# Illinois Institute of Technology

# Advanced Operating System (CS-550)

## Written Assignment 2

**Submitted By:**

**Batkhishig Dulamsurankhor (#A20543498)**
**Nitin Singh (#A20516824)**
**Vaishnavi Papudesi Babu (#A20527963)**

**Instructor: Professor Ioan Raicu**
**TA: Mr. Lan Nguyen**
**TA: Ms. Sonal Gaikwad**
**TA: Mr. Adarsh Agrawal**

**1. (10 points)** In this problem you are to compare reading a file using a single-threaded file server and a multithreaded server. It takes 15 msec to get a request for work, dispatch it, and do the rest of the necessary processing, assuming that the data needed are in a cache in main memory. If a disk operation is needed, as is the case one-third of the time, an additional 75 msec is required, during which time the thread sleeps. How many requests/sec can the server handle if it is single threaded? If it is multithreaded?

**Answer:**

For the **single-threaded** server:

Cache hits: It takes 15 msec to process a request if the data is in cache (happens 2/3 of the time)

Cache misses: It takes 15 + 75 = 90 msec to process a request if a disk operation is needed (happens 1/3 of the time)

Weighted average: each request takes (2/3) * 15 msec + (1/3) * 90 msec = 40 msec

Server capacity: In 1 second, the server capacity in single-threaded server will be

1000 ms / 40 ms per request = **25 requests/sec**

For the **multi-threaded** server:

Assume disk waiting overlapped in multiple threads

Average: Each thread still takes 15 msec on average to process a request

Server capacity: In one second, the server capacity in multi-threaded server will be 1000 ms / 15 ms per request = **66 2/3 requests per second**

Assume multithreaded server with preemptive scheduler will be,

Average: Each thread takes 15msec on average to process a request

Weighted average will be (2/3)*0 + (1/3)*75 = 25 msec

Probability of n threads sleeping is (25/40)^n = (5/8)^n.

Server Capacity: In one second, the Server capacity will be **(1-(5/8)^n)*(1000/15) reqs.**

When n = 1, (1-(5/8)^1)*(1000/15) = 25 requests/sec.

When n = 4, (1-(5/8)^4)*(1000/15) = 56.49 requests/sec.

**2. (5 points)** Would it make sense to limit the number of threads in a server process? Explain your answer.

**Answer:**

- If there is no limit in the threads, as the threads increase, the load on CPU increases due to many reasons including context switching and contention of resources like memory.
- More threads does not guarantee a better performance.
- It could lead to worse performance issues and slowness.
- Having too many threads could starve the other processes in the CPU.
- Hence, limiting the number of threads is a good practice to ensure efficient utilization of resources.
- A benefit of limiting the threads is that applications will be developed to reuse the threads efficiently.

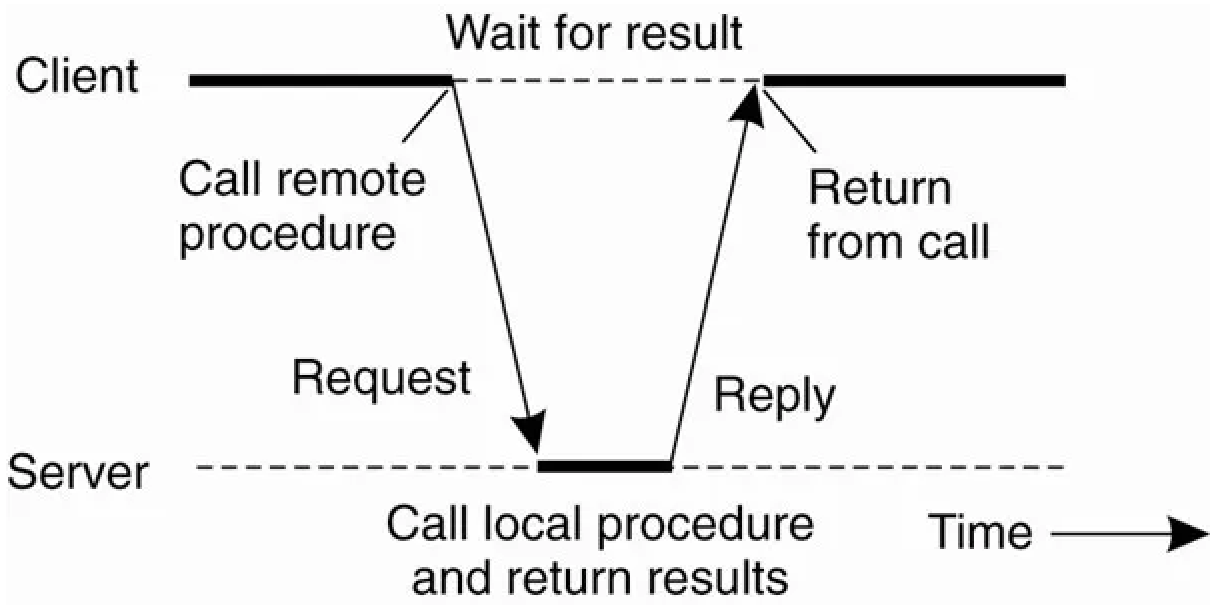
**3. (5 points)** Constructing a concurrent server by spawning a process has some advantages and disadvantages compared to multithreaded servers. Discuss 2 advantages and 2 disadvantages.
**Answer:**
Advantage 1: Process Isolation: Processes have separate address/memory space. So, if one process terminates abruptly or crashes  due to a reason, other processes will not be affected. This ensures high process isolation and fault tolerance.
Advantage 2: Better usage of multi-core CPUs: Processes can run in a truly concurrent environment compared to threads that are time sliced in multi-core CPU architectures as well.
Disadvantage 1: Higher overhead: individual processes creation and management is an additional overhead compared to threads that are lightweight. Context switching is also slower between processes.
Disadvantage 2: Inter-process communication:  Processes should use IPCs like sockets and pipes to communicate with each other as the processes have individual memory spaces, unlike threads that share common memory space in process

**4. (5 points)** Is a server that maintains a TCP/IP connection to a client stateful or stateless? Justify your answer.

A TCP/IP connection to a client is stateful because the transport layer is stateful. Every message is transmitted in the form of packets and the connection is maintained until all the packets are delivered and reassembled at the destination. TCP maintains state in the form of window size where the destination informs how much size of data it can receive at a time and source sends the data in chunks in the form of packets in that particular size. Source also maintains the sequence numbers in order to reorder them in the destination. It is also useful to identify the lost packets. The source waits for the acknowledgement (ACKs) of the packets sent and retransmits again if needed. This state allows TCP to be stateful and reliable.

**5. (5 points)** Describe how connectionless communication between a client and a server proceeds when using sockets.
A connectionless client-server communication works as below using sockets:
- A client creates a UDP socket and binds it with any available port
- A server creates a UDP socket with a specific port.
- Client sends a message to server on server's IP and port via UDP datagram.
- The message goes to the socket queue.
- The server receives incoming datagrams in a blocking read mechanism on the socket.
- Server reads the message in the datagram payload.
- Server processes the request and send back a response if required.
- Client performs a read on socket to receive response.
- This to and fro is repeated for all the messages.
- Each message is independent, there is no connection maintained between them.
- Client or the server could terminate independently without acknowledging each other.

**6. (5 points)** Does it make sense to implement persistent asynchronous communication by means of RPCs?



RPCs are frequently utilized for synchronous communications. This implies that a client will contact the server with a request, and the server will then await the reply.

However, using an RPC to establish persistent asynchronous communication might not be the best choice. For sustained asynchronous communication, other technologies, such as message queues or WebSocket, are more appropriate. Asynchronous applications benefit more from these technologies' efficient non-blocking and event-driven communication capabilities.

**7. (5 points)** With persistent communication, a receiver generally has its own local buffer where messages can be stored when the receiver is not executing. To create such a buffer, we may need to specify its size. Give an argument why this is preferable, as well as one against specification of the size.

Advantage of specifying buffer size:

1. Reduced latency: If a sufficiently big buffer is set up, the receiver can continue to receive messages even when the program isn't operating. This means that the sender need not wait for the recipient to be prepared before sending a message as a result we have reduced latency.
2. Simple to manage: You don't need to worry about the buffer's size because the system knows exactly how much of it there is.

Disadvantage of specifying buffer size:

1. Message can be lost:
   New messages will be misplaced if the buffer fills to capacity. If the sender sends messages more quickly than the recipient can receive them, new messages may be lost. If the recipient takes a long time to process new messages, they may potentially be lost.
2. Underutilization:
   The buffer could not be utilized as frequently as it should be if it is too large. System resources can be lost as a result of this.

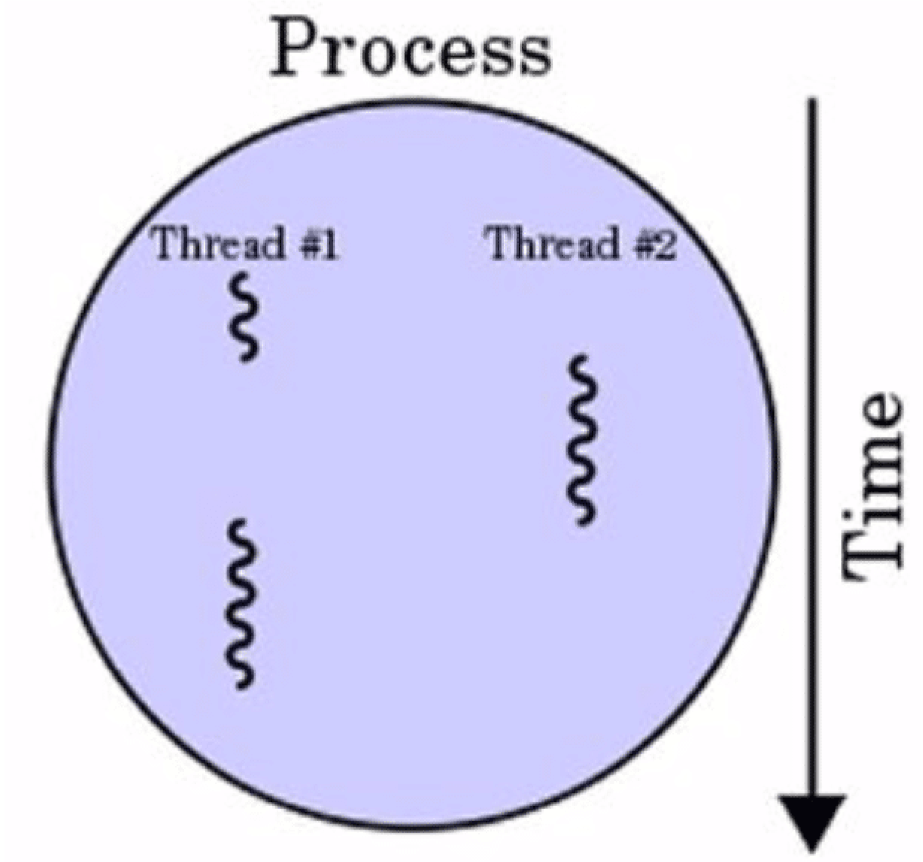
**8. (10 points)** When searching for files in an unstructured peer-to-peer system, it may help to restrict the search to nodes that have files similar to yours. Explain how gossiping can help to find those nodes. What is the worst-case scenario for locating the object in an unstructured overlay network in terms of number of nodes visited, where n is the number of nodes?

*How gossiping can help to find nodes:*
In an unstructured peer-to-peer network, gossiping is an efficient approach to locate files. All you have to do is tell your peers about the file you're looking for, and they'll tell their peers, and so on, until the message reaches the node that stores the file and it's sent to you. Of course, there's always the chance that the node containing the file isn't directly connected to you. In that situation, you must notify all nodes in the peer-to-peer network about the file you are looking for. This method may take some time, but it is preferable to doing nothing at all.

*Example:*
Assume there are 100 nodes in a peer-to-peer network. A node_A wishes to access a file on another node_B.
It asks its neighbors if they have access to the file, and if none of them have, it asks their neighbors.
The request is eventually distributed over the network. When node_B receives the request, it can send the file to node_A.

*Worst Case scenario:*
Assume the object is in an unstructured overlay network. If the file is on a node other than node_A, node_A must ask its neighbors for the file.
If no node has the file, it will request it from its neighbors. This will continue until node_A locates the file or reaches the network's end.
In the worst-case scenario, node_A must query all nodes in the network for the file before it can be located.
Example:
Assume an unordered overlay network with 100 nodes. One network node is looking for a file that is stored at node 99.
The node wants to find the file, therefore it will first ask its neighbors if they have it. If not, they will question their neighbors, and so on.
In the worst-case situation, the node must query all 99 nodes before locating the file.

**9. (5 points)** What is the difference between a process and a thread?



Process:
A process is a running program that has its own memory address space. Each process has its own version of the program's code, data and stack.
Thread:
Threads are lightweight processes that share the same memory address space as other threads within the same process. Threads can easily talk to each other and exchange data.

| Characteristic | Process | Thread |
|---|---|---|
| Memory Space | Has own memory space | Shares memory space with other thread in the same process |
| Isolation | Isolated from other processes | Not Isolated from other threads in the same process |
| Communication | Communication between the processes is complex | Communication between the threads is efficient |
| Resources | Process consumes more resource | Threads consumes lesser resource |
| Creation Time | Takes more time to create | Takes lesser time to create |
| Context Switching Time | Takes longer for context switching | Takes lesser time for context switching |

**10. (5 points)** What is one advantage of a non-blocking form of Send? What about one advantage of blocking communication?

Advantage of non-blocking send:
With non-blocking Send, the sender can continue to compute while sending data. This can help improve performance by freeing up the sender's CPU resources.

Advantage of blocking communication:
When sending data, blocking communication ensures that the data is delivered to the recipient. That's because the sender won't continue until the recipient confirms that they've received it.

Let's say you're sending a package to your friend.
You can do two things:

> Blocking send:
> Go to the post office to drop off the package and wait for it to be delivered. If you do this, you'll have to wait until the package is delivered before you can do anything else. This

means you'll be wasting your time and your CPU resources.

Non-blocking send:
If you don't have to wait for the package to be delivered, you can do other things while it's being delivered. You'll be able to use your time and CPU more efficiently.

Thus, there are situations where you need to use blocking communication. For example, you might want to send a critical piece of information that can't be lost, and you'll want to make sure it's delivered successfully. But in most cases, non-blocking communications is the better option.

**11. (5 points)** Give an example of an application that should use an unbuffered communication approach? Explain in detail why buffered communication would not be appropriate.

An example of an application that uses an unbuffered communication is an online gaming. In online gaming, especially in competitive ones, data must be delivered from one computer to another as quickly as possible. Low latency communication to a server is essential to provide players with seamless gaming experience. Even a little latency difference can have a huge impact on a player's performance.

If buffered communication is used in online gaming, real-time updates of actions in game can no longer be guaranteed. Therefore, a gaming experience will be unengaging.

**12. (5 points)** Explain how asynchronous communication allows for more scalable systems?

In asynchronous communication, a processor can work on a different task while waiting for an asynchronous response and processes it after it arrives. This greatly improves processor utilization. If a processor waits for a long request to respond and has queues of other requests and jobs on it, the processor will have so much more redundancy and the system won't be able to achieve concurrency.

**13. (10 points)** Remote procedure call (RPC), remote method invocation 2 points (RMI), and web services (WS) are all abstractions to allow inter-process communication across a network to access remote resources. Compare and contrast RPC, RMI, and WS.

Remote Procedure Call is supported on procedural programming and used to execute procedures on remote machines. It is OS dependent, meaning that different OSs have different software implementations of RP.

Remote Method Invocation is supported on any machine that has JVM installed and is dedicated to object oriented programming. Compared to RPC, it is efficient and the cost of development is cheaper. Both RPC and RMI are used to call remote procedures/methods like local ones.

Unlike RPC and RMI, web services is platform independent. In other words, any programming language or platform can be used to form web services in distributed computing. It communicates with other applications through protocols like HTTP and SOAP. Even thougth WS is more dynamic in terms of programming languages, RMI and RPC are more efficient and faster because they are more tightly coupled.

**14. (20 points)** Describe Moore’s Law in your own words. Do you expect it to continue indefinitely? If yes, justify your answer. If no, why not, and when do you expect it will end? Be as detailed as possible in your response.

Moore’s law states that the number of transistors in processors is doubled every two years. In other words, the computation power of computers will keep increasing exponentially and processors of a few years prior will be incomparable to the latest ones. Processors 4 years later will have 16 times more computational power than the current ones according to Moore’s law.

Although this trend has been valid from the 1970s until now, I don’t expect this trend to continue indefinitely. Firstly, exponential growth in the number of transistors will slow down because of obvious natural limitations, soon it will be hard to fit billions or even trillions of transistors in a same sized circuit board. It is true that on a physical level, the transistors are becoming smaller and reaching down to a few nanometers in size, but manufacturing even smaller transistors will be scientifically challenging and costly.

Secondly, the more processing power means the more power is required to run the processor and consequently it will generate more heat. Therefore powering processors and cooling them down is becoming more and more expensive, and it will be one of the important reasons that slows down this trend.

Lastly, to achieve higher computation power, we no longer need to build costly super processors. Instead the trend is moving towards distributed and parallel computing, in which big problems/jobs are divided into smaller subproblems/jobs so that multiple processors or machines can work concurrently and finish the task. Purchasing multiple moderate processors is much cheaper than buying a single processor with the same cumulative computation power. For these reasons, I think Moore’s law will slow down in the future.