**Lab – Wrapping a shared Object to become**

**Multithreading-Safe**

**Introduction**

In a system, there is an image capture card (ICC) which is responsible for capturing an image into hardware frame buffers. To provide higher parallelism, the ICC has two frame buffers – 0,1. So, while ICC is capturing image from VGA source and writing into frame buffer 1, other processes can read previously captured image from frame buffer 0 at the same time. The ICC hardware is designed to switch between frame buffer 0 and frame buffer 1 iteratively.

There is no proper hardware synchronization implemented so far. For example, if ICC is invoked to capture image #1, it will be stored in frame buffer 0. Image #2 will be stored in frame buffer 1. However, image #3 will be stored into frame buffer 0 again. At this time, ICC will simply write the image #3 over the frame buffer 0.

The manufacturer (Company SEPTortureME) of ICC provide a poor sample program which shows how a client can interact with the hardware.

The pseudo code of the sample program is as follows:  
Please read the behaviors of ICC carefully

|  |
| --- |
| char userbuffer[1000][1000] ; // the ICC can capture fixed resolution  // at 1000x1000 pixels image once at a time  ICC imc = new ICC();  while (imc.isbusy() == false) ;  // busy waiting until imc.status == true  fno = imc.capture(); // 1. it takes about 100 ms to grab image data from VGA source  // to the hardware’s framebuffer on the ICC  // 2. During this 100ms. ICC’s status will be set to “false”  // by hardware.  // 3. This method is a blocking call.  // 4. The ICC has 2 hardware framebuffer (0,1) onboard.  // ICC returns the framebuffer no which is written to  // 5. Once a framebuffer is written its internal Boolean   // variable “framestatus” is marked as “true”  If (imc.isframestatus(fno) == true) ;  imc.memcopy(userbuffer, fno) ;  // copy image data from hardware framebuffer #fno  // to userbuffer (user memory) which takes 20 ms.   // The framestatus should be marked as “false”  // after this call is completed.  // It is a blocking call |

Of course, **SEPTortureMe** is actually a hardware company. They cannot hire talented programmers who are familiar with concurrent programming. You know if this code is made multithreaded, it is very dangerous. For example, it is busy waiting, which makes CPU occupied. Second, if two or more threads (or processes) run the code the at same time, two threads can enter imc.capture() concurrently. That is, a race condition. Third, two or more threads are capable of invoking imc.memcopy() concurrently. Races are triggered and memory copy from the hardware framebuffer to user memory can be wrong.

Your talents, somehow, have been heard by **SEPTortureMe.** They pay you a lot of money to redesign the software into a multithread-safe shared object.

**Your Goals:**

1. You can do it by Java or C#. Choose your favorite programming language wisely and use the synchronization constructs provided by the programming language.
2. Since no real hardware is available. We provide a ICC emulator class for the lab. The pseudo code of ICC emulator class is attached to the end of this handout. Please rewrite it into your favorite programming language.
3. Rewrite and wrap the ICC emulator class into a shared object and become ***multithreading-safe****.* You should not touch the original behaviors of ICC but you can add any process/thread synchronizations into it. If your work is correct, your new ICC object can be accessed by arbitrary clients in your system.
4. Your code must be executable with least concurrency errors as much as possible.

**Tasks to complete**

1. Write a test driver program which create 10 clients, each with a thread. The purpose of the clients are to invoke your multithread-safe ICC object at the same time. The client behaviors (pseudo code) are as follows: (You should complete the incomplete parts in the pseudo code)

|  |
| --- |
| 1: a client {  2: char mybuffer[1000][1000] ;  3: ICC imc = …. // get the unique object of ICC  4: for (;;) {  5: fno = imc.capture();  6: imc.memcopy(mybuffer, fno);  7: // assert all elements in mybuffer[x][y] are the same  // to prove that your synchronization is correct.  char c = mybuffer[0][0];  for (int x=0;x<1000;x++) {  for (int y=0;y<1000;y++) {  assert(mybuffer[x][y] == c);  }  }  8: }  8: } |

1. The correct behaviors are:
   * When ICC is busy, line 5 should block and wait.
   * When ICC is free but both framebuffer 0 and 1 are newly written and are never copied, line 5 should block and wait until frame buffer is free.
   * When a memcopy(mybuffer, fno) begins, you should prevent ICC from writing new image to framebuffer #fno
   * When memcopy ends, framebuffer #fno should be marked as “false” to be used by future image capturing.
   * Only the two methods *capture()* and *memcopy()* are public to clients.
2. **IMPORTANT:** It is possible that a client captures an image but never copy the image data to user program. However, take these cases into consideration make the lab too hard for you. To simplify the lab, you do not need to consider the case. Just assume all the clients are benevolent and not malicious

**Summary**

1. You should invoke several clients at different timing to access and test your multithreading-safe ICC. Of course, ICC can only handle your clients one by one. Eventually, your clients should all be served and the assertion failure in line 7 is never triggered.
2. Place different delays to make your debugging easier

Good Luck !!! Wish you become a synchronization master.

**The pseudo code of ICC emulator**

The pseudo code is written in a C++ like style. It is not syntax-error for any compiler. However, please modify it into your programming language. Since there is real ICC hardware, please implement an ICC hardware emulator class as follows.

Each time, capture() is invoked, a character from ‘A’-‘Z’ is written to the hardware buffer for your debugging purpose. When character ‘Z’ is written, the character will restart from ‘A’.

Your goals of this lab is actually add new code to wrap this class to become multithread-safe shared object. However, you should not modify the behaviors of the emulator class.

|  |
| --- |
| class ICC {  bool busy ;  int currentframe = 0 ;  HWframebuffer hwframe[2] ;  bool framestatus[2] ; // “true” means the image data in HWframebuf is new  char c = ‘A’ ; // character c will be rotated from ‘A’ to ‘Z’ always  public isbusy() { return busy } ;  public int capture() {  int w = currentframe  busy = true ;  hwframe[w].write(c) ;  sleep(100ms); // put a reasonable delay for your own debugging  framestatus[w] = true ;  currentframe = 1 – currentframe ;  busy = false ;  if (c==’Z’) c = ‘A’ ;  else c++ ;  return w ;  }  public int memcopy(char\* userbuffer, int fno) {  // let’s simulate the memory copy  for (int x=0; x<1000 ; x++) {  for (int y=0;y<1000; y++) {  userbuffer[x][y] = hwframe[fno].getpixel(x,y) ;  framestatus[fno] = false ;  }  class HWframebuffer {  char hwdata[1000][1000] ;  public write(char c) {  // let’s simulate writing data into hardware framebuffer  int i,j ;  for (int i=0;i<1000; i++) {  for (intj=0;j<1000;j++) {  hwdata[i][j]= c ;  }  }  public char getpixel(int x, int y) { return hwdata[x][y] ; }  } |