

# HW2: Reader & Writer Problem



과목명 | 운영체제

담당교수 | 황선태교수님

학과 | 소프트웨어학부

학년 | 3학년

학번 | 20143050

이름 | 김현석

제출일 | 2017.04.15

```

mucs@ubuntu:~/Desktop/Untitled Folder$ ./reader 1 5 &./reader 2 5 &./writer 3 3 &./reader 4 5 &./reader 5 3 &./writer 6 3 &
[1] 31645
[2] 31646
[3] 31647
[4] 31648
[5] 31649
[6] 31650
mucs@ubuntu:~/Desktop/Untitled Folder$
process 31645 before critical section
process 31645 in critical section
process 31646 before critical section
process 31647 before critical section
process 31648 before critical section
process 31649 before critical section
process 31650 before critical section
process 31645 leaving critical section
process 31646 in critical section
process 31645 exiting
process 31646 leaving critical section
process 31646 exiting
process 31647 in critical section
process 31648 in critical section
process 31649 in critical section
process 31650 in critical section
process 31645 in critical section
process 31645 leaving critical section
process 31646 in critical section
process 31646 leaving critical section
process 31646 exiting
process 31645 exiting
process 31647 leaving critical section
process 31647 exiting
process 31647 in critical section
process 31647 leaving critical section
process 31647 exiting
process 31650 leaving critical section
process 31650 exiting
process 31650 in critical section
process 31650 leaving critical section
process 31650 exiting
process 31648 leaving critical section
process 31648 exiting
process 31649 leaving critical section
process 31649 exiting
process 31648 in critical section
process 31648 leaving critical section
process 31648 exiting
process 31649 in critical section
process 31649 leaving critical section

```

명령어는 [1] reader 1 5  
 [2] reader 2 5  
 [3] writer 3 3  
 [4] reader 4 5  
 [5] reader 5 3  
 [6] writer 6 3  
 이렇게 되어있다.

이때 실행 순서는 1 -> 2 -> 3 -> 6 -> 5 -> 4 가 된다.

이유는 새로 들어오는 reader 는 writer, reader가 실행중이라면 대기 상태로 가게 되는데 이때 [3] 이 실행 중 이기 때문에 [4],[5],[6]은 대기 상태에 들어가게된다. 이때 대기상태에 들어가는 것은 wait를 통해 구현하였다. 대기열에 writer와 reader가 있다면 writer가 먼저 실행되게 되있기 때문에 [4]가 먼저 실행되게 된다. [4]가 실행 될 때 대기상태에 있는 [4]를 활성화 시켜줘야 하는데

이때 signal이 그역할을 해준다. [5],[6]은 마지막에 실행되게 된다. 이 때 남은 [5],[6]를 활성화 시키는 것은 broadcast가 하게된다. 또한 reader가 가장 처음 실행되고 가장 마지막에 실행 되는것도 reader 이기 때문에 AR.txt를 보면 총 실행시간은 24초가 된다. 또한 매 시작전 ipcs를 통해 semaphore가 존재하는지 확인하여 지워줘야 한다.

결론적으로 본 프로그램은 세마포를 통해 각 프로그램을 연결하여 구동하며 이때 acquire와 release를 통해 critical section을 구성 하며 대기중 일때는 waiter 상태에 들어가게되며 활성화 시킬때는 signal,broadcast가 사용된다. 그 모든 결과를 AW.txt, AR.txt, WW.txt, WR.txt 과 terminal로 통해 확인할 수 있다.

# source 설명

## -목차-

- 1.코드 목적
- 2.headerfile 설명
- 3.코드 & 함수 설명
- 4.전체 출력화면
- 5.source 코드

### 1.코드목적

본 코드는 세마포를 이용하여 Lock과 Condition Variable을 만들어 구성된 Mesa Style Monitor로 Reader & Writer Problem을 구현하는 것이다.

### 2.header file 설명

“stdio.h” 표준 라이브러리의 하나로서 입출력에 관한 함수 참조하기 위해 사용된다.

“stdlib.h” 표준 라이브러리의 하나로 exit() 함수 참조하기 위해 사용된다.

“time.h” 표준 라이브러리의 하나로 시간에 관한 함수 참조하기 위해 사용된다.

“string.h” 표준 라이브러리의 하나로 문자열과 문자열 함수에 관하여 참조하기 위해 사용된다.

### 3. 함수 설명

#### 3-1. 사용한 함수

```
int initsem (key_t semkey, int n)
int p (int semid)
int v (int semid)
void reset(char *fileVar)
void Store(char *fileVar, int i)
int Load(char *fileVar)
void add(char *fileVar, int l)
void sub(char *fileVar, int l)
void initLock(Lock *l, key_t semkey)
void Acquire(Lock *l)
void Release(Lock *l)
void initCondVar(CondVar *c, key_t semkey, char* queueLength)
void Wait(CondVar *c, Lock *lock)
void Signal(CondVar *c)
void Broadcast(CondVar *c)
```

### 3-2. 함수 설명

int initsem (key\_t semkey, int n) : semaphore의 초기 값을 설정

```
int initsem (key_t semkey, int n) {
    int status = 0, semid;
    // 세마포어 값을 설정
    if ((semid = semget (semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == -1)
    {
        if (errno == EEXIST)
            semid = semget (semkey, 1, 0);
    }
    else
    {
        semun arg;
        arg.val = n;
        // 세마포어 제어( setval: 값 설정 )
        status = semctl(semid, 0, SETVAL, arg);
    }
    if (semid == -1 || status == -1)
    {
        perror("initsem failed");
        return (-1);
    }
    return (semid);
}
```

int p (int semid) : p연산을 통해 semaphore의 값을 감소시킨다.

```
int p (int semid) {
    struct sembuf p_buf;
    p_buf.sem_num = 0;
    p_buf.sem_op = -1;
    p_buf.sem_flg = SEM_UNDO;
    if (semop(semid, &p_buf, 1) == -1)
    {
        printf("p(semid) failed");
        exit(1);
    }
    return (0);
}
```

int v (int semid): semaphore의 값을 증가시킨다.

```
int v (int semid) {
    struct sembuf v_buf;
    v_buf.sem_num = 0;
    v_buf.sem_op = 1;
    v_buf.sem_flg = SEM_UNDO;
    if (semop(semid, &v_buf, 1) == -1)
    {
        printf("v(semid) failed");
        exit(1);
    }
    return (0);
}
```

void reset(char \*fileVar): 만약 filevar라는 이름의 파일이 없다면 생성하고 0을 기록한다.

```
void reset(char *fileVar) { // fileVar라는 이름의 텍스트 파일을 새로 만들고 0값을 기록한다.
    if( access(fileVar, F_OK) == -1){
        FILE *fp = fopen(fileVar, "w");
        time_t timer;
        time(&timer);
        fprintf(fp, "PID: %d time: %s0\n", getpid(), ctime(&timer));
        fclose(fp);
    }
}
```

void Store(char \*fileVar, int I):fileVar 파일 끝에 I 값을 기록한다.

```
void Store(char *fileVar, int i) { // fileVar 파일 끝에 i 값을 append한다
    FILE *fp= fopen(fileVar, "a");
    fprintf(fp, "%d\n", i);
    fclose(fp);
}
```

int Load(char \*fileVar): fileVar 파일의 마지막 값을 읽어 온다.

```
int Load(char *fileVar) { // fileVar 파일의 마지막 값을 읽어 온다.
    int num;
    FILE *fp = fopen(fileVar, "r");
    fseek(fp, -4, SEEK_END);
    while(1) {
        fscanf(fp, "%d", &num);
        if(feof(fp)) {
            break;
        }
    }

    fclose(fp);
    return num;
}
```

void add(char \*fileVar, int i): fileVar 파일의 마지막 값을 읽어서 i를 더한 후에 이를 끝에 기록한다.

```
void add(char *fileVar, int i) { // fileVar 파일의 마지막 값을 읽어서 i를 더한 후에 이를 끝에 append 한다.
    FILE *fp = fopen(fileVar, "a");
    time_t timer;
    time(&timer);
    int add_num = Load(fileVar);
    add_num += i;
    fprintf(fp, "PID: %d time: %s %d\n", getpid(), ctime(&timer), add_num);
    fclose(fp);
}
```

void sub(char \*fileVar, int i):

```
void sub(char *fileVar, int i) { // fileVar 파일의 마지막 값을 읽어서 i를 뺀 후에 이를 끝에 append 한다.
    FILE *fp = fopen(fileVar, "a");
    int sub_num = Load(fileVar);
    time_t timer;
    time(&timer);
    sub_num -= i;
    fprintf(fp, "PID: %d time: %s %d\n", getpid(), ctime(&timer), sub_num);
    fclose(fp);
}
```



void initLock(Lock \*l, key\_t semkey):semaphore를 연결한다.

```
void initLock(Lock *l, key_t semkey) {
    if ((l->semid = initsem(semkey,1)) < 0)
        // 세마포를 연결한다.(없으면 초기값을 1로 주면서 새로 만들어서 연결한다.)
        exit(1);
}
```

void Acquire(Lock \*l):p연산을 이용해서 acquire(잠금)를 구현

```
void Acquire(Lock *l) {
    p(l->semid);
}
```

void Release(Lock \*l):v연산을 이용해서 Release(잠금해제)를 구현

```
void Release(Lock *l) {
    v(l->semid);
}
```

void initCondVar(CondVar \*c, key\_t semkey, char\* queueLength)  
:condvar를 초기화 하고 condvar의 queueLength를 매개변수의 queueLength로 하여준다. 이때 semaphore가 있다면 연결한다 없다면 새로 생성

```
void initCondVar(CondVar *c, key_t semkey, char* queueLength) {
    c->queueLength = queueLength;
    reset(c->queueLength); // queueLength=0
    if ((c->semid = initsem(semkey,0)) < 0)
        // 세마포를 연결한다.(없으면 초기값을 0로 주면서 새로 만들어서 연결한다.)
        exit(1);
}
```

void Wait(CondVar \*c, Lock \*lock): wait는 대기상태를 표현하는 것으로써 이를 구현하는 방법은 add연산을 통해 queueLength의 개수를 늘리고 매개변수로 들어오는 condvar를 p연산하면서 구현한다.

```
void Wait(CondVar *c, Lock *lock) {
    add(c->queueLength,1);
    Release(lock);
    p(c->semid);
    Acquire(lock);
}
```

void Signal(CondVar \*c):wait(대기 상태)인 것을 활성화 시키는 역할을 하는데 이때 매개변수로 들어오는 condvar를 v연산하고 condvar의 queueLength의 값을 감소시킨다.

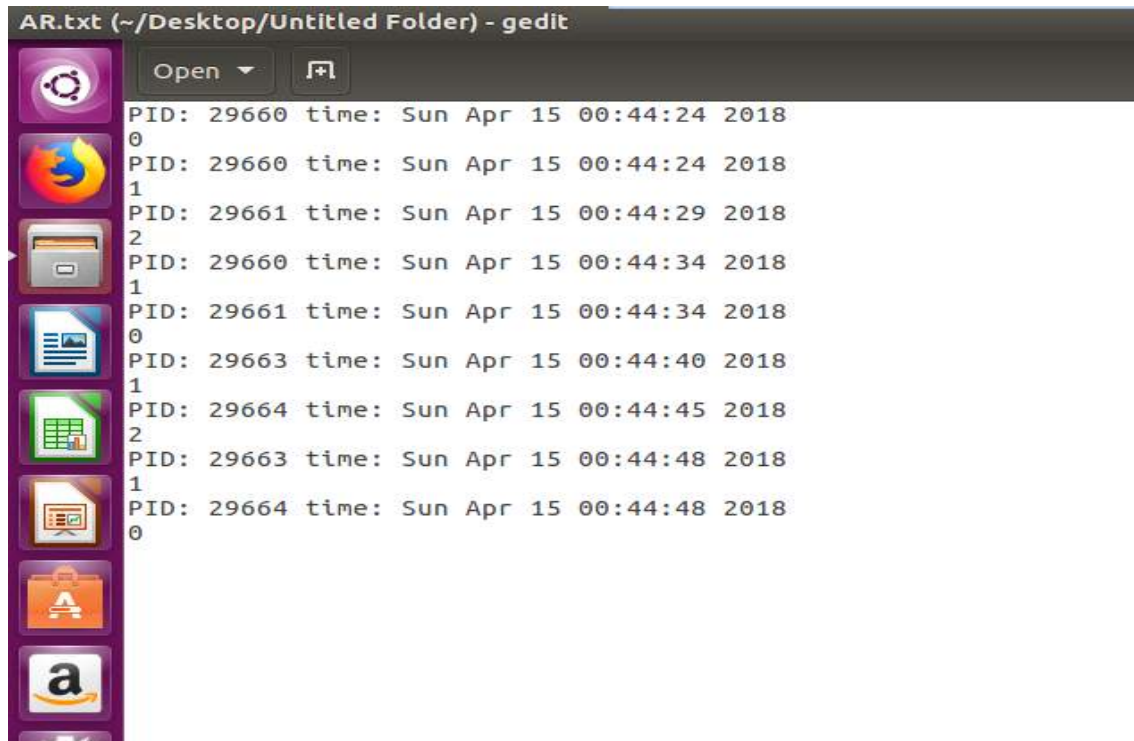
```
void Signal(CondVar *c) {  
    if(Load(c->queueLength) > 0){  
        v(c->semid);  
        sub(c->queueLength,1);  
    }  
}
```

void Broadcast(CondVar \*c):broadcast는 signal과 비슷한 역할을 하지만 signal과 다르게 broadcast는 대기중인 모든 wait를 활성화시키는 것에 차이가 있다. 코드에서 보면 알 수 있듯이 while문을 통해 queueLength에 있는 0이 나올 때 까지 감소시키면서 값을 감소시키는 것을 알 수 있다.

```
void Broadcast(CondVar *c) {  
    while(Load(c->queueLength) > 0){  
        v(c->semid);  
        sub(c->queueLength,1);  
    }  
}
```

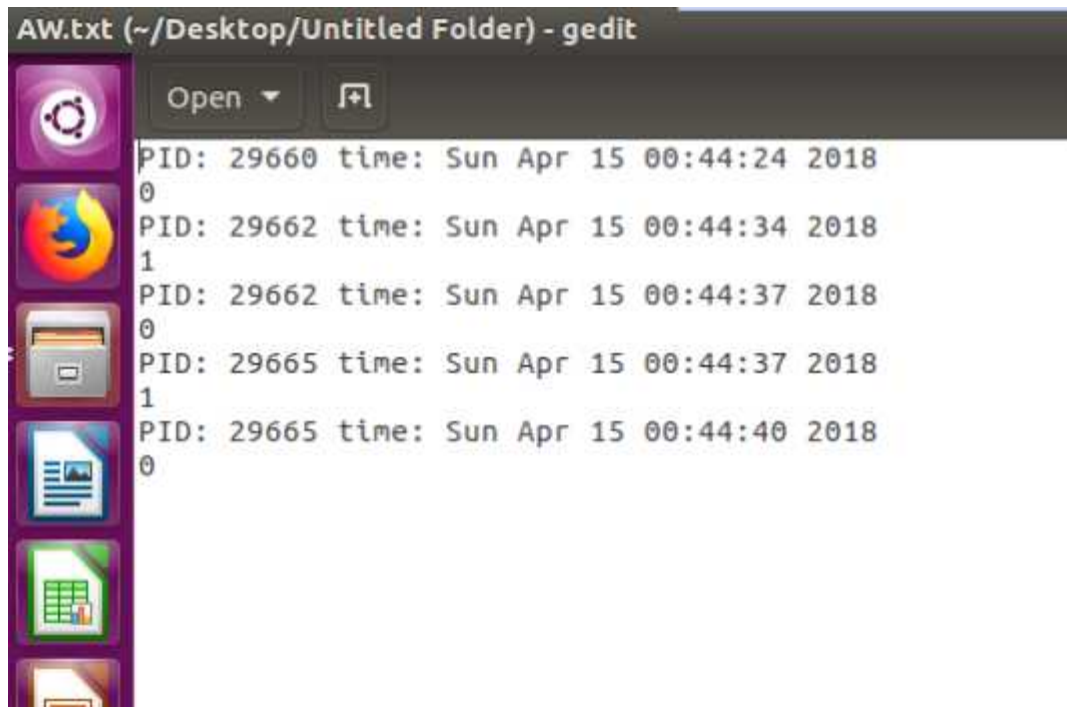
#### 4.전체화면 출력

##### 1.AR 파일의 출력화면



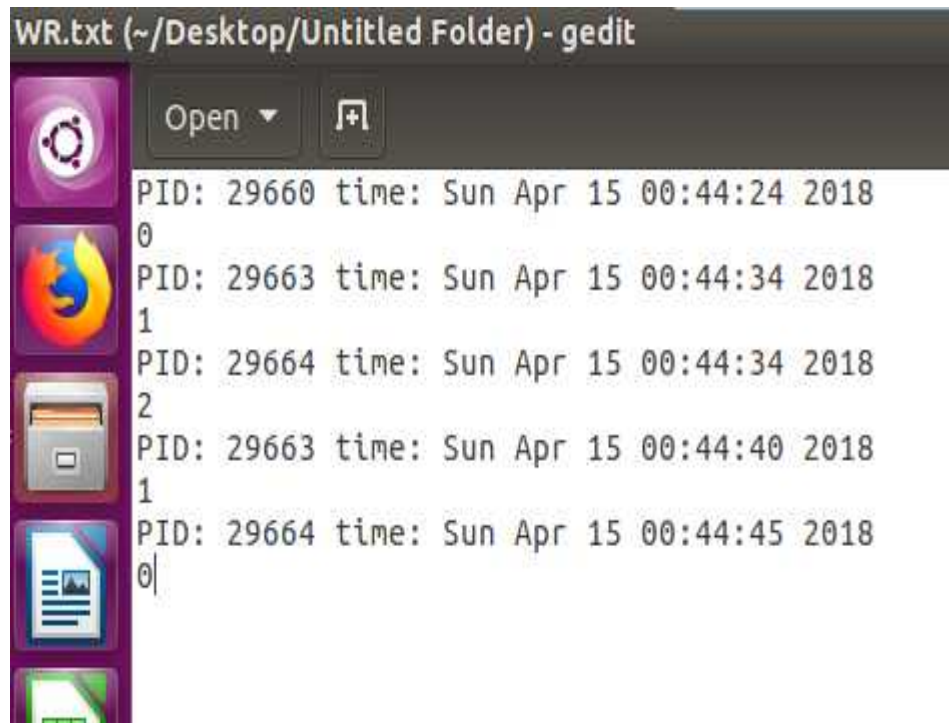
```
AR.txt (~/Desktop/Untitled Folder) - gedit
PID: 29660 time: Sun Apr 15 00:44:24 2018
0
PID: 29660 time: Sun Apr 15 00:44:24 2018
1
PID: 29661 time: Sun Apr 15 00:44:29 2018
2
PID: 29660 time: Sun Apr 15 00:44:34 2018
1
PID: 29661 time: Sun Apr 15 00:44:34 2018
0
PID: 29663 time: Sun Apr 15 00:44:40 2018
1
PID: 29664 time: Sun Apr 15 00:44:45 2018
2
PID: 29663 time: Sun Apr 15 00:44:48 2018
1
PID: 29664 time: Sun Apr 15 00:44:48 2018
0
```

##### 2.AW 파일의 출력화면



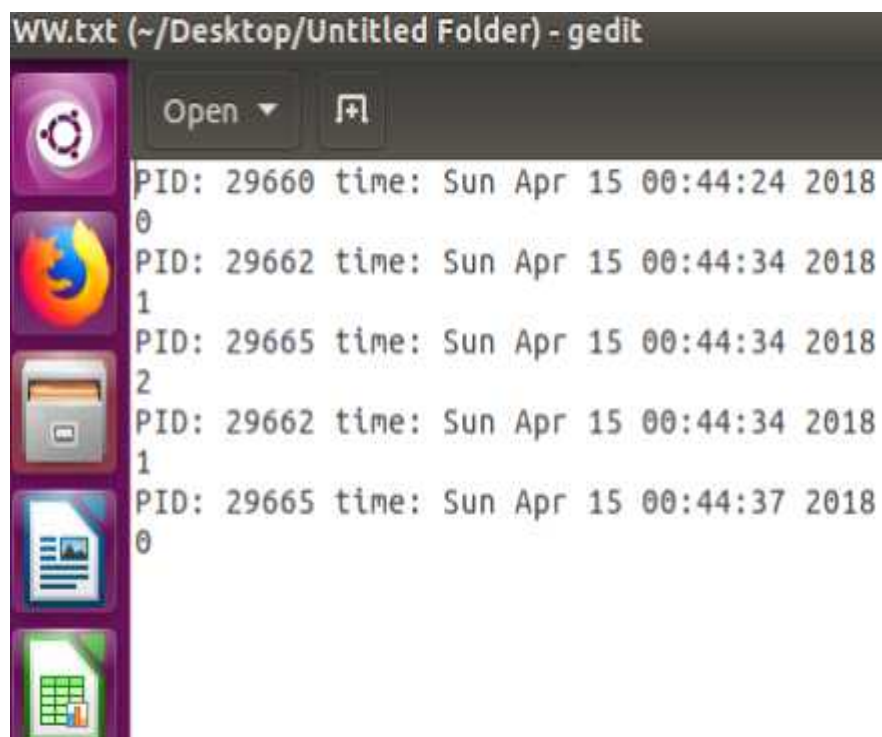
```
AW.txt (~/Desktop/Untitled Folder) - gedit
PID: 29660 time: Sun Apr 15 00:44:24 2018
0
PID: 29662 time: Sun Apr 15 00:44:34 2018
1
PID: 29662 time: Sun Apr 15 00:44:37 2018
0
PID: 29665 time: Sun Apr 15 00:44:37 2018
1
PID: 29665 time: Sun Apr 15 00:44:40 2018
0
```

### 3.WR 파일의 출력화면



```
WR.txt (~/Desktop/Untitled Folder) - gedit
PID: 29660 time: Sun Apr 15 00:44:24 2018
0
PID: 29663 time: Sun Apr 15 00:44:34 2018
1
PID: 29664 time: Sun Apr 15 00:44:34 2018
2
PID: 29663 time: Sun Apr 15 00:44:40 2018
1
PID: 29664 time: Sun Apr 15 00:44:45 2018
0|
```

### 4.WW 파일의 출력화면



```
WW.txt (~/Desktop/Untitled Folder) - gedit
PID: 29660 time: Sun Apr 15 00:44:24 2018
0
PID: 29662 time: Sun Apr 15 00:44:34 2018
1
PID: 29665 time: Sun Apr 15 00:44:34 2018
2
PID: 29662 time: Sun Apr 15 00:44:34 2018
1
PID: 29665 time: Sun Apr 15 00:44:37 2018
0
```

## 5.source 코드

### 1.reader.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define SEMPERM 0600
#define TRUE 1
#define FALSE 0

typedef union _semun { //semaphore의구조체
int val;
struct semid_ds *buf;
ushort *array;
} semun

int initsem(key_t semkey, int n) { //semaphore를초기화시킬때사용
int status = 0, semid;
// 세마포어값을설정
if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == -1)
{
if (errno == EEXIST)
semid = semget(semkey, 1, 0);
}
else
{
semun arg;
arg.val = n
// 세마포어제어( setval: 값설정)
status = semctl(semid, 0, SETVAL, arg);
}
if (semid == -1 || status == -1)
```

```

{
perror("initsem failed");
return (-1);
}
return (semid);
}

```

```

int p(int semid) { //p연산을하며acquire 할때사용된다.
struct sembuf p_buf;
p_buf.sem_num = 0;
p_buf.sem_op = -1;
p_buf.sem_flg = SEM_UNDO;
if (semop(semid, &p_buf, 1) == -1)
{
printf("p(semid) failed");
exit(1);
}
return (0);
}

```

```

int v(int semid) { //v연산을하며release 할때사용된다.
struct sembuf v_buf;
v_buf.sem_num = 0;
v_buf.sem_op = 1;
v_buf.sem_flg = SEM_UNDO;
if (semop(semid, &v_buf, 1) == -1)
{
printf("v(semid) failed");
exit(1);
}
return (0);
}

```

```

// Shared variable by file
void reset(char *fileVar) { // fileVar라는이름의텍스트화일을새로만들고0값을기록한다.
if (access(fileVar, F_OK) == -1) {
FILE *fp = fopen(fileVar, "w");
time_t timer;
time(&timer);
fprintf(fp, "PID: %d time: %s0\n", getpid(), ctime(&timer));
fclose(fp);
}
}

```

```
}  
}
```

```
void Store(char *fileVar, int i) { // fileVar 파일끝에 i 값을 append한다  
FILE *fp = fopen(fileVar, "a");  
fprintf(fp, "%d\n", i);  
fclose(fp);  
}
```

```
int Load(char *fileVar) { // fileVar 파일의 마지막 값을 읽어온다.  
int num;  
FILE *fp = fopen(fileVar, "r");  
fseek(fp, -4, SEEK_END);  
while (1) {  
fscanf(fp, "%d", &num);  
if (feof(fp)) {  
break  
}  
}  
}
```

```
fclose(fp);  
return num;  
}
```

```
void add(char *fileVar, int i) { // fileVar 파일의 마지막 값을 읽어서 i를 더한 후에 이를 끝에  
append 한다.  
FILE *fp = fopen(fileVar, "a");  
time_t timer;  
time(&timer);  
int add_num = Load(fileVar);  
add_num += i  
fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer), add_num);  
fclose(fp);  
}
```

```
void sub(char *fileVar, int i) { // fileVar 파일의 마지막 값을 읽어서 i를 빼 후에 이를 끝에  
append 한다.  
FILE *fp = fopen(fileVar, "a");  
int sub_num = Load(fileVar);  
time_t timer;  
time(&timer);
```

```

sub_num -= i
fprintf(fp, "PID: %d time: %s %d\n", getpid(), ctime(&timer), sub_num);
fclose(fp);

}

// Class Lock
typedef struct _lock { //lock의구조체semid로이루어졌다.
int semid;
} Lock

void initLock(Lock *l, key_t semkey) { //lock 초기화함수로세마포를연결하며만약없다면
초기값을1로주면서새로만들어서연결한다.
if ((l->semid = initsem(semkey, 1)) < 0)
exit(1);
}

void Acquire(Lock *l) { //p연산을이용해acquire를구현한다.
p(l->semid);
}

void Release(Lock *l) { //v연산을이용해release를구현한다.
v(l->semid);
}

// Class CondVar
typedef struct _cond { //condvar의구조체대기열과semid를포함한다.
int semid;
char* queueLength;
} CondVar

void initCondVar(CondVar *c, key_t semkey, char* queueLength) { // queueLength를
받아condvar queueLength로지정한다. 또한세마포를연결하는데없으면초기값을0로주면서새로
만들어서연결한다.
c->queueLength = queueLength
reset(c->queueLength); // queueLength=0
if ((c->semid = initsem(semkey, 0)) < 0)
exit(1);
}

void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를구현한다. add를통해

```



queueLength를증가시켜준다.

```
add(c->queueLength, 1);
```

```
Release(lock);
```

```
p(c->semid);
```

```
Acquire(lock);
```

```
}
```

```
void Signal(CondVar *c) { //condvar를받아대기중인queueLength가있다면활성화시켜주고  
sub를통해queueLength를감소시켜준다.
```

```
if (Load(c->queueLength) > 0) {
```

```
v(c->semid);
```

```
sub(c->queueLength, 1);
```

```
}
```

```
}
```

```
void Broadcast(CondVar *c) { //condvar를받아대기중인queueLength가있다면"모두" 활성  
화시켜주고sub를통해queueLength를감소시켜준다.
```

```
while (Load(c->queueLength) > 0) {
```

```
v(c->semid);
```

```
sub(c->queueLength, 1);
```

```
}
```

```
}
```

```
int main(int argc, char * argv[]) {
```

```
    // 서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지않게해야한다.
```

```
    // 실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을사용해야한다.
```

```
    // $ ipcs                // 남아있는세마포확T인
```

```
    // $ ipcrm -s <semid>    // <semid>라는세마포제거
```

```
    sleep(atoi(argv[1])); //처음sleep 인자를받아온다.
```

```
    key_t semkey = 0x200; //semkey값을지정한다.
```

```
    key_t semkey_writer = 0x300; //writer의semkey값을지정한다.
```

```
    key_t semkey_reader = 0x400; //reader의semkey값을지정한다.
```

```
    reset("WW.txt"); //WW.txt파일생성만약있다면생성하지않는다.
```

```
    reset("AW.txt"); //AW.txt파일생성만약있다면생성하지않는다.
```

```
    reset("WR.txt"); //WR.txt파일생성만약있다면생성하지않는다.
```

```
    reset("AR.txt"); //AR.txt파일생성만약있다면생성하지않는다.
```

```
    pid_t pid; //pid 값을받기위한변수
```

```

Lock lock; //wait, acquire, release에사용된다.
CondVar okToWrite; //writer에사용되는condvar
CondVar okToRead; //reader에사용되는condvar

int AW = 0, AR = 0, WW = 0, WR = 0; //초기값은0으로지정
pid = getpid(); //getpid를통해pid의값을pid 변수에저장

initLock(&lock, semkey); //lock 초기화함수
initCondVar(&okToWrite, semkey_writer, "queueLength.txt"); //writer에사용하는
condvar 함수초기화이때queueLength의파일명을지정한다.
initCondVar(&okToRead, semkey_reader, "queueLength.txt"); //reader에사용하는
condvar 함수초기화이때queueLength의파일명을지정한다.

printf("\nprocess %d before critical section\n", pid);
Acquire(&lock); // lock.Acquire()
printf("process %d in critical section\n",pid);

AW = Load("AW.txt");
WW = Load("WW.txt");

while((AW+WW)>0){ //만약Aactive Writer나Wating Writer이존재한다면wait 해준다.
    add("WR.txt",1);
    Wait(&okToRead,&lock);
    sub("WR.txt",1);
    AW = Load("AW.txt");
    WW = Load("WW.txt");
}
add("AR.txt",1); //한번읽었기때문에더해준다. /* 화일에서읽어서1 더하기*/

sleep(atoi(argv[2]));//두번째슬립인자를받는다.

printf("process %d leaving critical section\n", pid);
Release(&lock); // lock.Release()
printf("process %d exiting\n",pid);
Acquire(&lock); // lock.Acquire()
printf("process %d in critical section\n",pid);
sub("AR.txt",1);

AR = Load("AR.txt");
WW = Load("WW.txt");

```

if( AR == 0 && WW > 0 ){ //Active reader가없고Wating Writer가있다면signal을통해  
활성화시켜준다.

```
Signal(&okToWrite);
}
printf("process %d leaving critical section\n", pid);
Release(&lock); // lock.Release()
printf("process %d exiting\n",pid);
return 0;
}
```

## 2.writer.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#define SEMPERM 0600
#define TRUE 1
#define FALSE 0
```

```
typedef union _semun { //semaphore의구조체
    int val;
    struct semid_ds *buf;
    ushort *array;
} semun
```

```
int initsem (key_t semkey, int n) { //semaphore를초기화시킬때사용
    int status = 0, semid;
    // 세마포어값을설정
    if ((semid = semget (semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == -1)
    {
        if (errno == EEXIST)
            semid = semget (semkey, 1, 0);
    }
    else
    {
```

```

        semun arg;
        arg.val = n
// 세마포어제어( setval: 값설정)
        status = semctl(semid, 0, SETVAL, arg);
    }
    if (semid == -1 || status == -1)
    {
        perror("initsem failed");
        return (-1);
    }
    return (semid);
}

```

```

int p (int semid) { //p연산을하며acquire 할때사용된다.
    struct sembuf p_buf;
    p_buf.sem_num = 0;
    p_buf.sem_op = -1;
    p_buf.sem_flg = SEM_UNDO;
    if (semop(semid, &p_buf, 1) == -1)
    {
        printf("p(semid) failed");
        exit(1);
    }
    return (0);
}

```

```

int v (int semid) { //v연산을하며release 할때사용된다.
    struct sembuf v_buf;
    v_buf.sem_num = 0;
    v_buf.sem_op = 1;
    v_buf.sem_flg = SEM_UNDO;
    if (semop(semid, &v_buf, 1) == -1)
    {
        printf("v(semid) failed");
        exit(1);
    }
    return (0);
}

```

// Shared variable by file

```

void reset(char *fileVar) { // fileVar라는이름의텍스트화일을새로만들고0값을기록한다.

```

```

if( access(fileVar, F_OK) == -1){
    FILE *fp = fopen(fileVar, "w");
    time_t timer;
    time(&timer);
    fprintf(fp, "PID: %d time: %s0\n", getpid(), ctime(&timer));
    fclose(fp);
}
}

```

```

void Store(char *fileVar, int i) { // fileVar 파일끝에 i 값을 append한다
    FILE *fp= fopen(fileVar, "a");
    fprintf(fp, "%d\n", i);
    fclose(fp);
}

```

```

int Load(char *fileVar) { // fileVar 파일의 마지막값을 읽어온다.
int num;
    FILE *fp = fopen(fileVar, "r");
    fseek(fp,-4, SEEK_END);
    while(1) {
        fscanf(fp,"%d",&num);
        if(feof(fp)){
            break
        }
    }
}

```

```

fclose(fp);
return num;
}

```

```

void add(char *fileVar, int i) { // fileVar 파일의 마지막값을 읽어서 i를 더한 후에 이를 끝에
append 한다.
    FILE *fp = fopen(fileVar, "a");
    time_t timer;
    time(&timer);
    int add_num = Load(fileVar);
    add_num += i
    fprintf(fp, "PID: %d time: %s%d\n", getpid(), ctime(&timer), add_num);
    fclose(fp);
}

```

void sub(char \*fileVar, int i) { // fileVar 파일의 마지막값을읽어서i를뺀후에이를끝에 append 한다.

FILE \*fp = fopen(fileVar, "a");

int sub\_num = Load(fileVar);

time\_t timer;

time(&timer);

sub\_num -= i

fprintf(fp, "PID: %d time: %s %d\n", getpid(), ctime(&timer), sub\_num);

fclose(fp);

}

// Class Lock

typedef struct \_lock { //lock의구조체semid로이루어졌다.

int semid;

} Lock

void initLock(Lock \*l, key\_t semkey) { //lock 초기화함수로세마포를연결하며만약없다면 초기값을1로주면서새로만들어서연결한다.

if ((l->semid = initsem(semkey,1)) < 0)

exit(1);

}

void Acquire(Lock \*l) { //p연산을이용해acquire를구현한다.

p(l->semid);

}

void Release(Lock \*l) { //v연산을이용해release를구현한다.

v(l->semid);

}

// Class CondVar

typedef struct \_cond { //condvar의구조체대기열과semid를포함한다.

int semid;

char\* queueLength;

} CondVar

void initCondVar(CondVar \*c, key\_t semkey, char\* queueLength) { // queueLength를 받아condvar queueLength로지정한다. 또한세마포를연결하는데없으면초기값을0로주면서새로 만들어서연결한다.

c->queueLength = queueLength

```

    reset(c->queueLength); // queueLength=0
    if ((c->semid = initsem(semkey,0)) < 0)
        exit(1);
}

void Wait(CondVar *c, Lock *lock) { //condvar와lock를받아wait를구현한다. add를통해
queueLength를증가시켜준다.
    add(c->queueLength,1);
    Release(lock);
    p(c->semid);
    Acquire(lock);
}

void Signal(CondVar *c) { //condvar를받아대기중인queueLength가있다면활성화시켜주고
sub를통해queueLength를감소시켜준다.
    if(Load(c->queueLength) > 0){
        v(c->semid);
        sub(c->queueLength,1);
    }
}

void Broadcast(CondVar *c) { //condvar를받아대기중인queueLength가있다면"모두" 활성
화시켜주고sub를통해queueLength를감소시켜준다.
    while(Load(c->queueLength) > 0){
        v(c->semid);
        sub(c->queueLength,1);
    }
}

int main(int argc, char * argv[]) {
    // 서버에서작업할때는자기학번등을이용하여다른사람의키와중복되지않게해야한다.
    // 실행하기전에매번세마포들을모두지우거나아니면다른semkey 값을사용해야한다.
    // $ ipcs                // 남아있는세마포확T인
    // $ ipcrm -s <semid>    // <semid>라는세마포제거

    sleep(atoi(argv[1])); //처음sleep 인자를받아온다.
    key_t semkey = 0x200; //semkey값을지정한다.
    key_t semkey_writer = 0x300; //writer의semkey값을지정한다.
    key_t semkey_reader = 0x400; //reader의semkey값을지정한다.

    reset("WW.txt"); //WW.txt파일생성만약있다면생성하지않는다.

```

```
reset("AW.txt"); //AW.txt파일생성만약있다면생성하지않는다.  
reset("WR.txt"); //WR.txt파일생성만약있다면생성하지않는다.  
reset("AR.txt"); //AR.txt파일생성만약있다면생성하지않는다.
```

```
pid_t pid; //pid 값을받기위한변수
```

```
Lock lock; //wait, acquire, release에사용된다.  
CondVar okToWrite; //writer에사용되는condvar  
CondVar okToRead; //reader에사용되는condvar
```

```
int AW=0,AR=0,WW=0,WR=0; //초기값은0으로지정  
pid = getpid(); //getpid를통해pid의값을pid 변수에저장
```

```
initLock(&lock,semkey); //lock 초기화함수  
initCondVar(&okToWrite, semkey_writer, "queueLength.txt"); //writer에사용하는  
condvar 함수초기화이때queuelength의파일명을지정한다.  
initCondVar(&okToRead,semkey_reader, "queueLength.txt"); //reader에사용하는  
condvar 함수초기화이때queuelength의파일명을지정한다.
```

```
printf("\nprocess %d before critical section\n", pid);  
Acquire(&lock); // lock.Acquire()  
printf("process %d in critical section\n",pid);
```

```
AW = Load("AW.txt");  
AR = Load("AR.txt");
```

```
while((AW + AR)>0){ //만약Aactive Writer나Active Reader가존재한다면wait 해준다.  
    add("WW.txt",1);  
    Wait(&okToWrite,&lock);  
    sub("WW.txt",1);  
    AW = Load("AW.txt");  
    AR = Load("AR.txt");  
}  
add("AW.txt",1);//한번써줬기때문에1을더해준다. /* 화일에서읽어서1 더하기*/
```

```
sleep(atoi(argv[2])); //두번째슬립인자를받아온다.
```

```
printf("process %d leaving critical section\n", pid);  
Release(&lock); // lock.Release()  
printf("process %d exiting\n",pid);  
Acquire(&lock); // lock.Acquire()
```



```

printf("process %d in critical section\n",pid);
sub("AW.txt",1);

WR = Load("WR.txt");
WW = Load("WW.txt");

if( WW > 0 ){ //Wating Writer가있다면활성화시켜준다.
Signal(&okToWrite);
}
else if(WR > 0){ //Wating Writer가없고Wating Reader가있다면현시점에서는읽기만하면
되기때문에broadcast로전부다활성화시켜준다.
Broadcast(&okToRead);
}
printf("process %d leaving critical section\n", pid);
Release(&lock); // lock.Release()
printf("process %d exiting\n",pid);
return 0;
}

```