Camille Chow ECE357 PS7

Problem 1 - Spinlock

spinlock.h

```
#ifndef ____spinlock__
#define ____spinlock__
#include <sys/types.h>
#include <unistd.h>
struct spinlock {
   char lock;
   pid_t pid;
};
void sp_init(struct spinlock *l);
int tas(volatile char* lock);
void spin_lock(struct spinlock *l);
void spin_unlock(struct spinlock *l);
#endif
```

spinlock.c

```
#include "spinlock.h"
void sp_init(struct spinlock *l) {
   l->lock = 0;
   l->pid = 0;
}
void spin_lock(struct spinlock *l) {
   while (tas(&(l->lock))!=0){}
   l->pid = getpid();
}
void spin_unlock(struct spinlock *l) {
   l->lock = 0;
}
```

Problem 2 - Test the TAS

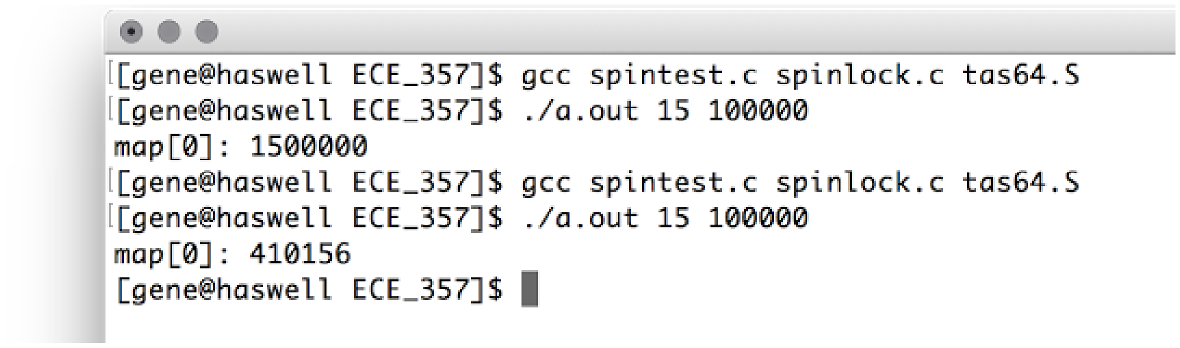
spintest.c

```
#include "spinlock.h"
#include <stdio.h>
#include <sys/mman.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

int main(int argc, char**argv) {
   if (argc < 3)
   {
      fprintf(stderr, "not enough input arguments\n");
      return 0;
   }
   int nchild = atoi(argv[1]);
   int niter = atoi(argv[2]);
   int* map;
   if ((map = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_ANON|MAP_SHARED, -1, 0)) < 0)
   {
      fprintf(stderr, "Error mmap-ing: %s\n", strerror(errno));
      exit(255);
   }
   map[0] = 0;
   struct spinlock *slp = (struct spinlock *) &map[1];
   sp_init(slp);
```

```
    pid_t pid;
    for (int i = 0; i < nchild; i++) {
        switch (pid = fork()) {
            case -1:
                fprintf(stderr,"Error forking: %s\n", strerror(errno));
                break;
            case 0:
                spin_lock(slp);
                for (int j = 0; j < niter; j++) {
                    map[0]++;
                }
                spin_unlock(slp);
                _exit(0);
                break;
        }
    }
    for (int k = 0; k < nchild; k++) {
        wait(NULL);
    }
    printf("map[0]: %d\n", map[0]);
    return 0;
}
```

Output with and without spinlock



```
[gene@haswell ECE_357]$ gcc spintest.c spinlock.c tas64.S
[gene@haswell ECE_357]$ ./a.out 15 100000
map[0]: 1500000
[gene@haswell ECE_357]$ gcc spintest.c spinlock.c tas64.S
[gene@haswell ECE_357]$ ./a.out 15 100000
map[0]: 410156
[gene@haswell ECE_357]$
```

Problem 3 - Condition Variables

cv.h

```
#ifndef ____cv__
#define ____cv__
#include "spinlock.h"
#define CV_MAXPROC 64
struct cv {
        struct spinlock sl;
        pid_t waitlist[CV_MAXPROC];
        int wait_cnt;
};
void handler(int sig);
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);
#endif
```

cv.c

```
#include "cv.h"
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>

void handler(int sig) {
}
void cv_init(struct cv* cv) {
        for (int i = 0; i < CV_MAXPROC; i++) {
                cv->waitlist[i] = 0;
        }
        sp_init(&cv->sl);
        cv->wait_cnt = 0;
}
void cv_wait(struct cv* cv, struct spinlock* mutex) {
        sigset_t mask, oldmask;
        sigfillset(&mask);
        sigdelset(&mask, SIGUSR1);
        if (sigprocmask(SIG_BLOCK, &mask, &oldmask) < 0)
   {
      fprintf(stderr, "Error with sigprocmask: %s\n", strerror(errno));
      _exit(255);
   }
   signal(SIGUSR1, handler);
        spin_lock(&cv->sl);
        if (cv->wait_cnt >= CV_MAXPROC) {
                fprintf(stderr, "Error: too many processes!\n");
      _exit(255);
        }
        cv->waitlist[cv->wait_cnt++] = getpid();
        spin_unlock(mutex);
        spin_unlock(&cv->sl);
        sigsuspend(&oldmask);
        spin_lock(mutex);
}
int cv_broadcast(struct cv* cv) {
        spin_lock(&cv->sl);
        for (int i = 0; i < cv->wait_cnt; i++) {
```

```
			if (kill(cv->waitlist[i],SIGUSR1) < 0) {
				fprintf(stderr, "Error killing process %d: %s\n", cv->waitlist[i],
strerror(errno));
		_exit(255);
			}
			cv->waitlist[i] = 0;
		}
		int sleepers = cv->wait_cnt;
		cv->wait_cnt = 0;
		spin_unlock(&cv->sl);
		return sleepers;
}
int cv_signal(struct cv* cv) {
		if (cv->wait_cnt > 0) {
			spin_lock(&cv->sl);
			if (kill(cv->waitlist[cv->wait_cnt - 1],SIGUSR1) < 0) {
				fprintf(stderr, "Error killing process %d: %s\n",
cv->waitlist[cv->wait_cnt - 1], strerror(errno));
		_exit(255);
			}
			cv->waitlist[cv->wait_cnt - 1] = 0;
			cv->wait_cnt--;
			spin_unlock(&cv->sl);
			return 1;
		}
		return 0;
}
```

Problem 4 - FIFO

fifo.h

```
#ifndef ____fifo__
#define ____fifo__
#include "cv.h"
#define MYFIFO_BUFSIZE 1000
struct fifo {
        unsigned long data[MYFIFO_BUFSIZE];
        int wr_ind;
        int rd_ind;
        int item_cnt;
        struct cv full, empty;
        struct spinlock mutex;
};
void fifo_init(struct fifo *f);
void fifo_wr(struct fifo *f,unsigned long d);
unsigned long fifo_rd(struct fifo *f);
#endif
```

fifo.c

```
#include "fifo.h"

void fifo_init(struct fifo *f) {
        f->wr_ind = 0;
        f->rd_ind = 0;
        f->item_cnt = 0;
        cv_init(&f->full);
        cv_init(&f->empty);
        sp_init(&f->mutex);
}
```

```
void fifo_wr(struct fifo *f,unsigned long d) {
        spin_lock(&f->mutex);
        while(f->item_cnt >= MYFIFO_BUFSIZE) {
                cv_wait(&f->full,&f->mutex);
        }
        f->data[f->wr_ind++] = d;
        f->wr_ind %= MYFIFO_BUFSIZE;
        f->item_cnt++;
        cv_signal(&f->empty);
        spin_unlock(&f->mutex);
}
unsigned long fifo_rd(struct fifo *f) {
        unsigned long d;
        spin_lock(&f->mutex);
        while(f->item_cnt <= 0) {
                cv_wait(&f->empty,&f->mutex);
        }
        d = f->data[f->rd_ind++];
        f->rd_ind %= MYFIFO_BUFSIZE;
        f->item_cnt--;
        cv_signal(&f->full);
        spin_unlock(&f->mutex);
        return d;
}
```

Problem 4 - FIFO Test

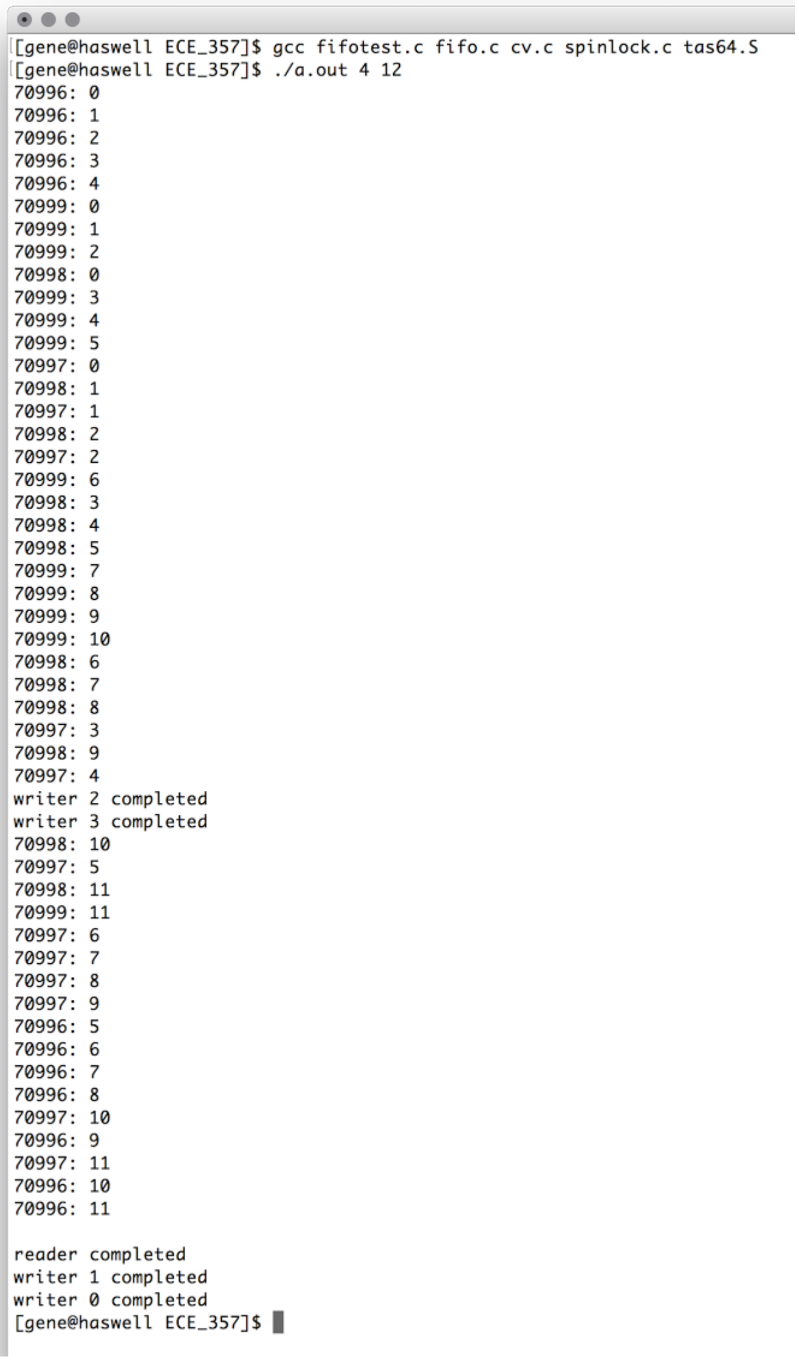
fifotest.c

```
#include "fifo.h"
#include <stdlib.h>
#include <sys/mman.h>
#include <signal.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <sys/wait.h>

int main(int argc, char**argv) {
        if (argc < 3)
   {
      fprintf(stderr, "not enough input arguments\n");
      return 0;
   }
   int nwriters = atoi(argv[1]);
   int nitems = atoi(argv[2]);
        struct fifo* f;
        void *map;
        if ((map = mmap(NULL, 16*MYFIFO_BUFSIZE, PROT_READ|PROT_WRITE,
MAP_ANON|MAP_SHARED, -1, 0)) < 0)
   {
      fprintf(stderr, "Error mmap-ing: %s\n", strerror(errno));
      exit(255);
   }
   f = (struct fifo *) map;
   fifo_init(f);
```

```
     pid_t pid;
     for (int i = 0; i < nwriters; i++) { //create writers
   switch (pid = fork()) {
      case -1:
         fprintf(stderr,"Error forking: %s\n", strerror(errno));
         break;
      case 0:
         for (long i = 0; i < nitems; i++) {
              fifo_wr(f,(getpid()<<8)|i);
         }
         fprintf(stderr, "writer %d completed\n", i);
         _exit(0);
         break;
   }
}
switch (pid = fork()) { //create reader
   case -1:
      fprintf(stderr,"Error forking: %s\n", strerror(errno));
      break;
   case 0:
     for (long i = 0; i < nitems*nwriters; i++) {
              long val = fifo_rd(f);
         fprintf(stderr, "%d: %d\n", val >> 8, val & 0xff);
      }
      fprintf(stderr, "\nreader completed\n");
      _exit(0);
      break;
}
for (int k = 0; k < nwriters + 1; k++) {
        wait(NULL);
    }
        return 0;
}
```

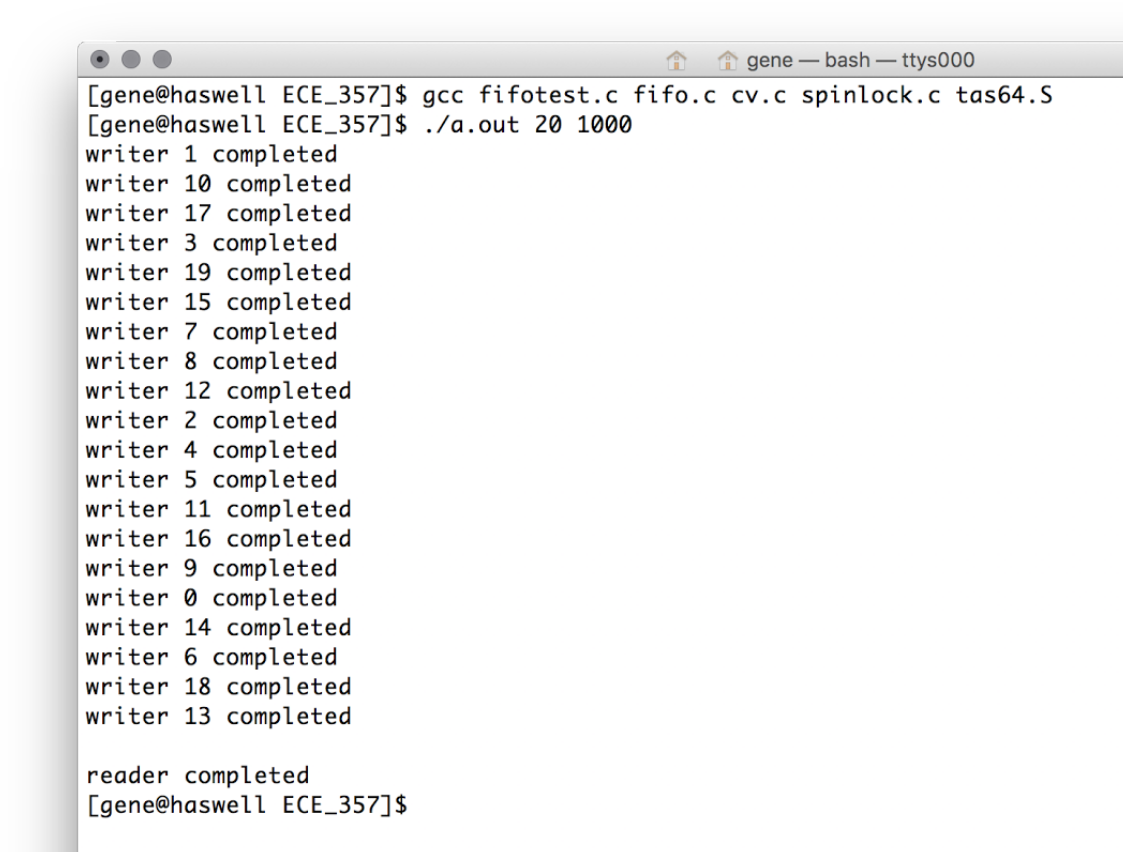
Output (MYFIFO_BUFSIZE = 10)



```
[gene@haswell ECE_357]$ gcc fifotest.c fifo.c cv.c spinlock.c tas64.S
[gene@haswell ECE_357]$ ./a.out 4 12
70996: 0
70996: 1
70996: 2
70996: 3
70996: 4
70999: 0
70999: 1
70999: 2
70998: 0
70999: 3
70999: 4
70999: 5
70997: 0
70998: 1
70997: 1
70998: 2
70997: 2
70999: 6
70998: 3
70998: 4
70998: 5
70999: 7
70999: 8
70999: 9
70999: 10
70998: 6
70998: 7
70998: 8
70997: 3
70998: 9
70997: 4
writer 2 completed
writer 3 completed
70998: 10
70997: 5
70998: 11
70999: 11
70997: 6
70997: 7
70997: 8
70997: 9
70996: 5
70996: 6
70996: 7
70996: 8
70997: 10
70996: 9
70997: 11
70996: 10
70996: 11

reader completed
writer 1 completed
writer 0 completed
[gene@haswell ECE_357]$
```

Output (MYFIFO_BUFSIZE = 1000, read output commented out)



```
[gene@haswell ECE_357]$ gcc fifotest.c fifo.c cv.c spinlock.c tas64.S
[gene@haswell ECE_357]$ ./a.out 20 1000
writer 1 completed
writer 10 completed
writer 17 completed
writer 3 completed
writer 19 completed
writer 15 completed
writer 7 completed
writer 8 completed
writer 12 completed
writer 2 completed
writer 4 completed
writer 5 completed
writer 11 completed
writer 16 completed
writer 9 completed
writer 0 completed
writer 14 completed
writer 6 completed
writer 18 completed
writer 13 completed

reader completed
[gene@haswell ECE_357]$
```