Fun with File Descriptors 레포트

2019-13674
양현서
바이오시스템소재학부

1 Producer-Consumer on an n-element Buffer

Producer와 Consumer가 하나씩밖에 없을 때, 세마포어 사용하지 않고 큐 만들어서 동기화 문제 해결하는 방법을 생각해 보세요. 큐의 크기는 문제를 단순화하여 n-sized array 로 하자. 이런 경우에 어떻게 하면 락 없이 잘 동작을 할 수 있겠는지 코드를 짜 보시오.

`read_index`와 `write_index`를 사용하고, `read_index < write_index`이 되게 관리한다.

```
#define NITERS 5
#define n NITERS
int arr[n];
int read_index=0;
int write_index=0;
int main() {
    pthread_t tid_producer;
    pthread_t tid_consumer;
    /* Create threads and wait */
     Pthread_create(&tid_producer, NULL, producer, NULL);
     Pthread_create(&tid_consumer, NULL, consumer, NULL);
     Pthread_join(tid_producer, NULL);
     Pthread_join(tid_consumer, NULL);
     exit(0);
}

void *producer(void *arg) {
    int i, item;
    for (i=0; i<NITERS; i++) {
        /* Produce item */
        item = i;
        printf("produced %d\n",item);
        /* Write item to buf */
        arr[write_index%n] = item;
        write_index++; // int overflow는 n이 작으므로 고려하지 않는다.
    }
    return NULL;
}

void *consumer(void *arg) {
    int i, item;
    for (i=0; i<NITERS; i++) {
        /* Read item from buf */
        while(read_index>= write_index) {
            sleep(1); // wait 등
        }
        item = arr[read_index%n];
        read_index++; // int overflow는 n이 작으므로 고려하지 않는다.
        /* Consume item */
        printf("consumed %d\n\, item);
    }
    return NULL;
```

```
}
```

2 Second readers-writers problem (favors writers)

우리가 배운 예인 first readers-writers problem에서는 reader가 읽고 있으면 writer는 쓸 수 없으며 기다려야 한다. 그러나 writer가 기다리는 동안에 새로운 reader가 들어오면 그 reader는 기다리고 있는 writer를 앞지른다. 이런 상황에서는 계속 reader가 들어올 경우 writer는 무한히 기다리게 되는 starvation 상황이 생길지도 모른다.
이번에는 writer에 priority를 주자. writer가 기다리고 있으면 뒤에 있는 reader는 앞지를 수 없다. 복잡한 예로, 만약에 reader 뒤에 writer 뒤에 reader 뒤에 writer가 오면 전에는 reader reader writer writer 순으로 처리했지만, 이번에는 reader writer writer reader 순으로 처리되게 하는 코드를 작성해 본다.

원리는 대기하는 reader와 writer, 작업하는 reader와 writer 수를 관리하여 reader가 새로 진입할 때는 작업하거나 기다리는 writer가 없을 때 진입하고, writer가 새로 진입할 때는 작업하거나 기다리는 writer나 작업중인 reader가 없어지면 진입하게 하는 것이다. first readers-writers problem에서는 writer가 진입할 때 기다리는 reader까지 끝날 때까지 기다려야 했다.

```
int realwritecnt; /* Initially 0 */
int waitwritecnt;/* initially 0 */
int realreadcnt; /* initially 0 */
int waitreadcnt;/* initially 0 */
sem_t mutex, readdata_mutex, writedata_mutex; /* initially 1 */
void reader(void) {
    while (1) {
        P(&mutex);
        if(realwritecnt + wailtwritecnt == 0) {
            V(&readdata_mutex); // 미리 락을 푼다.
            realreadcnt++;
        } else
            waitreadcnt++;
        V(&mutex);
        P(&readdata_mutex); // 밀린 reader들이 모이는 곳
        // read

        P(&mutex);
        realreadcnt--;
        if(realreadcnt == 0 && waitwritecnt > 0) { // 다 읽었고 writer가 기다리고
        ↪  있는 것이 있다면 락을 풀어준다.
            V(&writedata_mutex);
            waitwritecnt--;
            realwritecnt++;
        }
        V(&mutex);
    }
}


void writer(void) {
    while (1) {
        P(&mutex);
        if(realreadcnt + realwritecnt + waitwritecnt == 0) {
            V(&writedata_mutex); // 미리 락을 푼다
            realwritecnt++;
        } else
            waitwritecnt++;
        V(&mutex);
        P(&writedata_mutex); // 밀린 writer들이 모이는 곳
        /* Writing here */
        P(&mutex);
        realwritecnt--;
        if(waitwritecnt > 0) { // 기다리는  writer가 있다면 writer 락을 풀어준다.
```

```
            V(&writedata_mutex);
            realwritecnt++;
            waitwritecnt--;
        } else if(waitreadcnt >0) { // 기다리는 writer가 없고 기다리는 reader가
        ↪  있다면 락을 풀어준다
            V(&readdata_mutex);
            realreadcnt++;
            waitreadcnt--;
        }
        V(&mutex);
    }
}
```