Task 1  
/\*

\* Author: Samuel Casto

\* PantherID: 6330314

\* Program Description: This program will take a command line argument not greater than 12 which will be the max number

\* of reader threads that will access and print a shared value. This value will be updated by a single writer thread. This

\* will continue until each reader thread has accessed the shared value 2,000,000 times and the writer will increment the

\* shared value 25,000 times. Readers will print the shared value when they access it along with their name, and the writer

\* will print that it has finished writing each time it updates the shared values.

\*/

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

//defining our max number of readers

#define MAX\_READERS 12

//defining the max number of reads by readers

#define MAX\_READS 2000000

//defining our max number of writer incrememnts

#define MAX\_INCR 25000

//defining our struct holding the shared value

struct sharedData {

int value;

};

//creating two semaphores, one for readers and one for writers

sem\_t writeSem, readSem;

//global sharedData variable

struct sharedData \*sharedValue;

//global curReaders variable tracking the number of threads currently reading

int curReaders = 0;

//reader thread

void\* readT(void\* arg){

//getting our thread number

int readID = \*((int\*)arg);

//the thread needs to run 2000000 times

for(int i = 0; i < MAX\_READS; i++){

//waiting section

//decreasing our semaphore to get permission to enter CS

sem\_wait(&readSem);

//if this is the first reader, we need to get the other semaphore to stop the writer from writing

//while readers are reading

curReaders++;

if(curReaders == 1){

sem\_wait(&writeSem);

}

//we need to increment our reader semaphore so other readers can join

sem\_post(&readSem);

//printf("read %i entered\n",readID);

//critical section

printf("I'm reader%i, counter = %i\n",readID,sharedValue->value);

//exiting section

//printf("read %i left\n",readID);

//decreasing the semaphore so that we can get control to decrement curReaders

sem\_wait(&readSem);

curReaders--;

//if this reader was the last reader then we want to increment the writeSem for the writer

if(curReaders == 0){

sem\_post(&writeSem);

}

//incrementing readSem so another reader can enter

sem\_post(&readSem);

}

//Remainder section

}

//writer thread

void\* writeT(void\* arg){

int i;

//we need to update sharedValue MAX\_INCR times

for(i = 0; i < MAX\_INCR; i++){

//waiting to decrement writeSem and write

sem\_wait(&writeSem);

//printf("writer entered\n");

//critical section

//we are incremented shared value here 25000 times which was already defined

sharedValue->value += 1;

//exiting

printf("Writer Done!\n");

sem\_post(&writeSem);

}

//Remainder section

}

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

//verifying we have our command line argument

if(argc != 2){

printf("Error: One command line argument expected\n");

exit(1);

}

//we have an argument to assign

int numReaders = atoi(argv[1]);

if(numReaders < 1 || numReaders > MAX\_READERS){

printf("Error: Command line argument needs to be greater than 0 and less than 13\n");

exit(1);

}

//initializing our semaphores

sem\_init(&writeSem, 0, 1);

sem\_init(&readSem, 0, 1);

//allocating memory and setting our global variables value to 0

sharedValue = (struct sharedData \*) malloc(sizeof(struct sharedData));

sharedValue->value = 0;

//creating our number of reader threads

pthread\_t tid[numReaders];

int rc[numReaders];

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

//updating the reader threads id numbers

rc[i] = i + 1;

pthread\_create(&tid[i], NULL, readT, &rc[i]);

}

//creating our writer thread

pthread\_t writerThread;

pthread\_create(&writerThread, NULL, writeT, NULL);

//joining our threads

for(int i = 0; i < numReaders; i++)

pthread\_join(tid[i], NULL);

pthread\_join(writerThread, NULL);

//cleaning up our memory

sem\_destroy(&writeSem);

sem\_destroy(&readSem);

free(sharedValue);

return 0;

}

Task 2:

This code is fair because I use two semaphores to provide mutual exclusion to readers and writers. Readers can only concurrently enter their critical sections provided there are no writers. With the writer semaphore there can only be one writer writing and no readers can access their critical sections. This code allows multiple readers to access the shared data concurrently, while also not starving writers because they are given exclusive access when they need to write. This approach ensures that neither dominate the other or are given exclusive access to the shared resource.