SCSC 운영체제 및 실습 보고서

9주차 - 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

pthread_mutext_lock()
critical section에 진입하기 위한 함수
쓰레드가 critical section에 진입하면서 이 함수를 지나가게 된다면 lock이 걸려 다른 스레드는 접근할 수 없게 된다.
이미 다른 쓰레드가 진입해있으면 lock이 걸려있는 상태이기 때문에 진입하지 못하고 대기한다.

pthread_mutext_unlock()
critical section을 빠져나오면서 다른 쓰레드가 critical section에 접근할 수 있도록 lock을 푼다.

pthread_cond_wait(cond, mutex)
thread를 휴면 상태로 만들기 위해 사용하는 함수.
cond로 시그널이 전달되기를 기다림
이 함수를 사용하면 mutex_unlock을 호출하여 mutex를 풀어줌
시그널을 받으면 mutex_lock을 호출하여 mutex를 잠금

pthread_cond_signal(cond)
cond에 signal을 보내어 pthread_cond_wait를 콜하여 대기중인 쓰레드를 깨움

기본 상황

```
62 //생산자 쓰레드 함수
63 void *producer(void *arg)
64 {
65         //필요한 변수 선언
66         int i;
67         int input;
68         int id;
69         id = pc++;
70
71         for(i = 0; i<P_COUNT; i++)
72         {
73
74                 input = random()%100; //생산할 랜덤 숫자
75                 usleep(input);
76
77
78
79 1              pthread_mutex_lock(&item_lock);
80
81 2              while(CQ_count > 9) {   //if buffer is over Capacity
82                 pthread_cond_wait(&slots,&item_lock);//wait signal
83                 }
84
85                 /*뮤텍스를 이용하여 버퍼에 삽입하는 함수 구현*/
86 3               addQ(input);
87                 printf("producer %d add Q %d\n",id, input);
88                 /*뮤텍스를 이용하여 버퍼에 삽입하는 함수 구현*/
89
90 4               pthread_mutex_unlock(&item_lock);//
91 5               pthread_cond_signal(&items);
92
93
94 }
95 }
```

1. 스레드가 critical section에 진입하면서 다른 스레드는 들어오지 못하도록 lock 을 한다
2. 버퍼를 확인한다. 버퍼가 꽉 차 있지 않다면 그냥 통과한다.
3. 버퍼에 데이터를 입력하고 출력문을 실행한다.
4. 스레드가 나오면서 다른 스레드가 접근할 수 있도록 unlock한다.
5. 데이터를 넣었다는 신호를 보낸다. 이때 cpu가 그대로 생산자에게 할당되어 있다면 for문의 처음으로 돌아가 다시 시작한다.

```
96 //소비자 쓰레드 함수
97 void *consumer(void *arg)
98 {
99         //필요한 변수 선언
100        int i;
101        int output;
102        int id;
103        id = cc++;
104
105        for(i = 0; i<C_COUNT; i++)
106        {
107
108                usleep(random()%100);
109
110 6              pthread_mutex_lock(&item_lock);
111
112 7              while(CQ_count < 1 ) {
113                pthread_cond_wait(&items, &item_lock);
114
115                }
116
117
118                /*뮤텍스를 이용하여 버퍼에서 꺼내오는 함수 구현*/
119 8              output=getQ();
120                printf("consumer %d get Q %d\n",id, output);
121
122                /*뮤텍스를 이용하여 버퍼에서 꺼내오는 함수 구현*/
123
124 9              pthread_mutex_unlock(&item_lock);
125 10             pthread_cond_signal(&slots);
126
127
128        }
129 }
130
```

6. Cpu가 소비자에게 넘어온다면, 소비자 또한 생산자와 비슷한 방식으로 lock을 걸며 진행한다.
7. 버퍼 내에 데이터가 없는지 확인한다. 있으면 그냥 통과한다.
8. 버퍼내의 아이템을 꺼내어 출력한다.
9. 다른 스레드가 접근할 수 있도록 unlock한다.
10. 데이터를 빼갔다는 신호를 보낸다.

버퍼에 데이터가 꽉 찼을 때

```
62 //생산자 쓰레드 함수
63 void *producer(void *arg)
64 {
65         //필요한 변수 선언
66         int i;
67         int input;
68         int id;
69         id = pc++;
70
71         for(i = 0; i<P_COUNT; i++)
72         {
73
74                 input = random()%100; //생산할 랜덤 숫자
75                 usleep(input);
76
77
78
79                 pthread_mutex_lock(&item_lock);
80
81                 while(CQ_count > 9) {    //if buffer is over Capacity
82                 pthread_cond_wait(&slots,&item_lock); //wait signal
83                 }
84
85                 /*뮤텍스를 이용하여 버퍼에 삽입하는 함수 구현*/
86                 addQ(input);
87                 printf("producer %d add Q %d\n",id, input);
88                 /*뮤텍스를 이용하여 버퍼에 삽입하는 함수 구현*/
89
90                 pthread_mutex_unlock(&item_lock);//
91                 pthread_cond_signal(&items);
92
93
94 }
95 }
```

```
96 //소비자 쓰레드 함수
97 void *consumer(void *arg)
98 {
99         //필요한 변수 선언
100        int i;
101        int output;
102        int id;
103        id = cc++;
104
105        for(i = 0; i<C_COUNT; i++)
106        {
107
108                usleep(random()%100);
109
110                pthread_mutex_lock(&item_lock);
111
112                while(CQ_count < 1 ) {
113                pthread_cond_wait(&items, &item_lock);
114
115                }
116
117
118                /*뮤텍스를 이용하여 버퍼에서 꺼내오는 함수 구현*/
119                output=getQ();
120                printf("consumer %d get Q %d\n",id, output);
121
122                /*뮤텍스를 이용하여 버퍼에서 꺼내오는 함수 구현*/
123
124                pthread_mutex_unlock(&item_lock);
125                pthread_cond_signal(&slots);
126
127
128        }
129 }
130
```

1. 스레드가 critical section으로 들어오면서 lock을 건다.
2. 이 때 버퍼 내에 데이터가 꽉 차있다면 더 이상 생산하면 안되므로 while내에서 block상태에 빠진다. 이 때 lock을 걸어놓은 mutex를 cond_wait을 하면서 unlock한다.
3. 모든 생산자가 busy waiting 상태로 cpu할당시간이 지나 소비자에게 간다면, 소비자는 lock을 걸고 critical section내부로 진입하여 소비를 시작한다.
4. 소비를 다 하고 나오면 &slots에 signal을 보낸다

5. Signal을 받은 wait()함수는 다시 다른 스레드가 접근하지 못하도록 mutex를 lock한다.
6. 버퍼에 데이터를 삽입한다.
7. 스레드가 빠져나오면서 mutex를 unlock한다.

버퍼에 데이터가 하나도 없을 때

```
62 //생산자 쓰레드 함수
63 void *producer(void *arg)
64 {
65         //필요한 변수 선언
66         int i;
67         int input;
68         int id;
69         id = pc++;
70
71         for(i = 0; i<P_COUNT; i++)
72         {
73
74                 input = random()%100; //생산할 랜덤 숫자
75                 usleep(input);
76
77
78
79    1, 8         pthread_mutex_lock(&item_lock);
80
81    2            while(CQ_count > 9) {   //if buffer is over Capacity
82                 pthread_cond_wait(&slots,&item_lock);//wait signal
83                 }
84
85                 /*뮤텍스를 이용하여 버퍼에 삽입하는 함수 구현*/
86    3            addQ(input);
87                 printf("producer %d add Q %d\n",id, input);
88                 /*뮤텍스를 이용하여 버퍼에 삽입하는 함수 구현*/
89
90    4            pthread_mutex_unlock(&item_lock);//
91    5, 9         pthread_cond_signal(&items);
92
93
94 }
95 }
```

```
96 //소비자 쓰레드 함수
97 void *consumer(void *arg)
98 {
99         //필요한 변수 선언
100        int i;
101        int output;
102        int id;
103        id = cc++;
104
105        for(i = 0; i<C_COUNT; i++)
106        {
107
108                usleep(random()%100);
109
110           6    pthread_mutex_lock(&item_lock);
111
112             7, while(CQ_count < 1 ) {
113             10 pthread_cond_wait(&items, &item_lock);
114
115                }
116
117
118                /*뮤텍스를 이용하여 버퍼에서 꺼내오는 함수 구현*/
119                output=getQ();
120                printf("consumer %d get Q %d\n",id, output);
121
122                /*뮤텍스를 이용하여 버퍼에서 꺼내오는 함수 구현*/
123
124                pthread_mutex_unlock(&item_lock);
125                pthread_cond_signal(&slots);
126
127
128        }
129 }
130
```

1. 스레드가 critical section에 진입하면서 다른 스레드는 들어오지 못하도록 lock 을 한다
2. 버퍼를 확인한다. 버퍼가 꽉 차 있지 않다면 그냥 통과한다.
3. 버퍼에 데이터를 입력하고 출력문을 실행한다.
4. 스레드가 나오면서 다른 스레드가 접근할 수 있도록 unlock한다.
5. 데이터를 넣었다는 신호를 보낸다. 이때 cpu가 그대로 생산자에게 할당되어 있다면 for문의 처음으로 돌아가 다시 시작한다.

6. 소비자 또한 mutex에 lock을 걸고 critical section으로 진입한다.
7. 이때, 버퍼 내에 데이터가 없다면 while문에서 소비자가 아이템을 넣어줄 때 까지 block상태에 빠진다. 또한 mutex의 lock을 해제한다.
8. Cpu 할당 시간이 다 지나 소비자에게 cpu가 넘어가면, cpu는 다시 생산을 시작한다.
9. 생산이 끝난 소비자는 signal을 보낸다.
10. Signal을 받은 wait()함수는 mutex를 다시 lock시키고 소비를 진행한다.

결과분석

addQ()와 getQ() 내에 있는 메세지의 출력없이 잘 실행되고 있다.
버퍼내 데이터 갯수가 10개, 0개 일때 메세지 출력

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

Mutex와 signal을 이용하여 critical section에는 단 1개의 스레드만 접근할 수 있다.

```
parallels@ubuntu:~/Desktop/OS/9$ ./homework
producer 4 add Q 15
producer 1 add Q 83
producer 5 add Q 93
producer 2 add Q 86
consumer 1 get Q 15
consumer 2 get Q 83
producer 3 add Q 77
producer 5 add Q 21
producer 2 add Q 62
producer 4 add Q 92
producer 1 add Q 49
consumer 1 get Q 93
producer 3 add Q 59
producer 2 add Q 26
producer 5 add Q 63
producer 4 add Q 40
consumer 1 get Q 86
consumer 2 get Q 77
producer 2 add Q 11
producer 1 add Q 26
consumer 1 get Q 21
consumer 2 get Q 62
producer 5 add Q 68
producer 3 add Q 36
consumer 1 get Q 92
producer 4 add Q 67
consumer 2 get Q 49
producer 1 add Q 62
consumer 1 get Q 59
producer 2 add Q 82
consumer 2 get Q 26
producer 5 add Q 67
consumer 1 get Q 63
producer 3 add Q 29
consumer 2 get Q 40
producer 4 add Q 22
consumer 1 get Q 11
consumer 2 get Q 26
producer 2 add Q 93
producer 1 add Q 69
consumer 1 get Q 68
producer 5 add Q 56
consumer 2 get Q 36
producer 4 add Q 73
consumer 1 get Q 67
producer 3 add Q 29
consumer 1 get Q 62
consumer 2 get Q 82
producer 1 add Q 84
consumer 1 get Q 67
producer 3 add Q 70
consumer 2 get Q 29
consumer 1 get Q 22
consumer 2 get Q 93
consumer 2 get Q 69
consumer 1 get Q 56
consumer 2 get Q 73
consumer 1 get Q 29
consumer 2 get Q 84
consumer 2 get Q 70
parallels@ubuntu:~/Desktop/OS/9$
```