Problem Set 6 :Synchronization

Scott Jin

2017-12-20

Contents

1 Code Listings **1**

2 Experimental Screenshots **13**

3 Narrative **14**

3.1 Problem 5 14

List of Figures

1 Test result for spintest 13
2 Test result for cv_test 14
3 verbose result for fifoAcidTest 15
4 concise result for fifoAcidTest 16

1 Code Listings

Scott Jin——TAS64.S

```
.text

.globl _tas

_tas:
pushq %rbp
movq  %rsp, %rbp
movq  $1, %rax
#APP
lock;xchgb  %al,(%rdi)
#NO_APP
movsbq  %al,%rax
pop %rbp
ret
.Lfe1:

# .size tas,.Lfe1-tas
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <errno.h>
#include <signal.h>
#include <sys/wait.h>


typedef struct spinlock{
  volatile char primitive_lock;
}spinlock;

void spin_lock(struct spinlock *l);
void spin_unlock(struct spinlock *l);
int tas(volatile char *lock);
```

```c
#include "spinlock.h"


void spin_lock(struct spinlock *l){
  while(tas(&(l->primitive_lock))!=0){
    ;  /*we dont have the lock*/
  }
  /*critical region [lock set]*/
}

void spin_unlock(struct spinlock *l){
  l->primitive_lock=0;
}
```

```c
/*
 * testspin.c
 *
 *  Created on: Dec 21, 2017
 *      Author: scott
 */
#include "spinlock.h"

int main(int argc, char * argv[]) {
  if(argc!=3){
     fprintf(stderr,"Usage:%s [num of process] [num of iteration]\n",argv[0]);
     exit(EXIT_FAILURE);
  }
  long long unsigned int fknum = atoll(argv[1]);
  long long unsigned int iternum = atoll(argv[2]);
  fprintf (stderr, "fknum=%llu\n", fknum);
  fprintf (stderr, "iternum=%llu\n", iternum);
    int * mapped_area = mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED, 0, 0 );
    if(mapped_area==MAP_FAILED){
         fprintf(stderr,"Failed to mmap ANONYMOUS page: %s\n",strerror(errno));
         exit(EXIT_FAILURE);
      }
    mapped_area[0] = 0;
    spinlock * lock;
    lock=(spinlock *)(mapped_area+sizeof(spinlock)); /*important:make sure lock is fixed*/
    lock->primitive_lock= mapped_area[1]; /*paged all 0 at first can do spin_unlock*/
    pid_t pids[fknum];

    for (int i = 0; i < fknum; i++) {
        if ((pids[i] = fork()) < 0) {
            fprintf (stderr, "ERROR-->fork failure for fork number #[%d]: %s\n", i, strerror(errno));
            return EXIT_FAILURE;
          }
        if (pids[i] == 0) {       /*child*/
            spin_lock(lock);
            for (int j = 0; j < iternum; j++) {
                mapped_area[0]++;
            }
            spin_unlock(lock);
            exit(0);
        }
    }
    for (int i = 0; i < fknum; i++) {
        wait(0);
    }
    fprintf(stderr,"%d\n",mapped_area[0]);
}
```

```c
/*
 * cv.h
 *
 *  Created on: Dec 21, 2017
 *      Author: scott
 */
#include "spinlock.h"
#define CV_MAXPROC 64

static int wait_count;
static int swakeup_count;
static int wakeup_count;

typedef struct cv {
  int count;
  spinlock lock;
  pid_t pids[CV_MAXPROC];
  sigset_t sigmask;
} cv;

void cv_init(struct cv *cv);
/* Initialize any internal data structures in cv so that it is ready for
 * use.  The initial condition is that nobody is waiting for this cv.
 * You can probably arrange your struct cv so that all-0 bytes is
 * the initialization condition.
 */
void cv_wait(struct cv *cv, struct spinlock *mutex);
/* This will be called with the spinlock mutex held by the caller (otherwise
 * results will be undefined).  Atomically record within the internals
 * of cv that the caller is going to sleep, release the mutex, and
 * go to sleep (see text below).  After waking up, re-acquire the mutex
 * before returning to the caller
*/
int cv_broadcast(struct cv *cv);
/* Wake up any and all waiters (sleepers) on this cv.  If there are no waiters
 * the call has no effect and is not "remembered" for the next time that
 * someone calls cv_wait.  cv_broadcast should be called with the same mutex
 * held that protects cv_wait, as discussed in lecture notes under "Lost
 * Wakup", but note that cv_broadcast does not take a mutex as a parameter.
 * Return value: the number of sleepers that were awoken.
 */
int cv_signal(struct cv *cv);
/* Exactly the same as cv_broadcast except at most one sleeper is awoken.
 * Your choice how to pick which one if more than one candidate
 */
```

```c
/*
 * cv.c
 *
 *  Created on: Dec 21, 2017
 *      Author: scott
 *      u need a lot of error checking
 */
#include "cv.h"

void handler(int signo) {;}

void cv_init(struct cv *cv){
    int * mapped_area = mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_ANONYMOUS | MAP_SHARED, 0, 0 );
    if(mapped_area==MAP_FAILED){
            fprintf(stderr,"Failed to mmap ANONYMOUS page[cv_init]: %s\n",strerror(errno));
            exit(EXIT_FAILURE);
        }
    spinlock * lock;
    lock=(spinlock *)(mapped_area+sizeof(spinlock)); /*important:make sure lock is fixed*/
  cv->lock=*lock;
  for(int i=0;i<CV_MAXPROC;i++){
    cv->pids[i] = 0;
  }
  cv->count = 0;
  signal(SIGUSR1, handler);/*check error*/
  sigfillset(&cv->sigmask);
  sigdelset(&cv->sigmask, SIGUSR1);
}
/*the queue must be accessed by only one thread at a time. [producer /consumer]
 */
void cv_wait(struct cv *cv, struct spinlock *mutex){
  if(cv->count>=CV_MAXPROC){   /*sanity check is there still spot?*/
    fprintf(stderr,"Error[cv_wait]-->too many processes\n");
    exit(EXIT_FAILURE);
  }
  spin_lock(&cv->lock);
  cv->pids[cv->count] = getpid(); /* Putting process to sleep*/
  cv->count++;
  spin_unlock(&cv->lock);
    spin_unlock(mutex);/* I am going to sleep, let others deal with the fifo*/
  //fprintf(stderr,"cv_wait:process Going to sleep....\n");
  sigprocmask(SIG_BLOCK, &cv->sigmask, NULL);/*block all other signals and wait for sigUSR*/
    sigsuspend(&cv->sigmask);
    /*now signal returns*/
    if(cv->count>0){ /*is there still process waiting*/
    spin_lock(&cv->lock);
        //fprintf(stderr,"cv_wait: process[sigsuspended] woke up!\n");
        cv->pids[cv->count-1] = 0; /* now our process is awake, remove it from the list*/
        cv->count--;
        spin_unlock(&cv->lock);
          spin_lock(mutex); /*once return the lock should be acquired by the user[who is responsible to unlock it]*/
        return;
    }
    sigprocmask(SIG_UNBLOCK, &cv->sigmask, NULL);
    spin_lock(mutex);
}

int cv_broadcast(struct cv *cv){
  spin_lock(&cv->lock);
  int wakupcount = 0;
  if(cv->count == 0){ /*no effect on no waiter*/
    //fprintf(stderr,"cv_broadcast:No one is waiting, returning\n");
    spin_unlock(&cv->lock);
    return 0;
```

```
  }
  for(int i=0;i<CV_MAXPROC;i++){
    if(cv->pids[i]>0){
      kill(cv->pids[i],SIGUSR1);/*wake when all up*/
      wakupcount++;
    }
  }
  spin_unlock(&cv->lock);
  return wakupcount;
}

int cv_signal(struct cv *cv){
  spin_lock(&cv->lock);
  //fprintf(stderr, "now we have the cv->lock\n");
  int wakupcount = 0;
  if(cv->count==0){
    //fprintf(stderr,"cv_signal:No one is waiting, returning\n");
    spin_unlock(&cv->lock);
    return 0;
  }
  kill(cv->pids[cv->count-1],SIGUSR1);
  wakupcount++;
  spin_unlock(&cv->lock);
  //fprintf(stderr, "now we unlock the cv->lock\n");
  return wakupcount;
}
```

```c
/*
 * cv_test.c
 *
 *  Created on: Dec 22, 2017
 *      Author: scott
 */


#include "cv.h"
int main(){
  spinlock useless_lock;
  cv* c;
  c = (struct cv *) mmap (NULL, sizeof (cv), PROT_READ | PROT_WRITE, MAP_SHARED |
      MAP_ANONYMOUS, -1, 0);
  cv_init(c);
  int pid;
  for(int i=0;i<2;i++){
    if ((pid = fork()) < 0) {
      fprintf (stderr, "ERROR-->fork failure for fork number #[%d]: %s\n", i, strerror (
          errno));
      return EXIT_FAILURE;
    }
    if (pid==0){
      fprintf (stderr, "Successfully forked; putting it to wait\n");
      cv_wait(c,&useless_lock);
      exit(0);
    }
  }
  sleep(2);
  fprintf (stderr, "waking them all up\n");
  int wakeup=cv_broadcast(c);
  for (int i = 0; i < 2; i++) {
      fprintf(stderr,"Waiting for children to die\n");
      wait(0);
    }
  fprintf(stderr,"wakeup number=%d\n",wakeup);
  return 0;
}
```

```c
/*
 * fifo.h
 *
 *  Created on: Dec 22, 2017
 *      Author: scott
 */
#include "cv.h"
#define MYFIFO_BUFSIZ 1000

typedef struct fifo{
    unsigned long buf[MYFIFO_BUFSIZ];
    int occupied;
    int next_read;
    int next_write;
    spinlock FIFO_lock;
    cv wr;
    cv rd;
}fifo;

void fifo_init(struct fifo *f);
/* Initialize the shared memory FIFO *f including any required underlying
 * initializations (such as calling cv_init).  The FIFO will have a static
 * fifo length of MYFIFO_BUFSIZ elements.   #define this in fifo.h.
 * A value of 1K is reasonable.
*/
void fifo_wr(struct fifo *f,unsigned long d);
/* Enqueue the data word d into the FIFO, blocking unless and until the
 * FIFO has room to accept it. (i.e. block until !full)
 */
unsigned long fifo_rd(struct fifo *f);
/* Dequeue the next data word from the FIFO and return it.  Block unless
 * and until there are available words.  (i.e. block until !empty)
 */
```

```c
/*
 * fifo.c
 *
 *  Created on: Dec 22, 2017
 *      Author: scott
 */
#include "fifo.h"

int I=0;

void fifo_init(struct fifo *f){
  cv* rdp=NULL;
  cv* wrp=NULL;
  rdp = (cv *) mmap (NULL, sizeof (cv), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
  wrp = (cv *) mmap (NULL, sizeof (cv), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if(rdp==MAP_FAILED||wrp==MAP_FAILED){
        fprintf(stderr,"Failed to mmap ANONYMOUS page: %s\n",strerror(errno));
        exit(EXIT_FAILURE);
    }
  f->rd=*rdp;
  f->wr=*wrp;
  cv_init(&f->rd);
  cv_init(&f->wr);
  f->next_write=0;
  f->next_read=0;
    f->occupied=0;
    f->FIFO_lock.primitive_lock=0; /*lock initalization*/
}

void fifo_wr(struct fifo *f,unsigned long d){
    spin_lock(&f->FIFO_lock);
    //fprintf(stderr, "now we have the lock\n");

    while (f->occupied >= MYFIFO_BUFSIZ) {
     //fprintf(stderr, "the fifo is full put current writer to sleep");
     cv_wait(&f->wr, &f->FIFO_lock); /*when it return lock is acquired*/
   }
   f->buf[f->next_write++] = d;
   f->next_write %= MYFIFO_BUFSIZ;
   f->occupied++;
   /* as now: either f->occupied < MYFIFO_BUFSIZ and f->next_write is the index
         of the next empty slot in the buffer, or
         f->occupied == MYFIFO_BUFSIZ and f->next_write is the index of the
         next (occupied) slot that will be emptied by a consumer
         (such as b->next_write == b->next_read) */
    cv_signal(&f->rd);
    spin_unlock(&f->FIFO_lock);
    //fprintf(stderr, "now we unlock the lock\n");

}

unsigned long fifo_rd(struct fifo *f){
  unsigned long item;
  spin_lock(&f->FIFO_lock);
  while(f->occupied <= 0) {
    //fprintf(stderr, "the fifo is empty put current reader to sleep\n");
        fprintf(stderr, "read stream %d complete\n",++I);
        cv_wait(&f->rd, &f->FIFO_lock);
  }
  item = f->buf[f->next_read++];
  f->next_read %= MYFIFO_BUFSIZ;
  f->occupied--;
    /* now: either f->occupied > 0 and b->next_read is the index
       of the next occupied slot in the buffer, or
```

```
        f->occupied == 0 and f->nextout is the index of the next
        (empty) slot that will be filled by a producer (such as
        f->next_read == b->next_write) */
  cv_signal(&f->wr);
  spin_unlock(&f->FIFO_lock);
  return(item);
}
```

```c
/*
 * fifoAcidTest.c
 *
 *  Created on: Dec 22, 2017
 *      Author: scott
 */
#include "fifo.h"
int my_procnum;
int main() {
    fifo * f;
  f = (fifo *) mmap (NULL, sizeof (fifo), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if(f==MAP_FAILED){
        fprintf(stderr,"Failed to mmap ANONYMOUS page: %s\n",strerror(errno));
        exit(EXIT_FAILURE);
    }
  fifo_init(f);
  int nWriters = 5;
  int writeLength = MYFIFO_BUFSIZ; /* Twice the fifo buffer size*/
  fprintf (stderr, "Beginning test with %d writers, %d items each\n", nWriters, writeLength);
  pid_t pids[nWriters];
  for (int i = 0; i < nWriters; i++) {
    //fprintf (stderr,"Forking writer number %d\n", i);
    if ((pids[i] = fork()) < 0) {
                fprintf (stderr, "ERROR--> WRITER fork failure for fork number #[%d]: %s\n", i, strerror (errno));
                return EXIT_FAILURE;
    }
    if (pids[i] == 0) {
      my_procnum = i;
      unsigned long writeBuf[writeLength];
      for (int j = 0; j <writeLength; j++) {
                writeBuf[j] = j + getpid()*10000;
        fifo_wr(f, writeBuf[j]);
        //fprintf(stderr,"Writer %d wrote %lu\n",i,writeBuf[j]);
      }
      fprintf(stderr,"Writer %d completed\n",i);
      exit(EXIT_SUCCESS);
    }
  }
  //fprintf (stderr,"Forking the lonely reader\n");
  int lonelyreader = fork();
  if (lonelyreader < 0) {
    fprintf (stderr, "ERROR--> READER fork failure: %s\n",strerror (errno));
    return EXIT_FAILURE;
  }
  if (lonelyreader == 0) {
    my_procnum = nWriters; /*one procnum higher that last writer*/
    unsigned long readBuf[nWriters*writeLength];
    int rd = nWriters*writeLength;
    for (int i = 0; i < rd; i++) {
      readBuf[i]=fifo_rd(f);
    }
        fprintf(stderr,"ALL streams done\n");
        exit(0);
  }
  /*collect children*/
  for (int i = 0; i < nWriters+1 ; i++) { /*+1*/
    //fprintf(stderr,"Waiting for writer children & reader to die\n");
    wait(0);
  }
    fprintf(stderr,"Byebye!\n");
    return 0;
}
```

2 Experimental Screenshots

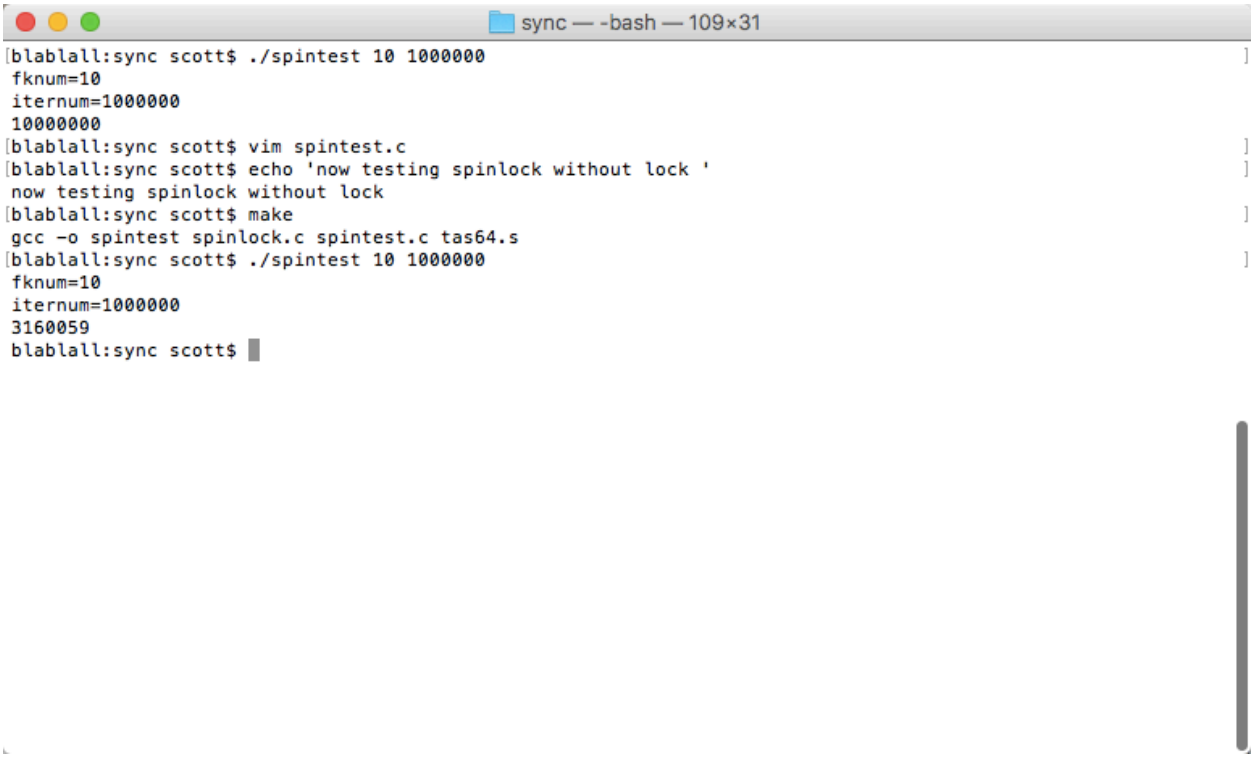

Figure 1: Test result for spintest

```
/Users/scott/Documents/CDT/sync/cmake-build-debug/cv_test
Successfully forked; putting it to wait
Successfully forked; putting it to wait
waking them all up
Waiting for children to die
Waiting for children to die
wakeup number=2

Process finished with exit code 0
```

Figure 2: Test result for cv_test

3 Narrative

3.1 Problem 5

The dileberate wronging of the program would defintely cause the program to go err. Turns out to be my bug, The mistake of forgetting to unlock the cv-¿lock in the count=0 [waiting process] case casued fifo$_{w}rtohaltwithcv_{s}ignalcantacquirethelock.Theresultistrivialtoshowhere$.

```
sync — -bash — 101×51
Last login: Fri Dec 22 18:33:52 on console
You have mail.
blablall:~ scott$ cd /Users/scott/Documents/CDT/sync
blablall:sync scott$ make
gcc fifoAcidTest.c tas64.s cv.c fifo.c -o fifotest -g
gcc cv_test.c tas64.s cv.c -o cv_test -g
blablall:sync scott$ ./fifltest
-bash: ./fifltest: No such file or directory
blablall:sync scott$ ./fifotest
Beginning tset with 5 writers, 1000 items each
Writer 0 completed
the fifo is full put current writer to sleepcv_wait:process Going to sleep....
the fifo is full put current writer to sleepcv_wait:process Going to sleep....
Forking the lonely reader
the fifo is full put current writer to sleepcv_wait:process Going to sleep....
Waiting for writer children & reader to die
Waiting for writer children & reader to die
the fifo is full put current writer to sleepcv_wait:process Going to sleep....
the fifo is empty put current reader to sleep
cv_wait: process[sigsuspended] woke up!
cv_wait:process Going to sleep....
cv_wait: process[sigsuspended] woke up!
the fifo is empty put current reader to sleep
cv_wait:process Going to sleep....
cv_wait: process[sigsuspended] woke up!
cv_wait: process[sigsuspended] woke up!
the fifo is full put current writer to sleepcv_wait:process Going to sleep....
the fifo is full put current writer to sleepcv_wait:process Going to sleep....
cv_wait: process[sigsuspended] woke up!
the fifo is empty put current reader to sleep
cv_wait:process Going to sleep....
cv_wait: process[sigsuspended] woke up!
Writer 3 completed
Waiting for writer children & reader to die
Writer 4 completed
cv_wait: process[sigsuspended] woke up!
Waiting for writer children & reader to die
cv_wait: process[sigsuspended] woke up!
the fifo is empty put current reader to sleep
cv_wait: process[sigsuspended] woke up!
cv_wait:process Going to sleep....
Writer 1 completed
cv_wait: process[sigsuspended] woke up!
the fifo is empty put current reader to sleep
cv_wait:process Going to sleep....
Waiting for writer children & reader to die
Writer 2 completed
cv_wait: process[sigsuspended] woke up!
Waiting for writer children & reader to die
Byebye!
blablall:sync scott$
```

Figure 3: verbose result for fifoAcidTest

```
/Users/scott/Documents/CDT/sync/cmake-build-debug/fifotest
Beginning test with 5 writers, 1000 items each
Writer 0 completed
read stream 1 complete
Writer 4 completed
read stream 2 complete
Writer 3 completed
read stream 3 complete
Writer 1 completed
read stream 4 complete
Writer 2 completed
ALL streams done
Byebye!

Process finished with exit code 0
```

Figure 4: concise result for fifoAcidTest