

## PA02: Analysis of Network I/O primitives using “perf” tool [31 points]

Deadline: **February 7, 2026**

### Important Note:

- Use of AI to generate code is allowed. However, you should be able to understand every line of code and answer during your Viva. Provide the prompts used while using GenAI for code, analysis, etc.
- **If AI is used for any aspect, you need to declare the components that are AI-generated. Failure to do so will result in a plagiarism penalty. Merely writing “AI used” won’t work, be very detailed in the report that where you are using & how you are using**
- The code should be modular, well-commented, and structured (well-aligned). This non-functional aspect carries marks.
- Add your GRS assignment to the public github repo created for PA02
- Read the deliverables and assignment statement carefully before starting the assignment.

### Objective

The goal of this assignment is to experimentally study the cost of data movement in network I/O by implementing and comparing:

- Standard two-copy socket communication
- One-copy optimized socket communication
- Zero-copy socket communication

You will implement **multithreaded client-server C programs**, profile them on your **own machine**, and **analyze micro-architectural effects such as CPU cycles and cache behavior**.

### Part A: Multithreaded Socket Implementations

You must implement a **TCP-based client-server application** (**run client and server on separate namespaces** (**VM will not work**)) with the following properties:

#### Common Requirements (All Versions)

- Server: **[3]**
  - Accepts multiple **concurrent clients**
  - Uses **one thread per client**
  - Transfers fixed-size messages repeatedly
  - The message sent by the server is a structure comprising **8 dynamically allocated string fields**
    - Heap-allocated buffers (**malloc**) for each message field
- Client: **[2]**
  - Sends data continuously **for a fixed duration**
- Message sizes:
  - Must be **parameterized at runtime**
- Thread count:

- Must be **parameterized at runtime**

### **A1. Two-Copy Implementation (Baseline)**

Implement a baseline socket program using: **[1]**

- `send()` / `recv()` socket primitives

**You must explain in your report:**

- Where do the two copies occur? Is it actually only two copies?
- Which components (kernel/user) perform the copies?

### **A2. One-Copy Implementation**

Implement a modified version that **reduces one copy**: **[2]**

- Use `sendmsg()` with pre-registered buffer

**You must explain **[1]****

- You must explicitly demonstrate **which copy has been eliminated**.

### **A3. Zero-Copy Implementation**

Implement a zero-copy version using: **[2]**

- `sendmsg()` with `MSG_ZEROCOPY`

**Requirements:**

- You must explain kernel behavior using a diagram **[1]**

## **Part B: Profiling and Measurement**

You must collect **quantitative measurements** for each implementation. **[2.5+2.5]**

### **Metrics to Measure**

Metric	Tool
Throughput (Gbps)	Application-level
Latency ( $\mu$ s)	Application-level
CPU cycles	<code>perf stat</code>

Cache misses (L1, LLC) `perf stat`

Context switches `perf stat`

Measurements must be taken for:

- At least **4 distinct message sizes**
- At least **4 thread counts**

## **Part C: Automated Experiment Script**

Write a **bash script** that: **[4]**

1. Compiles all implementations
2. Runs experiments across:
  - Message sizes
  - Thread counts
3. Collects profiling output automatically
4. Stores results in CSV format

**Constraints:**

- No manual intervention allowed after the script starts
- Script must re-run experiments cleanly
- Output filenames must encode experiment parameters

## **Part D: Plotting and Visualization**

Using **gnuplot / matplotlib / R (your choice)**: **[4]**

- Plot throughput vs message size
- Plot latency vs thread count
- Plot cache misses vs message size
- Plot CPU cycles per byte transferred

Each plot must:

- Clearly label axes
- Include legends
- Mention system configuration

## **Part E: Analysis and Reasoning**

Answer the following **questions**: [6]

1. Why does zero-copy not always give the best throughput?
2. Which cache level shows the most reduction in misses and why?
3. How does thread count interact with cache contention?
4. At what message size does one-copy outperform two-copy on *your* system?
5. At what message size does zero-copy outperform two-copy on *your* system?
6. Identify one unexpected result and explain it using OS or hardware concepts.

## Deliverables:

- Add the bulleted items to a folder named <roll\_num>\_PA02, and upload the compressed file, <roll\_num>\_PA02.zip
  - Source code (e.g., C code, bash scripts, etc. )
  - A Makefile for compiling the program.
  - Raw CSV data containing the measurements
  - Plots (PDF/PNG)
  - Documentation (README) explaining the implementation and usage.
  - Report comprising (a) screenshots, plots, and analysis for each question, (b) AI usage declaration, and (c) Your github repo URL
- Upload the above to the github repo with the folder name “GRS\_PA02”(not the zip, but the folder)

## Note -

1. **Every** file (except Readme & Makefile) must have a name like MT25xxx\_Part A\_Subpart(if any)\_Sub-subpart(if any)\_NameOfFile.extsnion(.c/.h/.pdf/.py/.CSV etc). It is the same **as it was done** in assignment 1. Do mention client file or server file wherever you implement like MT25001\_Part\_A1\_Client.c or MT25001\_Part\_A1\_Server.c
2. The naming convention of the zip must be <roll\_num>\_PA02.zip. GRS\_PA02 or 25081(*instead of MT25081*)\_PA02 etc will lead to a **3%** penalty.
3. Submission must be on **both** google classroom and on github. Only a report submitted on google classroom & code on github, whether timely done, will lead to a **3%** penalty.
4. Merely writing “AI used” in the report won’t suffice. It is asked to mention the “**components**” (read instructions at the top) in which-which part and where you used the LLM tool. Do mention the prompts also. **If not done then a 5% penalty will be levied and a plagiarism case will be embarked.**
5. **No PNG/JPEG etc files of plots must be submitted.** The plots should be present in the report **only**.
6. **For generating the plots, use ‘matplotlib’ only** (.py or .pynb etc). The values must be **hardcoded** in the python script. **Don’t supply the CSVs to generate the plots.** For example if a CSV has a column say ‘x’ having readings (based on the task done) as 1,278,62,89,... in the CSV, then these values must be in array format in the python files

like  $x = [1, 278, 62, 89, \dots]$  and then you need to plot graphs using this 'x'. You need to run this matplotlib file to generate the plots during the demo. It must match with plots present in the report. Failing to do this will **lead to 0 marks** in the components where the plot is needed.

7. The zip should **ONLY** contain code files, python plot files, Readme, Makefile, CSVs, report (**all must be named in the prescribed manner only**). **No graph plots** should be there. **No subfolders** should be there, only 1 folder having the listed files must be zipped. Failing to do so will levy a penalty of **1%**. **No binary files must be present in the zip or on github, else 0 will be awarded.**
8. Only a 'zip' file must be submitted. If 'rar' or any file other than 'zip' extension is submitted, then **0 marks** will be awarded.
9. The github repo must be **public, not private**.
10. The Readme & Makefile **must have your roll no.** as a comment at the very beginning (starting) of the file.