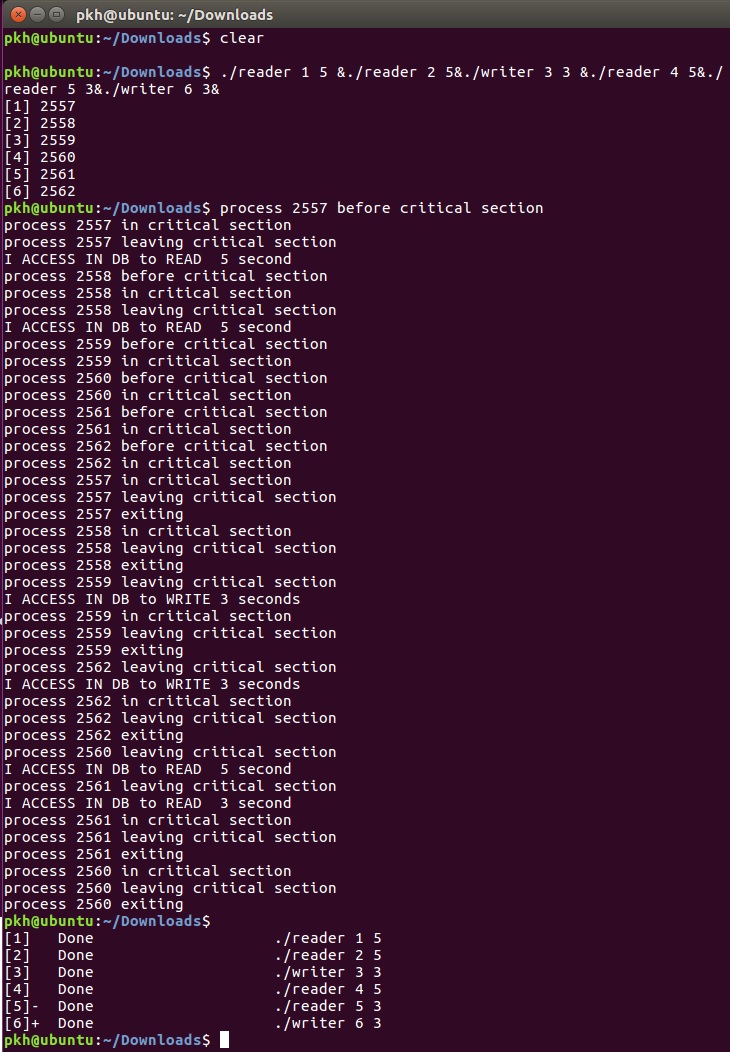
운영체제 (02)

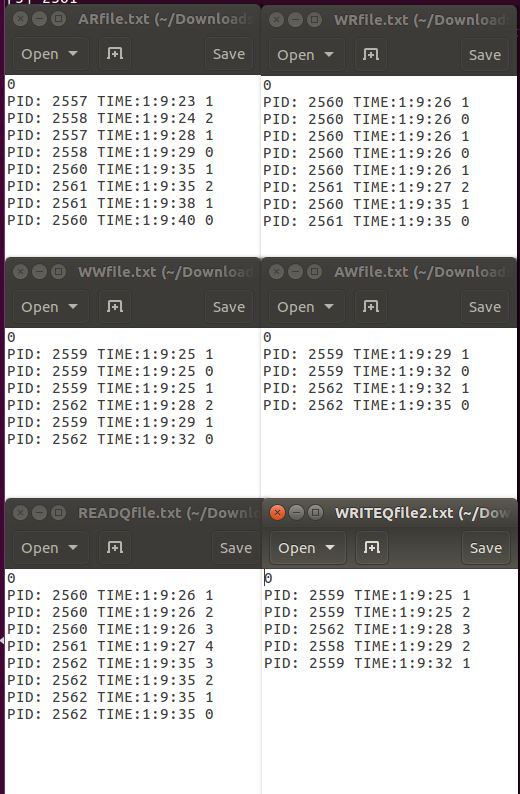
HW2: Reader & Writer Problem

소프트웨어학부

20143060

박기훈

1. 실행 화면



1. 실행 및 분석 서술

Reader 는 writer가 작성 중이 아닐 때는 중복으로 DB에 접근 할 수 있으며 writer는 중복으로 DB에 접근하지 못하며, reading이 끝나야 실행되며 DB접근 중 외부의 간섭을 차단한다.  
6개의 프로그램은 1초 단위로 차례대로 실행되게 되며  
시작 시간을 0초라고 하면 pid1 는 1초에 실행되어 AR=1로 ,pid2 는 2초에 실행되며 AR=2로 바꾼다. 3초에 writer프로세스가 실행되지만, 아직 pid1,2가 reading 작업 중이므로 pid2의 reading이 끝나기 기다리며 WW=1를 만들고, reading이 끝나는 7초에 writing을 시작하게 된다(pid1는 6초에(AR=1),pid2는 7초(AR=0)에 종료). 4초,5초에 실행된 pid4, pid5 는 pid3와 ,6초에 시작된 pid6 이 맞물린 writing 작업에 의해 pid6의 writing이 끝난 뒤에 (13초) 같이 시작하고, 5초간 읽는 pid4가 가장 늦은 시간(18초)에 끝나게 된다.

실행 시간:

Pid1(./reader 1 5) : 0~1초(sleep) 1초~6초(read)

Pid2(./reader 2 5) : 0~2초(sleep) 2초~7초(read)

Pid3(./writer. 3 3) : 0~3초(sleep) 3초~7초(wait) 7초~10초(write)

Pid4(./reader 4 5) : 0~4초(sleep) 4초~13초(wait) 13초~18초(read)

Pid5(./reader 5 3) : 0~5초(sleep) 5초~13초(wait) 13초~16초(read)

Pid6(./writer 6 3) : 0~6초(sleep) 6초~10초(wait) 10초~13초(write)

결론적으로, 전체 실행 시간은 18초, 가장 늦게 끝나는 프로세스는 reader 4 5 (pid:2560) 이다.

1. 소스코드

<reader.c>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <errno.h>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define SEMPERM 0600

#define TRUE 1

#define FALSE 0

typedef union \_semun {

int val;

struct semid\_ds \*buf;

ushort \*array;

} semun;

int initsem (key\_t semkey, int n) {

int status = 0, semid;

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;

status = semctl(semid, 0, SETVAL, arg);

}

if (semid == -1 || status == -1)

{

perror("initsem failed");

return (-1);

}

return (semid);

}

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) {

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

reset(char \*fileVar) {

// fileVar라는 이름의 텍스트 화일을 새로 만들고 0값을 기록한다.

int isfile = access(fileVar,0);

if(isfile == -1){

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

fprintf(fp,"0");

fclose(fp);

}

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

int n;

n = fgetc(fp);

if(n == "-1"){

fprintf(fp,"0");

}

fclose(fp);

}

Store(char \*fileVar,int i) {

// fileVar 화일 끝에 i 값을 append한다.

int n;

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

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",i);

fclose(fp);

}

int Load(char \*fileVar) {

int tmp,id;

char timet;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %s %d", &tmp,&id,&timet, &n);

}

fclose(fp);

return n;

}

add(char \*fileVar,int i) {

// fileVar 화일의 마지막 값을 읽어서 i를 더한 후에 이를 끝에 append 한다.

int tmp,id;

char timet;

int n;

time\_t t;

t = time(NULL);

struct tm\* ti;

ti = localtime(&t);

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %s %d", &tmp,&id,&timet, &n);

}

fclose(fp); //store n

n = n + i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"TIME:%d:%d:%d ",ti->tm\_hour,ti->tm\_min,ti->tm\_sec);

fprintf(fp,"%d\n",n);

fclose(fp);

}

sub(char \*fileVar,int i) {

// fileVar 화일의 마지막 값을 읽어서 i를 뺀 후에 이를 끝에 append 한다.

int tmp,id;

char timet;

int n;

time\_t t;

t = time(NULL);

struct tm\* ti;

ti = localtime(&t);

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %s %d", &tmp,&id,&timet, &n);

}

fclose(fp); //store n

n = n - i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"TIME:%d:%d:%d ",ti->tm\_hour,ti->tm\_min,ti->tm\_sec);

fprintf(fp,"%d\n",n);

fclose(fp);

}

// Class Lock

typedef struct \_lock {

int semid;

} Lock;

initLock(Lock \*l, key\_t semkey) {

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

// 세마포를 연결한다.(없으면 초기값을 1로 주면서 새로 만들어서 연결한다.)

exit(1);

}

Acquire(Lock \*l) {

p(l->semid);

}

Release(Lock \*l) {

v(l->semid);

}

// Class CondVar

typedef struct \_cond {

int semid;

char \*queueLength;

} CondVar;

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);

}

Wait(CondVar \*c, Lock \*lock) {

add(c->queueLength,1);

Release(lock);

p(c->semid);

Acquire(lock);

}

Signal(CondVar \*c) {

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

v(c->semid);

sub(c->queueLength,1);

}

}

Broadcast(CondVar \*c) {

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

v(c->semid);

sub(c->queueLength,1);

}

}

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

int sleepT = atoi(argv[1]);

int runningT = atoi(argv[2]);

key\_t semkey = 0x200;

key\_t semkey2 = 0x210;

key\_t semkey3 = 0x220;

// 서버에서 작업할 때는 자기 학번 등을 이용하여 다른 사람의 키와 중복되지 않게 해야 한다.

// 실행하기 전에 매번 세마포들을 모두 지우거나 아니면 다른 semkey 값을 사용해야 한다.

// $ ipcs // 남아 있는 세마포 확인

// $ ipcrm -s <semid> // <semid>라는 세마포 제거

sleep(sleepT);

int semid;

pid\_t pid;

Lock lock;

CondVar okToRead;

CondVar okToWrite;

char \*queueREAD = "READQfile.txt";

char \*queueWRITE = "WRITEQfile2.txt";

char \*AR = "ARfile.txt";

char \*WR = "WRfile.txt";

char \*AW = "AWfile.txt";

char \*WW = "WWfile.txt";

reset(AR);

reset(WR);

reset(AW);

reset(WW);

pid = getpid();

initLock(&lock,semkey);

initCondVar(&okToRead, semkey2,queueREAD);

initCondVar(&okToWrite, semkey3,queueWRITE);

printf("process %d before critical section\n", pid);

Acquire(&lock); // lock.Acquire()

printf("process %d in critical section\n",pid);

/\* 화일에서 읽어서 1 더하기 \*/

while((Load(AW) + Load(WW)) > 0){

add(WR,1);

Wait(&okToRead,&lock);

sub(WR,1);

}

add(AR,1);

printf("process %d leaving critical section\n", pid);

Release(&lock); // lock.Release()

//ACCESS DB

printf("I ACCESS IN DB to READ %d second\n",runningT);

sleep(runningT);

Acquire(&lock); // lock.Acquire()

printf("process %d in critical section\n",pid);

sub(AR,1);

if(Load(AR) == 0 && Load(WW) > 0){

Signal(&okToWrite);

}

printf("process %d leaving critical section\n", pid);

Release(&lock); // lock.Release()

printf("process %d exiting\n",pid);

exit(0);

}

<writer.c>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <errno.h>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define SEMPERM 0600

#define TRUE 1

#define FALSE 0

typedef union \_semun {

int val;

struct semid\_ds \*buf;

ushort \*array;

} semun;

int initsem (key\_t semkey, int n) {

int status = 0, semid;

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;

status = semctl(semid, 0, SETVAL, arg);

}

if (semid == -1 || status == -1)

{

perror("initsem failed");

return (-1);

}

return (semid);

}

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) {

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

reset(char \*fileVar) {

// fileVar라는 이름의 텍스트 화일을 새로 만들고 0값을 기록한다.

int isfile = access(fileVar,0);

if(isfile == -1){

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

fprintf(fp,"0");

fclose(fp);

}

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

int n;

n = fgetc(fp);

if(n == "-1"){

fprintf(fp,"0");

}

fclose(fp);

}

Store(char \*fileVar,int i) {

// fileVar 화일 끝에 i 값을 append한다.

int n;

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

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"%d\n",i);

fclose(fp);

}

int Load(char \*fileVar) {

int tmp,id;

char timet;

int n;

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %s %d", &tmp,&id,&timet, &n);

}

fclose(fp);

return n;

}

add(char \*fileVar,int i) {

// fileVar 화일의 마지막 값을 읽어서 i를 더한 후에 이를 끝에 append 한다.

int tmp,id;

char timet;

int n;

time\_t t;

t = time(NULL);

struct tm\* ti;

ti = localtime(&t);

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %s %d", &tmp,&id,&timet, &n);

}

fclose(fp); //store n

n = n + i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"TIME:%d:%d:%d ",ti->tm\_hour,ti->tm\_min,ti->tm\_sec);

fprintf(fp,"%d\n",n);

fclose(fp);

}

sub(char \*fileVar,int i) {

// fileVar 화일의 마지막 값을 읽어서 i를 뺀 후에 이를 끝에 append 한다.

int tmp,id;

char timet;

int n;

time\_t t;

t = time(NULL);

struct tm\* ti;

ti = localtime(&t);

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

fscanf(fp,"%d",&n);

while(!feof(fp)){

fscanf(fp,"%s %s %s %d", &tmp,&id,&timet, &n);

}

fclose(fp); //store n

n = n - i;

fp = fopen(fileVar, "a");

fprintf(fp,"PID: %ld ",getpid());

fprintf(fp,"TIME:%d:%d:%d ",ti->tm\_hour,ti->tm\_min,ti->tm\_sec);

fprintf(fp,"%d\n",n);

fclose(fp);

}

// Class Lock

typedef struct \_lock {

int semid;

} Lock;

initLock(Lock \*l, key\_t semkey) {

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

// 세마포를 연결한다.(없으면 초기값을 1로 주면서 새로 만들어서 연결한다.)

exit(1);

}

Acquire(Lock \*l) {

p(l->semid);

}

Release(Lock \*l) {

v(l->semid);

}

// Class CondVar

typedef struct \_cond {

int semid;

char \*queueLength;

} CondVar;

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);

}

Wait(CondVar \*c, Lock \*lock) {

add(c->queueLength,1);

Release(lock);

p(c->semid);

Acquire(lock);

}

Signal(CondVar \*c) {

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

v(c->semid);

sub(c->queueLength,1);

}

}

Broadcast(CondVar \*c) {

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

v(c->semid);

sub(c->queueLength,1);

}

}

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

int sleepT = atoi(argv[1]);

int runningT = atoi(argv[2]);

key\_t semkey = 0x200;

key\_t semkey2 = 0x210;

key\_t semkey3 = 0x220;

// 서버에서 작업할 때는 자기 학번 등을 이용하여 다른 사람의 키와 중복되지 않게 해야 한다.

// 실행하기 전에 매번 세마포들을 모두 지우거나 아니면 다른 semkey 값을 사용해야 한다.

// $ ipcs // 남아 있는 세마포 확인

// $ ipcrm -s <semid> // <semid>라는 세마포 제거

sleep(sleepT);

int semid;

pid\_t pid;

Lock lock;

CondVar okToRead;

CondVar okToWrite;

char \*queueREAD = "READQfile.txt";

char \*queueWRITE = "WRITEQfile2.txt";

char \*AR = "ARfile.txt";

char \*WR = "WRfile.txt";

char \*AW = "AWfile.txt";

char \*WW = "WWfile.txt";

reset(AR);

reset(WR);

reset(AW);

reset(WW);

pid = getpid();

initLock(&lock,semkey);

initCondVar(&okToRead, semkey2,queueREAD);

initCondVar(&okToWrite, semkey3,queueWRITE);

printf("process %d before critical section\n", pid);

Acquire(&lock); // lock.Acquire()

printf("process %d in critical section\n",pid);

/\* 화일에서 읽어서 1 더하기 \*/

while((Load(AW) + Load(AR)) > 0){

add(WW,1);

Wait(&okToWrite,&lock);

sub(WW,1);

}

add(AW,1);

printf("process %d leaving critical section\n", pid);

Release(&lock); // lock.Release()

//ACCESS DB

printf("I ACCESS IN DB to WRITE %d seconds\n",runningT);

sleep(runningT);

Acquire(&lock); // lock.Acquire()

printf("process %d in critical section\n",pid);

sub(AW,1);

if(Load(WW) > 0){

Signal(&okToWrite);

}

else if(Load(WR) > 0){

Broadcast(&okToRead);

}

printf("process %d leaving critical section\n", pid);

Release(&lock); // lock.Release()

printf("process %d exiting\n",pid);

exit(0);

}