Benjamin Kaplan - PS 6

Problem 2
==============================================
Without TAS:

```
binyamin@BenjaminButtox:~/Documents$ ./a.out 300 300
...
memory = 89969
binyamin@BenjaminButtox:~/Documents$ ./a.out 300 300
...
memory = 89924
binyamin@BenjaminButtox:~/Documents$ ./a.out 300 300
...
memory = 89941
binyamin@BenjaminButtox:~/Documents$ ./a.out 300 300
...
memory = 89970
```

With TAS:

```
binyamin@BenjaminButtox:~/Documents$ ./a.out 300 300
...
memory = 90000
binyamin@BenjaminButtox:~/Documents$ ./a.out 300 300
...
memory = 90000
binyamin@BenjaminButtox:~/Documents$ ./a.out 300 300
...
memory = 90000
```

cv.h
===============================================

```
#ifndef CV_H_
#define CV_H_
#include <unistd.h>
#define CV_MAXPROC 65
struct cv{
  pid_t waiters[CV_MAXPROC];
  int num_waiters;
  pid_t sleeping_caller;
  struct spinlock lock;
};

void cv_init(struct cv *cv);

void cv_wait(struct cv *cv, struct spinlock *mutex);

int cv_boradcast(struct cv *cv);

int cv_signal(struct cv *cv);

#endif
```

cv.c
===========================================

```
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include "cv.h"
#include "spinlock.h"
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
```

```
#include <unistd.h>

void cv_init(struct cv *cv){

  cv->num_waiters = 0;
  cv->sleeping_caller = 0;
}

void cv_wait(struct cv *cv, struct spinlock *mutex){
  cv->sleeping_caller = getpid();
  spin_unlock(mutex);
  sigset_t set;
  sigfillset(&set);
  sigsuspend(&set);
}

int cv_broadcast(struct cv *cv){
  int j = 0;
  int awoken = 0;
  for(j= 0 ; j<cv->num_waiters; j++){
    if(kill(cv->waiters[j], SIGUSR1) <0)
      fprintf(stderr, "Error %s, Errno: %d\n", strerror(errno), errno);
    awoken++;
  }
  return awoken;
}

int cv_signal(struct cv *cv){
  int sig;
  while(kill(cv->waiters[sig], SIGUSR1)<0){
    fprintf(stderr, "Error: Unable to signal - %s, Errno: %d\n", strerror(errno), errno);
    sig++;                //if first process cannot be signaled,
                          //then next process is chosen
    if(sig > numwaiters){ //if the end of the list is reached,
      sig = 0;            //it will start again. This may end up in an endless loop,
    }                     //but this is unlikely. More likely that some process will act
  }
  return 1;
}



fifo.h
=============================================
#ifndef FIFO_H_
#define FIFO_H_
#include "spinlock.h"
#define MYFIFO_SIZE 1024
struct fifo{
  struct spinlock lock;
  struct cv full, empty;
  unsigned long buf[MYFIFO_SIZE];
  int next_write, next_read;
  int item_count;
}fifo;

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"
#include <fcntl.h>
#include <errno.h>
```

```
#include <string.h>
#include <unistd.h>
#include <stdio.h>

void fifo_init(struct fifo *f){
  //fifo->lock->lock = 0;
  fifo->full = cv_init(&fifo->full);
  fifo->full=cv_init(&fifo->empty);
  fifo->next_write = 0;
  fifo->next_read = 0;
  fifo->item_count = 0;
}

void fifo_wr(struct fifo *f, unsigned long d){
  spin_lock(&fifo->lock);
  while(fifo->item_count >= MYFIFO_SIZE)
    cv_wait(&fifo->full, &fifo->lock);
  fifo->buf[fifo->next_write++] = d;
  fifo->next_write %=MYFIFO_SIZE;
  fifo->item_count++;
  cv_signal(&fifo-> empty);
  spin_unlock(&fifo_lock);
}

unsigned long fifo_rd(struct fifo *f){
  unsigned long d;
  spin_lock(&fifo->lock);
  while(&fifo->item_count<=0)
    cv_wait(&fifo->empty, &fifo->lock);
  d = fifo->buf[fifo->next_read++];
  fifo->next_read %= MYFIFO_SIZE;
  fifo->item_count--;
  cv_signal(&fifo->full);
  spin_unlock(&fifo->lock);
  return d;
}

#include <errno.h>
#include <string.h>
#include "cv.h"
#include <signal.h>
#include "spinlock.h"
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include "fifo.h"
#include <sys/mman.h>

void sigusr1_handler(int signo){
  ;
}

int main(int argc, char **argv){
  if(argv[1] == NULL){
    fprintf(stderr, "missing arguments!\n");
    return -1;
  }
  int num = atoi(argv[1]);
  if(0> signal( SIGUSR1, sigusr1_handler)){
    fprintf(stderr, "Error: %s, Errno: %d\n", strerror(errno), errno);
    exit(EXIT_FAILURE);
  }

  char * memory = mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANON, -1, 0);
  if(memory == NULL){
    fprintf(stderr, "Error: %s, Errno: %d\n", strerror(errno), errno);
    exit(EXIT_FAILURE);
  }
  struct fifo fifo1;
  memory = fifo1;
  fifo_init(&fifo1);
  pid_t child1;
  pid_t child2;
  int k = 0;
  //for(k = 0; k<2; k++){
    if((child1 = fork())<0){
      fprintf(stderr, "Error: %s, Errno: %d\n", strerror(errno), errno);
      return -1;
    }
    if(child1 ==0){//CHILD1 - WRITE
      int l = 0;
      for(l = 0; l< 2048; l++){
        fifo_wr(&fifo1, l);
      }

    }
    else{// PARENT
      if((child2 = fork())<0){
      fprintf(stderr, "Error: %s, Errno: %d\n", strerror(errno), errno);
      return -1;
      }
      if(child2 ==0){//CHILD2 - READ
        fprintf(stderr, "%li\n", fifo_rd(&fifo1));
      }
      else  //Parent
        ;

    }
  //}

}
```