



12 Pages

1. Please give the information requested in the boxes below.
2. Enter your name in the space provided to the right, fold over and seal down (using the label provided) to hide your name.
3. The answers must be legibly written. The Examiners can either ignore illegible script or have it typed and the expense of typing charged to the candidate.
4. Write your answers in ink.
5. Begin each answer on a fresh page. Do all rough work in the answer book and cross it through.
6. No part of this book is to be torn out; it must be handed in complete.
7. If more books than one are used, fasten them together using the tags available from the invigilator(s).
8. **Every book must be handed in. In no circumstances may answer books, used or unused, be taken out of the examination venue.**

Complete, fold and seal
(see note 2 below)

- 17019

SURNAME: Byrno

FORENAME(S): Joshua

1	Student Number	P144833
2	Title of Examination Paper	Network multiplayer games
3	Module Code	M6X01105
4	Date and Time of Examination	12 pm 13/05/2014
5	Examination Venue	302D
6	Seat Number	1
7	Number of Answer Books handed in	

[illegible]

(Gweler yr ochr arall am y fersiwn Cymraeg)

Sum

Question 01.A)

Within the figure shown, there appears to be no method of verifying that the data coming in has been received which could result in dropped packets of information as the client has sent a packet, potentially containing the user's input, to the server which has not obtained it or incorrectly stored it in such a way that it ends up not being recognised or processed. 2

One of the features I would implement is some form of information stack, most likely of a list type, that would take incoming data to be processed so that the possibility of something not being handled, most likely due to the threads handling the incoming request being unable to take on any additional pieces of information, would be largely avoided since it is now being retained server side. As an example;

Client sends a data request -> Server receives request and adds to stack -> Server processes the stack bottom up to deal with the oldest requests first -> Server sends the required data to the client.

A second feature that should be developed and implemented is associating the data request to the user's IP for validation purposes. This could be incorporated into the stack method described above to help ensure that the correct data that is requested is sent to the correct client.

Another feature that I would implement is multithreading the server's handling of requests and replies if it is not already doing so, specifically in terms of having a set of threads only handling incoming data and another set handling outgoing. This would help in dealing with multiple users at once and avoid bottlenecking the system due to the number of requests if everything is handled by one thread in a go, which in turn results in a higher user satisfaction as they won't be waiting as long. This would have to be implemented in tandem with the stack feature or else you would run the risk of the sending threads obtaining either the same data and sending it multiple times, or not obtaining data as they start to block each other.

One final feature I'd implement after the multithreaded request / reply system is that of a confirmation system that would send an automatic response that the data has been obtained and, if the other side does not receive confirmation, it is automatically resent again a few times before timing out. This would serve primarily to ensure that the data has been delivered properly and is being processed by the server. Without some form of confirmation system implemented, the figure shows that there is a single attempt at sending and then receiving the data, whilst if the feature is implemented there exists the possibility that if it wasn't obtained properly before, there would be attempts to correct the issue. The reason as to why this would need to be implemented after multithreading is to firstly avoid lengthening the data request / reply process, and secondly because it would be more efficient to have a separate process handling the data request / reply confirmations rather than the process handling the data requests / replies.

With these four features implemented, the chances of data not being obtained by the client or the server diminish as well as the probability that incorrect data for the requests received are sent by virtue of having a stack to contain data so that none of the requests are "dropped" or "disappear", a validation system to ensure that when data is sent, it's to the IP address associated with the data on the stack being processed at that time, a system to attempt re-sends of data when it has not been received as well as a system that enables better scalability for the server in terms of handling clients that would ultimately benefit whoever deployed the networked system.

4 ~~It's~~ An interesting analysis
but a little off
the point!

Question 01.B)

One of the primary advantages to having a pre-emptive kernel is that when handling the same types of information constantly for any duration of time, the pre-emptive kernel is able to "speed up" the process by predicting what the resulting information is. For example, if Player A is in a robot with 10 Lasers that deal 10 damage each but only up to 50 meters, past which they deal 0 damage, and player B is 90 meters away, a pre-emptive kernel would process through Player A firing at Player B faster as because all the weaponry information is the same, by the time it reaches weapon 7 as an example, it would already know that "No, it doesn't do any damage" and can start outputting that as the answer to a calculation involving damage results.

However, a downside of this form of pre-emptive calculation is that it can be prone to attacks of the software and hardware variety, best demonstrated with the SPECTER / SPECTRE and MELTDOWN vulnerabilities found in Intel CPUs in recent years. The general way that the vulnerabilities worked is that by manipulating the part of the CPU that would predict the outcome of a given function, for example the robot example above, a specially written form of code could peak at the information kept in the cache by forcing the CPU to execute it by tricking the predictor into thinking it was part of the function it had seen a few times already before any form of validation can take place.

For how this could relate to a games console, it could result in the video game platform being subject to misuse by malicious third-party individuals looking to gain information and / or control over aspects of the machine that they shouldn't be able to during gameplay, or otherwise implementing third-party software such as custom operating systems, similar to how the PlayStation Portable was able to have the base Sony Operating System replaced by a third-party one to enable applications such as emulators of old video game consoles to be installed.

Generally, it could cause the platform to be insecure.

When it comes to re-entrant functions, a pre-emptive kernel is theoretically less likely to encounter delays in executing the safe functions as it could "predict" when, in functions ran multiple times before, it needs to hop in and out which in turn results in faster processing times and an overall better gameplay experience. An obvious issue that can arise from this however is that if the console is processing a series of physics based calculations it's possible that the predicted result could be one that results in a jarring miscalculation. For example, if a ball has been bouncing along a path for a duration of time and then suddenly hits a wall, a poorly implemented pre-emptive kernel could potentially allow the ball to momentarily phase through the wall before shooting off in a random direction due to a physics collision, though sometimes the issue is related more to the game engine itself than the kernel.

Non-pre-emptive kernels however are more robust than their pre-emptive counterparts largely in part to the fact that they execute code thoroughly every time, rather than assuming after a while what a result could be. This means that there is no chance of a function being predicted incorrectly and causing some form of an issue in the software being ran, nor is there any form of vulnerability pertaining to the abuse of the prediction method.

Admittedly however this would mean that for multithreaded applications that it would take longer overall to execute the same code as a pre-emptive kernel as it would have to execute all 2+ threads. As an example as to the possible time discrepancy between a pre-emptive and a non-pre-emptive kernel in a round robin scheduling format, suppose that there are 24 robots as outline above each handled by their own thread, each with 10 lasers dealing 10 damage at 50 meters, 0 any further. All

(1b) 24 robots decide that they dislike the appearance of robot 25 which is 52 meters away from all but one of them which we will call Robby.

For a pre-emptive Kernel, as stated before, after the first robot or so the kernel would know that all the robots are going to miss. Except Robby, who's just in range but the kernel may accidentally miss this bit of information on a few of his shots as the kernel figures out that he is in range and not the same as the other 24. This means that Robby in his robot is robbed of 10 to maybe 30 of the damage he should have dealt to robot 25. However, because of the predictive nature of the kernel, all of the missed shots are dealt with much faster.

The non-pre-emptive kernel however has to process each thread and though it may take longer it ultimately means that Robby's robot doesn't lose out on dealing any damage which results in a better user experience as Robby's player doesn't feel like the game is unfair.

When it comes to supporting priority scheduling however, the pre-emptive Kernel would be better than the non-pre-emptive if time is the only factor unless it requires that the pre-emptive Kernel swap around a bunch in which case all advantages of a pre-emptive Kernel are lost.

Question 02.A)

The main benefit to a non-deterministic MAC is that it is changeable and can be assigned as needed to on a network. Where this can come in handy is when working with multiple virtual machines on one physical machine as it allows all of them to utilise their own MAC address for the network and received information, rather than having to utilise the physical MAC address and have some form of system after that to correctly divide up and assign the data obtained through a network connection. An obvious disadvantage of this however is when try to connect to a machine by using the MAC address it can be difficult when the virtual machine in question is shut-down or otherwise temporarily inaccessible, or when the MAC address is being used by a different machine.

As for how this can be a disadvantage for the performance of a networked multiplayer game, it can cause data to not be sent to the correct machine if the packets are relying on the MAC address handle rather than an IP address one or some other form of identification process. This in turn results in users at best losing the occasional set of data and at worst having their actions be completely mishandled, examples including a player walking forward instead of back, not reloading a weapon or choosing the wrong option on a menu.

It's also not a deterministic MAC address.

Question 02.B)

Subnetting works as follows:

Main network has X amount of bits.

Main network has too many machines for the network to assign addresses to.

A subnet takes N bits from the main network and uses them to create a smaller network to have machines assigned to it. E.G, 199.7.10.1

The subnet can support a number of machines inversely proportional to the number of subnets that exist. That is to say, the more subnets there are, the less hosts per subnet. The formula for calculating a subnet is $N^2 - 2$, whilst Hosts are $H - N$, followed by $h^2 - 2$.

2. b

	128	64	32	16	8	4	2	1
199.7.10.0	0	0	0	0	0	0	0	0

199.7.10.0 = 199 1 1 0 0 0 1 1 1

199 is Class B X

7	0	0	0	0	0	1	1	1
10	0	0	0	0	1	0	1	0

11000111 / 00000111 / 00001010 / 00000000

12 hosts = 14

4 bits reserved ✓

0 0 0 0

$4^2 - 2 = 14$

3

4 bits host ✓

4 bits Subnet

minimum for Subnet is 4

8 - 4 = 4
 Subnet Subnet host bits

$4^2 \geq$ host and subnet
 12 9

For calculations pertaining to this question, please view the answer book.

Question 3.A)

The term deadlock refers to when a thread is stuck waiting for a particular piece of data or a function to finish being used by another thread. This is typically caused when a function has run longer than anticipated, or another has completed faster. Either way, the thread that requires the in-use function or data is having to wait until it has been freed up before it can continue. In effect, imagine a system of wheels where they have to pass a stick from one wheel to another by rotating in such a way that the stick points to an opening on the next wheel. In the case of deadlock, the stick never faces the opening on the next wheel as the wheel has stopped turning.

Starvation meanwhile is when there are more threads than required for a particular process, so they sit around and don't contribute anything. Typically the way to handle this is to terminate or otherwise free the thread at the end of the process so that it can either be built again when needed or if its causing a deadlock it can free the data inside which can be incredibly dangerous as you can't be sure what state the thread was in when processing the function, I.E if it was handling a vector and reorganising data inside of it, then you can't be sure at where the data is at the time.

To alleviate deadlock conditions, one of the methods that can be employed is a lock and key system that operates on a timer, where-in a thread has X time to finish a process before it is handed over to another one. This ensures that data is constantly flowing rather than bottlenecking on one thread which can reduce the chances of deadlock and starvation taking place.

Another method to avoid deadlock is in a lock and key system where each function or type of data has an associated key that permits certain threads to work on it. This both frees other threads from the free-for-all situation that can occur whilst also ensuring that should a deadlock occur it affects a smaller number of threads.

It should be noted that sometimes a deadlock can be intentional. For example, if all of the calculations for X program need to be completed before proceeding. This occurs within graphics and physics calculations handled within CUDA on Nvidia graphics cards, as an example of real-world application.

Question 3.B)

Mutexes and semaphores are two ways of handling multithreading in terms of ensuring that there is minimal chance that threads run into one another or encounter deadlock.

Mutexes are effectively a pool of keys that are given out to threads that allows them to work on processes defined within the pool. Once a process is complete, the key is returned to the pool so that another thread can take over. Where this could be useful in a multiplayer game is when it comes to movement calculations. If any thread can grab the data for moving a player, it could be misinterpreted and executed in such a way that the player character exhibits strange movement patterns such as sliding right then forwards instead of stepping diagonally to the right. By locking this to a thread until the calculation is complete it avoids this problem from occurring.

Semaphores meanwhile work in a more rigid fashion, effectively to the degree of "Work on this until it is done". Where-as Mutexes can operate from a pool, semaphores are assigned to the thread and typically work in system critical areas where you cannot afford the process to be interrupted by another thread. They are thus utilised for more important areas of multithreading in multiplayer games such as handling the packets being obtained and sent to the players, calculations that can

Tamm

affect multiple players at once server side such as if part of the map is to break away into a bottomless chasm and exiting the game with the relevant memory clean-up.

Question 04.A)

The importance of consistency and responsiveness is that if a game you have developed takes a notable amount of time to conduct a simple action, such as move, then the players are going to quickly grow frustrated with it and no-longer play your game, tell their friends and other prospective buyers to not purchase the game and ultimately lead to a large loss of sale. ✓

For a game to be consistent it needs to work the same time, every time. If a player driving a car at 88 miles per hour hits a wall, they should not see some strange and different things occur every time. If they hit the wall and slide along it until they manage to regain control of the car the first time, then it should be like that all the time. If the car launches itself into the atmosphere the second time, phases through the wall the third, comes to a complete stop the fourth and so on then there is little consistency to be had.

Responsiveness in a multiplayer sense pertains more to obtaining "feedback" for doing something. If Robby in his robot shoots Paul in his car, but doesn't see anything happen to Paul for a few seconds before the car has a visible dent in the side door then the responsiveness is lacking. 6

To improve both the consistency and the responsiveness, the packets containing the relevant data need to be sent frequently and, fittingly, consistently. They should also be processed as fast as possible on the server or client side of the game, depending on how the game itself has been developed. An issue with this however is that it can require a fast internet connection to handle this transmission of data frequently, especially if the information being sent is large. ✓

Question 04.B)

Some approaches that could be taken to reduce the amount of data being sent in a networked multiplayer game include the following:

Have more of the data be contained client-side and calculated there, then the result sent to the server for processing rather than sending all of the information for the server to process. Examples of where this can be useful are in graphically derived calculations such as lighting and reflective rendering. Examples where this should not be implemented include the effect a player has on the world in relation to destructible terrain, which is to say that if a wall is broken in a certain way on the player's screen then it should be like that on every screen. 1

Question 04.C)

A technique that may improve responsiveness by offloading work to the player's machine is the handling of all non-essential calculations and processes to the client rather than the server. This results in a few performance increases due to the following:

By having processes such as lighting, rendering and sound handled by the client machine, it reduces the amount of data that needs to be sent over the network connection down from potentially all of the data that requires processing down to a few flags telling the client what to process. For example, rather than sending the entirety of a texture's brightness as a result of the sun hitting it, the server can send a bool and maybe some other bits saying that the sun can see that texture. In terms of gameplay, if the data for all Robots is kept on the client, then if Robby shoots at Ronny all the server has to do is pass messages around. Robby's computer has calculated it can't hit Ronny, so it tells the

server. The server then tells Ronny's computer it was shot at but wasn't damaged, so the visual effect of the lasers appears on Ronny's screen but nothing else really happens.

This leads to the second area of an increased performance, which results in improved responsiveness, which is that the results of the calculations are right in-front of the player rather than having to wait for a server to process them and send it to the client which then has to interpret and display the results. 2

A main drawback however to this technique is that it can start to increase the multiplayer game's file size on the client computer which can be an issue for people using small drive sizes.

Question 04.D)

Interpolation means that something is interlinked with another thing, whilst extrapolation means it isn't. Examples of interpolation include video game graphics on TVs and certain types of monitor that try to interweave two images together to create the illusion of fluid movement by breaking up the two images into lines and quickly layering the second image lines over the first.

For networking this could refer to when different types of data are in the same packet, such as the IP address of a client, the client's username, the client's password, the choice for their avatar and their favourite colour. 0.5

Question 04.E)

An approach that can be taken to alleviate the problem of "ghosting", the client's locations of all moving components being slightly behind that of the server's, is to transmit the packets to and from the client at a faster rate. This is done in modern video games such as Battlefield 4 where-in servers operate at what's called 30hz and 60hz, referring to how many times per second the server updates the clients on everyone's position. A downside to this is that it vastly increases the amount of traffic through a network connection and can negatively affect those with slow connections. It can also cause the server to work slower as more resources are dedicated simply to sending and receiving packets instead of processing functions and calculations.yuj6k