A Minor Proposal on

NixMon: Real-Time System Monitor

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Software Engineering

under Pokhara University

Submitted by:

Abhishek Man Basnet (211703)

Under the supervision of

Er. Madan Kadariya

Date:

February 4, 2026

Department of Software Engineering

NEPAL COLLEGE OF INFORMATION TECHNOLOGY

Balkumari, Lalitpur, Nepal.

Abstract

Standard socket programming projects often focus on simple chat applications, which, while educational, are ubiquitous and limited in scope. This project proposes **NixMon**, a real-time remote system monitoring tool designed to provide system administrators and users with live visibility into Linux server performance. Unlike static reporting tools, NixMon utilizes a client-server architecture where a lightweight server application reads kernel statistics directly from the `/proc` filesystem (including `/proc/stat`, `/proc/meminfo`, and `/proc/loadavg`). The collected data is streamed via TCP sockets to a connected client, which parses and visualizes the information in a live-updating dashboard. By implementing concurrency handling using `poll()` or `select()`, the system ensures efficient data transmission and responsiveness. NixMon demonstrates core networking concepts such as TCP/IP communication, I/O multiplexing, and system-level data parsing, offering a relevant and distinct alternative to traditional chat applications.

Keywords: System Monitoring, Socket Programming, TCP/IP, Linux Kernel, Real-time Dashboard.

Contents

List of Abbreviations

CPU	Central Processing Unit
I/O	Input/Output
JSON	JavaScript Object Notation
RAM	Random Access Memory
TCP	Transmission Control Protocol
UI	User Interface
GUI	Graphicsl User Interface
TUI	Text-Based User Interface
CLI	Command Line Interface

1. Introduction

In the realm of network programming education, chat applications are the standard submission. While they demonstrate basic connectivity, they often lack distinctiveness and real-world application utility. **NixMon** is proposed as a distinct, real-time system monitoring tool that utilizes the same underlying socket concepts—TCP, concurrency, and I/O handling—but applies them to a practical system administration problem.

NixMon operates on a Client-Server architecture. The **Server** component runs on a host Linux machine (e.g., Fedora), where it interfaces directly with the Linux kernel's pseudo-filesystem (`/proc`) to gather vital statistics without requiring root privileges. The **Client** component connects remotely or locally, receiving a stream of data which is then parsed and rendered into a live dashboard. This project shifts the complexity budget from building complex UI logic to robust networking and system-level data parsing, making it an ideal candidate for demonstrating network programming proficiency.

2. Problem Statement

Effective system administration requires real-time visibility into server health. While enterprise tools exist, they are often heavy, resource-intensive, or require complex configuration. Conversely, student projects in network programming frequently saturate the ”Chat App” genre, which, while functional, fails to stand out or challenge the developer with diverse data handling requirements. There is a need for a lightweight, custom monitoring solution that:

- Demonstrates mastery of socket programming and concurrency beyond simple text messaging.
- Provides actionable, real-time insights into system performance (CPU, RAM, Load).
- Operates efficiently on Linux systems using standard kernel interfaces.

NixMon addresses this by providing a ”Visual Demo” that is immediately impressive and technically grounded.

3. Project Objectives

- **Develop a Server Module:** Create a server application that reads system statistics from `/proc/stat`, `/proc/meminfo`, and `/proc/loadavg`.
- **Implement Concurrency:** Utilize `poll()` or `select()` to handle multiple client connections simultaneously.
- **Data Streaming:** Establish a TCP connection to stream JSON or delimited data updates every 1-2 seconds.
- **Develop a Client Dashboard:** Build a client-side interface that parses the incoming stream and displays live-updating statistics using ANSI escape codes or ncurses.

4. Significance of the Project

NixMon offers a refreshing alternative to the common chat application project while covering the same fundamental concepts of TCP, concurrency, and I/O. Its significance lies in its:

- **Educational Value:** It forces engagement with the Linux file system architecture and binary/structured data transmission.
- **Practical Utility:** It mimics actual sysadmin tooling, providing a foundation for understanding how tools like `htop` or `glances` operate under the hood.

5. Scope & Limitations

5.1 Scope

- **Operating System:** The server side is designed specifically for Linux (tested on Fedora) due to reliance on the `/proc` filesystem.
- **Metrics:**
  - CPU Usage (calculated from `/proc/stat`).
  - RAM Usage (Total/Free/Available from `/proc/meminfo`).
  - Load Averages (1/5/15 min from `/proc/loadavg`).
  - System Uptime (`/proc/uptime`).
- **Networking:** Implementation of raw TCP sockets with concurrency support.

5.2 Limitations

- The server is not portable to non-Linux operating systems (Windows/macOS) without significant modification (due to `/proc` dependency).
- The current proposal focuses on monitoring; active system management (killing processes, restarting services) is outside the initial scope.
- Alert thresholds and historical data visualization are considered stretch goals and may not be in the initial prototype.

6. Literature Review

To establish a benchmark for terminal-based system monitoring, two prominent open-source tools were analyzed. These tools represent the standard for performance, usability, and visual presentation in Command Line Interface (CLI) environments.

6.1 htop

htop is widely regarded as the industry-standard interactive process viewer for Unix systems, designed as a superior alternative to the legacy `top` utility. Written in C and built upon the `ncurses` library, it transforms the terminal into a functional Text-Based User Interface (TUI). Unlike its predecessor, *htop* provides a scrolling list of all running processes and utilizes visual bar gauges to represent CPU and memory usage per core.

Its significance to this project lies in its efficiency; it demonstrates how to parse the `/proc` filesystem rapidly while maintaining a low resource footprint. Key features relevant to our research include its handling of user input (allowing users to kill or ”renice” processes without typing PIDs) and its use of color-coding to convey system health instantly.

6.2 btop

btop represents the modern evolution of system monitoring, prioritizing high-performance visualization and aesthetics. Written in C++, it leverages modern terminal capabilities such as TrueColor support and mouse interaction to create a dashboard that rivals Graphical User Interface (GUI) applications.

btop is distinct for its use of dynamic, historical graphing for network traffic, disk I/O, and CPU frequency. It serves as a visual benchmark for the NixMon client, showing how raw data streams can be rendered into compelling, readable graphs using only standard terminal characters. While *htop* focuses on utilitarian process management, *btop* highlights the potential for rich user experiences in a terminal environment.

7. Methodology

The development of NixMon will follow an waterfall model, focusing first on data acquisition, then transmission, and finally visualization.

7.1 System Architecture

1. Data Acquisition Layer (Server)
The server will act as a daemon that periodically queries the `/proc` filesystem. Since these are text files, parsing is straightforward and does not require root access .

2. Communication Layer
The system will use standard BSD sockets. The server will bind to a port and listen for incoming connections. To support multiple clients (e.g., a desktop client and a mobile client simultaneously), the server will use I/O multiplexing functions like `select()` or `poll()`. Data will be formatted as JSON or a lightweight delimited string for ease of parsing.

3. Presentation Layer (Client)
The client connects to the server IP. Upon receiving a data packet, it clears or updates the specific console region to reflect the new values. Libraries like `ncurses` or simple ANSI escape codes will be used to create a static dashboard feel within the terminal.

8. Technical Description

8.1 Data Sources

The core functionality relies on the Linux Kernel's virtual filesystem:

- `/proc/stat`: Provides raw CPU tick counts (user, nice, system, idle). By comparing two snapshots taken 1 second apart, accurate CPU percentage can be calculated.
- `/proc/meminfo`: Provides memory details. We will extract `MemTotal`, `MemFree`, and `MemAvailable` to calculate usage percentages.
- `/proc/loadavg`: Provides the system load average for the past 1, 5, and 15 minutes, offering a quick metric for system stress.

8.2 Proposed Stack

- **Language:** C or C++ (for native socket and file handling).
- **Networking:** TCP/IP Sockets.
- **Concurrency:** `poll()` / `select()` / `pthreads`.
- **Data Format:** JSON or Custom Delimited Protocol.
- **Tools:** GCC/G++, Make/CMake, Git.

9. Proposed Deliverables

The project aims to produce a fully functional monitoring system along with comprehensive documentation. The deliverables are categorized into two main areas:

9.1 Software Artifacts

- **NixMon Server Application:** A C/C++ based daemon capable of reading `/proc` statistics and handling concurrent TCP connections.
- **NixMon Client Dashboard:** A terminal-based client application that connects to the server and visualizes real-time system metrics (CPU, RAM, Load).
- **Source Code Repository:** A complete Git repository containing all source files, makefiles, and build scripts.

9.2 Documentation

- **Project Proposal:** The initial document outlining the scope, objectives, and methodology (Current Document).
- **Final Report:** A final report detailing the system architecture, socket protocols used, challenges faced, and testing results.

10. Project Task and Time Schedule

- Networking & Concurrency Implementation – 1 days
- Client Dashboard Development – 5 days
- Documentation and Final Submission – 1 day