

운영체제 프로젝트

2016003618 송현수

1. 알고리즘

Writer 선호 알고리즘

큐에 추가된 Writer를 필요이상으로 오래 기다리게 하지 않는다

Integer readcount, writecount 0으로 초기화

Semaphore readmutex, writemutex, rd, contents 1로 초기화

Writer

```
while(1){
    wait(writemutex)
    writecount++
    if(writecount == 1)
        wait(rd)
    signal(writemutex)
    wait(contents)
    <Critical Section>
    signal(contents)
    wait(writemutex)
    writecount--
    if(writecount == 0)
        signal(rd)
    signal(writemutex)
}
```

Reader

```
while(1){
    wait(rd)
    wait(readmutex)
    readcount++
    if(readcount == 1)
        wait(contents)
    signal(readmutex)
    signal(rd)
    <Critical Section>
    wait(readmutex)
    readcount--
    if(readcount == 0)
        signal(contents)
    signal(readmutex)
}
```

Annotations:

- Writer가 진입함을 알림
- 첫번째 writer를 확인
- CS진입할때 Reader를 막음
- race condition을 피하기위함
- 다른 writer와 CS를 같이 쓸수 없음
- Writer가 끝났음을 알림
- 마지막 writer를 확인
- lock을 풀고 CS에 Reader가 들어갈수있게 함
- Reader가 진입하는것을 확인
- reader가 들어온다
- 첫번째 reader임을 확인
- CS에 진입할때 lock을 건다.
- race condition을 피하기위함
- reader CS에 진입하려고 한것이 끝났다고 알려줌
- reader가 종료된다.
- 마지막 reader임을 확인
- lock을 풀다.

writer 선호 알고리즘

공정한 Reader-Writer 알고리즘

어떤 스레드도 Starvation을 허용하지 않는다.

```

Integer readercount = 0
Semaphore rd, contents, fair = 1
Writer
while(1){
    wait(fair)           일단 대기
    wait(contents)        CS에 혼자 들어가기 위함 lock
    signal(fair)
    <Critical Section>
    signal(contents)      다른 reader와 writer를 위해 lock을 풀어줌
}
Reader
while(1){
    wait(fair)           일단 대기
    wait(rd)             readercount에 혼자만 접근하기 위함
    if (readercount == 0) 첫번째 reader임을 확인
        wait(contents)   reader가 CS에 진입함 lock을 걸어서 writer는 못들어감
    readercount++         reader의 개수를 확인
    signal(fair)
    signal(rd)
    <Critical Section>
    wait(rd)             readercount에 혼자만 접근하기 위함
    readercount--         reader가 종료되는 것을 확인
    if(readercount == 0) 마지막 reader임을 확인
        signal(contents) lock을 풀다
    signal(rd)
}

```

공정한 reader-writer 알고리즘

2. 소스 파일

a. Reader-prefer

중복되는 부분은 생략함(이 코드 위쪽으로 똑같음)

```

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t wrt = PTHREAD_MUTEX_INITIALIZER;
int readcount = 0;

void *reader(void *arg) {
    int id, i;
    id = *(int *)arg;
    while (1) {
        pthread_mutex_lock(&mutex);
        readcount++;
        if(readcount == 1) pthread_mutex_lock(&wrt);    //block writer
    }
}

```

```

        pthread_mutex_unlock(&mutex); //reader follow
        /*
         * Begin Critical Section
         */
        printf("<");
        for (i = 0; i < N; ++i)
            printf("%c", 'A'+id);
        printf(">");
        /*
         * End Critical Section
         */
        pthread_mutex_lock(&mutex);
        readcount--;
        if(readcount == 0) pthread_mutex_unlock(&wrt); //enable writer
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(0);
}

void *writer(void *arg) {
    int id, i;
    struct timespec req, rem;
    id = *(int *)arg;
    req.tv_sec = 0;
    req.tv_nsec = 100000000L;
    while (1) {
        pthread_mutex_lock(&wrt);
        /*
         * Begin Critical Section
         */
        printf("\n");
        if (id == 0)
            for (i = 0; i < L; ++i)
                printf("%s\n", t[i]);
        else
            for (i = 0; i < M; ++i)
                printf("%s\n", d[i]);
        /*
         * End Critical Section
         */
        pthread_mutex_unlock(&wrt);
        nanosleep(&req, &rem);
    }
    pthread_exit(0);
}

int main(void)
{
    int i;
    int rarg[RNUM], warg[WNUM];
    pthread_t rthid[RNUM];
    pthread_t wthid[WNUM];
    pthread_attr_t attr;
    struct timespec req, rem;
    pthread_mutex_init(&mutex, NULL); //mutex initiaite

```

```

pthread_mutex_init(&wrt, NULL);
/*
 * Initialize thread attributes
 */
if (pthread_attr_init(&attr) != 0) {
    fprintf(stderr, "pthread_init error\n");
    exit(-1);
}
/*
 * Create RNUM reader threads
 */
for (i = 0; i < RNUM; ++i) {
    rarg[i] = i;
    if (pthread_create(rthid+i, &attr, reader, rarg+i) != 0) {
        fprintf(stderr, "pthread_create error\n");
        exit(-2);
    }
}
/*
 * Create WNUM writer threads
 */
for (i = 0; i < WNUM; ++i) {
    warg[i] = i;
    if (pthread_create(wthid+i, &attr, writer, warg+i) != 0) {
        fprintf(stderr, "pthread_create error\n");
        exit(-2);
    }
}
/*
 * Wait for 0.5 second and terminate
 */
pthread_join(wthid[1], NULL);
pthread_mutex_destroy(&mutex);
pthread_mutex_destroy(&wrt);
req.tv_sec = 0;
req.tv_nsec = 500000000L;
nanosleep(&req, &rem);
exit(0);
}

```

b. writer-prefer

```
#include <semaphore.h> // 맨 위에 추가됨

sem_t read_mutex, write_mutex, contents, rd;
int readcount = 0;
int writecount = 0;

void *reader(void *arg)
{
    int id, i;
    id = *(int *)arg;
    while (1) {
        sem_wait(&rd);
        sem_wait(&read_mutex);
        readcount++;
        if(readcount == 1) sem_wait(&contents);
        sem_post(&read_mutex);
        sem_post(&rd);
        /*          Critical Section          */
        sem_wait(&read_mutex);
        readcount--;
        if(readcount == 0) sem_post(&contents);
        sem_post(&read_mutex);
    }
    pthread_exit(0);
}

void *writer(void *arg)
{
    int id, i;
    struct timespec req, rem;
    id = *(int *)arg;
    req.tv_sec = 0;
    req.tv_nsec = 100000000L;
    while (1) {
        sem_wait(&write_mutex);
        writecount++;
        if(writecount == 1) sem_wait(&rd);
        sem_post(&write_mutex);
        sem_wait(&contents);
        /*          Critical Section          */
        sem_post(&contents);
        sem_wait(&write_mutex);
        writecount--;
        if(writecount == 0) sem_post(&rd);
        sem_post(&write_mutex);
        nanosleep(&req, &rem);
    }
    pthread_exit(0);
}

int main(void) //중복된 것은 삭제하고 추가한 것만 씀
{
```

```

    sem_init(&contents, 0, 1);
    sem_init(&rd, 0, 1);
    sem_init(&read_mutex, 0, 1);
    sem_init(&write_mutex, 0, 1);
    err, "pthread_init error\n");
        exit(-1);
    }

    pthread_join(wthid[1], NULL);
    sem_destroy(&contents);
    sem_destroy(&rd);
    sem_destroy(&write_mutex);
    sem_destroy(&read_mutex);
}

```

c. 공정한 reader-writer 알고리즘

```

#include <semaphore.h> // 맨 위에 추가됨

int readcount = 0;
sem_t contents, fair, rd;

void *reader(void *arg) {
    중복삭제
    while (1) {
        sem_wait(&fair);
        sem_wait(&rd);
        if(readcount == 0) sem_wait(&contents);
        readcount++;
        sem_post(&fair);
        sem_post(&rd);
        /*          Critical Section          */
        sem_wait(&rd);
        readcount--;
        if(readcount == 0) sem_post(&contents);
        sem_post(&rd);
    }
    pthread_exit(0);
}

void *writer(void *arg)
{
    중복삭제
    while (1) {
        sem_wait(&fair);
        sem_wait(&contents);
        sem_post(&fair);
        /*          Critical Section          */
        sem_post(&contents);
        nanosleep(&req, &rem);
    }
}

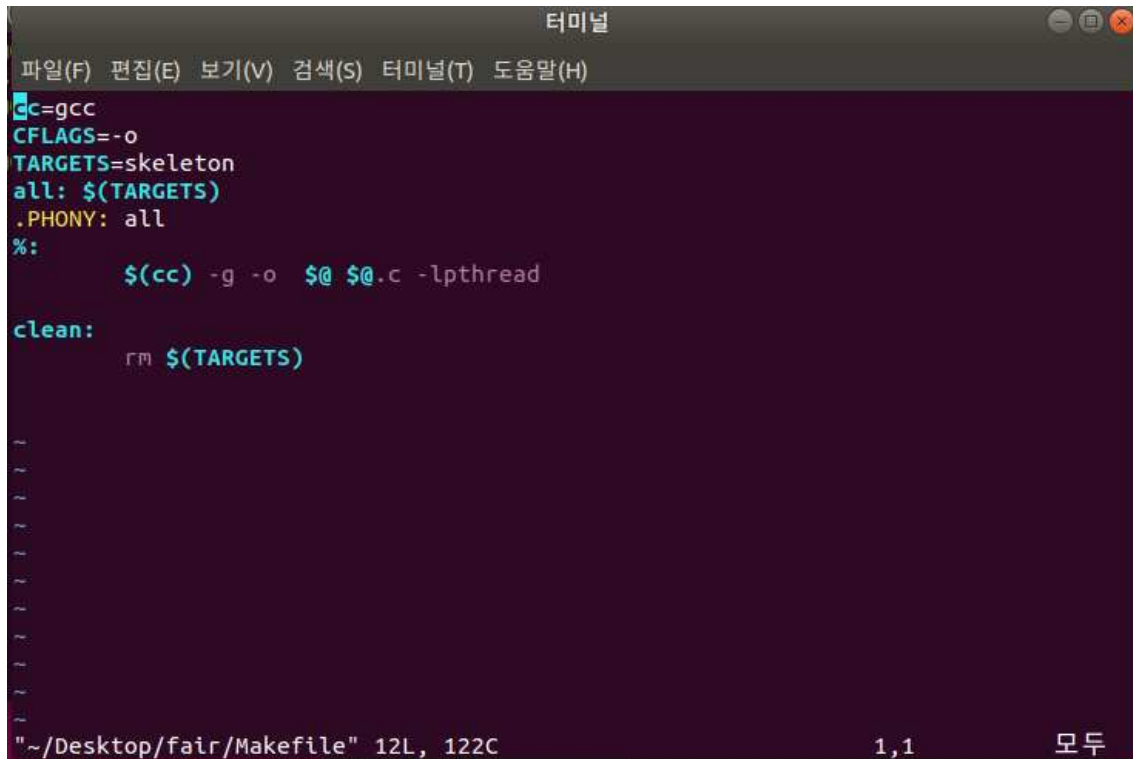
```

```
    pthread_exit(0);
}
int main(void)          //중복된 것은 삭제하고 추가한 것만 씀
{
    sem_init(&contents, 0, 1);
    sem_init(&rd, 0, 1);
    sem_init(&fair, 0, 1);

    pthread_join(wthid[1], NULL);
    sem_destroy(&contents);
    sem_destroy(&rd);
    sem_destroy(&fair);
}
```

3. 컴파일 과정 화면 캡처

Make file을 만들었다.

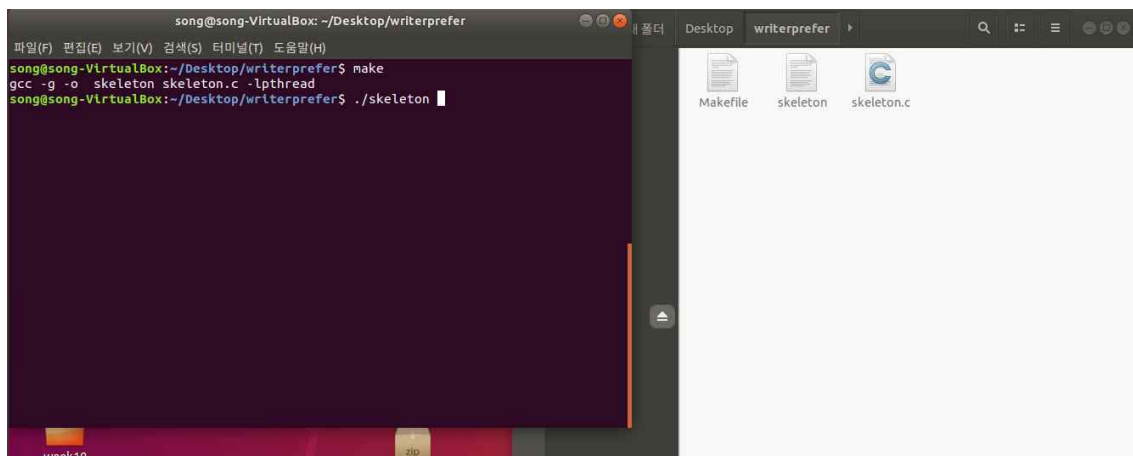


```
터미널
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
c=gcc
CFLAGS=-o
TARGETS=skeleton
all: $(TARGETS)
.PHONY: all
%:
    $(cc) -g -o $@ $@.c -lpthread

clean:
    rm $(TARGETS)

~/Desktop/fair/Makefile" 12L, 122C 1,1 모두
```

make로 컴파일하였다.

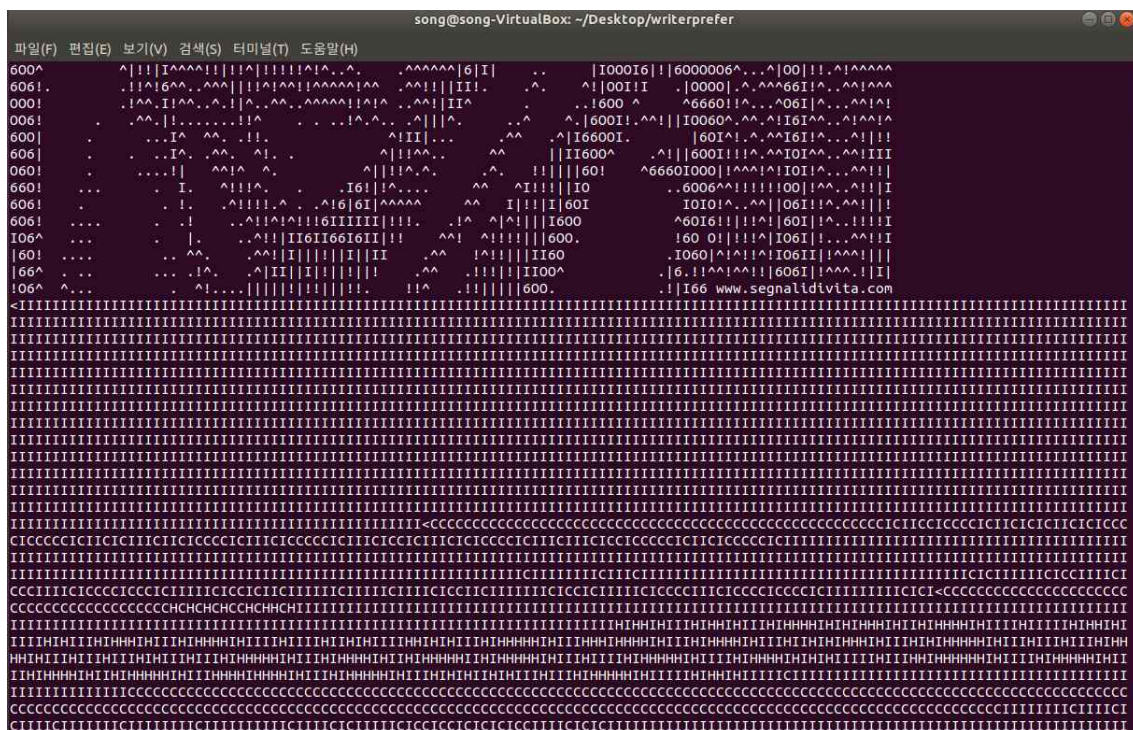
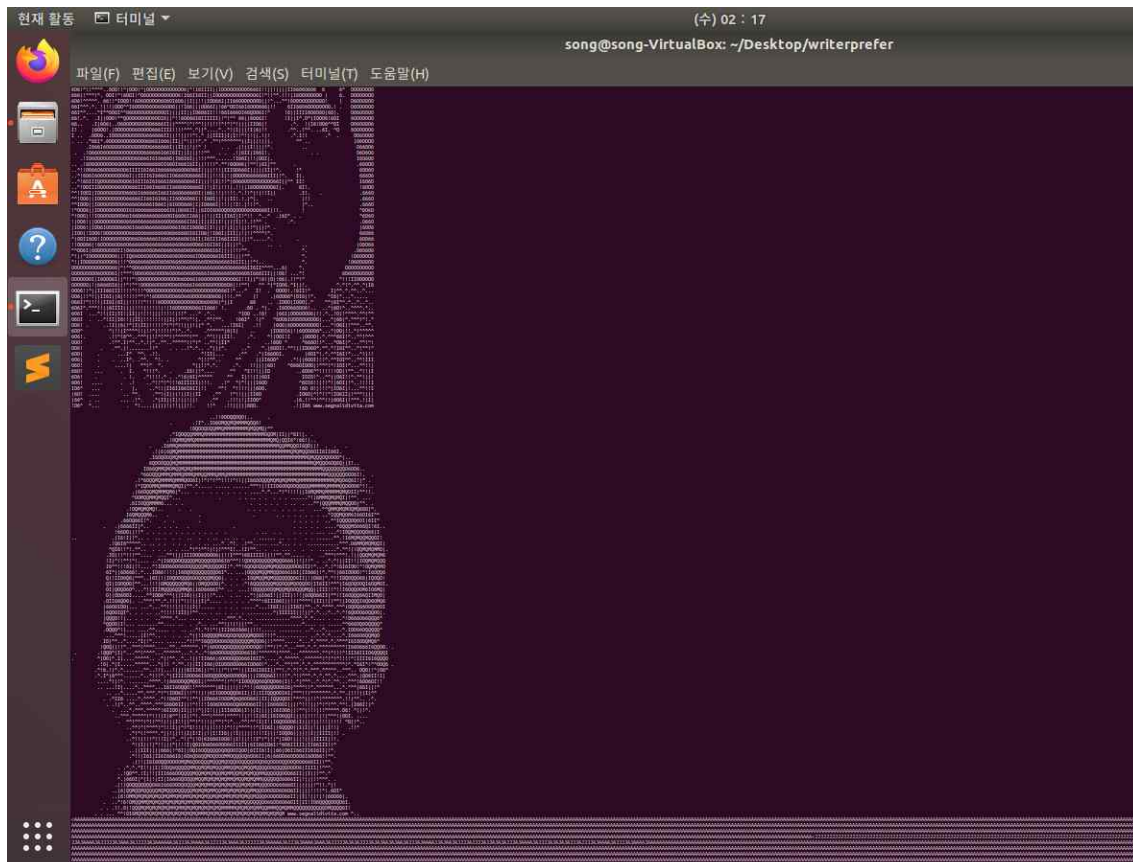


a. read-prefer

The image shows a Windows desktop environment. At the top, there is a taskbar with icons for a file explorer, a web browser, and a terminal. The terminal window is titled "song@song-VirtualBox: ~/Desktop/gotang" and shows a Go program running. The program's output is a long sequence of memory addresses and values, indicating a memory leak or a large data structure being processed. The output is formatted in a way that suggests it might be a hex dump or a log of memory operations. The desktop background is a solid light blue color. The terminal window is the primary focus, showing the execution of a Go program and its output.

b. writer-prefer

Writer 선호에서는 두 writer 사이에 reader가 올 수 없다. 따라서 그림이 항상 붙어서 출력된다.
reader는 중복으로 접근할 수 있으므로 문자는 섞여서 출력된다.



c. 공정한 reader-writer

공정한 reader-writer에서는 reader와 writer 모두 starvation이 없고 lock을 획득하고 해제하는 것에
공정한 기회를 얻는다. 따라서 그림이 연속으로 나지 않을 때도 있다. 이때에도 reader는 중복될 수 있으므로
문자는 식에서 출력된다.

