Artem Tolstov

CSC 415

4/15/2015

Homework 5

Win32.1.race.c and **posix.1.race.c** create the specified amount of threads that all attempt to increment the same asciiCount values simultaneously. This creates a race condition where the final values are over written without taking into consideration the counts of the threads that are simultaneously working on the same counts, producing inaccurate counts.

Win32.1.sync.c and posix.2.sync.c is the exact same program and the .race version with the exception that win32 uses CreateSemaphore that allows 1 process at a time to modify the asciiCount, thereby eliminating the race condition created in the previous program. The Posix version uses a mutex to the same effect, a semaphore with only 1 max process functions similar to a mutex.

Win32.2.c and **posix.2.c** receives arguments from the command line and uses the pow function to set the number of producers, consumers and items with their binary log. They create the win32 and posix versions of threads (the same as hw4), the number of producers is determined by the argument specified, and each producer creates the specified number of items. The consumers all calculate the total number of items they should consume by this equation: (producers*items)/consumers.

As the producers are creating items they are being stored in a 16 slot buffer, which is being printed from by the consumers (considering a printed value consumed). The semaphore “empty” monitors the number of empty slots the buffer currently has, and does not allow the producers to continue until more empty slots are created. While the semaphore “full” monitors the number of slots in the buffer with a produced number in them, and does not allow the consumer to go on until more slots are filled in the buffer. The producer threads waits on permission from the empty semaphore and then gives its signal to the full semaphore, while the consumer thread does the opposite: waits on permission from the full semaphore and signals the empty semaphore.

Win32.1.race.c code:

```
//Author: Artem Tolstov
//Purpose: CSC415 HW5




#include <stdio.h>
#include <stdlib.h>
#include <windows.h> //for threads
#define BUFFER_SIZE 65536
#define NUM_THREADS 4
void printOccurences(int array[128]);
int charCounts[128];
char buffer[BUFFER_SIZE];
struct threadParams
{
       int threadIndex;
       int stringLengthInBuffer;

};
DWORD WINAPI asciiCounter (LPVOID param);

int main(int argc, char *argv[])
{
    //Declarations
    FILE *filePointer;
    int i,j,x;
       int partition;
       struct threadParams params[NUM_THREADS];
       DWORD ThreadId;
```

```
   HANDLE ThreadHandle;



//Makes sure there are 2 arguments in the command line.
if (argc != 2 )
   {
          printf( "proper usage: %s filename.extension", argv[0] );
   }
else
   {
          //Reports if file does not exist.
          filePointer = fopen( argv[1], "r" );


          if (filePointer == 0)
          {
                printf ("Could not open file\n");
          }
          else
          {
              i = 0;
                while (((x = fgetc(filePointer)) != EOF) && (i < BUFFER_SIZE))
                {
                       //Put next character into i position of buffer, increment
i.
                    buffer[i] = x; i++;
                       //printf( "%c", x);
                }
                fclose( filePointer );


                //For debugging
```

```
                    //printf( "\nbuffer contains: %s\n", buffer );
                    printf("length of string in file is: %d\n", strlen(buffer));



                    //Initialize array charCounts to 0
                    for(j = 0; j < 128; j++)
                    {
                           charCounts[j] = 0;
                    }



                    //Sends params with thread indexi to threads
                    for(i = 0; i < NUM_THREADS; i++)
                    {
                           params[i].threadIndex = i;
                           params[i].stringLengthInBuffer = strlen(buffer);
                           //ThreadHandle = CreateThread(NULL, 0, asciiCounter,
&params[i], 0, &ThreadId);

                    }



                    //Im trying to create a race condition, so I made the thread get
created
                    //in its own for loop.
                    for (i = 0; i < NUM_THREADS; i++)
                    {
                           ThreadHandle = CreateThread(NULL, 0, asciiCounter,
&params[i], 0, &ThreadId);
                    }
```

```
                    //waits for threads to finish
                    if (ThreadHandle != NULL)
                    {
                           WaitForSingleObject(ThreadHandle, INFINITE);
                    }

                    CloseHandle(ThreadHandle);
                    printOccurences(charCounts);
             }
       }



}

//Prints letter if printable, Hex value if unprintable character.
//Combines the values of the different threads into [0][] section of the array.
void printOccurences(int array[128])
{
       int i,j, holdIt;
       /*
       // Adds up the different instances of the array in array[0][i];

    for(i = 0; i < 128; i++)
    {
        //holdIt = array[0][i];
             //array[0][i] = 0;
        for(j = 1; j < NUM_THREADS; j++)
        {
           array[0][i] +=  array[j][i];
```

```
        }
        //array[0][i] = array[0][i] + holdIt;
    }
       */

    for(i = 0; i < 128; i++)
    {
        if((i > 32) && (i != 127))
        {
            printf("%d occurrences of %c\n", array[i], (char)i);
        }
        else
        {
            printf("%d occurrences of 0x%x\n", array[i], i);
        }
    }
}

DWORD WINAPI asciiCounter (LPVOID param)
{
       struct threadParams thisThreadsParams = *(struct threadParams*)param;
       int i;
       //printf("asciiCounter has threadIndex = %d\n", thisThreadsParams.threadIndex);
       for(i = thisThreadsParams.threadIndex *
(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i <
(thisThreadsParams.threadIndex+1)*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS)
; i++)
       {
              //printf("%c is value of %d\n", buffer[i], (int)buffer[i]);

              charCounts[(int)buffer[i]]++;
```

```
		//Keeps the null characters in the buffer from writing

	}

	//final check to count remaining characters due to truncation in division

	if(thisThreadsParams.threadIndex + 1 == NUM_THREADS)

	{

		if(NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS) !=
thisThreadsParams.stringLengthInBuffer)

		{

			for(i =
NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i <
thisThreadsParams.stringLengthInBuffer; i++)

			{

				charCounts[(int)buffer[i]]++;

			}

		}

	}



	return 0;

}
```

Win32.1.sync.c code:

```
/*********************************************************************\
| Program : win32.1.sync.c                                            |
| Problem : Write to a buffer using multiple threads and use a        |
|           semaphore to protect the counts in the array you are      |
|           writing to so that the counts remain accurate.           |
| Purpose : Practice implementing semaphores.                         |
| Author  : Artem Tolstov                                             |
| Date    : 4/15/2015                                                 |
\*********************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <windows.h> //for threads
#define BUFFER_SIZE 65536
#define NUM_THREADS 4
HANDLE ghSemaphore;
void printOccurences(int array[128]);
int charCounts[128];
char buffer[BUFFER_SIZE];
struct threadParams
{
       int threadIndex;
       int stringLengthInBuffer;

};
DWORD WINAPI asciiCounter (LPVOID param);

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

```
    //Declarations
    FILE *filePointer;
    int i,j,x;
       int partition;
       struct threadParams params[NUM_THREADS];
       DWORD ThreadId;
       HANDLE ThreadHandle;



    //Makes sure there are 2 arguments in the command line.
    if (argc != 2 )
       {
              printf( "proper usage: %s filename.extension", argv[0] );
       }
    else
       {
              //Reports if file does not exist.
              filePointer = fopen( argv[1], "r" );


              if (filePointer == 0)
              {
                     printf ("Could not open file\n");
              }
              else
              {
                  i = 0;
                     while (((x = fgetc(filePointer)) != EOF) && (i < BUFFER_SIZE))
                     {
                            //Put next character into i position of buffer, increment
i.
```

```
            buffer[i] = x; i++;
               //printf( "%c", x);
            }
            fclose( filePointer );

            //For debugging
            //printf( "\nbuffer contains: %s\n", buffer );
            printf("length of string in file is: %d\n", strlen(buffer));


            //Initialize array charCounts to 0
            for(j = 0; j < 128; j++)
            {
                  charCounts[j] = 0;
            }


            //Sends params with thread indexi to threads
            for(i = 0; i < NUM_THREADS; i++)
            {
                  params[i].threadIndex = i;
                  params[i].stringLengthInBuffer = strlen(buffer);
                  //ThreadHandle = CreateThread(NULL, 0, asciiCounter,
&params[i], 0, &ThreadId);

            }

            ghSemaphore = CreateSemaphore(
                                          NULL,           // default security
attributes
                                          1,              // initial count
```

```
                                                1,              // maximum count
                                                NULL);          // unnamed semaphore


                        if (ghSemaphore == NULL)
                        {
                                printf("CreateSemaphore error: %d\n", GetLastError());
                                return 1;
                        }


                        //Im trying to create a race condition, so I made the thread get
created
                        //in its own for loop.
                        for (i = 0; i < NUM_THREADS; i++)
                        {
                                ThreadHandle = CreateThread(NULL, 0, asciiCounter,
&params[i], 0, &ThreadId);
                        }


                        //waits for threads to finish
                        if (ThreadHandle != NULL)
                        {
                                WaitForSingleObject(ThreadHandle, INFINITE);
                        }


                        CloseHandle(ThreadHandle);
                        CloseHandle(ghSemaphore);
                        printOccurences(charCounts);
                }
        }
```

```
}

//Prints letter if printable, Hex value if unprintable character.
//Combines the values of the different threads into [0][] section of the array.
void printOccurences(int array[128])
{
       int i,j, holdIt;
       /*
       // Adds up the different instances of the array in array[0][i];

    for(i = 0; i < 128; i++)
    {
        //holdIt = array[0][i];
             //array[0][i] = 0;
        for(j = 1; j < NUM_THREADS; j++)
        {
           array[0][i] +=  array[j][i];
        }
        //array[0][i] = array[0][i] + holdIt;
    }
       */

    for(i = 0; i < 128; i++)
    {
        if((i > 32) && (i != 127))
        {
            printf("%d occurrences of %c\n", array[i], (char)i);
        }
        else
```

```
        {
            printf("%d occurrences of 0x%x\n", array[i], i);
        }
    }
}

DWORD WINAPI asciiCounter (LPVOID param)
{
       struct threadParams thisThreadsParams = *(struct threadParams*)param;
       int i;
       //printf("asciiCounter has threadIndex = %d\n", thisThreadsParams.threadIndex);
       for(i = thisThreadsParams.threadIndex *
(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i <
(thisThreadsParams.threadIndex+1)*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS)
; i++)
       {
              //printf("%c is value of %d\n", buffer[i], (int)buffer[i]);

              WaitForSingleObject(ghSemaphore, INFINITE);
                        // (handle to semaphore, zero-second time-out interval)
              charCounts[(int)buffer[i]]++;
              //Keeps the null characters in the buffer from writing

              ReleaseSemaphore(ghSemaphore, 1, NULL);
                                  // (handle to semaphore, increase count by one, not
interested in previous count)
       }
       //final check to count remaining characters due to truncation in division
       if(thisThreadsParams.threadIndex + 1 == NUM_THREADS)
       {
              if(NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS) !=
thisThreadsParams.stringLengthInBuffer)
```

```
		{
			for(i = NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i < thisThreadsParams.stringLengthInBuffer; i++)
			{
				WaitForSingleObject(ghSemaphore, INFINITE);
				charCounts[(int)buffer[i]]++;
				ReleaseSemaphore(ghSemaphore, 1, NULL);
			}
		}
	}

	return 0;
}
```

Win32.2.c code:

```
/*********************************************************************\
| Program : win32.2.c                                                 |
| Problem : Create a program that reads and writes to a bounded buffer|
|           without overwriting any of the array by using semaphores  |
|           to control access to the buffer.                          |
| Purpose : Practice the consumer-producer problem.                   |
| Author  : Artem Tolstov                                             |
| Date    : 4/15/2015                                                 |
\*********************************************************************/


#include <stdio.h>
#include <windows.h>     //semaphores and threads
#include <math.h>        //pow


#define BUFFER_SIZE 16


DWORD WINAPI consumerThread (LPVOID param);
DWORD WINAPI producerThread (LPVOID param);


struct threadParams
{
       int thread_number;
       int producers, consumers, items;


};


//Global Variables
int buffer[BUFFER_SIZE];
int bufferPTR = 0;
```

```
HANDLE full; //tells producer if there is no room to produce

HANDLE empty; //should tell consumer if there is anything to consume

HANDLE binarySemaphore; //to allow access in and out of the buffer

//int producers, consumers, items;

int numProduced = 0;

int numConsumed = 0;


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

{

	//Declarations

	int producers = pow(2, atoi(argv[1]));

	int consumers = pow(2, atoi(argv[2]));

	int items = pow(2, atoi(argv[3]));

	HANDLE *prodHandle = (HANDLE*) malloc(sizeof(HANDLE)*producers);

	HANDLE *conHandle = (HANDLE*) malloc(sizeof(HANDLE)*consumers);

	int i = 0;

	struct threadParams *params;

	params = (struct threadParams*) malloc(sizeof(struct threadParams) *
producers);


	//1. Parse all of the command-line parameters

	//Makes sure there are exactly 3 commandline arguments

	if (argc != 4)

	{

		printf("propers usage: %s int int int", argv[0] );

	}

	else

	{

		//2. Print them in a message

		//Prints to screen our command line args (string form)
```

```
		//printf("\ncommand line arguments are: %s %s %s\n\n", argv[1], argv[2],
argv[3]);


		//Prints to screen to contents of producers, consumers, items
		printf("producers: 2^%d = %d\n", atoi(argv[1]), producers);
		printf("consumers: 2^%d = %d\n", atoi(argv[2]), consumers);
		printf("    items: 2^%d = %d\n", atoi(argv[3]), items);


		//3. Initialize the synchronization objects
		full = CreateSemaphore(NULL, 0, BUFFER_SIZE, NULL); //tells consumer that
there are 0 full slots, consumer can't consume
		empty = CreateSemaphore(NULL, BUFFER_SIZE, BUFFER_SIZE, NULL); //starts
with 16 since there are 16 empty slots that producers can fill
		binarySemaphore = CreateSemaphore(NULL, 1, 1, NULL); //allows access 1 at
a time to global buffer


		//4. Spawn all of the threads/
		//printf("producers = %d\nconsumers = %d\n", producers, consumers);
		for(i = 0; i < producers; i++)
		{
			params[i].thread_number = i;
			params[i].producers = producers;
			params[i].consumers = consumers;
			params[i].items = items;
			prodHandle[i] = CreateThread(NULL, 0, producerThread, &params[i],
0, NULL);
			if(prodHandle[i] == NULL)
			{
				fprintf(stderr, "Error: CreateThread, producer thread
%i.\n",prodHandle[i]);
				ExitProcess(i);
			}
		}
```

```
		for(i = 0; i < consumers; i++)
		{
			conHandle[i] = CreateThread(NULL, 0, consumerThread, &params[0], 
0, NULL);
			if(conHandle[i] == NULL)
			{
				fprintf(stderr, "Error: CreateThread, producer thread 
%i.\n",conHandle[i]);
				ExitProcess(i);
			}
		}

		//5. Wait for all the threads to complete,
		WaitForMultipleObjects(producers, prodHandle, TRUE, INFINITE);
		WaitForMultipleObjects(consumers, conHandle, TRUE, INFINITE);

		//6. Print a final message
		printf("\nAll producer and consumer threads have finished.\n");
		printf("Number of producers: %d\nNumber of consumers: %d\nNumber of items 
per producers: %d\n\n", producers, consumers, items);
		printf("Number Produced: %d\nNumber Consumed: %d", numProduced, 
numConsumed);

	}
	return 0;
}

DWORD WINAPI consumerThread (LPVOID param)
{
	struct threadParams thisThreadsParams = *(struct threadParams*)param;
	int producers = thisThreadsParams.producers;
```

```
	int consumers = thisThreadsParams.consumers;
	int items = thisThreadsParams.items;

	int ItemsToConsume = (producers*items)/consumers;
	int i;
	for(i = 0; i < ItemsToConsume; i++)
	{
		WaitForSingleObject(full, INFINITE);
		WaitForSingleObject(binarySemaphore, INFINITE);
		/*critical section */
		bufferPTR--;
		printf("%d consumed from buffer[%d]\n", buffer[bufferPTR], bufferPTR);
		numConsumed++;
		/*end critical section */
		ReleaseSemaphore(empty, 1, NULL);
		ReleaseSemaphore(binarySemaphore, 1, NULL);
	}
	return 0;
}

DWORD WINAPI producerThread (LPVOID param)
{
	struct threadParams thisThreadsParams = *(struct threadParams*)param;
	int producers = thisThreadsParams.producers;
	int consumers = thisThreadsParams.consumers;
	int items = thisThreadsParams.items;
	int threadNum = thisThreadsParams.thread_number;

	int counter = 0;
	int item;
```

```
	int totalItems = items;
	int i = 0;


	for(i = 0; i < totalItems; i++)
	{
		WaitForSingleObject(empty, INFINITE); //producers wait for empty slots to
fill
		WaitForSingleObject(binarySemaphore, INFINITE); //wait for binary
semaphore
		/*critical section */
		item = threadNum * numProduced++ + counter++;
		buffer[bufferPTR++] = item;
		printf("Item: %d created\n", item);
		/*critical section ends */
		ReleaseSemaphore(full, 1, NULL);
		ReleaseSemaphore(binarySemaphore, 1, NULL);
	}
	return 0;
}
```

Posix.1.race.c code:

```
/*********************************************************************\
| Program : posix.1.race.c                                            |
| Problem : Takes name of file from command line and counts the       |
|           number of ascii characters using predefined number of     |
|           threads then prints to the screen occurences of said      |
|           characters.                                               |
| Purpose : This version is designed to show an example of race       |
|           conditions.                                               |
| Author  : Artem Tolstov                                             |
| Date    : 4/15/2015                                                 |
\*********************************************************************/


//Author: Artem Tolstov
//Purpose: CSC415 HW4
//Function: Takes name of file from command line and counts the number of ascii
characters using
//                  predefined number of threads then prints to the screen occurences
of said characters.




#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include <windows.h> //for threads
#include <pthread.h> //for posix threads
#define BUFFER_SIZE 65536
#define NUM_THREADS 4
void printOccurences(int array[128]);
int charCounts[128];
char buffer[BUFFER_SIZE];
```

```
struct threadParams
{
	int threadIndex;
	int stringLengthInBuffer;

};
//DWORD WINAPI asciiCounter (LPVOID param);
void *asciiCounter (void *param);

int main(int argc, char *argv[])
{
    //Declarations
    FILE *filePointer;
    int i,j,x;
       int partition;
       struct threadParams params[NUM_THREADS];
       pthread_t ThreadId;
       //HANDLE ThreadHandle;
       pthread_attr_t attr;



    //Makes sure there are 2 arguments in the command line.
    if (argc != 2 )
       {
              printf( "proper usage: %s filename.extension", argv[0] );
       }
    else
       {
              //Reports if file does not exist.
              filePointer = fopen( argv[1], "r" );
```

```
if (filePointer == 0)
{
	printf ("Could not open file\n");
}
else
{
    i = 0;
	while (((x = fgetc(filePointer)) != EOF) && (i < BUFFER_SIZE))
	{
		//Put next character into i position of buffer, increment
i.
	    buffer[i] = x; i++;
		//printf( "%c", x);
	}
	fclose( filePointer );

	//For debugging
	//printf( "\nbuffer contains: %s\n", buffer );
	printf("length of string in file is: %d\n", (int)strlen(buffer));


	//Initialize array charCounts to 0
	for(j = 0; j < 128; j++)
	{
		charCounts[j] = 0;
	}


	//Sends params with thread indexi to threads
```

```
                    pthread_attr_init(&attr);
                    for(i = 0; i < NUM_THREADS; i++)
                    {
                           params[i].threadIndex = i;
                           params[i].stringLengthInBuffer = strlen(buffer);
                           pthread_create(&ThreadId, &attr, asciiCounter, &params[i]);
                           //ThreadHandle = CreateThread(NULL, 0, asciiCounter,
&params[i], 0, &ThreadId);

                    }


                    //Im trying to create a race condition, so I made the thread get
created
                    //in its own for loop.
                    for (i = 0; i < NUM_THREADS; i++)
                    {
                           pthread_create(&ThreadId, &attr, asciiCounter, &params[i]);
                    }
                    pthread_join(ThreadId, NULL);
                    //waits for threads to finish
                    //if (ThreadHandle != NULL)
                    //{
                    //     WaitForSingleObject(ThreadHandle, INFINITE);
                    //}

                    //CloseHandle(ThreadHandle);
                    printOccurences(charCounts);
                    pthread_exit(0);
             }
      }
```

```
}

//Prints letter if printable, Hex value if unprintable character.
//Combines the values of the different threads into [0][] section of the array.
void printOccurences(int array[128])
{
       int i,j, holdIt;
       /*
       // Adds up the different instances of the array in array[0][i];

    for(i = 0; i < 128; i++)
    {
        //holdIt = array[0][i];
             //array[0][i] = 0;
        for(j = 1; j < NUM_THREADS; j++)
        {
           array[0][i] +=  array[j][i];
        }
        //array[0][i] = array[0][i] + holdIt;
    }
       */

    for(i = 0; i < 128; i++)
    {
        if((i > 32) && (i != 127))
        {
            printf("%d occurrences of %c\n", array[i], (char)i);
```

```
        }
        else
        {
            printf("%d occurrences of 0x%x\n", array[i], i);
        }
    }
}


void *asciiCounter (void *param)
{
       struct threadParams thisThreadsParams = *(struct threadParams*)param;
       int i;
       //printf("asciiCounter has threadIndex = %d\n", thisThreadsParams.threadIndex);
       for(i = thisThreadsParams.threadIndex *
(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i <
(thisThreadsParams.threadIndex+1)*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS)
; i++)
       {
              //printf("%c is value of %d\n", buffer[i], (int)buffer[i]);


              charCounts[(int)buffer[i]]++;
              //Keeps the null characters in the buffer from writing
       }
       //final check to count remaining characters due to truncation in division
       if(thisThreadsParams.threadIndex + 1 == NUM_THREADS)
       {
              if(NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS) !=
thisThreadsParams.stringLengthInBuffer)
              {
                     for(i =
NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i <
thisThreadsParams.stringLengthInBuffer; i++)
                     {
```

```
                        charCounts[(int)buffer[i]]++;
                    }
                }
            }

            return 0;
        }
```

Posix.1.sync.c code:

```
/*********************************************************************\
| Program : posix.1.sync.c                                            |
| Problem : Takes name of file from command line and count the number |
|           of ascii characters using predefined number of threads    |
|           then prints to the screen occurences of said character,   |
|           this version uses semaphores to protect access to the     |
|           buffer when it is being printed to.                       |
| Purpose : Use semaphores on a posix system.                         |
| Author  : Artem Tolstov                                             |
| Date    : 4/15/2015                                                 |
\*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include <windows.h> //for threads on win32
#include <pthread.h> //for posix threads
#define BUFFER_SIZE 65536
#define NUM_THREADS 4
//define handle to ghsemaphore here
pthread_mutex_t myMutex;


void printOccurences(int array[128]);
int charCounts[128];
char buffer[BUFFER_SIZE];
struct threadParams
{
       int threadIndex;
       int stringLengthInBuffer;
```

```
};
//DWORD WINAPI asciiCounter (LPVOID param);
void *asciiCounter (void *param);

int main(int argc, char *argv[])
{
    //Declarations
    FILE *filePointer;
    int i,j,x;
       int partition;
       struct threadParams params[NUM_THREADS];
       pthread_t ThreadId;
       //HANDLE ThreadHandle;
       pthread_attr_t attr;



    //Makes sure there are 2 arguments in the command line.
    if (argc != 2 )
       {
              printf( "proper usage: %s filename.extension", argv[0] );
       }
    else
       {
              //Reports if file does not exist.
              filePointer = fopen( argv[1], "r" );

              if (filePointer == 0)
              {
                     printf ("Could not open file\n");
              }
```

```
        else
        {
            i = 0;
               while (((x = fgetc(filePointer)) != EOF) && (i < BUFFER_SIZE))
               {
                      //Put next character into i position of buffer, increment
i.
                   buffer[i] = x; i++;
                      //printf( "%c", x);
               }
               fclose( filePointer );

               //For debugging
               //printf( "\nbuffer contains: %s\n", buffer );
               printf("length of string in file is: %d\n", (int)strlen(buffer));


               //Initialize array charCounts to 0
               for(j = 0; j < 128; j++)
               {
                      charCounts[j] = 0;
               }


               //Sends params with thread indexi to threads
               pthread_attr_init(&attr);
               for(i = 0; i < NUM_THREADS; i++)
               {
                      params[i].threadIndex = i;
                      params[i].stringLengthInBuffer = strlen(buffer);
```

```
                        pthread_create(&ThreadId, &attr, asciiCounter, &params[i]);
                        //ThreadHandle = CreateThread(NULL, 0, asciiCounter,
&params[i], 0, &ThreadId);

                    //create mutex here
                    pthread_mutex_init(&myMutex, NULL);
                    }

                    //Im trying to create a race condition, so I made the thread get
created
                    //in its own for loop.
                    for (i = 0; i < NUM_THREADS; i++)
                    {
                        pthread_create(&ThreadId, &attr, asciiCounter, &params[i]);
                    }
                    pthread_join(ThreadId, NULL);
                    //waits for threads to finish
                    //if (ThreadHandle != NULL)
                    //{
                    //    WaitForSingleObject(ThreadHandle, INFINITE);
                    //}

                    //CloseHandle(ThreadHandle);
                    printOccurences(charCounts);
                    pthread_exit(0);
                    //close mutex
                    pthread_mutex_destroy(&myMutex);
                }
            }
```

```
}

//Prints letter if printable, Hex value if unprintable character.
//Combines the values of the different threads into [0][] section of the array.
void printOccurences(int array[128])
{
       int i,j, holdIt;
       /*
       // Adds up the different instances of the array in array[0][i];

    for(i = 0; i < 128; i++)
    {
        //holdIt = array[0][i];
             //array[0][i] = 0;
        for(j = 1; j < NUM_THREADS; j++)
        {
           array[0][i] +=  array[j][i];
        }
        //array[0][i] = array[0][i] + holdIt;
    }
       */

    for(i = 0; i < 128; i++)
    {
        if((i > 32) && (i != 127))
        {
            printf("%d occurrences of %c\n", array[i], (char)i);
        }
```

```
        else
        {
            printf("%d occurrences of 0x%x\n", array[i], i);
        }
    }
}

void *asciiCounter (void *param)
{
       struct threadParams thisThreadsParams = *(struct threadParams*)param;
       int i;
       //printf("asciiCounter has threadIndex = %d\n", thisThreadsParams.threadIndex);
       for(i = thisThreadsParams.threadIndex *
(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i <
(thisThreadsParams.threadIndex+1)*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS)
; i++)
       {
              //printf("%c is value of %d\n", buffer[i], (int)buffer[i]);
              pthread_mutex_lock(&myMutex);
              charCounts[(int)buffer[i]]++;
              pthread_mutex_unlock(&myMutex);
              //Keeps the null characters in the buffer from writing
       }
       //final check to count remaining characters due to truncation in division
       if(thisThreadsParams.threadIndex + 1 == NUM_THREADS)
       {
              if(NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS) !=
thisThreadsParams.stringLengthInBuffer)
              {
                     for(i =
NUM_THREADS*(thisThreadsParams.stringLengthInBuffer/NUM_THREADS); i <
thisThreadsParams.stringLengthInBuffer; i++)
                     {
```

```
                        pthread_mutex_lock(&myMutex);
                        charCounts[(int)buffer[i]]++;
                        pthread_mutex_unlock(&myMutex);
                    }
                }
            }

        return 0;
}
```

Posix.2.c code:

```
/*********************************************************************\
| Program : win32.2.c                                                 |
| Problem : Create a program that reads and writes to a bounded buffer|
|           without overwriting any of the array by using semaphores  |
|           to control access to the buffer.                          |
| Purpose : Practice the consumer-producer problem.                   |
| Author  : Artem Tolstov                                             |
| Date    : 4/15/2015                                                 |
\*********************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h> //for posix threads
#include <semaphore.h> //for semaphores
#include <math.h>        //pow

#define BUFFER_SIZE 16

void *consumerThread (void *param);
void *producerThread (void *param);
//DWORD WINAPI consumerThread (LPVOID param);
//DWORD WINAPI producerThread (LPVOID param);

pthread_mutex_t myMutex;
struct threadParams
{
       int thread_number;
       //int producers, consumers, items;
```

```
};

//Global Variables
int buffer[BUFFER_SIZE];
int bufferPTR = 0;
//HANDLE full; //tells producer if there is no room to produce
//HANDLE empty; //should tell consumer if there is anything to consume
pthread_mutex_t myMutex;
sem_t empty;
sem_t full;
//HANDLE binarySemaphore; //to allow access in and out of the buffer
//int producers, consumers, items;
int numProduced = 0;
int numConsumed = 0;
int globalProducers;
int globalConsumers;
int globalItems;

int main(int argc, char *argv[])
{
	//Declarations
	int producers = pow(2, atoi(argv[1]));
	int consumers = pow(2, atoi(argv[2]));
	int items = pow(2, atoi(argv[3]));
	pthread_t *ThreadIdProducers;
	pthread_t *ThreadIdConsumers;
	pthread_attr_t attr;
	//HANDLE *prodHandle = (HANDLE*) malloc(sizeof(HANDLE)*producers);
	//HANDLE *conHandle = (HANDLE*) malloc(sizeof(HANDLE)*consumers);
```

```
	int i = 0;

	globalProducers = producers;

	globalConsumers = consumers;

	globalItems = items;

	struct threadParams *params;

	ThreadIdProducers = (pthread_t*) malloc(sizeof(pthread_t) * producers);

	ThreadIdConsumers = (pthread_t*) malloc(sizeof(pthread_t) * consumers);

	params = (struct threadParams*) malloc(sizeof(struct threadParams) *
producers);


	//1. Parse all of the command-line parameters

	//Makes sure there are exactly 3 commandline arguments

	if (argc != 4)

	{

		printf("propers usage: %s int int int", argv[0] );

	}

	else

	{

		//2. Print them in a message

		//Prints to screen our command line args (string form)

		//printf("\ncommand line arguments are: %s %s %s\n\n", argv[1], argv[2],
argv[3]);


		//Prints to screen to contents of producers, consumers, items

		printf("producers: 2^%d = %d\n", atoi(argv[1]), producers);

		printf("consumers: 2^%d = %d\n", atoi(argv[2]), consumers);

		printf("    items: 2^%d = %d\n", atoi(argv[3]), items);


		//3. Initialize the synchronization objects
```

```
		//full = CreateSemaphore(NULL, 0, BUFFER_SIZE, NULL); //tells consumer
that there are 0 full slots, consumer can't consume

		//empty = CreateSemaphore(NULL, BUFFER_SIZE, BUFFER_SIZE, NULL); //starts
with 16 since there are 16 empty slots that producers can fill

		//binarySemaphore = CreateSemaphore(NULL, 1, 1, NULL); //allows access 1
at a time to global buffer



		sem_init(&empty, 0, BUFFER_SIZE);

		sem_init(&full, 0, 0);

		pthread_mutex_init(&myMutex, NULL);



		//4. Spawn all of the threads/

		//printf("producers = %d\nconsumers = %d\n", producers, consumers);

		for(i = 0; i < producers; i++)

		{

			pthread_attr_init(&attr);

			params[i].thread_number = i;

			//params[i].producers = producers;

			//params[i].consumers = consumers;

			//params[i].items = items;

			//prodHandle[i] = CreateThread(NULL, 0, producerThread,
&params[i], 0, NULL);

			pthread_create(&ThreadIdProducers[i], &attr, producerThread,
&params[i]);

			/*pthread_exit(0);

			if(prodHandle[i] == NULL)

			{

				fprintf(stderr, "Error: CreateThread, producer thread
%i.\n",prodHandle[i]);

				ExitProcess(i);

			}*/

		}
```

```
		for(i = 0; i < consumers; i++)
		{
			//conHandle[i] = CreateThread(NULL, 0, consumerThread, &params[0],
0, NULL);
			pthread_create(&ThreadIdConsumers[i], &attr, consumerThread,
&params[i]);
			/*
			if(conHandle[i] == NULL)
			{
				fprintf(stderr, "Error: CreateThread, producer thread
%i.\n",conHandle[i]);
				ExitProcess(i);
			}*/
		}

		//5. Wait for all the threads to complete,
		for(i = 0; i < producers; i++)
		{
			pthread_join(ThreadIdProducers[i], NULL);
		}
		for(i = 0; i < consumers; i++)
		{
			pthread_join(ThreadIdConsumers[i], NULL);
		}
		//pthread_exit(0);
		sem_destroy(&empty);
		sem_destroy(&full);
		pthread_mutex_destroy(&myMutex);
		//WaitForMultipleObjects(producers, prodHandle, TRUE, INFINITE);
		//WaitForMultipleObjects(consumers, conHandle, TRUE, INFINITE);
```

```
		//6. Print a final message

		printf("\nAll producer and consumer threads have finished.\n");

		printf("Number of producers: %d\nNumber of consumers: %d\nNumber of items
per producers: %d\n\n", producers, consumers, items);

		printf("Number Produced: %d\nNumber Consumed: %d\n", numProduced,
numConsumed);

		free(params);


	}

	return 0;

}


void *consumerThread (void *param)

{

	struct threadParams thisThreadsParams = *(struct threadParams*)param;

	int producers = globalProducers;//thisThreadsParams.producers;

	int consumers = globalConsumers;//thisThreadsParams.consumers;

	int items = globalItems;//thisThreadsParams.items;

	int ItemsToConsume = (producers*items)/consumers;

	int i;

	for(i = 0; i < ItemsToConsume; i++)

	{

		sem_wait(&full);

		//WaitForSingleObject(full, INFINITE);

		pthread_mutex_lock(&myMutex);

		//WaitForSingleObject(binarySemaphore, INFINITE);

		/*critical section */

		bufferPTR--;

		printf("%d consumed from buffer[%d]\n", buffer[bufferPTR], bufferPTR);

		numConsumed++;

		/*end critical section */
```

```
		sem_post(&empty);
		//ReleaseSemaphore(empty, 1, NULL);
		pthread_mutex_unlock(&myMutex);
		//ReleaseSemaphore(binarySemaphore, 1, NULL);
	}
	return 0;
}

void *producerThread (void *param)
{
	struct threadParams thisThreadsParams = *(struct threadParams*)param;
	int producers = globalProducers;//thisThreadsParams.producers;
	int consumers = globalConsumers;//thisThreadsParams.consumers;
	int items = globalItems;//thisThreadsParams.items;
	int threadNum = thisThreadsParams.thread_number;

	int counter = 0;
	int item;
	int totalItems = items;
	int i = 0;

	for(i = 0; i < totalItems; i++)
	{
		sem_wait(&empty);
		//WaitForSingleObject(empty, INFINITE); //producers wait for empty slots
to fill
		pthread_mutex_lock(&myMutex);
		//WaitForSingleObject(binarySemaphore, INFINITE); //wait for binary
semaphore
		/*critical section */
		item = threadNum * numProduced++ + counter++;
```

```
            buffer[bufferPTR++] = item;
            printf("Item: %d created\n", item);
            /*critical section ends */
            sem_post(&full);
            //ReleaseSemaphore(full, 1, NULL);
            pthread_mutex_unlock(&myMutex);
            //ReleaseSemaphore(binarySemaphore, 1, NULL);
        }
        return 0;
}
```

Output: win32.1.race

```
C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri
ncipals Kelly Cassidy\hw5>win32.1.race test.txt
length of string in file is: 65536
0 occurrences of 0x0
0 occurrences of 0x1
0 occurrences of 0x2
0 occurrences of 0x3
0 occurrences of 0x4
0 occurrences of 0x5
0 occurrences of 0x6
0 occurrences of 0x7
0 occurrences of 0x8
0 occurrences of 0x9
0 occurrences of 0xa
0 occurrences of 0xb
0 occurrences of 0xc
0 occurrences of 0xd
0 occurrences of 0xe
0 occurrences of 0xf
0 occurrences of 0x10
0 occurrences of 0x11
0 occurrences of 0x12
0 occurrences of 0x13
0 occurrences of 0x14
0 occurrences of 0x15
0 occurrences of 0x16
0 occurrences of 0x17
0 occurrences of 0x18
0 occurrences of 0x19
```

0 occurrences of 0x1a

0 occurrences of 0x1b

0 occurrences of 0x1c

0 occurrences of 0x1d

0 occurrences of 0x1e

0 occurrences of 0x1f

2687 occurrences of 0x20

31316 occurrences of !

0 occurrences of "

0 occurrences of #

0 occurrences of $

0 occurrences of %

0 occurrences of &

0 occurrences of '

0 occurrences of (

0 occurrences of)

0 occurrences of *

0 occurrences of +

0 occurrences of ,

0 occurrences of -

0 occurrences of .

0 occurrences of /

0 occurrences of 0

0 occurrences of 1

0 occurrences of 2

0 occurrences of 3

0 occurrences of 4

0 occurrences of 5

0 occurrences of 6

0 occurrences of 7

0 occurrences of 8

0 occurrences of 9

0 occurrences of :

0 occurrences of ;

0 occurrences of <

0 occurrences of =

0 occurrences of >

0 occurrences of ?

0 occurrences of @

0 occurrences of A

0 occurrences of B

0 occurrences of C

0 occurrences of D

0 occurrences of E

0 occurrences of F

0 occurrences of G

2681 occurrences of H

0 occurrences of I

0 occurrences of J

0 occurrences of K

0 occurrences of L

0 occurrences of M

0 occurrences of N

0 occurrences of O

0 occurrences of P

0 occurrences of Q

0 occurrences of R

0 occurrences of S

0 occurrences of T

0 occurrences of U

0 occurrences of V

2693 occurrences of W

0 occurrences of X

0 occurrences of Y

0 occurrences of Z

0 occurrences of [

0 occurrences of \

0 occurrences of]

0 occurrences of ^

0 occurrences of _

0 occurrences of `

0 occurrences of a

0 occurrences of b

0 occurrences of c

2692 occurrences of d

2682 occurrences of e

0 occurrences of f

0 occurrences of g

0 occurrences of h

0 occurrences of i

0 occurrences of j

0 occurrences of k

7867 occurrences of l

0 occurrences of m

0 occurrences of n

5226 occurrences of o

0 occurrences of p

0 occurrences of q

2693 occurrences of r

0 occurrences of s

```
0 occurrences of t

0 occurrences of u

0 occurrences of v

0 occurrences of w

0 occurrences of x

0 occurrences of y

0 occurrences of z

0 occurrences of {

0 occurrences of |

0 occurrences of }

0 occurrences of ~

0 occurrences of 0x7f


C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri

ncipals Kelly Cassidy\hw5>
```

Output: win32.1.sync

```
C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri

ncipals Kelly Cassidy\hw5>win32.1.sync test.txt

length of string in file is: 65536

0 occurrences of 0x0

0 occurrences of 0x1

0 occurrences of 0x2

0 occurrences of 0x3

0 occurrences of 0x4

0 occurrences of 0x5

0 occurrences of 0x6

0 occurrences of 0x7

0 occurrences of 0x8

0 occurrences of 0x9

0 occurrences of 0xa
```

0 occurrences of 0xb

0 occurrences of 0xc

0 occurrences of 0xd

0 occurrences of 0xe

0 occurrences of 0xf

0 occurrences of 0x10

0 occurrences of 0x11

0 occurrences of 0x12

0 occurrences of 0x13

0 occurrences of 0x14

0 occurrences of 0x15

0 occurrences of 0x16

0 occurrences of 0x17

0 occurrences of 0x18

0 occurrences of 0x19

0 occurrences of 0x1a

0 occurrences of 0x1b

0 occurrences of 0x1c

0 occurrences of 0x1d

0 occurrences of 0x1e

0 occurrences of 0x1f

2731 occurrences of 0x20

35492 occurrences of !

0 occurrences of "

0 occurrences of #

0 occurrences of $

0 occurrences of %

0 occurrences of &

0 occurrences of '

0 occurrences of (

0 occurrences of)

0 occurrences of *

0 occurrences of +

0 occurrences of ,

0 occurrences of -

0 occurrences of .

0 occurrences of /

0 occurrences of 0

0 occurrences of 1

0 occurrences of 2

0 occurrences of 3

0 occurrences of 4

0 occurrences of 5

0 occurrences of 6

0 occurrences of 7

0 occurrences of 8

0 occurrences of 9

0 occurrences of :

0 occurrences of ;

0 occurrences of <

0 occurrences of =

0 occurrences of >

0 occurrences of ?

0 occurrences of @

0 occurrences of A

0 occurrences of B

0 occurrences of C

0 occurrences of D

0 occurrences of E

0 occurrences of F

0 occurrences of G

2732 occurrences of H

0 occurrences of I

0 occurrences of J

0 occurrences of K

0 occurrences of L

0 occurrences of M

0 occurrences of N

0 occurrences of O

0 occurrences of P

0 occurrences of Q

0 occurrences of R

0 occurrences of S

0 occurrences of T

0 occurrences of U

0 occurrences of V

2731 occurrences of W

0 occurrences of X

0 occurrences of Y

0 occurrences of Z

0 occurrences of [

0 occurrences of \

0 occurrences of]

0 occurrences of ^

0 occurrences of _

0 occurrences of `

0 occurrences of a

0 occurrences of b

0 occurrences of c

2731 occurrences of d

```
2732 occurrences of e

0 occurrences of f

0 occurrences of g

0 occurrences of h

0 occurrences of i

0 occurrences of j

0 occurrences of k

8194 occurrences of l

0 occurrences of m

0 occurrences of n

5462 occurrences of o

0 occurrences of p

0 occurrences of q

2731 occurrences of r

0 occurrences of s

0 occurrences of t

0 occurrences of u

0 occurrences of v

0 occurrences of w

0 occurrences of x

0 occurrences of y

0 occurrences of z

0 occurrences of {

0 occurrences of |

0 occurrences of }

0 occurrences of ~

0 occurrences of 0x7f


C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri

ncipals Kelly Cassidy\hw5>
```

Output: win32.2

```
C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri
ncipals Kelly Cassidy\hw5>win32.2 1 1 5
producers: 2^1 = 2
consumers: 2^1 = 2
    items: 2^5 = 32
Item: 0 created
Item: 0 created
0 consumed from buffer[1]
Item: 3 created
Item: 1 created
1 consumed from buffer[2]
Item: 6 created
6 consumed from buffer[2]
Item: 2 created
2 consumed from buffer[2]
Item: 9 created
9 consumed from buffer[2]
Item: 3 created
3 consumed from buffer[2]
Item: 12 created
12 consumed from buffer[2]
Item: 4 created
4 consumed from buffer[2]
Item: 15 created
15 consumed from buffer[2]
Item: 5 created
5 consumed from buffer[2]
Item: 18 created
18 consumed from buffer[2]
```

```
Item: 6 created

6 consumed from buffer[2]

Item: 21 created

21 consumed from buffer[2]

Item: 7 created

7 consumed from buffer[2]

Item: 24 created

24 consumed from buffer[2]

Item: 8 created

8 consumed from buffer[2]

Item: 27 created

27 consumed from buffer[2]

Item: 9 created

9 consumed from buffer[2]

Item: 30 created

30 consumed from buffer[2]

Item: 10 created

10 consumed from buffer[2]

Item: 33 created

33 consumed from buffer[2]

Item: 11 created

11 consumed from buffer[2]

Item: 36 created

36 consumed from buffer[2]

Item: 12 created

12 consumed from buffer[2]

Item: 39 created

39 consumed from buffer[2]

Item: 13 created

13 consumed from buffer[2]
```

```
Item: 42 created
42 consumed from buffer[2]
Item: 14 created
14 consumed from buffer[2]
Item: 45 created
45 consumed from buffer[2]
Item: 15 created
15 consumed from buffer[2]
Item: 48 created
48 consumed from buffer[2]
Item: 16 created
16 consumed from buffer[2]
Item: 51 created
51 consumed from buffer[2]
Item: 17 created
17 consumed from buffer[2]
Item: 54 created
54 consumed from buffer[2]
Item: 18 created
18 consumed from buffer[2]
Item: 57 created
57 consumed from buffer[2]
Item: 19 created
19 consumed from buffer[2]
Item: 60 created
60 consumed from buffer[2]
Item: 20 created
20 consumed from buffer[2]
Item: 63 created
63 consumed from buffer[2]
```

```
Item: 21 created

21 consumed from buffer[2]

Item: 66 created

66 consumed from buffer[2]

Item: 22 created

22 consumed from buffer[2]

Item: 69 created

69 consumed from buffer[2]

Item: 23 created

23 consumed from buffer[2]

Item: 72 created

72 consumed from buffer[2]

Item: 24 created

24 consumed from buffer[2]

Item: 75 created

75 consumed from buffer[2]

Item: 25 created

25 consumed from buffer[2]

Item: 78 created

78 consumed from buffer[2]

Item: 26 created

26 consumed from buffer[2]

Item: 81 created

81 consumed from buffer[2]

Item: 27 created

27 consumed from buffer[2]

Item: 84 created

84 consumed from buffer[2]

Item: 28 created

28 consumed from buffer[2]
```

```
Item: 87 created

87 consumed from buffer[2]

Item: 29 created

29 consumed from buffer[2]

Item: 90 created

90 consumed from buffer[2]

Item: 30 created

30 consumed from buffer[2]

Item: 93 created

93 consumed from buffer[2]

Item: 31 created

31 consumed from buffer[2]

3 consumed from buffer[1]

0 consumed from buffer[0]


All producer and consumer threads have finished.

Number of producers: 2

Number of consumers: 2

Number of items per producers: 32


Number Produced: 64

Number Consumed: 64

C:\Users\czar__000\Google Drive\School\Spring 2015\CSC 415 Operating Systems Pri

ncipals Kelly Cassidy\hw5>
```

Output: posix.1.race

archie@Linux64-VirtualBox:~/Desktop/hw5$ ./posix.1.race test.txt
length of string in file is: 65536
0 occurrences of 0x0
0 occurrences of 0x1
0 occurrences of 0x2
0 occurrences of 0x3
0 occurrences of 0x4
0 occurrences of 0x5

0 occurrences of 0x6
0 occurrences of 0x7
0 occurrences of 0x8
0 occurrences of 0x9
0 occurrences of 0xa
0 occurrences of 0xb
0 occurrences of 0xc
0 occurrences of 0xd
0 occurrences of 0xe
0 occurrences of 0xf
0 occurrences of 0x10
0 occurrences of 0x11
0 occurrences of 0x12
0 occurrences of 0x13
0 occurrences of 0x14
0 occurrences of 0x15
0 occurrences of 0x16
0 occurrences of 0x17
0 occurrences of 0x18
0 occurrences of 0x19
0 occurrences of 0x1a
0 occurrences of 0x1b
0 occurrences of 0x1c
0 occurrences of 0x1d
0 occurrences of 0x1e
0 occurrences of 0x1f
5430 occurrences of 0x20
68148 occurrences of !
0 occurrences of "
0 occurrences of #
0 occurrences of $
0 occurrences of %
0 occurrences of &
0 occurrences of '
0 occurrences of (
0 occurrences of)
0 occurrences of *
0 occurrences of +
0 occurrences of ,
0 occurrences of -
0 occurrences of .
0 occurrences of /
0 occurrences of 0
0 occurrences of 1
0 occurrences of 2
0 occurrences of 3
0 occurrences of 4
0 occurrences of 5
0 occurrences of 6
0 occurrences of 7
0 occurrences of 8

0 occurrences of 9
0 occurrences of :
0 occurrences of ;
0 occurrences of <
0 occurrences of =
0 occurrences of >
0 occurrences of ?
0 occurrences of @
0 occurrences of A
0 occurrences of B
0 occurrences of C
0 occurrences of D
0 occurrences of E
0 occurrences of F
0 occurrences of G
5426 occurrences of H
0 occurrences of I
0 occurrences of J
0 occurrences of K
0 occurrences of L
0 occurrences of M
0 occurrences of N
0 occurrences of O
0 occurrences of P
0 occurrences of Q
0 occurrences of R
0 occurrences of S
0 occurrences of T
0 occurrences of U
0 occurrences of V
5446 occurrences of W
0 occurrences of X
0 occurrences of Y
0 occurrences of Z
0 occurrences of [
0 occurrences of \
0 occurrences of]
0 occurrences of ^
0 occurrences of _
0 occurrences of `
0 occurrences of a
0 occurrences of b
0 occurrences of c
5447 occurrences of d
5428 occurrences of e
0 occurrences of f
0 occurrences of g
0 occurrences of h
0 occurrences of i
0 occurrences of j
0 occurrences of k

16140 occurrences of l
0 occurrences of m
0 occurrences of n
10810 occurrences of o
0 occurrences of p
0 occurrences of q
5449 occurrences of r
0 occurrences of s
0 occurrences of t
0 occurrences of u
0 occurrences of v
0 occurrences of w
0 occurrences of x
0 occurrences of y
0 occurrences of z
0 occurrences of {
0 occurrences of |
0 occurrences of }
0 occurrences of ~
0 occurrences of 0x7f
archie@Linux64-VirtualBox:~/Desktop/hw5$

Output: posix.1.sync

archie@Linux64-VirtualBox:~/Desktop/hw5$ ./posix.1.sync test.txt
length of string in file is: 65536
0 occurrences of 0x0
0 occurrences of 0x1
0 occurrences of 0x2
0 occurrences of 0x3
0 occurrences of 0x4
0 occurrences of 0x5
0 occurrences of 0x6
0 occurrences of 0x7
0 occurrences of 0x8
0 occurrences of 0x9
0 occurrences of 0xa
0 occurrences of 0xb
0 occurrences of 0xc
0 occurrences of 0xd
0 occurrences of 0xe
0 occurrences of 0xf
0 occurrences of 0x10
0 occurrences of 0x11
0 occurrences of 0x12
0 occurrences of 0x13
0 occurrences of 0x14
0 occurrences of 0x15
0 occurrences of 0x16
0 occurrences of 0x17
0 occurrences of 0x18

0 occurrences of 0x19
0 occurrences of 0x1a
0 occurrences of 0x1b
0 occurrences of 0x1c
0 occurrences of 0x1d
0 occurrences of 0x1e
0 occurrences of 0x1f
5462 occurrences of 0x20
70984 occurrences of !
0 occurrences of "
0 occurrences of #
0 occurrences of $
0 occurrences of %
0 occurrences of &
0 occurrences of '
0 occurrences of (
0 occurrences of)
0 occurrences of *
0 occurrences of +
0 occurrences of ,
0 occurrences of -
0 occurrences of .
0 occurrences of /
0 occurrences of 0
0 occurrences of 1
0 occurrences of 2
0 occurrences of 3
0 occurrences of 4
0 occurrences of 5
0 occurrences of 6
0 occurrences of 7
0 occurrences of 8
0 occurrences of 9
0 occurrences of :
0 occurrences of ;
0 occurrences of <
0 occurrences of =
0 occurrences of >
0 occurrences of ?
0 occurrences of @
0 occurrences of A
0 occurrences of B
0 occurrences of C
0 occurrences of D
0 occurrences of E
0 occurrences of F
0 occurrences of G
5464 occurrences of H
0 occurrences of I
0 occurrences of J
0 occurrences of K

0 occurrences of L
0 occurrences of M
0 occurrences of N
0 occurrences of O
0 occurrences of P
0 occurrences of Q
0 occurrences of R
0 occurrences of S
0 occurrences of T
0 occurrences of U
0 occurrences of V
5462 occurrences of W
0 occurrences of X
0 occurrences of Y
0 occurrences of Z
0 occurrences of [
0 occurrences of \
0 occurrences of]
0 occurrences of ^
0 occurrences of _
0 occurrences of `
0 occurrences of a
0 occurrences of b
0 occurrences of c
5462 occurrences of d
5464 occurrences of e
0 occurrences of f
0 occurrences of g
0 occurrences of h
0 occurrences of i
0 occurrences of j
0 occurrences of k
16388 occurrences of l
0 occurrences of m
0 occurrences of n
10924 occurrences of o
0 occurrences of p
0 occurrences of q
5462 occurrences of r
0 occurrences of s
0 occurrences of t
0 occurrences of u
0 occurrences of v
0 occurrences of w
0 occurrences of x
0 occurrences of y
0 occurrences of z
0 occurrences of {
0 occurrences of |
0 occurrences of }
0 occurrences of ~

0 occurrences of 0x7f
archie@Linux64-VirtualBox:~/Desktop/hw5$

Output: posix.2

```
archie@Linux64-VirtualBox:~/Desktop/hw5$ ./posix.2 1 1 5
producers: 2^1 = 2
consumers: 2^1 = 2
   items: 2^5 = 32
Item: 0 created
Item: 1 created
Item: 2 created
Item: 3 created
Item: 4 created
Item: 5 created
Item: 6 created
Item: 7 created
Item: 8 created
Item: 9 created
Item: 10 created
Item: 11 created
Item: 12 created
Item: 13 created
Item: 14 created
Item: 15 created
15 consumed from buffer[15]
14 consumed from buffer[14]
13 consumed from buffer[13]
12 consumed from buffer[12]
11 consumed from buffer[11]
10 consumed from buffer[10]
Item: 17 created
17 consumed from buffer[10]
9 consumed from buffer[9]
8 consumed from buffer[8]
7 consumed from buffer[7]
6 consumed from buffer[6]
5 consumed from buffer[5]
4 consumed from buffer[4]
3 consumed from buffer[3]
2 consumed from buffer[2]
1 consumed from buffer[1]
0 consumed from buffer[0]
Item: 15 created
Item: 16 created
Item: 17 created
Item: 18 created
Item: 19 created
Item: 20 created
Item: 21 created
Item: 22 created
```

Item: 23 created
Item: 24 created
Item: 25 created
Item: 26 created
Item: 27 created
Item: 28 created
Item: 29 created
29 consumed from buffer[14]
28 consumed from buffer[13]
27 consumed from buffer[12]
26 consumed from buffer[11]
25 consumed from buffer[10]
24 consumed from buffer[9]
23 consumed from buffer[8]
22 consumed from buffer[7]
21 consumed from buffer[6]
20 consumed from buffer[5]
19 consumed from buffer[4]
18 consumed from buffer[3]
17 consumed from buffer[2]
16 consumed from buffer[1]
Item: 34 created
34 consumed from buffer[1]
15 consumed from buffer[0]
Item: 30 created
Item: 31 created
31 consumed from buffer[1]
30 consumed from buffer[0]
Item: 38 created
Item: 40 created
Item: 42 created
Item: 44 created
44 consumed from buffer[3]
42 consumed from buffer[2]
40 consumed from buffer[1]
38 consumed from buffer[0]
Item: 46 created
Item: 48 created
Item: 50 created
Item: 52 created
Item: 54 created
Item: 56 created
Item: 58 created
Item: 60 created
Item: 62 created
Item: 64 created
Item: 66 created
Item: 68 created
Item: 70 created
Item: 72 created
Item: 74 created

```
Item: 76 created
76 consumed from buffer[15]
74 consumed from buffer[14]
72 consumed from buffer[13]
70 consumed from buffer[12]
68 consumed from buffer[11]
66 consumed from buffer[10]
64 consumed from buffer[9]
Item: 78 created
Item: 80 created
Item: 82 created
Item: 84 created
Item: 86 created
Item: 88 created
Item: 90 created
90 consumed from buffer[15]
88 consumed from buffer[14]
86 consumed from buffer[13]
84 consumed from buffer[12]
82 consumed from buffer[11]
80 consumed from buffer[10]
78 consumed from buffer[9]
62 consumed from buffer[8]
60 consumed from buffer[7]
58 consumed from buffer[6]
56 consumed from buffer[5]
54 consumed from buffer[4]
52 consumed from buffer[3]
50 consumed from buffer[2]
48 consumed from buffer[1]
46 consumed from buffer[0]
Item: 92 created
Item: 94 created
94 consumed from buffer[1]
92 consumed from buffer[0]

All producer and consumer threads have finished.
Number of producers: 2
Number of consumers: 2
Number of items per producers: 32

Number Produced: 64
Number Consumed: 64
archie@Linux64-VirtualBox:~/Desktop/hw5$
```