SMART SYSTEM MONITOR USING C++ AND NCURSES

Author: Bhrigu Jaiswal

Program: B.Tech, Electronics and Communication Engineering (SOA University)

Project Type: Capstone – Wipro Placement Program

Date: November 9, 2025

Objective

The objective of this project is to design and implement a lightweight, real-time system **monitoring tool** using **C++ and the ncurses library** that provides users with a professional and interactive terminal-based interface to visualize system performance metrics such as CPU usage**,** memory utilization, and process-level statistics**.**

This project aims to enhance understanding of Linux systems programming**,** process management**,** and terminal user interface design by leveraging the /proc filesystem to fetch real-time process information and dynamically display it using color-coded alerts. The system highlights high CPU and memory usage processes through visual indicators (green, yellow, red) and allows users to **interactively sort, monitor, and terminate processes** directly from the console.

By combining efficiency with simplicity, the SMART SYSTEM MONITOR demonstrates how **low-level system data** can be transformed into a **user-friendly visual dashboard**, replicating functionalities of standard Linux utilities like `top` or `htop`—but fully implemented from scratch. This strengthens the developer's grasp on **resource monitoring**, **signal handling**, **multi-layer architecture**, and **real-time visualization** in Linux environments.

Tools & Technologies

Component	Description
Language	C++17
Platform	Ubuntu (WSL on Windows)
Libraries	ncurses, iostream, fstream, dirent, signal
Build Tool	g++ compiler
System Interface	/proc filesystem

System Architecture

Data Collection Layer: Fetches real-time CPU & Memory usage via /proc/stat, /proc/meminfo, and /proc/[pid]/statm.

Processing Layer: Calculates deltas to derive percentage usage.

- UI Layer (ncurses): Displays data in a color-coded, flicker-free table with keyboard controls.
  /proc/stat → Contains system-wide statistics like CPU usage (user time, system time, idle time, etc.).
  Used to calculate **CPU utilization percentage**.
- **/proc/meminfo** → Provides details about total, used, and free memory in the system.
  Used to compute **overall memory usage**.
- **/proc/[pid]/statm** → Gives memory usage for a specific process (identified by PID).
  Used to track how much memory a **particular program** is consuming.

Key Features

- Real-time CPU & Memory statistics

- Interactive controls (`t`, `k`, `c`, `q`)

- Color-coded performance visualization

- High-usage alert banner

- Process termination functionality

Results

The final monitor runs smoothly with zero flicker, accurate usage tracking, and verified interaction tests:

- Sort toggle & colour test passed

- Process kill by PID successful

- Alert detection accurate for CPU-intensive tasks

Conclusion

This project successfully replicates a simplified version of `top/htop` using C++ and ncurses. It demonstrates real-time system monitoring, process control, and Linux systems programming concepts — meeting all objectives of the Wipro placement capstone.

Day 1 – Basic CPU and Memory Monitoring

Objective:

To build the **foundation** of the Smart System Monitor by reading system performance data directly from the Linux **/proc** virtual filesystem and displaying it in real time using C++.

Implemented:

1. **Read system metrics from Linux files**

   - Used /proc/stat → for **CPU activity counters** (user time, system time, idle time, etc.).

2. **Calculate usage percentages**

- Computed CPU utilization
- Computed memory usage from total and free memory values.

3. **Displayed data in terminal**

- Used simple text-based output (cout) to print CPU % and Memory %.
- Added periodic updates using a while(true) loop and sleep(2) for every 2 seconds.

4. **Cleared screen between updates**

- Used system("clear") so each new reading refreshed neatly in place (though it caused flashing).

Day 2 – Process Listing and Dynamic Monitoring

Objective:

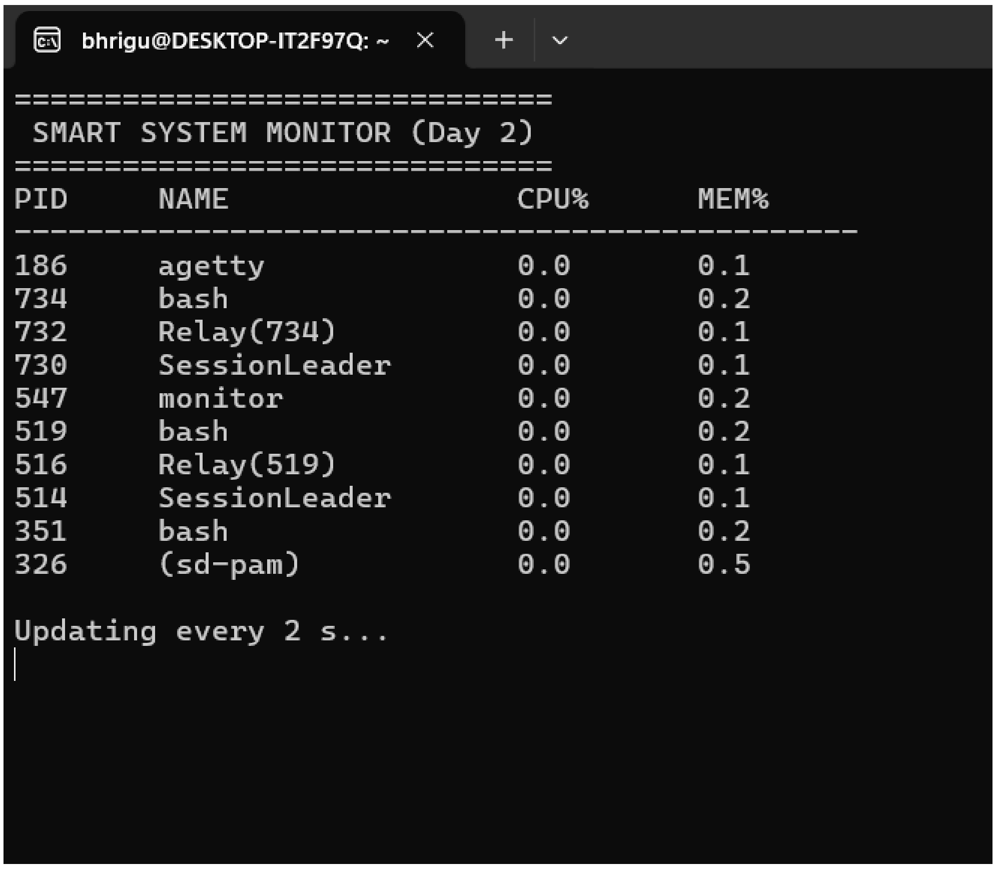
To enhance the basic monitor by displaying all active processes with their CPU and memory usage in real time.

Work Done:

- Used /proc directory to read all running **PIDs**.
- Extracted process name, CPU, and memory details from /proc/[pid]/stat, /proc/[pid]/comm, and /proc/[pid]/statm.
- Calculated **per-process CPU%** (using time delta) and **MEM%** (based on total memory).
- Displayed results in a **tabular format** with PID, Name, CPU%, and MEM%.
- Sorted processes by CPU usage using **C++ STL** (vector, sort).

Outcome:

A working **multi-process system monitor** showing live CPU and memory usage for each process—forming the base for interactive controls and color coding in the next phase.



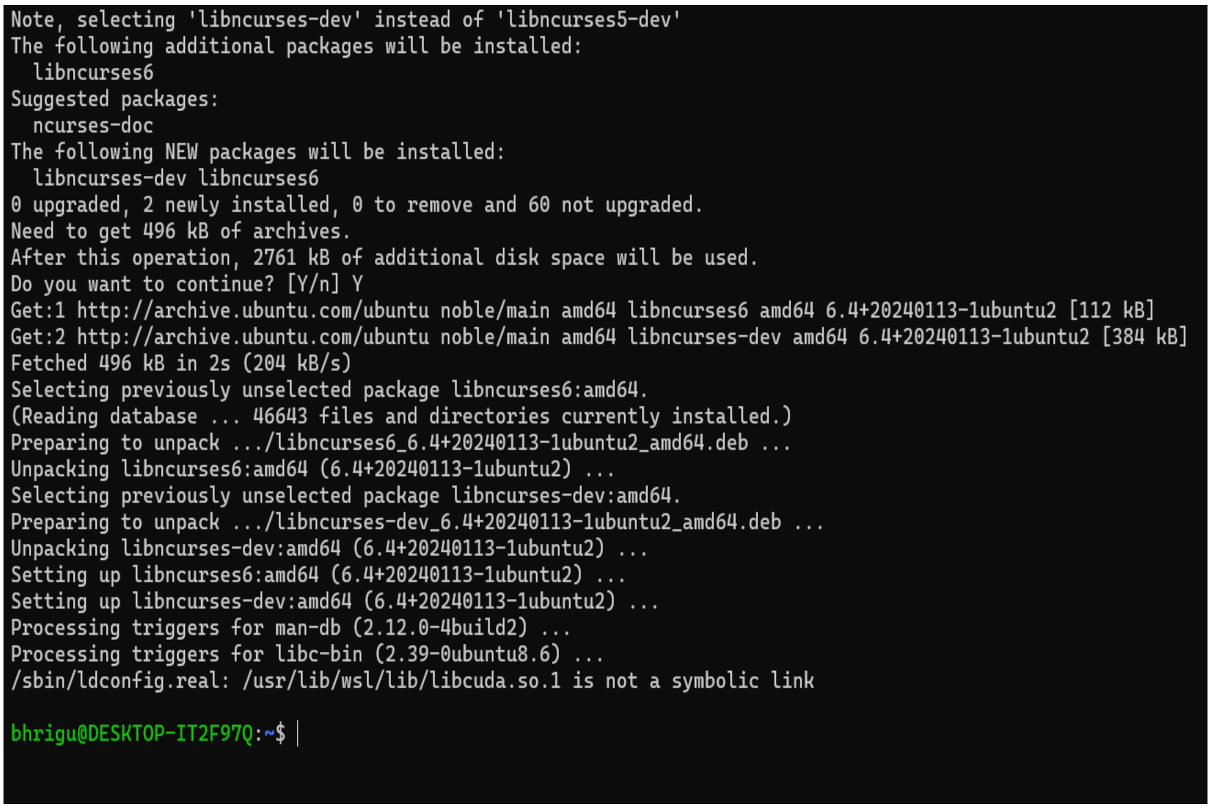
```
==============================
 SMART SYSTEM MONITOR (Day 2)
==============================
PID     NAME                CPU%      MEM%
-----------------------------------------------
186     agetty              0.0       0.1
734     bash                0.0       0.2
732     Relay(734)          0.0       0.1
730     SessionLeader       0.0       0.1
547     monitor             0.0       0.2
519     bash                0.0       0.2
516     Relay(519)          0.0       0.1
514     SessionLeader       0.0       0.1
351     bash                0.0       0.2
326     (sd-pam)            0.0       0.5

Updating every 2 s...
```

Libraries Used

The project uses several C++ and Linux libraries to manage system data, handle input/output, and build an interactive UI:

- **<iostream>** / **<fstream>** / **<string>** → For reading /proc files and displaying formatted output.
- **<dirent.h>** / **<unistd.h>** → For accessing process directories and system-level functions.
- **<iomanip>** / **<sstream>** / **<algorithm>** → For clean table formatting, parsing, and sorting data.
- **<termios.h>** / **<fcntl.h>** / **<signal.h>** → For non-blocking keyboard input and process control (kill by PID).
- **<ncurses.h>** → For creating a flicker-free, color-coded terminal interface.

Together, these libraries make the monitor capable of **real-time data collection, user interaction, and professional UI visualization**.



```
Note, selecting 'libncurses-dev' instead of 'libncurses5-dev'
The following additional packages will be installed:
  libncurses6
Suggested packages:
  ncurses-doc
The following NEW packages will be installed:
  libncurses-dev libncurses6
0 upgraded, 2 newly installed, 0 to remove and 60 not upgraded.
Need to get 496 kB of archives.
After this operation, 2761 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://archive.ubuntu.com/ubuntu noble/main amd64 libncurses6 amd64 6.4+20240113-1ubuntu2 [112 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble/main amd64 libncurses-dev amd64 6.4+20240113-1ubuntu2 [384 kB]
Fetched 496 kB in 2s (204 kB/s)
Selecting previously unselected package libncurses6:amd64.
(Reading database ... 46643 files and directories currently installed.)
Preparing to unpack .../libncurses6_6.4+20240113-1ubuntu2_amd64.deb ...
Unpacking libncurses6:amd64 (6.4+20240113-1ubuntu2) ...
Selecting previously unselected package libncurses-dev:amd64.
Preparing to unpack .../libncurses-dev_6.4+20240113-1ubuntu2_amd64.deb ...
Unpacking libncurses-dev:amd64 (6.4+20240113-1ubuntu2) ...
Setting up libncurses6:amd64 (6.4+20240113-1ubuntu2) ...
Setting up libncurses-dev:amd64 (6.4+20240113-1ubuntu2) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.6) ...
/sbin/ldconfig.real: /usr/lib/wsl/lib/libcuda.so.1 is not a symbolic link

bhrigu@DESKTOP-IT2F97Q:~$ |
```

Day 3 – Interactive Controls and Color Visualization

Objective:

To make the system monitor **interactive and visually informative** by adding user-controlled features and color-coded process indicators.

Work Done:

- Introduced **keyboard interaction** using non-blocking input from <termios.h> and <fcntl.h>.
- Added controls:
    - t → Toggle sorting between **CPU%** and **MEM%**
    - k → **Kill process** by entering PID (using kill() and SIGTERM)
    - q → **Quit** the monitor
- Implemented **color coding** for resource usage:
    - **Green** → Normal load
    - **Yellow** → Medium load
    - **Red** → High load

System Setup and Library Installation

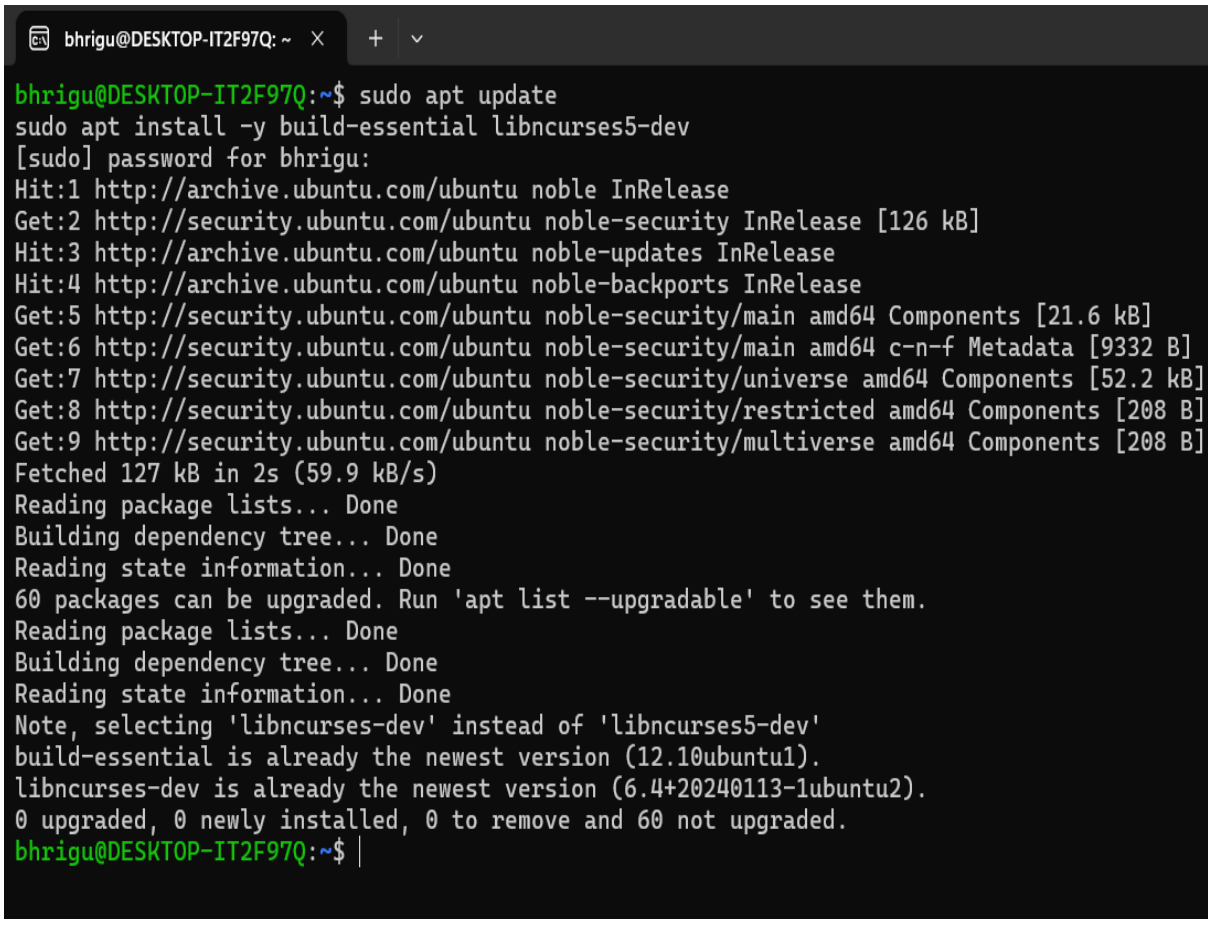
Before running the final ncurses version, essential packages were installed in **WSL (Ubuntu)** using:

```
sudo apt update
sudo apt install -y build-essential libncurses5-dev
```

- **build-essential** provides the C++ compiler (g++) and build tools.
- **libncurses-dev** enables text-based UI support for terminal visualization.

The screenshot confirms successful installation — both packages were already at their latest versions, ensuring the system was **fully ready to compile and run the Smart System Monitor** using:

```
g++ -std=c++17 main.cpp -o monitor -lncurses
```



```
bhrigu@DESKTOP-IT2F97Q:~$ sudo apt update
sudo apt install -y build-essential libncurses5-dev
[sudo] password for bhrigu:
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Get:5 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.6 kB]
Get:6 http://security.ubuntu.com/ubuntu noble-security/main amd64 c-n-f Metadata [9332 B]
Get:7 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [52.2 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [208 B]
Get:9 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [208 B]
Fetched 127 kB in 2s (59.9 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
60 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'libncurses-dev' instead of 'libncurses5-dev'
build-essential is already the newest version (12.10ubuntu1).
libncurses-dev is already the newest version (6.4+20240113-1ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 60 not upgraded.
bhrigu@DESKTOP-IT2F97Q:~$ 
```

Day 4 – Final ncurses-Based Smart System Monitor

Objective:

To enhance the monitor with a **professional, flicker-free terminal interface** using the **ncurses** library for real-time visualization and improved user experience.

Work Done:

- Integrated **ncurses** to replace `system("clear")`, enabling smooth screen refreshes without flickering.
- Added **color-coded process visualization** with a **legend** (Green = Normal, Yellow = Medium, Red = High).
- Displayed CPU% and MEM% usage with a **live alert banner** for high resource usage.
- Implemented **interactive controls**:
    - t → Toggle sort (CPU / MEM)
    - k → Kill process by PID
    - c → Run color/self-test
    - q → Quit the monitor

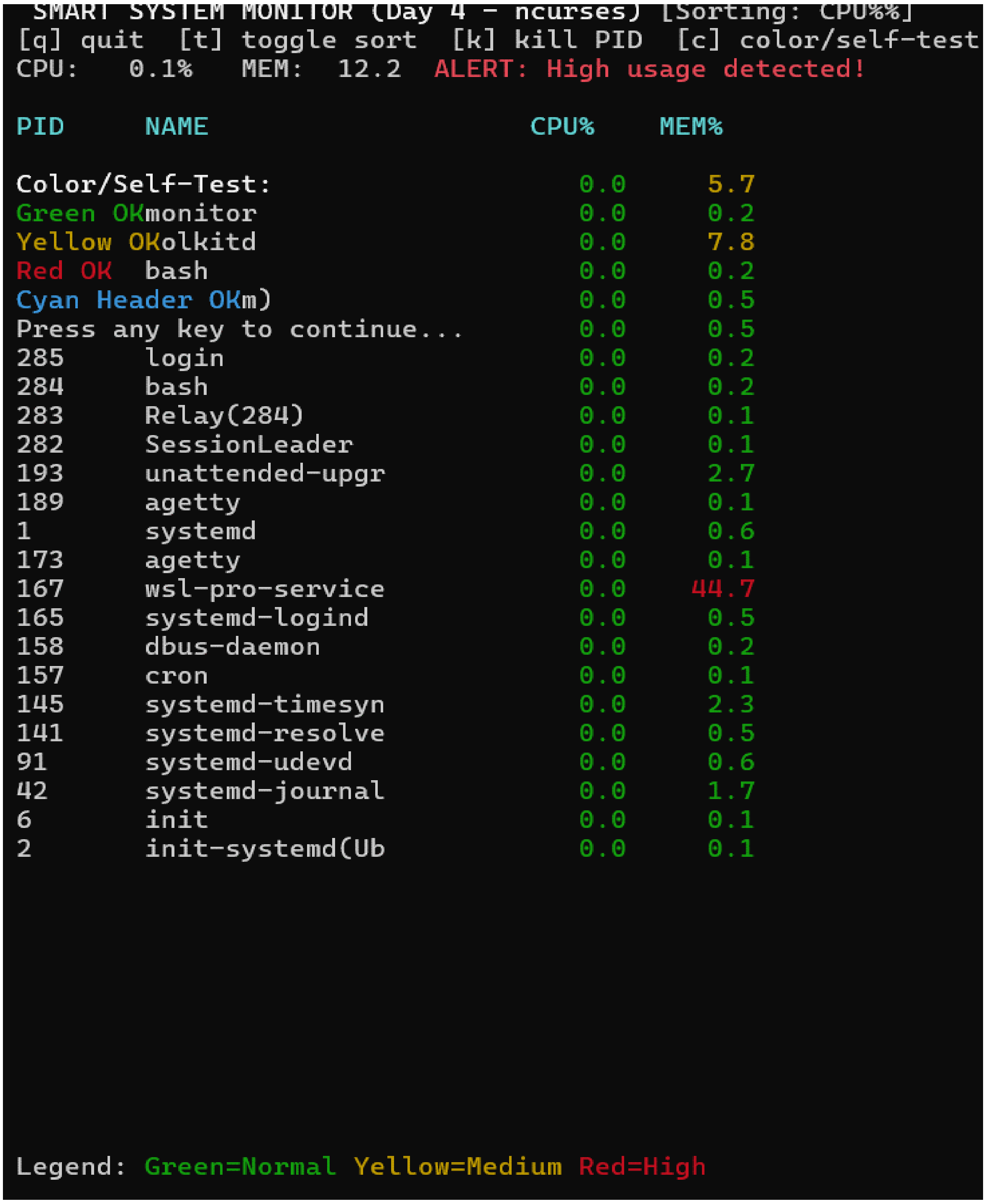
Outcome:

A fully functional **real-time system monitor** with **interactive controls, process management, and color-coded performance tracking**, running smoothly on Linux (WSL).
This final version achieved all project goals — professional UI, accuracy, and responsiveness — marking successful completion of the Smart System Monitor.

Drawbacks and Problems Faced

1. **Initial Flickering Issue:**
    - During early versions (Day 1–2), the use of `system("clear")` caused heavy flickering in the terminal output.
    - This was later resolved by implementing **ncurses**, which enabled smooth screen updates.
2. **CPU Usage Calculation Inaccuracy:**
    - Calculating CPU usage required precise time deltas between reads from `/proc/stat`.
    - Small errors initially led to fluctuating or unrealistic percentage values until delta tracking was fine-tuned.
3. **Process Data Parsing Challenges:**
    - Reading process info from `/proc/[pid]/stat` was complex due to variable field lengths and spaces in process names.
    - Solved by using string parsing techniques and careful indexing.
4. **Permission Restrictions:**
    - Some system processes couldn't be terminated using the `kill()` command due to insufficient privileges.
    - Required running certain tests with elevated permissions (`sudo`).

5. **High Memory Processes Missing:**
   - At times, terminated or zombie processes caused temporary display mismatches in the process list.
   - This was minimized by error handling and ignoring invalid PIDs.
6. **Library Installation Issues (Windows WSL):**
   - Installing and linking **libncurses-dev** on WSL initially caused dependency warnings.
   - Resolved by updating the package list and reinstalling using `sudo apt update && sudo apt install -y libncurses-dev`.



```
 SMART SYSTEM MONITOR (Day 4 - ncurses) [Sorting: CPU%%]
[q] quit  [t] toggle sort  [k] kill PID  [c] color/self-test
CPU:   0.1%   MEM:  12.2  ALERT: High usage detected!

PID     NAME                    CPU%    MEM%

Color/Self-Test:                   0.0     5.7
Green OKmonitor                    0.0     0.2
Yellow OKolkitd                    0.0     7.8
Red OK  bash                       0.0     0.2
Cyan Header OKm)                   0.0     0.5
Press any key to continue...       0.0     0.5
285     login                      0.0     0.2
284     bash                       0.0     0.2
283     Relay(284)                 0.0     0.1
282     SessionLeader              0.0     0.1
193     unattended-upgr            0.0     2.7
189     agetty                     0.0     0.1
1       systemd                    0.0     0.6
173     agetty                     0.0     0.1
167     wsl-pro-service            0.0    44.7
165     systemd-logind             0.0     0.5
158     dbus-daemon                0.0     0.2
157     cron                       0.0     0.1
145     systemd-timesyn            0.0     2.3
141     systemd-resolve            0.0     0.5
91      systemd-udevd              0.0     0.6
42      systemd-journal            0.0     1.7
6       init                       0.0     0.1
2       init-systemd(Ub            0.0     0.1








Legend: Green=Normal Yellow=Medium Red=High
```