Prashant Yadav

CS 4348.002 – Program 2

```
/*******************************************************************
 * File: Program2.c
 *Author: Prashant Yadav
 *Procedure:
 *read - Each reader thread starts execution from this routine.
 *write - Each writer thread starts execution from this routine.
 *writerarray - Write routine calls this routine to write data to shared array.
 *readarray - Read routine calls this routine to read data from shared array.
 *main - Main routine from where program starts executing.In this routine we initialize 10
reader
 * threads and 1 writer thread.
 *******************************************************************/
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>

int  MAX =  1000000;   //Max size of Shared array.
int sharedarr[1000000], start=0, end=0, readerdata[10];  //Initialize shared array.
int readercount, writercount, counter=0;  //These variables are used to mantain read and write
count.
sem_t x, y,z, wsem, rsem;  //Declaration of useful semaphores.
void readarray();  //Declaration of readarray routine
void writerarray();  //Declaration of writerarray routine

/*******************************************************************
 *void* read(void *id)
 *Author: Prashant Yadav
 *Date: 9/29/2019
 *Description: It check whether there is a writer thread currently accessing shared array.If yes
 *                  it waits for writer to finish, otherwise holds a lock on shared array to read
 *                 data from it. it allows multiple readers to read from the shared array.
 *Parameters:
 *id I/P int This is used as a thread id to identify each reader thread.
 *This routine does not return anything.
 *******************************************************************/
void*  read(void *id)
{
 while(1){
  sem_wait(&z);        //Reduce value of semaphore z by 1.
  sem_wait(&rsem);     //Reduce value of semaphore rsem by 1.
  sem_wait(&x);        //Reduce vlaue of semaphore x by 1.
  readercount++;       //Increase reader count by 1 at entry to critical section.
  if(readercount==1){  //Signal writer thread when first reader enter to critical section.
   sem_wait(&wsem);
  }
  sem_post(&x);        //Singal x i.e increase value of sempahore x by 1.
  sem_post(&rsem);     //Signal rsem. It sllows multiple readers to enter critical section.
  sem_post(&z);        //Singal semaphore z. Increase value by 1.
  readarray(id);       //Call to readarray routine.
  if(start==MAX){      //If start reaches to end of shared array exit from while loop.
   printf("\n Reader finshed..");
   break;
  }
  sem_wait(&x);        //Hold lock on semaphore x to exit from the critical section.
  readercount--;       //Decrease readercount at exit of critical section.
  if(readercount==0){  //Condition to check if there are no readers signal to wsem semaphore.
   sem_post(&wsem);
  }
  sem_post(&x);        //Unlock sempahore x and exit critical section..
 }
}

/*****************************************************************
 *void* write()
 *Author: Prashant Yadav
 *Date: 9/29/2019
 *Description: It checks if there are any reader thread currently reading from shared array. If
 *                    yes, writer thread wait for reader thread to finish, otherwise writer thread holds
 *                   lock on shared array and starts writing to it.
 *Parameters:
 *There are no Input args to this routine.
 *This routine does not return anything.
 *****************************************************************/
void* write()
{
 while(1){
  sem_wait(&y);        //Reduce value of semaphore by y.
  writercount++;       //Increase value of writercount by 1 at entry to critical section.
  if(writercount==1){  //If writercount=1 signal to rsem, i.e. wait for readers to exit.
   sem_wait(&rsem);
  }
  sem_post(&y);        //Increase value of semaphore by 1.
  sem_wait(&wsem);     //Reduce value of wsem by 1 at entry to critical section.
  writerarray();       //Call to writerarray routine.
  if(end==MAX){        //If writer finshed writing to shared array, gracefully exit from while
loop.
   printf("\n Writer finished...");
   sem_post(&wsem);    //Before exiting signal wsem and rsem. To let readers to read.
   sem_post(&rsem);
   break;
  }
  sem_post(&wsem);     //Signal wsem at exit from critical section.
  sem_wait(&y);        //Signal y at exit from critical section.
  writercount--;       //Reduce writer count at exit.
  if(writercount==0){  //If there are no writers signal to reader threads.
```

```
   sem_post(&rsem);
  }
  sem_post(&y);        //Unlock semaphore y.
 }
}


/*****************************************************************
 *void readarray(void *id)
 *Author: Prashant Yadav
 *Date: 9/29/2019
 *Description: Read routine calls this routine to read from shared array.This routine also update
 *                    readerdata array for each thread.
 *Parameters:
 *id I/P int This is used as a thread id to identify each reader thread.
 *This routine does not return anything.
 *****************************************************************/
void readarray(void *id){
 //Increase counter to corresponding thread by 1 in readerdata array.
 readerdata[(int)id-1] = readerdata[(int)id-1]+1;
 printf("\n Reader-%d at Value %d",(int)id, sharedarr[start++]); //Display data and thread id.
}

/*****************************************************************
 *void writerarray()
 *Author: Prashant Yadav
 *Date: 9/29/2019
 *Description: Write routine calls this routine to write data to shared array.
 *Parameters:
 *There are no input args to this routine.
 *This routine does not return anything.
 *****************************************************************/
void writerarray(){
 sharedarr[end]=counter++;  //Write to shared array
 printf("\n Writing value %d", sharedarr[end]); //Display data written by writer thread.
 end++;  //Increase end by 1 so that writer writes to next index in next iteration.
}

/*****************************************************************
 *void main()
 *Author: Prashant Yadav
 *Date: 9/29/2019
 *Description: Program execution starts from this routine, 10 reader threads and 1 writer thread
 *                    are spawed here and later joined. It also displays stats of read count for every
 *                    reader thread.
 *Parameters:
 *It does not take any input args.
 *This routine does not return anything.
 *****************************************************************/
void main()
{
 readercount=0;    //Initialize readercount to zero
 writercount=0;    //Initialize writercount to zero
 sem_init(&x,0,1); //Initialize all semaphores with intial value 1.
 sem_init(&y,0,1); //Here second argument 0 shows semaphore will be shared between threads
 sem_init(&z,0,1); //of a process.
 sem_init(&rsem,0,1);
 sem_init(&wsem,0,1);
 pthread_t readers[10], writer;  //10 readers and 1 writer pthread.
 int j;
 for(j=0;j<10;j++)
 { readerdata[j]=0;    //Initialize readerdata array to store read count for each thread.
 }
 printf("\n Execution Started....");
 pthread_create(&writer, NULL, write, NULL);  //Initalize writer pthread.
 int i;
 for(i=0;i<10;i++){
  pthread_create(&readers[i], NULL, read, (void *)i+1); //Initialize 10 reader pthreads.
 }
 pthread_join(writer,NULL);  //Join writer pthread.
 for(i=0;i<10;i++){
  pthread_join(readers[i], NULL);  //join each reader pthread.
 }
 for(j=0;j<10;j++){
  printf("\nReader-%d reads %d times",j+1, readerdata[j]);   //Print read count of each reader
pthread.
 }
 printf("\n Execution ends here...");
 sem_destroy(&x);  //Destroy all used semaphores.
 sem_destroy(&y);
 sem_destroy(&z);
 sem_destroy(&rsem);
 sem_destroy(&wsem);
}
```