Yacine Manseur

11/14/15

Operating Systems

Problem Set 7

Problem 1 Output:

yacine@yacine-N53SV ~/Documents/OS/OS/PS7 $ gcc tas64.s testandset.c -o P1 -lrt

yacine@yacine-N53SV ~/Documents/OS/OS/PS7 $ ./P1

First test complete.

The value in memory should be 40000000.

The value we have is: 12025555

Second test complete.

The value in memory should be 40000000.

The value we have is: 40000000

Problem 4 Output:

yacine@yacine-N53SV ~/Documents/OS/OS/PS7 $ make

gcc -c fifotest.c

gcc -o fifotest.exe fifotest.o fifo.o sem.o myQueue.o tas64.s

yacine@yacine-N53SV ~/Documents/OS/OS/PS7 $ ./fifotest.exe 64 1

|  |  |  |  |
| --- | --- | --- | --- |
| Process 1 wrote 0  Process 2 wrote 1  Process 3 wrote 2  Process 4 wrote 3  Process 6 wrote 5  Process 5 wrote 4  Process 7 wrote 6  Process 8 wrote 7  Process 9 wrote 8  Process 10 wrote 9  Process 11 wrote 10  Process 12 wrote 11  Process 13 wrote 12  Process 14 wrote 13  Process 15 wrote 14  Process 16 wrote 15  Process 17 wrote 16  Process 18 wrote 17  Process 19 wrote 18  Process 20 wrote 19  Process 22 wrote 21  Process 24 wrote 23  Process 25 wrote 24  Process 21 wrote 20  Process 26 wrote 25  Process 27 wrote 26  Process 23 wrote 22  Process 28 wrote 27  Process 29 wrote 28  Process 30 wrote 29  Process 31 wrote 30  Process 32 wrote 31  Process 33 wrote 32  Process 34 wrote 33  Process 35 wrote 34  Process 36 wrote 35  Process 37 wrote 36  Process 38 wrote 37  Process 39 wrote 38  Process 40 wrote 39  Process 41 wrote 40  Process 42 wrote 41  Process 43 wrote 42  Process 44 wrote 43  Process 45 wrote 44  Process 46 wrote 45  Process 47 wrote 46 | Process 48 wrote 47  Process 49 wrote 48  Process 50 wrote 49  Process 51 wrote 50  Process 52 wrote 51  Process 53 wrote 52  Process 54 wrote 53  Process 55 wrote 54  Process 56 wrote 55  Process 57 wrote 56  Process 58 wrote 57  Process 59 wrote 58  Process 60 wrote 59  Process 61 wrote 60  Process 62 wrote 61  Process 63 wrote 62  Process 64 wrote 63  Value #1 read: 0  Value #2 read: 1  Value #3 read: 2  Value #4 read: 3  Value #5 read: 4  Value #6 read: 5  Value #7 read: 6  Value #8 read: 7  Value #9 read: 8  Value #10 read: 9  Value #11 read: 10  Value #12 read: 11  Value #13 read: 12  Value #14 read: 13  Value #15 read: 14  Value #16 read: 15  Value #17 read: 16  Value #18 read: 17  Value #19 read: 18  Value #20 read: 19  Value #21 read: 21  Value #22 read: 23  Value #23 read: 24  Value #24 read: 20  Value #25 read: 25  Value #26 read: 26  Value #27 read: 22  Value #28 read: 27  Value #29 read: 28  Value #30 read: 29 | Value #31 read: 30  Value #32 read: 31  Value #33 read: 32  Value #34 read: 33  Value #35 read: 34  Value #36 read: 35  Value #37 read: 36  Value #38 read: 37  Value #39 read: 38  Value #40 read: 39  Value #41 read: 40  Value #42 read: 41  Value #43 read: 42  Value #44 read: 43  Value #45 read: 44  Value #46 read: 45  Value #47 read: 46  Value #48 read: 47  Value #49 read: 48  Value #50 read: 49  Value #51 read: 50  Value #52 read: 51  Value #53 read: 52  Value #54 read: 53  Value #55 read: 54  Value #56 read: 55  Value #57 read: 56  Value #58 read: 57  Value #59 read: 58  Value #60 read: 59  Value #61 read: 60  Value #62 read: 61  Value #63 read: 62  Value #64 read: 63  Total words received: 640  0  0  0  1  1  0  0  0  0  0  0 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  .  .  .  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  Total Errors found: 2 |

**Appendix:**

Makefile:

fifotest.exe: fifotest.o fifo.o sem.o myQueue.o tas64.s

gcc -o fifotest.exe fifotest.o fifo.o sem.o myQueue.o tas64.s

testandset.o: testandset.c

gcc -c testandset.c

fifotest.o: fifotest.c

gcc -c fifotest.c

fifo.o: fifo.c fifo.h

gcc -c fifo.c

sem.o: sem.c sem.h

gcc -c sem.c

myQueue.o: myQueue.c myQueue.h

gcc -c myQueue.c

clean:

rm -f \*.exe \*.o \*.stackdump \*~

backup:

test -d backups || mkdir backups

cp \*.c backups

cp \*.h backups

cp \*.s backups

cp Makefile backups

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 7

// Problem 1 -- Test the test-and-set

// gcc tas64.s testandset.c -o P1 -lrt

// testandset.c

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <errno.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <sys/wait.h>

int tas(volatile char \*lock);

void reportError(char \*e)

{

perror(e);

exit(-1);

}

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

{

int fd;

unsigned long \*sharedInt;

pid\_t pid;

size\_t size = sizeof(unsigned long)\*2;

int procNum = 0;

int numCores = 4;

int numIterations = 10000000;

int temp; // used for incrementing the int in shared memory

fd = shm\_open("/myregion", O\_RDWR | O\_CREAT | O\_TRUNC, 0766);

if (fd < 0)

reportError("Error creating shared memory object");

// Used to avoid bus error

if(ftruncate(fd,size) < 0)

reportError("Error truncating shared memory object");

sharedInt = mmap(NULL, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (sharedInt == MAP\_FAILED)

reportError("Error creating mmap");

\*sharedInt = 0L; /\* Initialize the number \*/

for(procNum = 0; procNum < numCores; procNum++)

{

pid=fork();

if(pid == 0)

{

for(temp = 0; temp < numIterations; temp++)

(\*sharedInt)++;

exit(0);

}

}

while(wait(NULL) != -1);

printf("First test complete.\n");

printf("The value in memory should be %d.\n", numCores\*numIterations);

printf("The value we have is: %lu\n\n", \*sharedInt);

volatile char \*lock = (char\*) sharedInt+sizeof(unsigned long);

\*lock = 0;

// Reset test...

\*sharedInt = 0L;

for(procNum = 0; procNum < numCores; procNum++)

{

pid = fork();

if (pid == 0)

{

for(temp = 0; temp < numIterations; temp++)

{

while(tas(lock));

(\*sharedInt)++;

\*lock = 0;

}

exit(0);

}

}

while(wait(NULL) != -1);

printf("Second test complete.\n");

printf("The value in memory should be %d.\n", numCores\*numIterations);

printf("The value we have is: %lu\n", \*sharedInt);

return 0;

}

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 7

// sem.h

#ifndef SEM\_H

#define SEM\_H

#include <signal.h>

#include <unistd.h>

#include <sys/types.h>

#define N\_PROC 64 /\* Maximum number of virtual processors required to accept \*/

int my\_procnum; /\* Small integer identifier \*/

struct sem

{

volatile char lock; /\* Lock \*/

int semval; /\* Number of Resources \*/

int semwait[N\_PROC]; /\* Waiting \*/

pid\_t sempid[N\_PROC]; /\* ID of process that did last op \*/

};

int tas(volatile char \*lock);

// Initialize the semaphore \*s with the initial count.

void sem\_init(struct sem \*s, int count);

// Attempt to atomically decrement the semaphore

int sem\_try(struct sem \*s);

// Atomically decrement the semaphore, blocking until successful.

void sem\_wait(struct sem \*s);

// Wake other processes

void sem\_inc(struct sem \*s);

#endif

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 7

// sem.c

#include "sem.h"

// Signal Handler{

void sig\_handler(int signo){}

void sem\_init(struct sem \*s, int count)

{

s->lock = 0;

s->semval = count;

}

int sem\_try(struct sem \*s)

{

while(tas(&(s->lock)) != 0);

if (s->semval > 0)

{

s->semval--;

s->lock = 0;

return 1;

}

else

{

s->lock = 0;

return 0;

}

}

void sem\_wait(struct sem \*s)

{

while(1)

{

while(tas(&(s->lock)) != 0);

if(s->semval > 0)

{

s->semval--;

//s->semwait[my\_procnum] = 0;

s->lock = 0;

return;

}

else

{

sigset\_t mask;

s->semwait[my\_procnum] = 1;

s->sempid[my\_procnum] = getpid();

sigfillset(&mask); /\* Intialize the mask to include all signals \*/

sigdelset(&mask, SIGUSR1);

sigdelset(&mask, SIGINT); /\* So that the program can be aborted manually \*/

sigprocmask(SIG\_BLOCK, &mask, NULL); /\* Block \*/

signal(SIGUSR1, sig\_handler);

s->lock = 0;

sigsuspend(&mask); /\* Wait for either SIGUSR1 or SIGINT \*/

sigprocmask(SIG\_UNBLOCK, &mask, NULL); /\* Unblock \*/

}

}

}

void sem\_inc(struct sem \*s)

{

int id;

while(tas(&(s->lock)) != 0);

s->semval++;

for(id = 0; id < N\_PROC; id++)

{

if(s->semwait[id])

{

s->semwait[id] = 0;

kill(s->sempid[id],SIGUSR1);

//break;

}

}

s->lock = 0;

}

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 7

// fifo.h

#ifndef FIFO\_H

#define FIFO\_H

#include "sem.h"

#define MYFIFO\_BUFSIZ 4096 /\* Length of the FIFO \*/

struct fifo

{

unsigned long buf[MYFIFO\_BUFSIZ]; /\* Data buffer \*/

int front, back; /\* Pointers to front and back of FIFO \*/

struct sem mutex, read, write; /\* Semaphores to use \*/

};

/\* Initialize the shared memory FIFO \*f \*/

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

/\* Enque the data word d into the FIFO \*/

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

/\* Deque the next data word from the FIFO and return it. \*/

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

#endif

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 7

// fifo.c

#include "fifo.h"

void fifo\_init(struct fifo \*f)

{

f->front = 0;

f->back = 0;

sem\_init(&(f->mutex), 1);

sem\_init(&(f->read), 0); /\* Because FIFO is empty \*/

sem\_init(&(f->write), MYFIFO\_BUFSIZ); /\* Because FIFO is empty \*/

}

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

{

sem\_wait(&(f->write)); /\* Block until FIFO has room \*/

sem\_wait(&(f->mutex));

f->buf[f->back] = d; /\* Insert d at the back of the FIFO \*/

f->back = (f->back + 1) % MYFIFO\_BUFSIZ; /\* Increment index value \*/

sem\_inc(&(f->read)); /\* Wake the rest \*/

sem\_inc(&(f->mutex));

}

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

{

sem\_wait(&(f->read)); /\* Block until there are words queued in the FIFO \*/

sem\_wait(&(f->mutex));

unsigned long d = f->buf[f->front];

f->front = (f->front + 1) % MYFIFO\_BUFSIZ; /\* Increment index value \*/

sem\_inc(&(f->write)); /\* Wake the rest \*/

sem\_inc(&(f->mutex));

return d;

}

// Yacine Manseur

// Cooper Union Fall 2015

// ECE 357: Operating Systems

// Problem Set 7

// fifotest.c

/\*

Instructions:

1. Establish a struct fifo in shared memory

2. Create two virtual processors, one of which will be the writer and the other the reader

3. Have the writer send a fixed number of sequentially-numbered data using fifo\_wr

4. Have the reader read these and verify that all were received

5. Next, create multiple writers but one reader

a. In a successful test, all of the writers' streams will be received by the reader

complete, in (relative) sequence, with no missing or duplicated items, and all

processes will eventually run to completion and exit (no hanging).

b. A suggested approach is to treat each datum (32-bit word) as a bitwise word

consisting of an ID for the writer and the sequence number

c. It is not necessary to test under multiple readers, but your fifo code should work

correctly for this case

Notes:

1. Use reasonable test parameters

2. Remember for the acid test to make the buffer fill and empty quite a few times

3. You should be able to demonstrate failure by deliberately breaking something in

your implementation (reversing the order of two operations)

4. Then demonstrate success under a variety of strenuous conditions

\*/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/mman.h>

#include "sem.h"

#include "fifo.h"

struct fifo \*f;

void reportError(char \*e)

{

perror(e);

exit(-1);

}

void createMultipleWriters(int numProc, int numWords)

{

FILE \*written;

written = fopen("written.txt", "a");

if(written==NULL)

reportError("Error opening temp file");

int ii, jj;

pid\_t pid;

for(ii = 0; ii < numProc; ii++)

{

switch(pid=fork())

{

case -1:

reportError("Error creating fork");

case 0:

my\_procnum = ii+1;

for(jj = ii\*numWords; jj < (ii+1)\*numWords; jj++)

{

fifo\_wr(f,jj);

printf("Process %d wrote %d\n", my\_procnum, jj);

fprintf(written, "%d\n", jj);

}

exit(0);

}

}

while(wait(NULL) != -1);

fclose(written);

}

void createSingleReader(int numProc, int numWords)

{

FILE \*read;

read = fopen("read.txt", "a");

int jj, total = 0;

pid\_t pid;

unsigned long value;

switch(pid=fork())

{

case -1:

reportError("Error creating fork");

case 0:

for(jj = 0; jj < numProc\*numWords; jj++)

{

value = fifo\_rd(f);

total++;

printf("Value #%d read: %lu\n", total, value);

fprintf(read, "%lu\n", value);

}

printf("Total words received: %d\n", total);

exit(0);

}

while(wait(NULL) != -1);

fclose(read);

}

void compareResults()

{

FILE \*f1, \*f2;

int c1, c2;

int totalErrors=0;

f1 = fopen("written.txt", "r");

f2 = fopen("read.txt", "r");

c1 = fgetc(f1);

c2 = fgetc(f2);

while(c1!=EOF || c2 != EOF)

{

if(c1 != '\n' && c2 != '\n'){

if(abs(c1-c2) > 0)

totalErrors ++;

printf("%d\n", abs(c1-c2));

//printf("%c, %c\n", c1, c1);

}

c1 = fgetc(f1);

c2 = fgetc(f2);

}

printf("Total Errors found: %d\n", totalErrors);

fclose(f1);

fclose(f2);

}

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

{

if (argc != 3)

reportError("Please enter only two arguments");

int numProc = atoi(argv[1]);

int numWords = atoi(argv[2]);

if(numProc > 64)

reportError("Error: Maximum number of virtual processors exceeded");

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

if(f == MAP\_FAILED)

reportError("Error creating mmap");

fifo\_init(f);

my\_procnum = 0;

createMultipleWriters(numProc, numWords);

createSingleReader(numProc, numWords);

compareResults();

remove("written.txt");

remove("read.txt");

return 0;

}