

# 리눅스 프로그래밍

Readers – Writers(소비자 – 생산자)문제

---

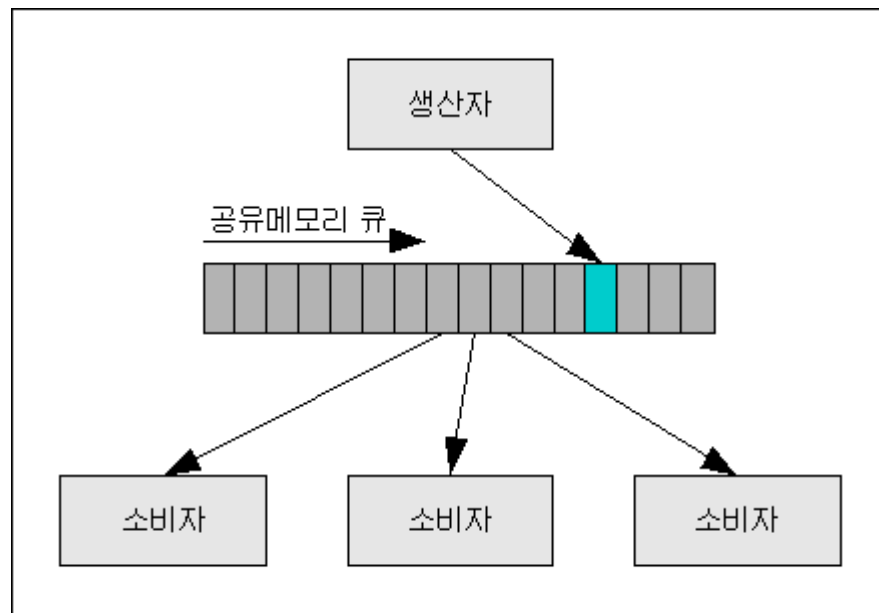
Mcalab

작성자 : 양평우

정용희

# Readers – Writers 문제

- 생산자와 소비자 문제
  - 공유 메모리



# Readers – Writers 문제

## □ 해결방안

### ■ 공유메모리 + 세마포어

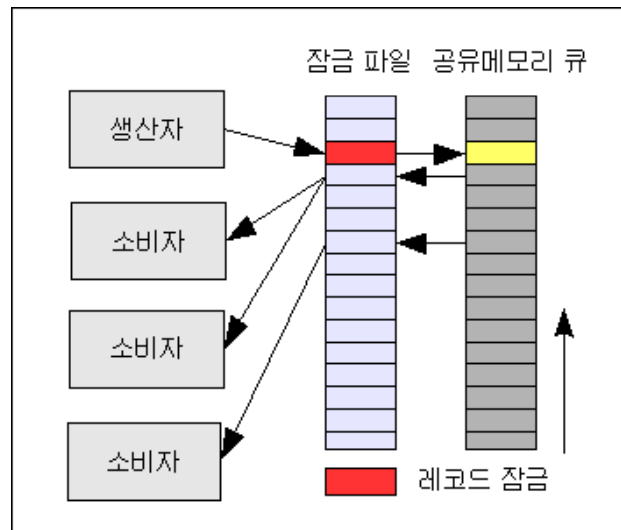
- 생산자가 데이터 쓰기 때 세마포어값 1증가, 소비자가 읽기시 값 1감소

### ■ 공유메모리 + 생산자 통지

- 공유메모리 앞부분에 생산자의 쓰기 위치를 남기는 방법

### ■ 공유메모리 + 파일 레코드 잠금(실습내용)

- 공유메모리와 동일한 잠금 검사 전용 파일 생성 후 생산자가 쓰기 중엔 사용중인 영역(배열)에 해당하는 파일 레코드를 잠그는 방식
- 소비자는 처음 잠금 파일을 확인 후 가장 최근의 생산자의 쓰기영역을 알 수 있음
- 소비자는 그 위치부터 데이터를 읽어나가면 됨



# 생산자 소스코드

## □ Writer.c

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <fcntl.h>

#include <string.h>
#include <unistd.h>

#define QUEUE_SIZE 10

// 생산자와 소비자간 공유할 데이터
struct data
{
    char name[80];
};

struct flock lock, unlock;

int lock_open(int fd, int index)
{
    lock.l_start  = index;
    lock.l_type  = F_WRLCK;
    lock.l_len    = 1;
    lock.l_whence = SEEK_SET;
    return fcntl(fd, F_SETLKW, &lock);
}
```

## 생산자 소스코드(2)

### ❑ Writer.c

```
int lock_close(int fd, int index)
{
    unlock.l_start  = index;
    unlock.l_type   = F_UNLCK;
    unlock.l_len    = 1;
    unlock.l_whence = SEEK_SET;
    return fcntl(fd, F_SETLK, &unlock);
}

void lock_init()
{
    lock.l_start  = 0;
    lock.l_type   = F_WRLCK;
    lock.l_len    = 1;
    lock.l_whence = SEEK_SET;
}

void unlock_init()
{
    unlock.l_start  = 0;
    unlock.l_type   = F_UNLCK;
    unlock.l_len    = 1;
    unlock.l_whence = SEEK_SET;
}
```

# 생산자 소스코드(3)

## □ Writer.c

```
int main()
{
    int shmid;
    int i = 0;
    int offset = 0;

    struct data *cal_num;
    void *shared_memory;
    struct data ldata;
    int fd;

    lock_init();
    unlock_init();

    // 잠금 파일을 생성한다.
    if ((fd = open("shm_lock", O_CREAT|O_RDWR)) < 0)
    {
        perror("file open error ");
        exit(0);
    }
    // 파일을 공유메모리 큐의 크기만큼 만든다.
    write(fd, (void *)'\0', sizeof(char)*QUEUE_SIZE);
```

## 생산자 소스코드(4)

### ❑ Writer.c

```
// 공유메모리를 생성한다.  
// 공유메모리의 크기는 QUEUE_SIZE * 원소의 크기가 된다.  
shmids = shmget((key_t)1234, sizeof(ldata)*QUEUE_SIZE, 0666|IPC_CREAT);  
if (shmids == -1)  
{  
    perror("shmget failed : ");  
    exit(0);  
}  
  
// 공유할 메모리의 크기를 할당하고 이를 공유 메모리영역에 붙인다.  
shared_memory = (void *)malloc(sizeof(ldata)*QUEUE_SIZE);  
shared_memory = shmat(shmids, (void *)0, 0);  
if (shared_memory == (void *)-1)  
{  
    perror("shmat failed : ");  
    exit(0);  
}
```

# 생산자 소스코드(5)

## ❑ Writer.c

```
while(1)
{
    // 공유할 데이터
    sprintf(ldata.name, "write Data : %d\n", i);
    // 이진 디버깅용 출력물
    printf("%d %s", (i==0)? QUEUE_SIZE - 1:i-1, ldata.name);

    // 레코드를 잠근다.
    if(lock_open(fd, i) < 0)
    {
        perror("lock error");
    }
    // 레코드 잠금을 얻었다면
    // 이전 레코드의 잠금을 푼다.
    if(lock_close(fd, (i==0)? QUEUE_SIZE - 1: i-1) < 0)
    {
        perror("flock error");
    }
    // 공유메모리에 데이터를 쓴다.
    memcpy((void *)shared_memory+offset, (void *)&ldata, sizeof(ldata));
    sleep(1);
    offset += sizeof(ldata);
    i++;

    // 이진 순환 큐이다. 만약 큐의 크기를 모두 채웠다면
    // offset과 인덱스 번호 i를 초기화 한다.
    if (i == QUEUE_SIZE) {i = 0; offset = 0;}
}
```



# 소비자 소스코드

## ❑ Reader.c

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <errno.h>
#include <fcntl.h>

#include <string.h>
#include <unistd.h>

#define QUEUE_SIZE 10

struct data
{
    char name[80];
};

struct flock lock, unlock;

int lock_open(int fd, int index)
{
    lock.l_start    = index;
    lock.l_type     = F_WRLCK;
    lock.l_len      = 1;
    lock.l_whence   = SEEK_SET;
    return fcntl(fd, F_SETLKW, &lock);
}
```

## 소비자 소스코드(2)

### ❑ Reader.c

```
int lock_isopen(int fd, int index)
{
    lock.l_start    = index;
    lock.l_type     = F_WRLCK;
    lock.l_len      = 1;
    lock.l_whence   = SEEK_SET;
    return fcntl(fd, F_SETLK, &lock);
}

int lock_close(int fd, int index)
{
    unlock.l_start  = index;
    unlock.l_type   = F_UNLCK;
    unlock.l_len    = 1;
    unlock.l_whence = SEEK_SET;
    return fcntl(fd, F_SETLK, &unlock);
}

void lock_init()
{
    lock.l_start    = 0;
    lock.l_type     = F_WRLCK;
    lock.l_len      = 1;
    lock.l_whence   = SEEK_SET;
}
```

## 소비자 소스코드(3)

### ❑ Reader.c

```
void unlock_init()
{
    unlock.l_start = 0;
    unlock.l_type = F_UNLCK;
    unlock.l_len = 1;
    unlock.l_whence = SEEK_SET;
}

int main()
{
    int shmid;
    int i = 0;
    int offset = 0;
    int fd;

    void *shared_memory;
    struct data *ldata;
    lock_init();
    unlock_init();

    if ((fd = open("shm_lock", O_RDWR)) < 0)
    {
        perror("file open error ");
        exit(0);
    }
```

# 소비자 소스코드(4)

## ❑ Reader.c

```
shmidx = shmget((key_t)1234, sizeof(struct data)*QUEUE_SIZE, 0666);
if (shmidx == -1)
{
    perror("shmget failed : ");
    exit(0);
}

shared_memory = (void *)malloc(sizeof(struct data)*QUEUE_SIZE);
shared_memory = shmat(shmidx, (void *)0, 0);
if (shared_memory == (void *)-1)
{
    perror("shmat failed : ");
    exit(0);
}
```

# 소비자 소스코드(5)

## ❑ Reader.c

```
// 이 부분은 생산자가 가장 최근에 쓴 데이터의 인덱스를
// 찾아내기 위한 코드다.
// 잠금 파일의 레코드를 차례대로 검사하면서 잠금이 있는 부분을 검사한다.
// 잠금이 검사되면, 리턴한다.
while(1)
{
    if(lock_isopen(fd, i)< 0)
    {
        if (errno == EAGAIN)
        {
            printf("Read index is %d %d %d\n", i, EAGAIN, errno);
            fcntl(fd, F_GETLK, &lock); // 코드 1
            offset = sizeof(struct data)*i;
            break;
        }
        else
        {
            printf("Init Error\n");
            exit(0);
        }
    }
    lock_close(fd, i);
    i++;
    if (i == QUEUE_SIZE)
    {
        printf("Server Error\n");
    }
}
```

# 소비자 소스코드(6)

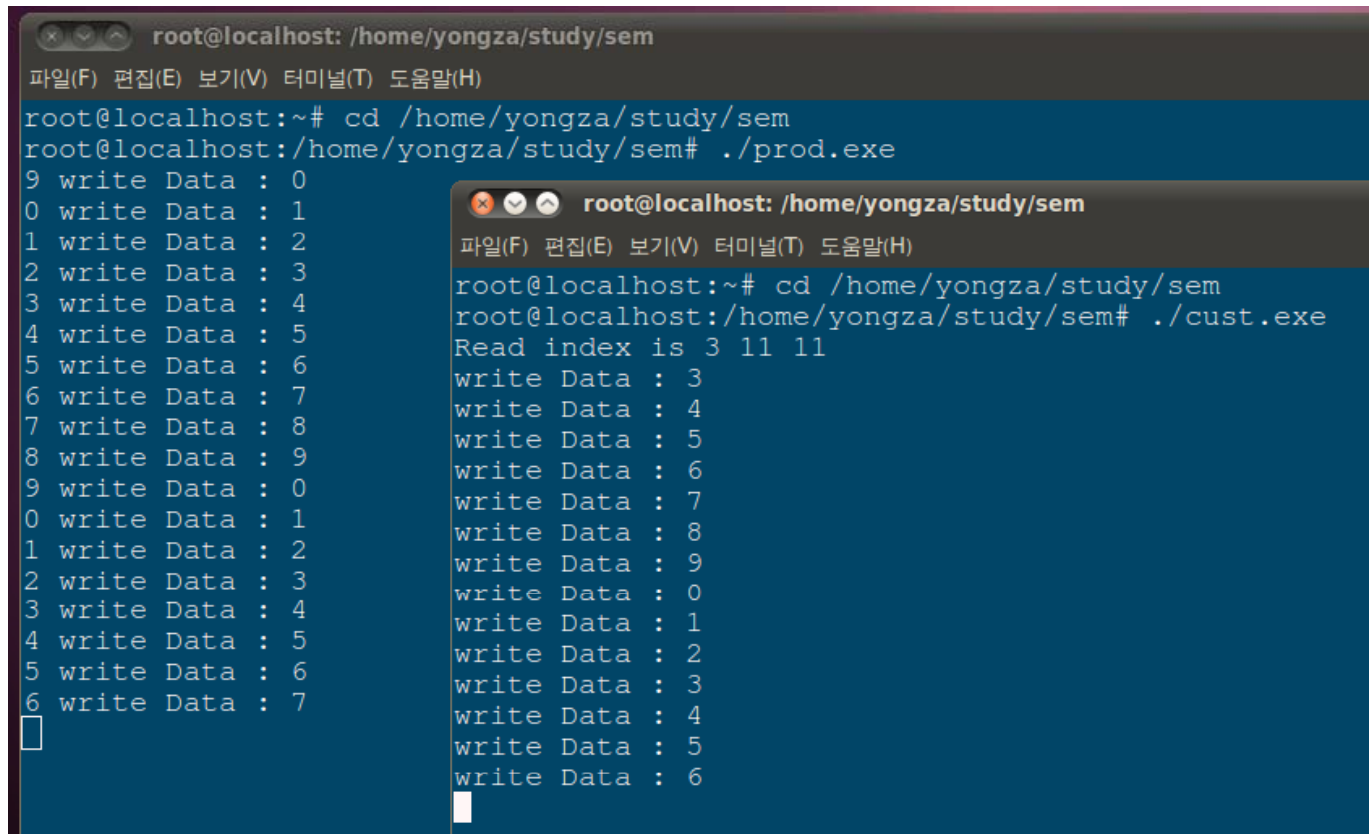
## ❑ Reader.c

```
// 공유 메모리로 부터 데이터를 읽는다.
while(1)
{
    if (lock_open(fd, i) < 0)
    {
        perror("flock error");
    }
    ldata = (struct data *) (shared_memory+offset);
    printf("%s", ldata->name);
    lock_close(fd, i);

    offset += sizeof(struct data);
    i++;
    if (i == QUEUE_SIZE) {i = 0; offset = 0;}
}
}
```

# 소스 코드 실행

- 생산자와 소비자 소스코드 컴파일
  - Reader.c, Writer.c 파일을 각각 gcc 컴파일러를 통해 실행파일 생성
- 또 하나의 터미널창(셸)을 실행하여 각각 생산자와 소비자 실행



The image shows two terminal windows side-by-side. The left window is titled 'root@localhost: /home/yongza/study/sem' and shows the execution of 'prod.exe'. It outputs a sequence of 'write Data : 0' through 'write Data : 7', followed by a blank line. The right window is also titled 'root@localhost: /home/yongza/study/sem' and shows the execution of 'cust.exe'. It outputs 'Read index is 3 11 11' followed by a sequence of 'write Data : 3' through 'write Data : 6'.

```
root@localhost: /home/yongza/study/sem
파일(F) 편집(E) 보기(V) 터미널(T) 도움말(H)
root@localhost:~# cd /home/yongza/study/sem
root@localhost:/home/yongza/study/sem# ./prod.exe
9 write Data : 0
0 write Data : 1
1 write Data : 2
2 write Data : 3
3 write Data : 4
4 write Data : 5
5 write Data : 6
6 write Data : 7
7 write Data : 8
8 write Data : 9
9 write Data : 0
0 write Data : 1
1 write Data : 2
2 write Data : 3
3 write Data : 4
4 write Data : 5
5 write Data : 6
6 write Data : 7
□

root@localhost: /home/yongza/study/sem
파일(F) 편집(E) 보기(V) 터미널(T) 도움말(H)
root@localhost:~# cd /home/yongza/study/sem
root@localhost:/home/yongza/study/sem# ./cust.exe
Read index is 3 11 11
write Data : 3
write Data : 4
write Data : 5
write Data : 6
write Data : 7
write Data : 8
write Data : 9
write Data : 0
write Data : 1
write Data : 2
write Data : 3
write Data : 4
write Data : 5
write Data : 6
```