Operating Systems Lab #2 Aleternate Fall 2013 Burris

To: Faithful Coding Warriors, our last hope!

Either complete this assignment ASAP or we all parish.

Due: Star Date: Thursday November, 1 2012 prior to the start of class. I will accept it late but please be on time to avoid a potential grade penalty. Please submit your lab early or on time. You will however receive more points for a “B” or “A” option lab submitted late than a “C” option lab submitted on time. Do not procrastinate. Additional labs will be assigned prior to the due date. This lab (or equivalent) has been completed by freshman using Ada, Java, and C++. ***If you complete the “A” option on this lab, 15% of the grade will be added to either your test or lab average (your choice) prior to computing the final class grade. If you complete the “B” option 10 % will be added, For the “C” option 5% will be added prior to computing the final grade.*** You may use Ada, C++, or Java to implement the lab. If you wish to use another language you must ask permission. I am willing to allow flexibility in requirements to meet implementation language restrictions but you must submit a written request for the substitution and receive permission.

An important part of the lab is to verify that you have implemented it correctly. When appropriate, I suggest you calculate by hand the correct answers for adequate test data and verify your program results. I will also verify your results by hand. If your results are incorrect, I will return the entire lab to you for corrections and deduct an appropriate penalty form your final grade. You must state on the first page of the lab (cover sheet with name, class, due date, and date submitted) which grading option you have completed. If you fail to state a grading option, the lab will be treated as a “C” option, i.e., I will not look for the additional features required for the “B” and “A” options while grading.

You cannot imagine our pain when the “Planet Search and Colonization” test group dropped out of hyperspace into Alderaan’s orbit to find nothing but space debris. We quickly jumped back into hyper space before Darth Vader could target us with the Death Star. We had been on a 6 month test run to prove it feasible to search for and colonize other planets in space.



<http://www.google.com/search?q=star+wars+alderaan&hl=en&client=firefox-a&hs=tUw&rls=org.mozilla:en-US:official&channel=np&prmd=imvns&tbm=isch&tbo=u&source=univ&sa=X&ei=kj0ST8--HJGqsAKuvqDYAw&sqi=2&ved=0CDUQsAQ&biw=1920&bih=938>

Alderaan was approaching its maximum supportable population and migration was necessary to prevent undesirable restriction on birth rates.

Upon dropping back into normal time/space we found ourselves hopelessly lost. There had not been sufficient time for navigation to plan a safe escape route into a sector of known space. We had sufficient supplies with recycling to allow about 2.5 generations to search for a habitable planet. Fortunately we have found one but time is running short. There is a vast asteroid field between us and the planet. Normally we would just go around the asteroid field. Unfortunately scarcity of resources requires we take the shortest possible path through the asteroid field to the planet.

Our Chief Engineer Scotty has told Captain Kirk (both nicknamed for famous explorers on the NCC 1701 of early Earth vintage) there is a potential solution to our problem but it is risky without sufficient simulations to provide us with the opportunity to make a safe transit. We have sufficient materials by cannibalizing some older ships to create two probes with photon torpedoes that can fire every 3 seconds and destroy 5 units of asteroid per shot. We have sufficient materials to build a much larger number of probes with phasers capable of firing every 2 seconds but only able to destroy 3 units of asteroid mass per shot. We will need one scout probe to lead the way reporting back to the main fleet an asteroid identifier (integer), the asteroid’s mass (integer) and time it was discovered in milliseconds. When free, each probe capable of destroying an asteroid will query the fleet for its next target. If the remaining time is not sufficient for the probe to destroy the target (time to collision <= current time + [time required to destroy the asteroid] ) the probe will ram the asteroid after sending a self destruction message to fleet.

What we need to know is the minimum number of phaser probes we must create in addition to the two photon torpedo probes to safely transit the asteroid field. Sensor measurements indicate we will encounter no more than 50 asteroids during the transit. For safety, we will assume 55 asteroids. Stop the simulation when we have destroyed 55 asteroids providing appropriate information on remaining defensive probes and probes lost.

If we lose all our defensive probes prior to destroying 55 asteroids, provide sufficient information for our next simulation.

**“D” Option (best grade is a 60):**

Mr. Spook has decided to store information on asteroids in a sequentially allocated stack size 15 when received from the scout probe. The stack will be popped on request to provide targeting information for defensive probes. Please ensure race conditions cannot occur! If new targeting information is received from the scout ship and the stack is full, discard the information. Each time asteroid targeting information must be discarded, the fleet’s inner shields incur 1 hit. When defensive probes request targeting information, Mr. Spock requires us to determine if the current time is greater than the time the asteroid was scheduled to collide. If a collision occurred, the inner shields incur another hit. If the shields incur 5 hits, the shields will fail and you must report the loss of the entire fleet. Whenever possible, defensive probes should be provided accurate targeting information. Please use monitors for all communications though semaphores or Ada rendezvous are acceptable. Central command would be impressed if you used sockets, RMI (remote method invocation), or CORBA (common object request broker architecture) to implement communications.

For convenience, targeting information may be stored in separate stacks (separate stacks for the asteroid ID, mass, and time discovered) or a single stack using logical records. Allow storage for exactly 15 asteroids.

You must provide a HUD (Heads-Up-Display) graphical display of all pertinent information updated in real time including the number asteroids awaiting destruction, number of asteroids destroyed, the number of asteroids in the stack, the number of defensive probes available, number of hits on the shields and current time. The time should be updated either each time an even occurs or once every second.

Information on generating random distributions may be found on Blackboard including code for the Exponential distribution in the epistle “Random Numbers.”

Provide sufficient output (at least the final HUD’s of multiple runs) to convenience me you have solved the “D” option. A discussion of results is expected with the output HUDs. The “D” Option code should appear next.

**“C” Option (best grade is a 70):**

The “D” Option sequentially allocated stack of 15 elements must be implemented as an array of targeting objects. The stack must be implemented as a separate “class” with methods to “insert” and “remove” objects from the stack. You may also wish to provide Boolean methods to indicate if the stack is empty or full. Another example appears in MyClass.java. You need not complete the “D” option. Rather implement the “C” Option subject to the new “C” option requirements.

Provide sufficient output (at least the final HUD’s of multiple runs) to convenience me you have solved the “C” option. A discussion of results is expected with the output HUDs. The “C” Option code should appear next.

**“B” Option (maximum grade is 85):**

First, implement the “C” Option. Next replace the “C” Option stack with a queue. The queue must be implemented as a separate “class” with methods to “insert” and “remove” objects from the queue. You may also wish to provide Boolean methods to indicate if the queue is empty or full. Note the queue containing targeting information is a critical resource. Management insists you not allow race conditions to occur. You must state explicitly why race conditions can or cannot occur in your solution as part of the lab! Hint: If you use the same API (methods) to define the queue used to define the stack, this is the only code that will have to be changed migrating from the “C” to the “B” option. Work smart, not hard!

Provide sufficient output (at least the final HUD’s of multiple runs) to convenience me you have solved the “C” option. A discussion of results is expected with the output HUDs. The “C” Option code should appear next. The HUD’s for the “B” Option solution should appear next followed by a discussion of “C” versus “B” option results. Explain why you should see a difference. *Include a block diagram exhibiting your strategy to prevent race conditions and explanation of why it will solve the problem! If a race condition does not exist, you must explain why it does not exist*! The “B” Option code should appear next.

**“A” Option (maximum grade is 100):**

First complete the “B” Option. Replace the “B” option queue with an array of 15 elements where each element is an object of a class. Each time new asteroid targeting information is received from the Scout probe, place it in descending order of time remaining to impact. When a defensive probe request targeting information always provide it the information from the last element in the array. That way we always assign defensive probes the asteroid that will impact the fleet next. The code to create and manipulate the array must be implemented as a separate class! Hint: If you use the same API (methods) to define the array operations used to define the stack and queue this is the only code that will have to be changed migrating from the “C” and “B” to the “A” option. Work smart, not hard!

Provide all “C” and “B” Option information in the specified order. Next provide sufficient output (at least the final HUD’s of multiple runs) to convenience me you have solved the “A” option. A discussion of results is expected comparing the “C,” “B,” and “A” Options. Which is the best strategy and why. Did the “A” Option strategy meet expectation? You must explain why or why not. The “A” Option code should appear next.

**You must make arrangements with Dr. B to demonstrate your labs works!**

**Initial Class Diagram**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  | Scout Probe (separate or thread) |
|  |  |  |  | probe ID;  parentIP;  probeType; |
|  |  |  |  | Initalize(Probe ID, IP, scout );  AcceptCommand( command); |
| Task Force Command |  |  |  |  |
| array of Probes (separate threads);  data structure to track asteroids;  data fields to maintain system HUD; |  |  |  |  |
| RequestTargetInfo(target info);  RequestInstructions(update info);  AcceptProbeMessage(new target);  AcceptProbeMessage(get target info);  AcceptProbeMessage(death); |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  | Defensive Probe (separate or thread) |
|  |  |  |  | probe ID;  probeType (phasor or photon torpeado); |
|  |  |  |  | Initalize(Probe ID, IP,  typeWeapon);  AcceptCommand( command);  AcceptTargetInfo( command); |
|  |  |  |  |  |

**Basic Logic (scenario):**

|  |  |
| --- | --- |
| **Task Force Command** |  |
| Prior to entering asteroid field:   1. Obtain start up information. 2. Create and launch probes. 3. Initialize graphical display (HUD). |  |
| Enter asteroid field:  Loop (till fleet destroyed or successful exit from asteroid field) {  Accept probe request from scout or defensive probe.  Case( request type){  Scout: process scout request adding  asteroid to targeting data structure, update clock;  Defensive: {  getTarget Loop{  if (target available)  if (time found + time to collision < current time)  Send target info to Defensive probe, update clock;  Iterate target loop;  Else  Record hit on main shields  End if  Else  Send no targets currently available;  Exit getTarget loop;  End if;  } End getTarget loop;  } end defensive loop  Update: Perform update;  If (asteroids destroyed exceeds 55)  Exit after recalling remaining probes and start celebration;  Else  If (hits on shields is >= 5)  report shields failed, allow remaining probes to self destruct and die  End if;  End if;  }End Loop |  |

The scout probe should detect new asteroids according to a Poison distributing with mean expected arrival time of 4 seconds (exponentially distributed).

The size of the asteroid is a step function. Generate a random variant 0 <= x <1.0. If x <= 0.2 the size is 3; if 0.20 < x <= 0.5 the size is 6 units; if 0.5 < x <= 0.7 the size 9 units; for x > 0.7 the size is 11 units. The time to impact must be uniformly distributed between 0.0 and 15 seconds.

Each distribution must be implemented as a unique function. The size of asteroids is of type integer.

**Basic Scout Probe (scenario):**

|  |  |
| --- | --- |
| **Scout Probe** |  |
| Obtain start up information including probe identification and IP of Task Force Command;  Loop {  Accept message from Task Force Command and process as required;  Wait random length of time exponentially distributed to detect next asteroid;  Get current time;  Determine time to impact (random, step function);  Determine asteroid mass ( random, uniformly distributed);  Send targeting information to Tack Force Command;  }End Loop |  |
|  |  |

**Basic Defensive Probe (scenario):**

|  |  |
| --- | --- |
| **Defensive Probe** |  |
| Obtain start up information including probe identification, IP of Task Force Command and weapon type;  Loop {  Request message from Task Force Command and process as required;  Request targeting information;  If(target available) {  If(remaining time till impact is sufficient to destroy target) {  Delay the time for required number of shots to destroy target;  Report target destroyed with our ID;  Delay any remaining time to allow weapon to recharge;  }  Else  }  Delay any remaining time till target impact after moving between the target and fleet;  Report our termination to Task Force Command when time expires;  Terminate self;  } // End if;  Else //no target  Rest prior to requesting target information 500 milliseconds;  //The process of making repetitive requesting is called “polling.”  End if;  }End Loop |  |
|  |  |

Typical Task Force Command/Scout Interaction:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Task Force | |  |  |  | Scout | |  |
|  |  |  | Accept message from Command |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | Send targeting info. |  |  |  |  |
|  |  |  |  |  |  |  |  |

Typical Task Force Command/Defensive Scout Interaction:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Task Force | |  |  |  | Defensive | |  |
|  |  |  | Accept message from Command |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | Request target information. |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | Send targeting information message. |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | Report of target or probe destruction. Report ready for action if there was no target. |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Architectural Considerations and Game Flexibility**

There are several approaches/strategies to solving the lab. You are free to select any of the following strategies or another so long as you preserve the basic structure and solve all problems described in the requirements.

**Strategy 1: A network game.**

In this strategy users would play the role of the scout and defensive probes. This would allow each player to start their scout or defensive probes separately on different computers and register via sockets, RMI, servlets, or another communications technology with the Task Force Command. A sample of this approach is the code for EchoServerJDK121 and EchoClientJDK121. The lab assumes a single Task Force Command (TFC) containing all data structures and control of the Command GUI which would be available (with updates from TFC). Since there is a single TFC requests from probes are serialized as they are serviced to completion one at a time. Note only one probe can access the single port at a time. With care it should be possible to prevent race conditions from occurring.

A primary drawback would be starting multiple probes on one or more computes during testing. I will allow you to just create probes artificially in the TFC if desired.

**Strategy 2: Multiple player game restricted to one machine.**

Unlike “C” and Ada where spawned task are completely independent of the parents, Java threads share portions of common memory. The shared memory allows direct access to global data structures and shared (reentrant methods) as shown by the code used for “Race1.java.” This shared access is substantially faster than using sockets. The code used to illustrate race condition (Race1 and Race2) would allow the main program to create all users as threads. Each thread could directly access the Command GUI as in Race1 with the potential for race conditions. To prevent Race conditions, the use of synchronized methods may be required. In turn, this could potentially result in reduced service times due to the interference caused by synchronization when the serialized access is enforced in synchronized methods. It is a trade off, accurate results versus a performance hit.

**Strategy 3: A multi-threaded server network game.**

If the service time for individual player/probe accessing the TFC is high, you may wish to multi-thread the server in solution 1. Again only a single probe may be serviced at a time using the single port in the design strategy for Solution 1. The solution to improve service is to multithread the services provided by the TFC. Assume you adopt this approach and all TFC servers have direct access to the data structures and Command GUI. You have reintroduced the potential for a race condition. Again, the only solution is to serialize access to shared resources. Once a user starts utilizing a resource all other users must be locked out till the first user releases the resource. Since the players are potentially on different machines and do not share memory, “synchronized methods” will not work.

Not part of the lab! There is no unique solution but a common approach to accomplish the synchronization might appear as follows using the Command GUI. This solution was proposed by Lt. Uhura.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | TFC 1 |  | Probe1 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | TFC 2 |  | Probe2 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | Serialization  Thread (synchronized) |  |  |  |  |  |
| Command  GUI |  | Local Data Structures |  |  |  |  |  |
|  |  | ReadGUIField1( ) {--}  ReadGUIField2( ){--}  WriteGUIField1( ){--}  WriteGUIField1( ){--} |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  | TFC N |  | ProbeN |
|  |  |  |  |  |  |  |  |

The code for a typical TFC would appear as follows:

|  |  |  |
| --- | --- | --- |
|  | TFC number J |  |
|  | Local data structures |  |
|  | Loop   1. Wait for service request; 2. Accept service request: 3. Code requiring substantial time. 4. Temp = ReadGUIField2( ); 5. Delay processing time. 6. WriteGUIField2( ); 7. WriteGUIField1( );   End Loop |  |
|  |  |  |

In this model each probe has its own copy of a TFC. Several probes could share the same TFC serially, one at a time. The “Read” and “Write” functions in the “Serialization Thread” should be as fast and efficient as possible to prevent TFS’s from have to wait extended lengths of time to obtain required information or update the specified field in the Command GUI.

There is a subtle potential problem with this solution. If read operations are not part of the decision process used to determine values written to the Command GUI this solution works. If the read operation influences values in the subsequent processing prior to the write operation there is a problem. Do you see it? This is not part of the lab grade. If you do not see the problem and have given the problem sufficient thought I would be happy to explain the problem. If you see the problem, do you see a solution using the basic approach? Hint: One consists of combining a read with a write operation. Another is called delayed locking. The general problem is referred to as a “critical section.”