**CS2106 Operating Systems**

**Lab 5 – Semaphores**

|  |  |
| --- | --- |
| Name 1: Quek Yang Sheng | Student Number 1: A0126400Y |
| Name 2: Ang Wei Ming | Student Number 2: A0168721B |
| Name 3: | Student Number 3: |

**Question 1 (1 mark)**

This program does not run correctly.

**Question 2. (3 marks)**

This program may not operate correctly in a multithreaded because multiple threads are able to run the enqueue and dequeue functions, the count, front, and back variables to be inaccurate. The value of count can be less than QLEN even when the buffer is full due to race conditions on the variable. Therefore, allowing multiple threads to overwrite the values inside the buffer. Race conditions on the len array also causes some buffer to not copy fully.

**Question 3. (4 marks)**

My code is shown below:

|  |
| --- |
| #ifndef \_\_Q\_HEADER\_\_  #define \_\_Q\_HEADER\_\_  #include <pthread.h>  #include <semaphore.h>  //# of entries in the queue  #define QLEN 10  // Length of each queue entry in bytes  #define ENTRY\_SIZE 64  typedef struct  {  char data[QLEN][ENTRY\_SIZE];  int len[QLEN];  int front, back;  int count;  pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;  sem\_t empty, full;  } TBuffer;  void initBuffer(TBuffer \*buffer);  void enq(TBuffer \*buffer, const char \*data, int len);  int deq(TBuffer \*buffer, char \*data);  #endif |

|  |
| --- |
| void enq(TBuffer \*buffer, const char \*data, int len)  {  // I only added lock and unlock and also initialisation in buffer.h for question 3  if(buffer->count >= QLEN) {  sem\_wait(&buffer->empty);  }        // This is the critical section  pthread\_mutex\_lock(&buffer->mutex);  unsigned int bytesToCopy = (len < ENTRY\_SIZE ? len : ENTRY\_SIZE);  memcpy(buffer->data[buffer->back], data, bytesToCopy);  buffer->len[buffer->back] = bytesToCopy;  buffer->count++;  buffer->back = (buffer->back + 1) % QLEN;    sem\_post(&buffer->full);  pthread\_mutex\_unlock(&buffer->mutex);  }  int deq(TBuffer \*buffer, char \*data)  {  // I only added lock and unlock and also initialisation in buffer.h for question 3  if(buffer->count == 0){  sem\_wait(&buffer->full);  }        // This is the critical section  pthread\_mutex\_lock(&buffer->mutex);  int len = buffer->len[buffer->front];  memcpy(data, buffer->data[buffer->front], len);  buffer->count--;  buffer->front = (buffer->front + 1) % QLEN;  sem\_post(&buffer->empty);  pthread\_mutex\_unlock(&buffer->mutex);  return len;  } |

**Question 4. (1 mark)**

The answer is not correct.

**Question 5. (3 marks)**

The problem in Q4 is the enqueue function in the buffer.cpp returns when there is no space in the buffer, causing the data being enqueued to be lost.

The sem\_wait and sem\_post calls fixed the problem because in the enqueue function in buffer.cpp, when the buffer is full, it will wait for the empty semaphore to be called. This empty semaphore will only be called when the dequeue function successfully dequeues data from the buffer, making space for the enqueue function to write data. Similarly, in the dequeue function, when there is no data in the buffer to be read, it will wait for the full semaphore to be called, which the enqueue function will when it finishes writing data into the empty buffer.

**Question 6. (4 marks)**

Changes to my web server are shown below with explanation:

|  |
| --- |
| TBuffer buffer;  int main(int ac, char \*\*av)  {  …        …        …  **initBuffer(&buffer);**  pthread\_t loggerThread;  pthread\_create(&loggerThread, NULL, logger, NULL);  startServer(PORTNUM);  }  Explanation: Buffer is initialized in main() |
| void \*logger(){  while(1){  /\*removed previous mutexes. relying on buffer internal mutex\*/  char data[ENTRY\_SIZE];  int len = deq(&buffer, data);  if(len>=0){  fwrite(data, strlen(data), 1, logfptr);  fflush(logfptr);  }  }  } |
| void writeLog(const char \*format, ...)  {  char myBuffer[LOG\_BUFFER\_LEN];  va\_list args;    va\_start(args, format);  vsprintf(myBuffer, format, args);  va\_end(args);  /\*removed previous mutexes. relying on buffer internal mutex\*/  char data[ENTRY\_SIZE];  sprintf(data, "%s: %s\n", getCurrentTime(), myBuffer);  enq(&buffer, data, strlen(data)+1);  } |

**Question 7.** **(1 mark)**

This is what’s happening in testbarrier.cpp: Different threads calls the barrier at different time and only when the last thread calls the barrier, all of them crosses the barrier together.

**Question 8. (3 marks)**

This is how reachBarrier works: Whenever reachBarrier is being called, it increments the number of threads that has reached the barrier, if the number is less than the required number of threads to continue, it will wait for its own semaphore to be called. Once the last thread reaches the barrier, it calls the semaphore of the thread with the highest procNum so that it can continue to run. This thread will then call the semaphore of the thread with id of procNum – 1 and this applies to every thread following that until procNum == 0, which allows all the threads to finish running reachBarrier and exit.

TOTAL: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / 20