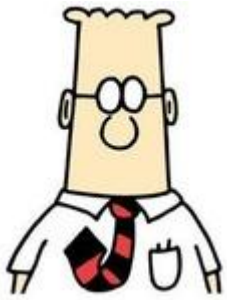

# GROUP PROJECT

**SysMonitor++ — Linux System Resource Monitoring Tool**

| | |
|---|---|
| **Project Name** | **:** System Programming Project |
| **No of group members** | **:** 4 students |
| **Release Date** | **:** Monday, 10th November 2025 (Week 6) |
| **Due Date** | : Friday, 19th December 2025, (Week 11) |
| **Weight** | : **20%** |
| **Hardware & Software Requirements** | : Access to terminal (in Ubuntu or any Linux distro<br>: text editor (vi, nano, etc)<br>: Access to **Ubuntu/Linux terminal** (physical or virtu<br>Text editor (e.g., vi, nano, gedit)<br>**GNU glibc** and **gcc compiler**<br>GitHub Desktop or CLI for version control |



## SCENARIO

You are a junior system programmer at a tech company in Johor Bahru.

Your manager calls you in:

**Manager:** "Our servers are running slow, and I need a simple monitoring tool that shows real-time CPU, memory, and process usage. I don't want to install heavy tools like *htop*, I want something written in C using system calls. Can you build that?"

You agree — and you'll name it **SysMonitor++**.

# TASK A: CODING TASK

Program Name: sysmonitor.c

Main Objective: Develop a command-line system monitoring tool using Linux system calls and /proc filesystem.

**1. Main Menu System**

Provide a simple menu allowing user selection:
1. CPU Usage
2. Memory Usage
3. Top 5 Processes
4. Continuous Monitoring
5. Exit

**2. CPU Usage Information**

Read data from /proc/stat using system calls (open, read, close). Calculate CPU usage percentage and display output.

**3. Memory Usage Information**

Read /proc/meminfo to show total, used, and free memory.

**4. Top 5 Active Processes**

Traverse /proc/ to read each PID's /stat and /comm, then display top 5 CPU consumers.

**5. Continuous Monitoring Mode**

Command: ./sysmonitor -c 2 (refresh every 2 seconds). Clear screen before each update.

**6. Logging**

Save snapshots in syslog.txt with timestamps.

**7. Signal Handling**

Capture SIGINT (Ctrl+C) to save logs and exit gracefully.

**8. Error Handling**

Display clear error messages using perror().

Suggested Functions:

```
void getCPUUsage();
void getMemoryUsage();
void listTopProcesses();
void continuousMonitor(int interval);
void handleSignal(int sig);
```

# TASK B: TESTING AND EVALUATION

All implemented features must be compiled, executed, and tested successfully from both the menu-driven mode and the command-line (parameter) mode.

Each test case must be documented with screenshots showing compilation, execution, output results, and log entries.

## 1 Compilation and Build Test

Before testing the program features, verify that your source code compiles correctly without errors. Use the following checklist to document your compilation process:

| Item | Requirement |
|---|---|
| **Command** | gcc sysmonitor.c -o sysmonitor |
| **Expected Result** | No compiler warnings or errors. |
| **Evidence** | Screenshot showing successful compilation output. |
| **Error Handling** | If compilation errors occur, show how they were corrected (before/after screenshots). |

## 2 Execution Tests – Menu Mode

**Command:**
./sysmonitor

**Expected Flow:**

1. Program displays the main menu with numbered options:
    1. CPU Usage
    2. Memory Usage
    3. Top 5 Processes
    4. Continuous Monitoring
    5. Exit
2. User inputs a choice (1–5).
3. Corresponding function executes and displays formatted output on screen.
4. Each operation result must also be logged into syslog.txt with timestamp.

**What to Include in Report:**

- Screenshot of main menu display.
- Screenshot of each option's result (CPU %, Memory info, Top 5 processes).
- Snapshot of appended log entries in syslog.txt.

## 3 Execution Tests – Command-Line Mode

The program must support **non-interactive execution** using flags and parameters.

This validates your command-line argument handling and function modularity.

| Example Command | Purpose | Expected Output |
|---|---|---|
| ./sysmonitor -m cpu | Display CPU usage only. | CPU usage % printed and saved to log. |
| ./sysmonitor -m mem | Display memory usage only. | Memory info (Total, Used, Free). |
| ./sysmonitor -m proc | List top 5 active processes. | Table with PID, Name, CPU%. |
| ./sysmonitor -c 2 | Continuous monitoring every 2 seconds. | Refreshing display + periodic log entries. |

**Verification Steps:**

1. Run each command individually.
2. Observe and record terminal outputs.
3. Check syslog.txt for correct, timestamped entries.
4. Validate that Ctrl + C triggers graceful exit message:
5. Exiting... Saving log.

**Evidence Required:**

- Screenshots for each command showing terminal output.
- Screenshot of syslog.txt with corresponding timestamps.

## 4 Signal Handling Test (Ctrl + C)

Test how your program responds to a keyboard interrupt (SIGINT).
This ensures that the program exits safely and properly records the final log entry.

| Action | Expected Behaviour |
|---|---|
| Press **Ctrl + C** during any mode | Program catches SIGINT, displays “Exiting... Saving log.”, writes final entry into syslog.txt, and terminates cleanly. |
| Verify Log | Last line in syslog.txt includes current timestamp and “Session ended” or similar marker. |

Include a screenshot of the signal interception and final log content.

## 5 Error-Handling and Edge-Case Tests

| Test Case | Expected System Response |
|---|---|
| Missing argument → ./sysmonitor -m | Display “Error: missing parameter. Use -m [cpu/mem/proc]”. |
| Invalid option → ./sysmonitor -x | Display “Invalid option. Use –h for help.” |
| No permission to read /proc | Display error via perror() message. |
| Log file not found | Program automatically creates new syslog.txt. |

Include screenshots showing how the program handles at least **two error cases**.

## 6 Log Verification

Open syslog.txt after several runs and confirm:

- Entries are **chronologically ordered**.
- Each entry includes **timestamp, CPU %, Memory usage**, and **mode (menu or CLI)**.
- The log file is **appended**, not overwritten.

Include 3–4 sample lines of your log file in the report.

## 7 Summary Table for Report

Students should summarize results in a table like this:

| Test ID | Command/Scenario | Expected Output | Actual Output | Status (Pass/Fail) |
|---|---|---|---|---|
| T1 | gcc sysmonitor.c -o sysmonitor | Compilation success | Success | ✅ |
| T2 | ./sysmonitor > CPU option | Display CPU % | Matches | ✅ |
| T3 | ./sysmonitor -m proc | Top 5 processes list | Matches | ✅ |
| T4 | Ctrl +C | Graceful exit + log saved | Matches | ✅ |

**Deliverables for Task B:**

1. Screenshots of compilation and every tested mode.
2. Snippets of output (terminal + log).
3. Table summarizing all test results.
4. Explanation discussing test coverage and program reliability.

# TASK C: REFLECTION

Write **your reflection** addressing the following:

1. Which AI tools (ChatGPT, Gemini, Copilot, etc.) you used.
2. What kind of help they provided (e.g., idea generation, debugging).
3. Which tasks were fully manual.
4. How using AI affected your understanding of system programming.
    a. Attach **screenshots of your AI interaction** in the Appendix.
5. List all group members and describe their individual tasks.
6. Include screenshots of your discussion using messaging app such as whatsapp (telegram, etc). In your opinion, what is the best method to coordinate and communicate when it comes to group projects?

# TASK D: USING GITHUB

GitHub Repository

- Repository Setup: Each group must create a GitHub repository for their project.
- Collaboration: Group members are encouraged to use GitHub for version control and collaborative coding.
- Commit History: Regular commits with meaningful messages should be made to demonstrate the development progress.

Each group must create a private/public repo titled:
TMN4133-GroupXX-SysMonitor

Include:

- Regular commits with meaningful messages
- README.md describing compilation and usage steps
- Screenshot of commit history in report

# TASK E: PROJECT PRESENTATION

- Record the group presentation using any suitable software.
- Prepare a slide maximum of 30 minutes presentation including demo session.
- Each group member has to present to obtain marks.
- Upload your recording to Youtube or Onedrive and share the link in submission.

# OTHER REQUIREMENTS

## 1. REPORT GUIDELINES

**Font:** Times New Roman, size 12, single spacing
**Include:**

1. **Cover Page (Faculty Format)**
2. **Introduction**
    - Project overview, objectives, and tools used
3. **Task A: Coding**
    - Explanation of each module
    - Properly formatted source code snippets
    - Screenshots of compilation & execution
4. **Task B: Testing Results**
    - Sample outputs and log excerpts
5. **Task C: Reflection on AI**
6. **Task D: GitHub Collaboration**
    - Repo link and screenshots
7. **Task E: Presentation Summary**
8. **Conclusion**
9. **Appendix:**

- AI chat screenshots
- Proof of collaboration (WhatsApp/Telegram)

## 2. SUBMISSION

Please submit the following items through the submission link provided through the course cube in eLEAP by the due date and time.

What needs to be submitted?

Submit through eLEAP with the following structure:

```
TMN4133SP-GroupXX-SysMonitor.zip
│
├── report.pdf
├── sysmonitor.c
├── syslog.txt
├── presentation_link.txt
├── README.md
```

## 3. GRADING RUBRICS

The group project carries 20% weight (marks) for the course and is divided as follows:

| Component | Description | Marks |
|---|---|---|
| **Report Format & Quality** | Structure, clarity, completeness | 3 |
| **Language & Flow** | Grammar, technical clarity | 1 |
| **Task A – Coding Implementation** | System calls, logic, menu, functionality | 7 |
| **Task B – Testing & Execution** | Successful compilation, test outputs | 2 |
| **Task C – AI Reflection** | Depth of reflection & screenshots | 2 |
| **Task D – GitHub Usage** | Commit history, teamwork evidence | 2 |
| **Task E – Presentation** | Demo quality & individual participation | 3 |
| | **Total** | **20 marks** |

.: End of Project Specification :.