System Monitor Tool Report

Wipro Project Report

Title: System Monitor Tool

Project No.: 3

Name: Abhishek Sandha

Technology Used: C++ (Linux Environment)

Organization: Wipro

Duration: November 2025

Abstract

The System Monitor Tool is a Linux-based command-line utility developed in C++ to track and manage system performance in real time. It displays CPU usage, Memory utilization, and active processes. Users can sort processes by memory usage, refresh data manually, and terminate unwanted processes directly through the terminal. This project deepens understanding of Linux process management and the /proc filesystem.

Objectives

■ Develop a real-time System Monitor Tool analyzing CPU, memory, and process statistics.

■ Implement process-level control (kill functionality).

■ Utilize C++ file I/O for reading /proc system files.

■ Provide an interactive command-based terminal interface.

■ Demonstrate Linux system programming concepts.

Tools and Technologies Used

Language: C++ (C++14 standard)

Operating System: Linux (Ubuntu)

Libraries: , , , , ,

Concepts Used: File I/O, Process Handling, Sorting, System Calls

Build Tool: g++ Compiler

Methodology

■ CPU Usage Calculation: Reads /proc/stat to extract CPU time fields and compute usage percentage.

■ Memory Usage Calculation: Reads /proc/meminfo for total and free memory usage.

■ Process Information Retrieval: Gathers process details (PID, memory) from /proc/[pid]/.

■ Sorting and Display: Displays top processes sorted by memory or PID.

■ Process Termination: Allows killing a process using its PID.

■ Refresh: Manual refresh using ENTER key.

Code

```
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
using namespace std;
void readMemoryInfo(long &totalMem;, long &freeMem;) {
ifstream file("/proc/meminfo");
string key, unit; long value;
totalMem = freeMem = 0;
while (file >> key >> value >> unit) {
if (key == "MemTotal:") totalMem = value / 1024;
if (key == "MemAvailable:") freeMem = value / 1024;
}
}
long long lastTotal = 0, lastIdle = 0;
float readCpuUsage() {
ifstream file("/proc/stat");
string cpu; long long user, nice, system, idle;
file >> cpu >> user >> nice >> system >> idle;
long long total = user + nice + system + idle;
long long totalDiff = total - lastTotal;
long long idleDiff = idle - lastIdle;
float cpuPercent = (totalDiff != 0) ? (float)(totalDiff - idleDiff) * 100.0 / totalDiff : 0;
lastTotal = total; lastIdle = idle;
return cpuPercent;
}
struct Process { int pid; string name; long memoryKB; float cpuPercent; };
bool isNumber(const string &s;) {
for (char c : s) if (!isdigit((unsigned char)c)) return false;
```

```
return true;
}
vector getProcesses() {
vector result; DIR *dir = opendir("/proc"); if (!dir) return result;
struct dirent *entry;
while ((entry = readdir(dir))) {
string dirname = entry->d_name;
if (!isNumber(dirname)) continue;
int pid = stoi(dirname); string pname; long mem = 0;
ifstream f1("/proc/" + dirname + "/comm");
if (f1.good()) getline(f1, pname);
ifstream f2("/proc/" + dirname + "/statm");
if (f2.good()) { long pages = 0; f2 >> pages; mem = pages * 4; }
result.push_back({pid, pname, mem, 0.0f});
}
closedir(dir); return result;
}
void killProcess(int pid) {
if (kill(pid, SIGKILL) == 0) cout << "Process " << pid << " killed.\n";
else cerr << "Failed: " << strerror(errno) << "\n";
}
char sortMode = 'n';
void display() {
long totalMem = 0, freeMem = 0; readMemoryInfo(totalMem, freeMem);
float cpu = readCpuUsage(); auto plist = getProcesses();
if (sortMode == 'm') sort(plist.begin(), plist.end(), [](auto &a;, auto &b;){ return a.memoryKB >
b.memoryKB; });
system("clear");
cout << "================== SYSTEM MONITOR ==================\n";
cout << fixed << setprecision(1);
cout << "CPU Usage: " << cpu << "%\n";
cout << "Memory: " << (totalMem - freeMem) << " MB / " << totalMem << " MB\n";
cout << "----------------------------------------------------\n";
cout << left << setw(8) << "PID" << setw(14) << "Memory(KB)" << setw(20) << "Name" << "\n";
cout << "----------------------------------------------------\n";
int limit = 120;
```

```
for (auto &p; : plist) {
cout << left << setw(8) << p.pid << setw(14) << p.memoryKB << setw(20) << p.name << "\n";
if (--limit <= 0) break;
}
cout << "----------------------------------------------------\n";
cout << "Enter = Refresh\n"
<< "m = Sort by memory\n"
<< "n = No sort\n"
<< "k = Kill process\n"
<< "q = Quit\n";
}
int main() {
readCpuUsage();
while (true) {
display();
cout << "\nCommand: ";
string input; getline(cin, input);
if (input == "") continue;
if (input == "q") break;
if (input == "m") { sortMode = 'm'; continue; }
if (input == "n") { sortMode = 'n'; continue; }
if (input[0] == 'k') {
try { int pid = stoi(input.substr(2)); killProcess(pid); }
catch (...) { cout << "Invalid format. Use: k 1234\n"; }
}
}
cout << "Exiting System Monitor.\n"; return 0;
}
```

Sample Output

Below are the screenshots showing real-time execution of the System Monitor Tool in Linux terminal:

Conclusion

The System Monitor Tool successfully retrieves live system information from the /proc filesystem and displays it interactively through the command line. It enables users to monitor CPU and memory usage, view active processes, and terminate unnecessary ones. This project demonstrates practical system-level programming and process control in Linux using C++.