**Problem Set 6 – Synchronization**

**Table of Contents**

|  |
| --- |
| 1. Problem 1 |
| * 1. Spinlock.h………………………………………….…………pg. 1 |
| * 1. Spinlock.c………………………………………….…………pg. 1 |
| 1. Problem 2 |
| * 1. Spintest.c………………………….……………….…………pg. 2 |
| 1. Problem 3 |
| * 1. Cv.h……………………………………….……….…………pg. 3 |
| * 1. Cv.c…….………………………………………….….………pg. 3 |
| 1. Problem 4 |
| * 1. Fifo.h…….……………………………………………………pg. 4 |
| * 1. Fifo.c…………………………………………….……………pg. 5 |
| 1. Problem 5 |
| * 1. Ftest.c…………………………………………….………...…pg. 6 |
| 1. Sample Runs |
| * 1. Spintest Sample Run………………………………...……..…pg. 7 |
| * 1. Ftest Sample Run……………….……………..…….……..…pg. 8 |
| * 1. Ftest Sample Run……………….…………..……….……..…pg. 9 |

**Source Code**

1. **Problem 1**
2. **Spinlock.h**

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <sched.h>

#include <fcntl.h>

#include <errno.h>

#include <string.h>

#include <signal.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/wait.h>

struct spinlock{

volatile char lock;

pid\_t pid;

pid\_t ppid;

int numlock;

int numunlock;

};

int tas(volatile char \*lock);

void spin\_lock(struct spinlock \*l);

void spin\_unlock(struct spinlock \*l);

1. **Spinlock.c**

#include "spinlock.h"

void spin\_lock(struct spinlock \*l){

while(tas(&l->lock)!=0) sched\_yield();

l->numlock++;

l->pid = getpid();

l->ppid = getppid();

}

void spin\_unlock(struct spinlock \*l){

l->lock = 0;

l->numunlock++;

}

1. **Problem 2**
   1. **Spintest.c**

#include "spinlock.h"

int main() {

int nchild, niter;

int status = 0;

pid\_t f, fpast;

int\* map;

struct spinlock \*l;

printf("Number of childs: ");

scanf("%i", &nchild);

if(nchild < 8) nchild = 8;

printf("Number of iterations: ");

scanf("%i", &niter);

if((map = mmap(NULL, sizeof(int), PROT\_READ|PROT\_WRITE, AP\_ANONYMOUS|MAP\_SHARED, -1, 0)) == MAP\_FAILED) {

fprintf(stderr, "ERRNO %d - Failed to map a int variable on virtual memory : %s\n", errno, strerror(errno));

return 1;

}

\*map = 0;

l = (struct spinlock \*) (map + sizeof(struct spinlock));

l->lock = map[1];

fpast = 1;

printf("Intial value : %d\n", \*map);

for(int i = 0; i < nchild; i++) {

if(fpast > 0){ //to guarantee that only parent process forks. If the child process forks as well, twice the number of wanted children will be spawned.

if((f = fork()) < 0) {

fprintf(stderr, "ERRNO %d - Failed to fork from parent : %s\n", errno, strerror(errno));

return 1;

}

}

fpast = f;

}

if(f > 0) {

while(wait(&status) > 0);

printf("Proc %i, Final value : %d\n", f, \*map);

} else {

for(int i = 0; i < niter; i++) {

spin\_lock(l);

map[0]++;

spin\_unlock(l);

}

printf("Pid: %i, Ppid %i - Spin locked %i times, unlocked %i times\n", l->pid, l->ppid, l->numlock, l->numunlock);

}

}

1. **Problem 3**
   1. **Cv.h**

#include "spinlock.h"

#define CV\_MAXPROC 64

struct cv{

pid\_t procs[CV\_MAXPROC];

int waiters;

sigset\_t mask;

struct spinlock mutex;

};

void cv\_init(struct cv \*cv);

void cv\_wait(struct cv \*cv, struct spinlock \*mutex);

int cv\_broadcast(struct cv \*cv);

int cv\_signal(struct cv \*cv);

* 1. **Cv.c**

#include "cv.h"

void sighandler(int signo){return;}

void cv\_init(struct cv \*cv){

int \*map;

struct spinlock \*l;

if((map = mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_ANONYMOUS|MAP\_SHARED, -1, 0)) == MAP\_FAILED) {

fprintf(stderr, "ERRNO %d - Failed to map a int variable on virtual memory : %s\n", errno, strerror(errno));

exit(EXIT\_FAILURE);

}

l = (struct spinlock \*) (map + sizeof(struct spinlock));

cv->mutex = \*l;

for(int i = 0; i < CV\_MAXPROC; i++)

cv->procs[i] = 0;

cv->waiters = 0;

if(signal(SIGUSR1, sighandler) == SIG\_ERR){

fprintf(stderr, "Failed to assign signal handler for SIGUSR1: %s\n", strerror(errno));

exit(EXIT\_FAILURE);

}

sigfillset(&cv->mask);

sigdelset(&cv->mask, SIGUSR1);

}

void cv\_wait(struct cv \*cv, struct spinlock \*m){

spin\_lock(&cv->mutex);

cv->procs[cv->waiters++] = getpid();

spin\_unlock(&cv->mutex);

spin\_unlock(m);

if (sigprocmask(SIG\_BLOCK, &(cv->mask), NULL) < 0) {

fprintf(stderr, "Sigprocmask from cv\_wait failed: %s", strerror(errno));

exit(EXIT\_FAILURE);

}

sigsuspend(&cv->mask);

if(cv->waiters > 0){

spin\_lock(&cv->mutex);

cv->procs[cv->waiters-1] = 0;

cv->waiters--;

spin\_unlock(&cv->mutex);

spin\_lock(m);

return;

}

if (sigprocmask(SIG\_UNBLOCK, &(cv->mask), NULL) < 0) {

fprintf(stderr, "Sigprocmask from cv\_wait failed: %s", strerror(errno));

exit(EXIT\_FAILURE);

}

spin\_lock(m);

}

int cv\_broadcast(struct cv \*cv){

if(cv->waiters == 0) return 0;

int awaken = 0;

spin\_lock(&cv->mutex);

for(int i = 0; i < CV\_MAXPROC;i++){

if(cv->procs[i] > 0){

kill(cv->procs[i], SIGUSR1);

awaken++;

}

}

spin\_unlock(&cv->mutex);

return awaken;

}

int cv\_signal(struct cv \*cv){

if(cv->waiters == 0) return 0;

spin\_lock(&cv->mutex);

kill(cv->procs[cv->waiters-1], SIGUSR1);

spin\_unlock(&cv->mutex);

return 1;

}

1. **Problem 4**
   1. **Fifo.h**

#include "cv.h"

#define MYFIFO\_BUFSIZ 1024

struct fifo{

unsigned long buf[MYFIFO\_BUFSIZ];

struct cv full, empty;

struct spinlock mutex;

int read, write, count;

};

void fifo\_init(struct fifo \*f);

void fifo\_wr(struct fifo \*f,unsigned long d);

unsigned long fifo\_rd(struct fifo \*f);

}

* 1. **Fifo.c**

#include "fifo.h"

int i;

void fifo\_init(struct fifo \*f) {

struct cv \*rd, \*wr ;

if((rd = (struct cv \*) mmap(NULL, sizeof(struct cv), PROT\_READ|PROT\_WRITE, MAP\_ANONYMOUS|MAP\_SHARED, -1, 0)) == MAP\_FAILED) {

fprintf(stderr, "Failed to mmap read cv: %s\n", strerror(errno));

exit(EXIT\_FAILURE);

}

if((wr = (struct cv \*) mmap(NULL, sizeof(struct cv), PROT\_READ|PROT\_WRITE, MAP\_ANONYMOUS|MAP\_SHARED, -1, 0)) == MAP\_FAILED) {

fprintf(stderr, "Failed to mmap write cv: %s\n", strerror(errno));

exit(EXIT\_FAILURE);

}

f->full = \*wr;

f->empty = \*rd;

cv\_init(&f->full);

cv\_init(&f->empty);

f->read = 0;

f->write = 0;

f->count = 0;

spin\_unlock(&f->mutex);

}

void fifo\_wr(struct fifo \*f, unsigned long d){

spin\_lock(&f->mutex);

while(f->count >= MYFIFO\_BUFSIZ)

cv\_wait(&f->full, &f->mutex);

f->buf[f->write++] = d;

f->write%=MYFIFO\_BUFSIZ;

f->count++;

cv\_signal(&f->empty);

spin\_unlock(&f->mutex);

}

unsigned long fifo\_rd(struct fifo \*f){

unsigned long d;

spin\_lock(&f->mutex);

while(f->count <= 0){

cv\_wait(&f->empty, &f->mutex);

printf("Reader %d completed\n", i++);

}

d = f->buf[f->read++];

f->read%=MYFIFO\_BUFSIZ;

f->count--;

cv\_signal(&f->full);

spin\_unlock(&f->mutex);

return d;

}

1. **Problem 5**
   1. **Ftest.c**

#include "fifo.h"

int proc, reader, status;

int writers = 8;

int length = 2048;

int main(){

struct fifo \*f;

if((f = (struct fifo \* ) mmap(NULL, sizeof (struct fifo), PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0)) == MAP\_FAILED){

fprintf(stderr, "Syscall mmap for fifo failed: %s\n", strerror(errno));

exit(EXIT\_FAILURE);

}

fifo\_init(f);

printf("Beginning test with %d writers, %d items each\n", writers, length);

pid\_t pid[writers];

for(int i = 0; i < writers; i++){

if((pid[i] = fork()) < 0){

fprintf(stderr, "Unable to fork id#%d: %s\n",i, strerror(errno));

exit(EXIT\_FAILURE);

}

if(pid[i] == 0){

proc = i;

unsigned long wr\_buf[length];

for(int j = 0; j < length; j++){

wr\_buf[j] = getpid()\*10000 + j;

fifo\_wr(f, wr\_buf[j]);

}

printf("Writer %d completed\n", i);

exit(EXIT\_SUCCESS);

}

}

if((reader = fork()) < 0){

fprintf(stderr, "Unable to fork: %s\n", strerror(errno));

exit(EXIT\_FAILURE);

}

if(reader == 0){

unsigned long rd\_buf[writers\*length];

int nread = writers\*length;

for(int i = 0; i < nread; i++){

rd\_buf[i] = fifo\_rd(f);

}

printf("All streams done\n");

}

printf("Waiting for writer children to die\n");

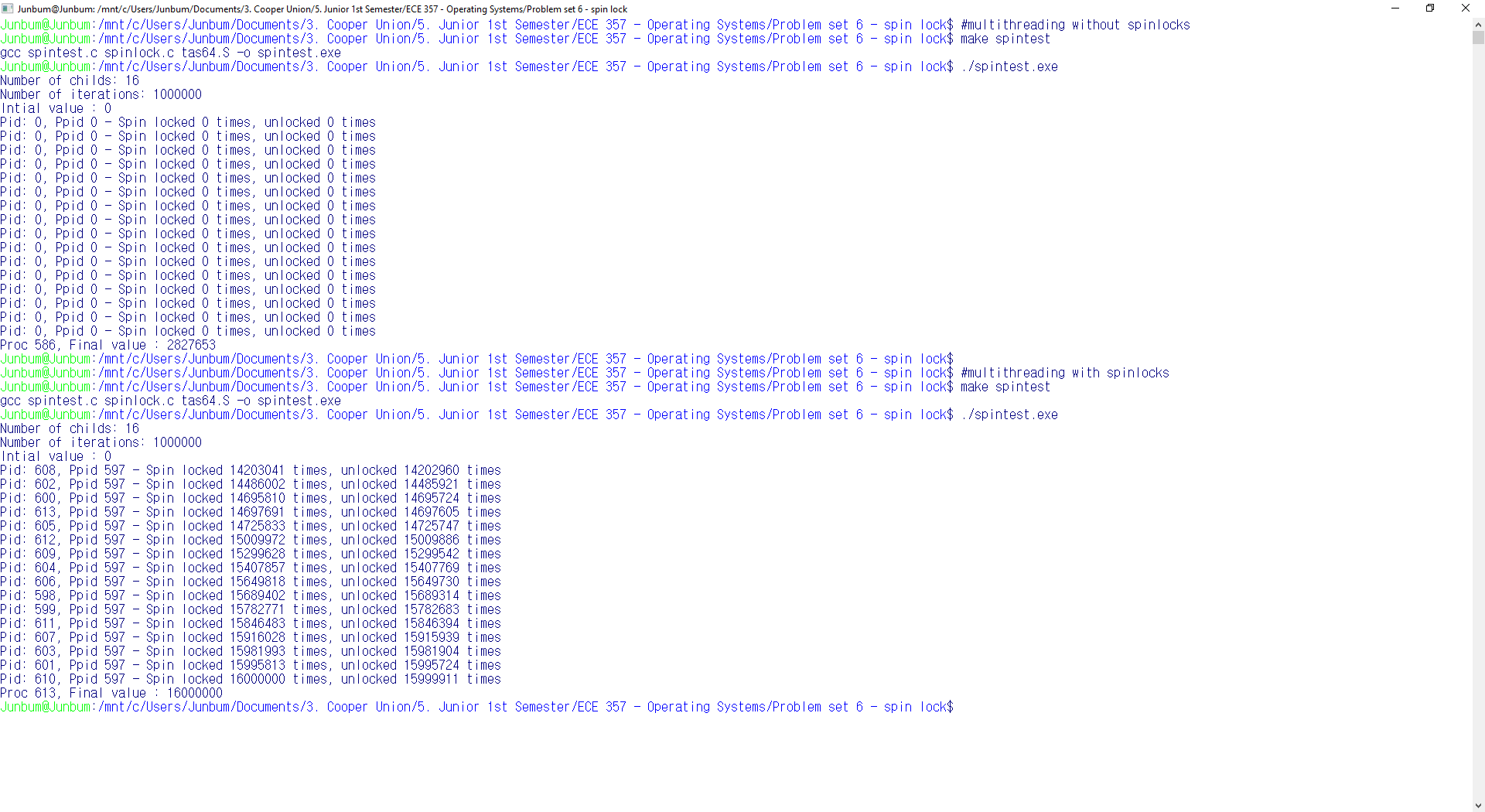
for(int i = 0; i < writers + 1; i++){

while(wait(&status)>0);

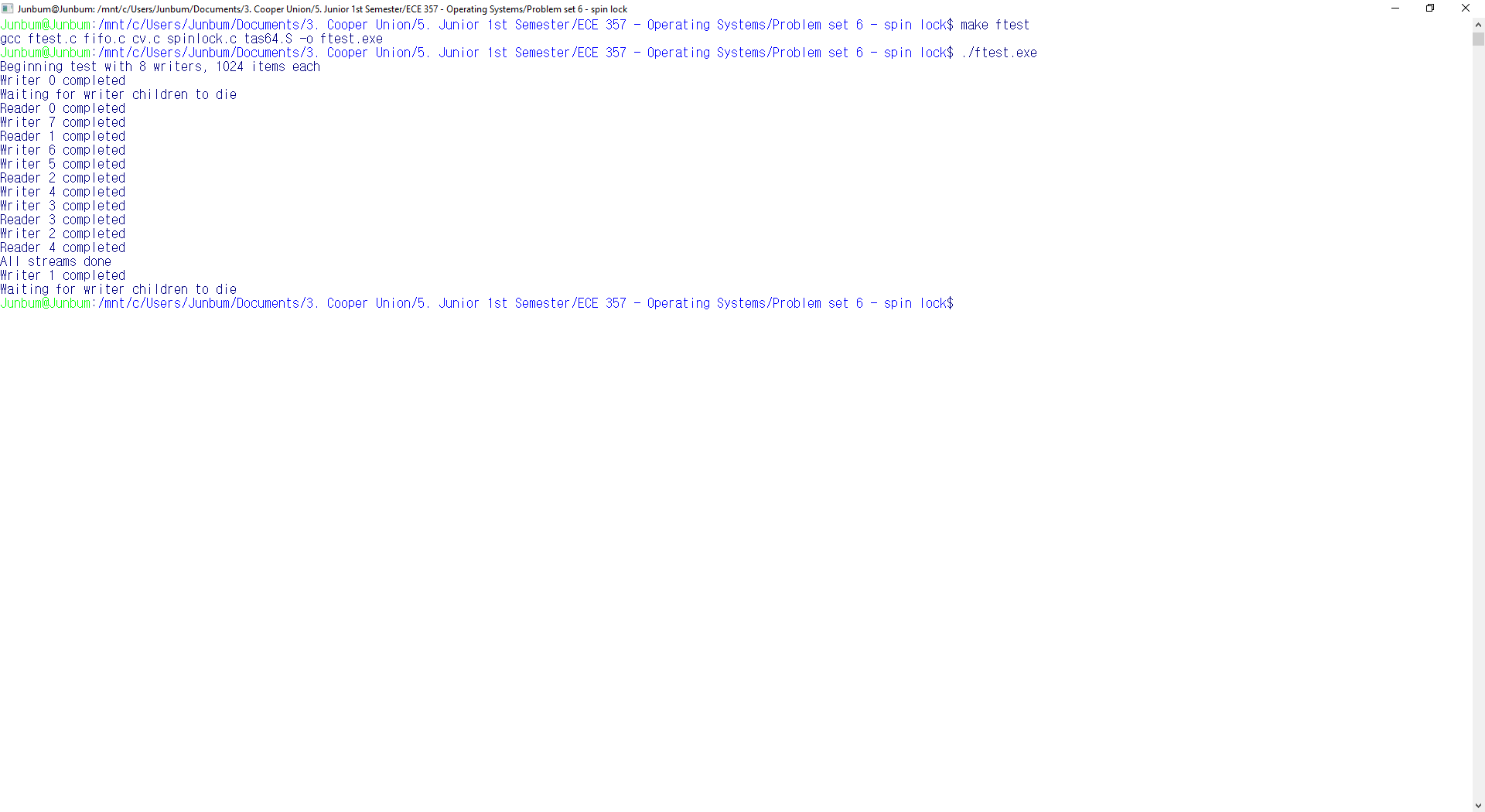
}

return 0;

} **Example 1. Spintest sample run – Multithreading with and without spinlock (Problem 2)**



**Example 2. Ftest sample run – 8 Writers with 1024 length (Problem 5)**



**Example 3. Ftest sample run – 16 Writers with 1024 length (Problem 5)**

