0){
}
sem_wait(&memory->empty);
sem_wait(&memory->sem);
output = getQ(memory);
//공유 메모리 버퍼에서 아이템 소비
printf("consumer %d get Q %d\n", id, output);
sem_post(&memory->sem);
sem_post(&memory->full);


   /* 적절한 세마포어 사용*/
```

1. 소비자는 생산을 시작한다. 버퍼내부 용량이 꽉 차면 while문 내부에서 busy waiting을 하지만 그렇지 않으면 통과한다.
2. 소비자 세마포어를 한 개 소비한다.
3. 생산자 혹은 다른 소비자가 critical section을 통과하지 못하도록 하는 세마포어를 한 개 소비한다.
4. critical section에서 아이템을 생산하여 버퍼에 추가한다.
5. 다른 스레드가 접근할 수 있도록 세마포어를 증가시킨다
6. 소비자에게 아이템 생산을 알리며 소비자의 세마포어를 증가시킨다.

6. Cpu가 소비자에게 넘어온다면, 소비자 또한 생산자와 비슷한 방식으로세마포어를 소비하며 진행한다.
7. 버퍼 내에 데이터가 없는지 확인한다. 있으면 그냥 통과한다.
8. 소비자 세마포어를 하나 감소 시킨다.
9. 다른 생산자 혹은 소비자가 접근할 수 없도록 하는 세마포어를 하나 감소 시킨다.
10. 데이터를 버퍼로부터 소비한다.
11. 다른 생산자 혹은 소비자가 접근할 수 있도록 세마포어를 증가 시킨다.
12. 생산자에게 아이템을 소비했다고 알리며 생산자의 세마포어를 증가시킨다.

버퍼에 데이터가 꽉 찼을 때

```
/* 적절한 세마포어 사용*/
while(memory->CQ_count == 10){

}
sem_wait(&memory->full);
sem_wait(&memory->sem);

state = addQ(memory, input);
//공유 메모리 버퍼에 아이템 생산
printf("producer %d add Q %d\n", id, input);
sem_post(&memory->sem);
sem_post(&memory->empty);


/* 적절한 세마포어 사용*/
```

```
 /* 적절한 세마포어 사용*/
while(memory->CQ_count == 0){
}
sem_wait(&memory->empty);
sem_wait(&memory->sem);
output = getQ(memory);
//공유 메모리 버퍼에서 아이템 소비
printf("consumer %d get Q %d\n", id, output);
sem_post(&memory->sem);
sem_post(&memory->full);


   /* 적절한 세마포어 사용*/
```

1. 생산자가 생산을 하고 아직 Time-slice가 다 되지 않았을 때, 다시 생산을 하려 진입한다.
2. 이 때 버퍼 내에 아이템이 꽉 차있다면 while문에서 busy-waiting을 한다.
3. 시간이 다 되어서 CPU가 소비자에게 할당되어 아이템을 소비한다.
4. 소비를 하면서 생산자의 세마포어를 증가 시켜 생산자가 진입할 수 있도록 한다.
5. 다시 생산자에게 CPU가 할당되면, 버퍼내에 아이템이 꽉 차있진 않으므로 while문에서 탈출하여 생산을 다시 시작한다.

버퍼에 데이터가 하나도 없을 때

```
/* 적절한 세마포어 사용*/
while(memory->CQ_count == 10){

}
sem_wait(&memory->full);
sem_wait(&memory->sem);

state = addQ(memory, input);
//공유 메모리 버퍼에 아이템 생산
printf("producer %d add Q %d\n", id, input);
sem_post(&memory->sem);
sem_post(&memory->empty);


/* 적절한 세마포어 사용*/
```

```
 /* 적절한 세마포어 사용*/
while(memory->CQ_count == 0){
}
sem_wait(&memory->empty);
sem_wait(&memory->sem);
output = getQ(memory);
//공유 메모리 버퍼에서 아이템 소비
printf("consumer %d get Q %d\n", id, output);
sem_post(&memory->sem);
sem_post(&memory->full);


   /* 적절한 세마포어 사용*/
```

1. 소비자가 소비를 하고 아직 Time-slice가 다 되지 않았을 때, 다시 소비를 하기 위해진입한다.
2. 이 때 버퍼 내에 아이템이 하나도 없다면, while문에서 busy-waiting을 한다.
3. 시간이 다 되어서 CPU가 생산자에게 할당되어 아이템을 생산한다.
4. 생산를 하면서 생산자의 세마포어를 증가 시켜 소비자가 진입할 수 있도록 한다.
5. 다시 소비자에게 CPU가 할당되면, 버퍼내에 아이템이 생겼으므로 while문에서 탈출하여 생산을 다시 시작한다.

결과분석

addQ()와 getQ() 내에 있는 메세지의 출력없이 잘 실행되고 있다.
버퍼내 데이터 갯수가 10개, 0개 일때 메세지 출력
만일 버퍼가 꽉 차거나 버퍼가 비어있다면 while문에서 busy-waiting을 하며 대기하므로 메세지 출력이 되질 않음.

메세지 출력 갯수가 60개 이다.
생산자 5개 x 6회씩 데이터 입력
소비자 2개 x 15회씩 데이터 소비

생산자 및 소비자가 정해진 횟수의 입력과 소비를 마친 후 End 메세지를 출력.

공유 메모리가 성공적으로 삭제되었다는 메세지 출력.

Semaphore를 이용하여 mutex개념을 구현.
공유 메모리를 사용하는 critical section에는 단 1개의 스레드만 접근할 수 있다.

```
parallels@ubuntu:~/Desktop/OS/11$ ./homework
producer 2 add Q 85
producer 3 add Q 86
producer 2 add Q 88
producer 1 add Q 84
producer 3 add Q 89
consumer 1 get Q 85
producer 1 add Q 87
consumer 2 get Q 86
producer 2 add Q 79
consumer 1 get Q 88
producer 5 add Q 88
consumer 2 get Q 84
producer 3 add Q 80
consumer 1 get Q 89
producer 4 add Q 87
consumer 1 get Q 87
producer 1 add Q 78
consumer 2 get Q 79
producer 2 add Q 17
consumer 2 get Q 88
producer 3 add Q 18
consumer 1 get Q 80
producer 5 add Q 91
consumer 2 get Q 87
consumer 1 get Q 78
producer 1 add Q 16
producer 3 add Q 96
consumer 2 get Q 17
producer 2 add Q 95
consumer 1 get Q 18
producer 5 add Q 82
consumer 2 get Q 91
producer 4 add Q 90
consumer 1 get Q 16
producer 3 add Q 38
****Producer 3 END****
consumer 2 get Q 96
producer 4 add Q 81
consumer 1 get Q 95
consumer 2 get Q 82
producer 2 add Q 37
****Producer 2 END****
consumer 2 get Q 90
consumer 1 get Q 38
producer 4 add Q 19
producer 1 add Q 94
producer 5 add Q 20
consumer 2 get Q 81
producer 1 add Q 36
****Producer 1 END****
consumer 1 get Q 37
producer 4 add Q 97
consumer 2 get Q 19
producer 5 add Q 98
consumer 1 get Q 94
producer 4 add Q 39
****Producer 4 END****
consumer 1 get Q 20
producer 5 add Q 40
****Producer 5 END****
consumer 2 get Q 36
consumer 1 get Q 97
consumer 2 get Q 98
consumer 1 get Q 39
****Consumer 1 END****
consumer 2 get Q 40
****Consumer 2 END****
Shared memory removed
```