Donghyun Park

Prof. Hakner

Operating Systems

12/27/17

PSet 6 – Synchronization

Problem 1 – Spin Lock

spinlock.h:

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <sys/wait.h>

typedef struct spinlock{

volatile char primlock;

}spinlock;

int tas(volatile char \*lock);

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

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

spinlock.c:

#include "spinlock.h"

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

while(tas(&(l->primlock)) != 0){

;

}

}

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

l->primlock = 0;

}

Problem 2 – Test the test-and-set

spintest.c:

#include "spinlock.h"

int main(int argc, char \*\*argv){

long long unsigned int nchild, niter;

pid\_t pid;

if(argc != 3){

fprintf(stderr, "Error: Please input 3 arguments\n");

exit(255);

}

nchild = atoll(argv[1]);

niter = atoll(argv[2]);

int \*map = mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_ANONYMOUS|MAP\_SHARED, 0, 0);

if(map < 0){

fprintf(stderr, "Error: failed to mmap() anonymous page - %s\n", strerror(errno));

exit(255);

}

map[0] = 0;

spinlock \*lock;

lock = (spinlock \*)(map + sizeof(spinlock));

lock -> primlock = map[1];

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

switch(pid = fork()){

case -1:

fprintf(stderr, "Error: failed to fork() #%dth time - %s\n", i, strerror(errno));

break;

case 0:

spin\_lock(lock);

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

map[0]++;

}

spin\_unlock(lock);

exit(0);

break;

}

}

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

wait(0);

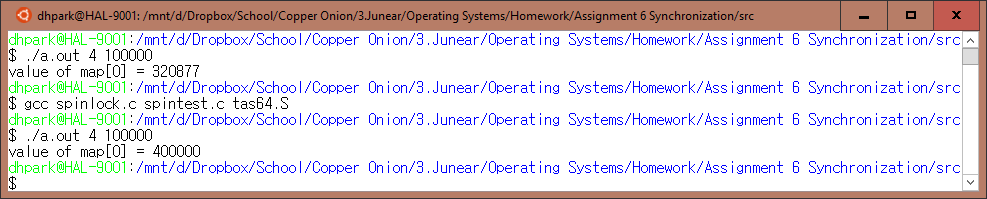
}

printf("value of map[0] = %d\n", map[0]);

return 0;

}

Output of without and then with spin lock protection:



Problem 3 – Implement Condition Variables

cv.h:

#include "spinlock.h"

typedef struct cv{

int i;

spinlock lock;

pid\_t pid[64];

sigset\_t sigMask;

}cv;

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);

cv.c:

#include "cv.h"

void sigHandle(int signum){

;

}

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

spinlock \*lock;

int \*map = mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_ANONYMOUS|MAP\_SHARED, 0, 0);

if(map < 0){

fprintf(stderr, "Error: failed to mmap() anonymous page - %s\n", strerror(errno));

exit(255);

}

lock = (spinlock \*)(map + sizeof(spinlock));

cv->lock = \*lock;

for(int i = 0; i < 64; i++){

cv->pid[i] = 0;

}

cv->i = 0;

signal(SIGUSR1, sigHandle);

sigfillset(&cv->sigMask);

sigdelset(&cv->sigMask, SIGUSR1);

}

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

if(cv->i >= 64){

fprintf(stderr, "Error: too many processes\n");

exit(255);

}

spin\_lock(&cv->lock);

cv->pid[cv->i] = getpid();

cv->i++;

spin\_unlock(&cv->lock);

spin\_unlock(mutex);

sigprocmask(SIG\_BLOCK, &cv->sigMask, NULL);

sigsuspend(&cv->sigMask);

if(cv->i > 0){

spin\_lock(&cv->lock);

cv->pid[cv->i - 1] = 0;

cv->i--;

spin\_unlock(&cv->lock);

spin\_lock(mutex);

return;

}

sigprocmask(SIG\_UNBLOCK, &cv->sigMask, NULL);

spin\_lock(mutex);

}

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

int wakeNum = 0;

spin\_lock(&cv->lock);

if(cv->i == 0){

spin\_unlock(&cv->lock);

return 0;

}

for(int j = 0; j < 64; j++){

if(cv->pid[j] > 0){

kill(cv->pid[j], SIGUSR1);

wakeNum++;

}

}

spin\_unlock(&cv->lock);

return wakeNum;

}

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

int wakeNum = 0;

spin\_lock(&cv->lock);

if(cv->i == 0){

spin\_unlock(&cv->lock);

return 0;

}

kill(cv->pid[cv->i - 1], SIGUSR1);

wakeNum++;

spin\_unlock(&cv->lock);

return wakeNum;

}

Problem 4 – A FIFO using condition variables

fifo.h:

#include "cv.h"

#define MYFIFO\_BUFSIZ 1024

typedef struct fifo{

int state, readNext, writeNext;

unsigned long buf[MYFIFO\_BUFSIZ];

spinlock lock;

cv w, r;

}fifo;

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

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

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

fifo.c:

#include "fifo.h"

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

cv \*readMap = NULL, \*writeMap = NULL;

readMap = (cv \*)mmap(NULL, sizeof(cv), PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANONYMOUS, -1, 0);

writeMap = (cv \*)mmap(NULL, sizeof(cv), PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANONYMOUS, -1, 0);

if(readMap < 0){

fprintf(stderr, "Error: failed to mmap() anonymous file for read %s\n", strerror(errno));

exit(255);

}

if(writeMap < 0){

fprintf(stderr, "Error: failed to mmap() anonymous file for write %s\n", strerror(errno));

exit(255);

}

fifo->r = \*readMap;

fifo->readNext = 0;

cv\_init(&fifo->r);

fifo->w = \*writeMap;

fifo->writeNext = 0;

cv\_init(&fifo->w);

fifo->state = 0;

fifo->lock.primlock = 0;

}

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

spin\_lock(&fifo->lock);

while(fifo->state >= MYFIFO\_BUFSIZ){

cv\_wait(&fifo->w, &fifo->lock);

}

fifo->buf[fifo->writeNext++] = x;

fifo->writeNext %= MYFIFO\_BUFSIZ;

fifo->state++;

cv\_signal(&fifo->r);

spin\_unlock(&fifo->lock);

}

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

unsigned long fifoRead;

spin\_lock(&fifo->lock);

while(fifo->state <= 0){

printf("Fifo read is %d complete\n", ++z);

cv\_wait(&fifo->r, &fifo->lock);

}

fifoRead = fifo->buf[fifo->readNext++];

fifo->readNext %= MYFIFO\_BUFSIZ;

fifo->state--;

cv\_signal(&fifo->w);

spin\_unlock(&fifo->lock);

return fifoRead;

}

Problem 5 – Test your FIFO

ftest.c:

#include "fifo.h"

int main(int argc, char \*\*argv){

int writers, items;

fifo \*fifo1;

pid\_t pid1, pid2;

if(argc < 3){

printf("Error: please input 3 arguments\n");

exit(255);

}

writers = atoi(argv[1]+2);

items = atoi(argv[2]+2);

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

fifo1 = (fifo \*)mmap(NULL, sizeof(fifo), PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANONYMOUS, -1, 0);

if(fifo1 < 0){

fprintf(stderr, "Error: failed to mmap() anonymous page for fifo - %s\n", strerror(errno));

exit(255);

}

fifo\_init(fifo1);

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

switch(pid1 = fork()){

case -1:

fprintf(stderr, "Error: failed to fork() #%dth time - %s\n", i, strerror(errno));

break;

case 0:;

unsigned long writeBuf[items];

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

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

fifo\_wr(fifo1, writeBuf[j]);

}

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

exit(0);

break;

}

}

switch(pid2 = fork()){

case -1:

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

break;

case 0:;

unsigned long readBuf[writers \* items];

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

readBuf[i] = fifo\_rd(fifo1);

}

printf("All streams done\n");

break;

}

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

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

wait(0);

}

return 0;

}

Output of testing the FIFO:

