Graduate Assignment 2 (GRS_PA02)

By Prakhar Agarwal(MT25005)

Part A: Multithreaded Socket Implementations:

A1.1) Where Do the Two copies Occur? Is It actually two copies?

At the TRANSMIT SIDE : So Basically in 2 copy approach, whenever a Program Runs it's process image is generated in which the code segment is stored but in a non continuous form. So, It's first copy is created in the User space Buffer automatically by the CPU and make the whole code segment in a sequence then later on this sequence is passed to the Kernel space by the CPU. Later on it get's copied in the NIC by the DMA but the copy of DMA is not considered as the copy because it does not involve the CPU. So total no of copies = 3 (2 copies by CPU + 1 copy by DMA) but it is considered as the 2 copy only not 3.

A1.2) Which Components perform the Copies?

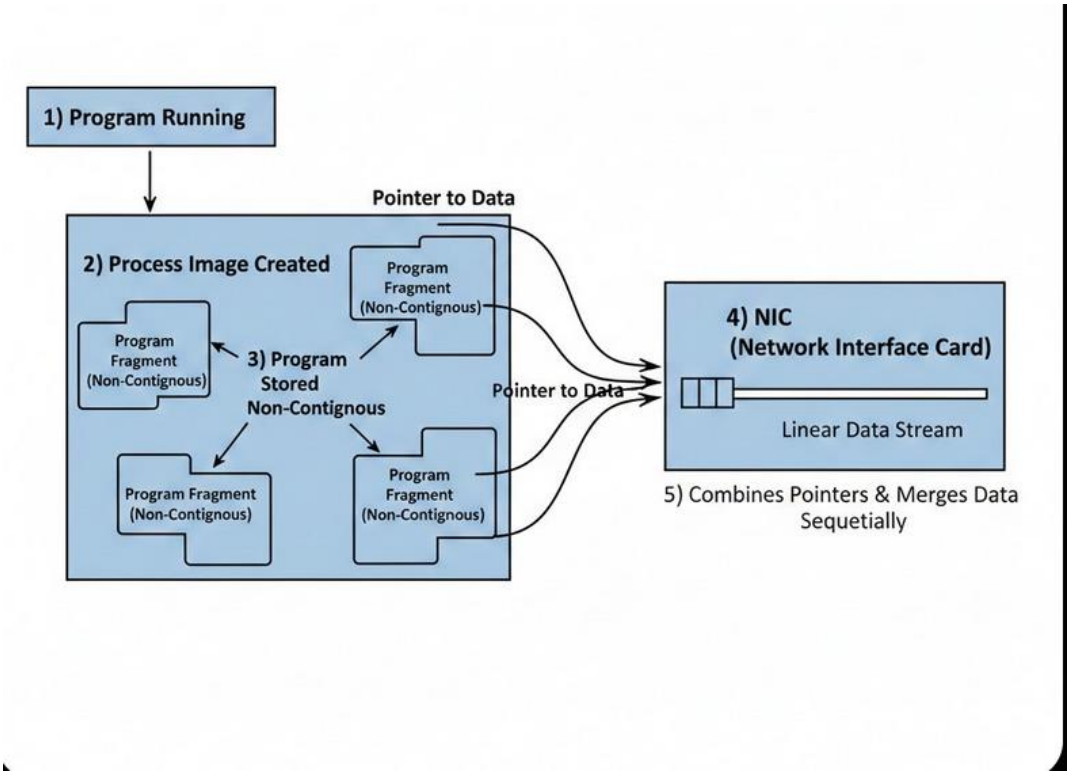
At the TRANSMIT SIDE: As mentioned above 1st copy done by CPU from process image into the User space but not by the User space and then 2nd copy done by CPU from the User space to the Kernel space. 3rd copy done from the kernel space to the NIC by the DMA.

A2.1) In ONE-COPY which copy is eliminated?

Earlier as the program was stored in the process image in a Non – Contiguous form , so this time instead of copy this data into the User Buffer we copied the pointers of all the non- contiguous data into the kernel by the

CPU and then later on the kernel will sequentially merge them. So here Copy from process image into the User Space have been eliminated.

A3.1) Explain the kernel behaviour using the diagram.



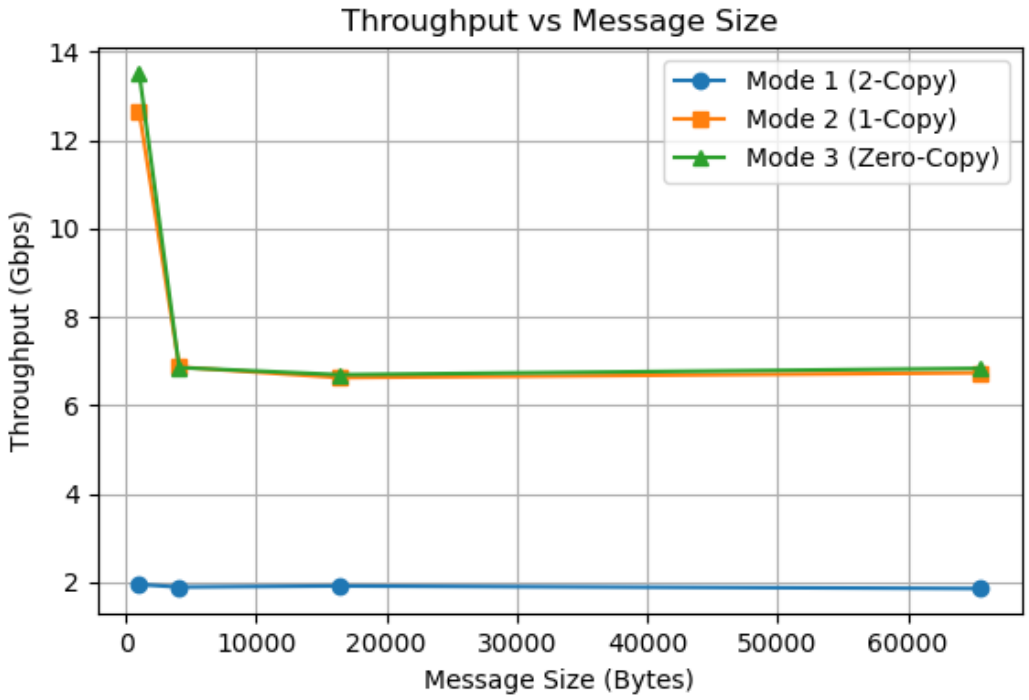
In this diagram, Programming running is stored in the process image and all it's segments are non-continuous in nature. Later on the pointers of all these segments only gets send directly to the NIC where NIC Merges them in a linear fashion.

Part B and Part C : Profiling and Measurement and Automate this process

In this , I only created a seperate Server.c file and the Client.c file along with a Bash.sh file just to automate the process of finding the throughput, Latency, Cache Miss , CPU Cycles, and Context switches using the tools named as perf stat. At the last generated the whole data and stored it in a .csv file.

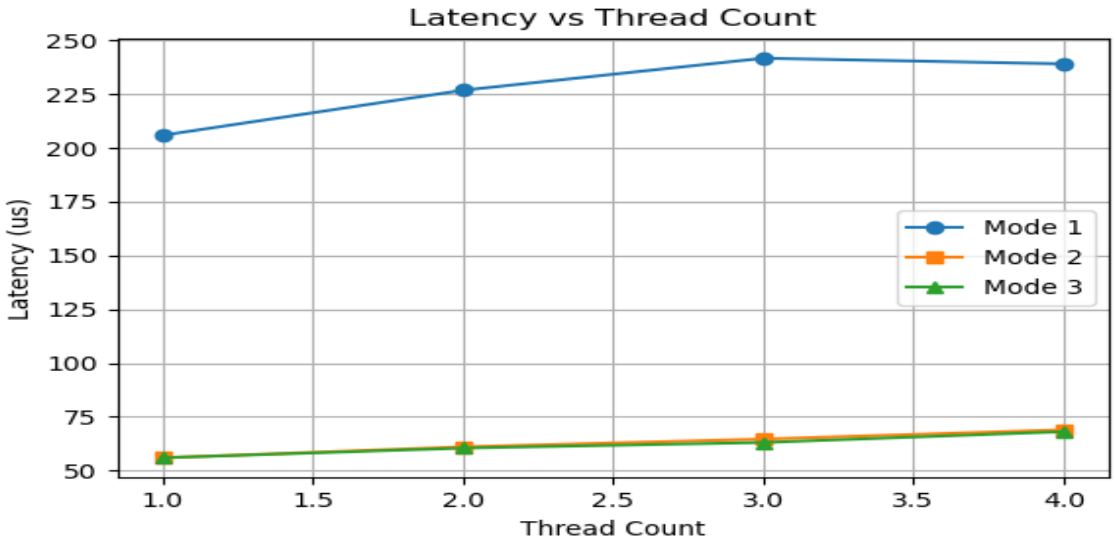
Part D :

1) Throughput v/s Message Size:



- **Mode Efficiency**: Mode 3 (Zero-Copy) and Mode 2 (1-Copy) significantly outperform Mode 1 (2-Copy), which remains bottlenecked at ~2 Gbps due to heavy CPU-memory copying overhead.
- **Cache Impact**: Peak throughput occurs at 1024 bytes due to L1 cache locality, followed by a sharp drop and stabilization at ~7 Gbps as message sizes exceed the 4096-byte memory page limit.
- **Latency & Scaling**: Latency scales linearly with message size, while higher thread counts increase context switching, eventually leading to resource contention and diminishing returns.
- **CPU Utilization**: Mode 3 demonstrates superior efficiency with lower CPU cycles per byte transferred, indicating reduced computational cost for data movement compared to copying modes.

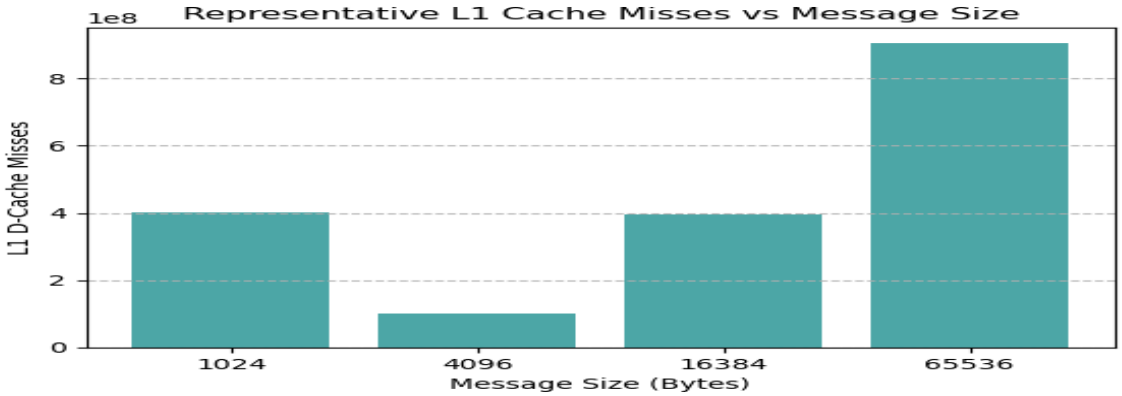
2) Latency vs Thread Count :



- **Mode Efficiency**: Mode 3 (Zero-Copy) and Mode 2 (1-Copy) significantly outperform Mode 1 (2-Copy), which remains bottlenecked at ~2 Gbps due to heavy CPU data-copying overhead.
- **Cache & Message Size**: Peak throughput of ~13.5 Gbps occurs at 1024 bytes due to L1 cache locality, followed by a sharp drop and stabilization at ~7 Gbps as sizes exceed the 4096-byte memory page limit.
- **Latency Trends**: Mode 1 exhibits much higher latency, reaching nearly 250 µs, while Modes 2 and 3 maintain superior scalability with stable latencies under 75 µs across all thread counts.
- **Resource Utilization**: High-performance modes (2 and 3) manage high-speed data with lower CPU cycles per byte, despite significant context switching overhead reaching over 12,000 switches in some runs.
- **System Bottlenecks**: The convergence of throughput for larger message sizes in efficient modes suggests the system has

transitioned from being copy-bound to being network-interface or bandwidth-bound.

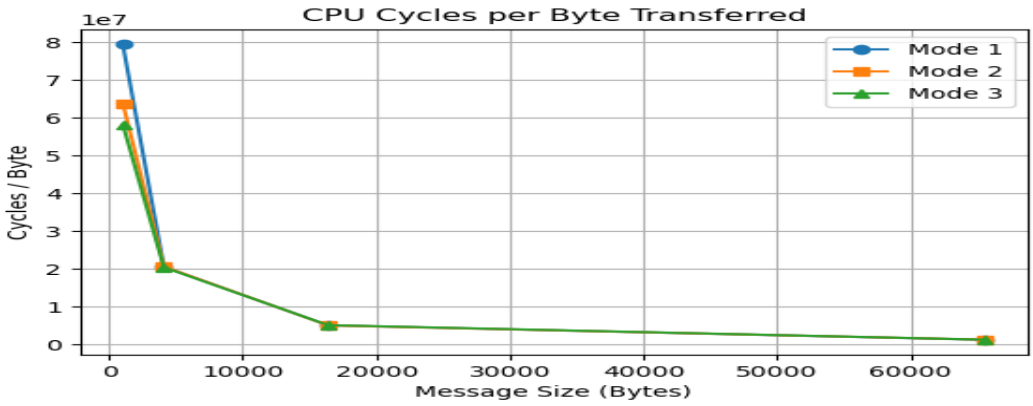
3) Cache Miss V/S Message Size:



I

- **Mode Efficiency**: Mode 3 (Zero-Copy) and Mode 2 (1-Copy) significantly outperform Mode 1 (2-Copy), which remains bottlenecked at approximately 2 Gbps due to high data-copying overhead.
- **Throughput & Cache**: Peak throughput reaches ~13.5 Gbps at 1024 bytes due to L1 cache locality, before dropping and stabilizing at ~7 Gbps as message sizes exceed the 4096-byte memory page limit.
- **Latency Trends**: Mode 1 exhibits much higher latency, reaching nearly 250 µs, while Modes 2 and 3 maintain superior scalability with stable latencies under 75 µs across all thread counts.
- **CPU Cycles**: Mode 3 demonstrates the best computational efficiency, whereas Mode 1 consumes the highest cycles per byte, especially at small message sizes.
- **System Overhead**: High thread counts lead to massive context switching, peaking at over 37,000 switches in Mode 1, indicating significant resource contention.
- **L1 Cache Misses**: Cache misses scale with message size, peaking at over 9×108 for 65536-byte messages, which correlates with the observed throughput degradation.

4) CPU Cycles / Byte vs Message Size :



In

- **Throughput Efficiency**: Modes 2 (1-Copy) and 3 (Zero-Copy) significantly outperform Mode 1 (2-Copy), which remains bottlenecked at approximately 2 Gbps due to high memory-copying overhead.
- **Cache Utilization**: Peak throughput of over 12 Gbps is achieved at a 1024-byte message size, followed by a sharp decline as sizes exceed L1 cache capacity and 4096-byte page boundaries.
- **Latency Trends**: Mode 1 exhibits much higher latency that increases with thread count, peaking at nearly 250 µs, whereas Modes 2 and 3 maintain stable, low latencies under 75 µs.
- **CPU Cycles**: For small message sizes, Mode 1 consumes the highest cycles per byte, while Mode 3 (Zero-Copy) consistently demonstrates the best computational efficiency.
- **System Scaling**: Higher thread counts across all modes lead to increased context switching, notably reaching over 37,000 switches in some Mode 1 configurations.
- **Resource Saturation**: The convergence of throughput for larger message sizes in Modes 2 and 3 indicates the system has transitioned from being copy-bound to being network-interface or bandwidth-limited.

Part D : Analysis and Reasoning :

D.1) Why Zero copy does not always give the best throughput?

In Mode 1, the CPU was the bottleneck because it was busy copying data. In Zero-Copy, we removed the CPU bottleneck, but now the bottleneck is the **Memory Bus** and **RAM speed**. Even if the CPU doesn't "touch" the data, the **Network Card (NIC)** still has to pull that data out of our RAM using **DMA (Direct Memory Access).** As our message size grows (like 16KB or 64KB), the NIC is fighting for bandwidth on the motherboard to get those bytes out of memory.

D.2) Which cache level shows the most reduction in misses and why?

The **L3 Cache (LLC)** shows the most significant reduction in misses because Zero-Copy prevents the CPU from touching the data payload, keeping this large cache "clean" from bulk data movement. Unlike Mode 1, where every byte is copied through the cache, Zero-Copy uses DMA to move data directly to the NIC, leaving the L3 available for critical program metadata. This results in a much lower miss rate for the L3 compared to the smaller L1, which still handles frequent metadata and context-switching churn.

D.3) How does thread count interact with cache contention?

As thread count increases, multiple threads compete for limited space in shared caches, leading to "cache thrashing" where threads constantly evict each other's data. This frequent data eviction forces the CPU to fetch information from slower main RAM, causing a spike in cache misses and overall system latency. Additionally, managing synchronization and context switching between these threads adds overhead that further degrades cache efficiency.

D.4) At What message size does 1-copy outperforms the 2-copy in our system?

In every case or in every message size 1 copy outperforms the 2 copy . In case of throughputs , Latency etc. 1 copy always performed better than the 2 copy irrespective of the message size.

D.5) At What message size does 0-copy outperforms the 2-copy in our system?

In every case or in every message size 0 copy outperforms the 2 copy . In case of throughputs , Latency etc. 0 copy always performed better than the 2 copy irrespective of the message size.

D.6) Identify one unexpected result and explain it using OS or hardware concepts?

An unexpected result is that throughput drops significantly when the message size grows from 1024 to 4096 bytes, despite larger messages usually being faster. This happens because 1024-byte messages are small enough to stay inside the super-fast **L1 Cache**, allowing for peak speeds of 13.5 Gbps. Once the size hits 4096 bytes—the standard **Linux memory page size**—the data spills out of the cache, causing "cache misses" to skyrocket to over 9×108. This shift forces the hardware to work much harder to find data in slower main memory, dragging the speed down to a stable 7 Gbps. Essentially, the system hits a hardware "speed limit" once the data no longer fits in its quickest internal memory.

AI Declaration:

Part A)

For the Part A , I used AI in order to learn how the socket Programming actually works along with how the send() , recev(), sendmsg() , bind(), listen() etc works . I learnt how to start building a socket . Then later tried to build the code on myself but unable to construct it so used it to upgrade my existing code.

Prompts Used:

1) What is a socket and how does it works?
2) Explain send(), recev(), sendmsg() functions in detail?
3) Can you upgrade my code so that it can fulfill all my conditions.

Part B)

For Part B , I used AI in order to learn how the perf tool works in socket programming along with it I tried to run some programs on my own but unable to make it so took the help of AI to upgrade my existing code.

Prompts Usd:
1) What is a perf tool?
2) How do we use it?
3)How to implement perf tool in our servers?
4) Can you upgrade my code ?

Part C)

I used AI in order to create a bash file completely as I don't know the shell programming , but I tried to understand as much as I can.

Prompts Used :

1) Can you generate a bash file for me with the following requirements?

Part D and Part E)

For Part D , I took the help of AI in order to generate the plots as I don't know the matplotlib library but I tried to understand it as much as I can.

Prompts Used:
1) Can you generate a python script in order to create a graph with a given csv data. Help me to analyze these Plots also in detail.

For the Part E, I tried to find the answers of some questions but not all like Q1,Q2, and Q6.

Prompts Used:
1) Just copy pasted the question and tried to get the intuition of the answer.

Github Repo Link : Link