< 운영체제 과제 5 >

2017253041_홍성우

#1

(1)

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define BUFSIZE 2
#define MAXDELAY 3000000
#define rand2() ((double)rand() / RAND_MAX)
struct buffer {
        int buf[BUFSIZE];
        int in;
        int out;
        int count;
};
void initbuf(struct buffer *p);
void *producer(void *arg);
void *consumer(void *arg);
void putmsg(struct buffer *p, int msg);
int getmsg(struct buffer *p);


int main()
{
        struct buffer bbuf;
        pthread_t tid1, tid2;

        initbuf(&bbuf);

        pthread_create(&tid1, NULL, producer, &bbuf);
        pthread_create(&tid2, NULL, consumer, &bbuf);

        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
        return 0;
}

void initbuf(struct buffer *p)
{
        p->in = p->out = p->count = 0;
}
void *producer(void *arg)
{
        struct buffer *p = (struct buffer *) arg;
        int msg = 0;
        while (1) {
                usleep(rand2() *MAXDELAY);
                msg++;
                printf("Producer inserts message %d\n", msg);
                putmsg(p, msg);
        }
}
void *consumer(void *arg)
{
        struct buffer *p = (struct buffer *) arg;
        int msg;
        while (1) {
                usleep(rand2() *MAXDELAY);
                msg = getmsg(p);
```

```
                msg = getmsg(p);
                printf("Consumer removes message %d\n", msg);
        }
}
void putmsg(struct buffer *p, int msg)
{
        while (p->count == BUFSIZE);
        p->count++;
        p->buf[p->in] = msg;
        p->in = (p->in + 1) % BUFSIZE;
        printf("putmsg: buffer size = %d\n", p->count);
}
int getmsg(struct buffer *p)
{
        int msg;
        while (p->count == 0);
        p->count--;
        msg = p->buf[p->out];
        p->out = (p->out + 1) % BUFSIZE;
        printf(" > getmsg: buffer size = %d\n", p->count);
        return msg;
}
```

```
u17041@solgae:~/hw$ cc -o prod-cons prod-cons.c -lpthread
u17041@solgae:~/hw$ ls
prod-cons  prod-cons.c
u17041@solgae:~/hw$ ./prod-cons
Producer inserts message 1
putmsg: buffer size = 0
 > getmsg: buffer size = 0
Consumer removes message 1
Producer inserts message 2
putmsg: buffer size = 1
 > getmsg: buffer size = 0
Consumer removes message 2
Producer inserts message 3
putmsg: buffer size = 1
 > getmsg: buffer size = 0
Consumer removes message 3
^C
```

*producer, *consumer 각각 두개의 쓰레드로 동작하는 프로그램이다, 이 두 쓰레드는 bbuf 라는 버퍼를 이용해 메시지를 주고받는다. *producer는 임의의 시간 지연 후 메시지를 보내는 동작을, *consumer는 메시지를 받는 동작을 반복한다. 이때, initbuf로 버퍼를 초기화하고 지연시간은 난수 발생 함수를 사용하여 0과 MAXDELAY 사이의 시간을 얻어서 사용하며 시간 지연은 usleep함수를 이용한다. putmsg는 count를 증가시키면서 메시지를 넣고 버퍼사이즈 만큼만 입력하기위해 멈춘다. getmsg는 count를 감소시키면서 메시지를 뽑고 버퍼가 비어 있을 때까지 가져오고 멈춘다.

(2)

POSIX Pthreads, Windows 및 Java의 세 가지 주요 스레드 라이브러리가 사용된다. POSIX 표준의 스레드 확장인 Pthread는 사용자 수준 또는 커널 수준 라이브러리로 제공될 수 있다. Windows 스레드 라이브러리는 Windows 시스템에서 사용할 수 있는 커널 수준 라이브러리다. Java 스레드 API를 사용하면 Java 프로그램에서 스레드를 직접 만들고 관리할 수 있다. UNIX 및 Linux 시스템은 주로 Pthread를 사용한다.

#2

(1)

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define BUFSIZE 2
#define MAXDELAY 3000000
#define rand2() ((double)rand() / RAND_MAX)
struct buffer {
        int buf[BUFSIZE];
        int in;
        int out;
        int count;
};
void initbuf(struct buffer *p);
void *producer(void *arg);
void *consumer(void *arg);
void putmsg(struct buffer *p, int msg);
int getmsg(struct buffer *p);

sem_t sem;

int main()
{
        struct buffer bbuf;
        pthread_t tid1, tid2;

        sem_init(&sem, 0, 1);
        initbuf(&bbuf);

        pthread_create(&tid1, NULL, producer, &bbuf);
        pthread_create(&tid2, NULL, consumer, &bbuf);

        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
        return 0;
}
void initbuf(struct buffer *p)
{
        p->in = p->out = p->count = 0;
}

void *producer(void *arg)
{
        struct buffer *p = (struct buffer *) arg;
        int msg = 0;
        while (1) {
                usleep(rand2() *MAXDELAY);
                msg++;
                printf("Producer inserts message %d\n", msg);
                putmsg(p, msg);
        }
}

void *consumer(void *arg)
{
        struct buffer *p = (struct buffer *) arg;
        int msg;
```

```
        int msg;
        while (1) {
                usleep(rand2() *MAXDELAY);
                msg = getmsg(p);
                printf("Consumer removes message %d\n", msg);
        }
}

void putmsg(struct buffer *p, int msg)
{
        while (p->count == BUFSIZE);
        sem_wait(&sem);
        p->count++;
        p->buf[p->in] = msg;
        p->in = (p->in + 1) % BUFSIZE;
        printf("[putmsg] buffer size = %d\n", p->count);
        sem_post(&sem);
}

int getmsg(struct buffer *p)
{
        int msg;

        while (p->count == 0);
        sem_wait(&sem);
        p->count--;
        msg = p->buf[p->out];
        p->out = (p->out + 1) % BUFSIZE;
        printf("[getmsg] buffer size = %d\n", p->count);
        sem_post(&sem);
        return msg;
}
```

```
u17041@solgae:~/prog$ cc -o synsema synsema.c -lpthread
u17041@solgae:~/prog$ ls
synsema  synsema.c
u17041@solgae:~/prog$ ./synsema
Producer inserts message 1
[putmsg] buffer size = 1
[getmsg] buffer size = 0
Consumer removes message 1
Producer inserts message 2
[putmsg] buffer size = 1
[getmsg] buffer size = 0
Consumer removes message 2
Producer inserts message 3
[putmsg] buffer size = 1
[getmsg] buffer size = 0
Consumer removes message 3
Producer inserts message 4
[putmsg] buffer size = 1
Producer inserts message 5
[putmsg] buffer size = 2
^C
```

putmsg에서 쓰기 작업을 하기 전에 sem_wait를 이용해서 세마포어를 제한한다. 그리고 버퍼에 메시지를 쓴 다음 sem_post를 이용해서 세마포어 제한을 푼다. 같은 방법으로 getmsg에서 쓰기 작업을 하기 전에 sem_wait를 이용해서 세마포어를 제한한다. 그리고 버퍼에 메시지를 읽은 다음 sem_post를 이용해서 세마포어 제한을 푼다. 이렇게 하면, 데이터를 읽고 쓰는 동안 임계구역에서 하나의 프로세스만 작업할 수 있게 된다.

(2)

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define BUFSIZE 2
#define MAXDELAY 3000000 // (3초)
#define rand2() ((double)rand() / RAND_MAX)
struct buffer {
        int buf[BUFSIZE];
        int in;
        int out;
        int count;
};
void initbuf(struct buffer *p);
void *producer(void *arg);
void *consumer(void *arg);
void putmsg(struct buffer *p, int msg);
int getmsg(struct buffer *p);

pthread_mutex_t mutex_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t thread_cond = PTHREAD_COND_INITIALIZER;

int main()
{
        struct buffer bbuf;
        pthread_t tid1, tid2;

        pthread_mutex_init(&mutex_lock, NULL);
        pthread_cond_init(&thread_cond, NULL);
        initbuf(&bbuf);

        pthread_create(&tid1, NULL, producer, &bbuf);
        pthread_create(&tid2, NULL, consumer, &bbuf);

        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
        return 0;
}

void initbuf(struct buffer *p)
{
        p->in = p->out = p->count = 0;
}
void *producer(void *arg)
{
        struct buffer *p = (struct buffer *) arg;
        int msg = 0;
        while (1) {
                usleep(rand2() *MAXDELAY);
                msg++;
                printf("Producer inserts message %d\n", msg);
                putmsg(p, msg);
        }
}
void *consumer(void *arg)
{
        struct buffer *p = (struct buffer *) arg;
```

```
        struct buffer *p = (struct buffer *) arg;
        int msg;
        while (1) {
                usleep(rand2() *MAXDELAY);
                msg = getmsg(p);
                printf("Consumer removes message %d\n", msg);
        }
}
void putmsg(struct buffer *p, int msg)
{
        while (p->count == BUFSIZE);
        pthread_mutex_lock(&mutex_lock);
        p->count++;
        p->buf[p->in] = msg;
        p->in = (p->in + 1) % BUFSIZE;
        printf("[putmsg] buffer size = %d\n", p->count);
        pthread_cond_signal(&thread_cond);
        pthread_mutex_unlock(&mutex_lock);
}
int getmsg(struct buffer *p)
{
        int msg;
        pthread_mutex_lock(&mutex_lock);
        while (p->count == 0)
                pthread_cond_wait(&thread_cond, &mutex_lock);
        p->count--;
        msg = p->buf[p->out];
        p->out = (p->out + 1) % BUFSIZE;
        printf("[getmsg] buffer size = %d\n", p->count);
        pthread_cond_signal(&thread_cond);
        pthread_mutex_unlock(&mutex_lock);
        return msg;
}
```

```
u17041@solgae:~/prog$ cc -o synmu synmu.c -lpthread
u17041@solgae:~/prog$ ls
synmu  synmu.c
u17041@solgae:~/prog$ ./synmu
Producer inserts message 1
[putmsg] buffer size = 1
[getmsg] buffer size = 0
Consumer removes message 1
Producer inserts message 2
[putmsg] buffer size = 1
[getmsg] buffer size = 0
Consumer removes message 2
Producer inserts message 3
[putmsg] buffer size = 1
[getmsg] buffer size = 0
Consumer removes message 3
Producer inserts message 4
[putmsg] buffer size = 1
Producer inserts message 5
[putmsg] buffer size = 2
^C
```

putmsg에서는 count, buf, in을 수정하는 문장을 mutex lock을 획득상태에서 수행하고 수행 종료 후 lock을 반환한다. lock반환 후에 buffer에 데이터가 하나가 채워졌음을 알리고, getmsg에서는 반대로 비워졌음을 알린다. 이렇게 하면 공유 자원 공간에 대한 접근 시간 제어로 동기화를 달성하므로 문제를 해결할 수 있다.