

Wipro Project Report

Title: System Monitor Tool

Project No.: 3

Name: Pradyumna Kumar Sahu

Technology Used: C++ (Linux Environment)

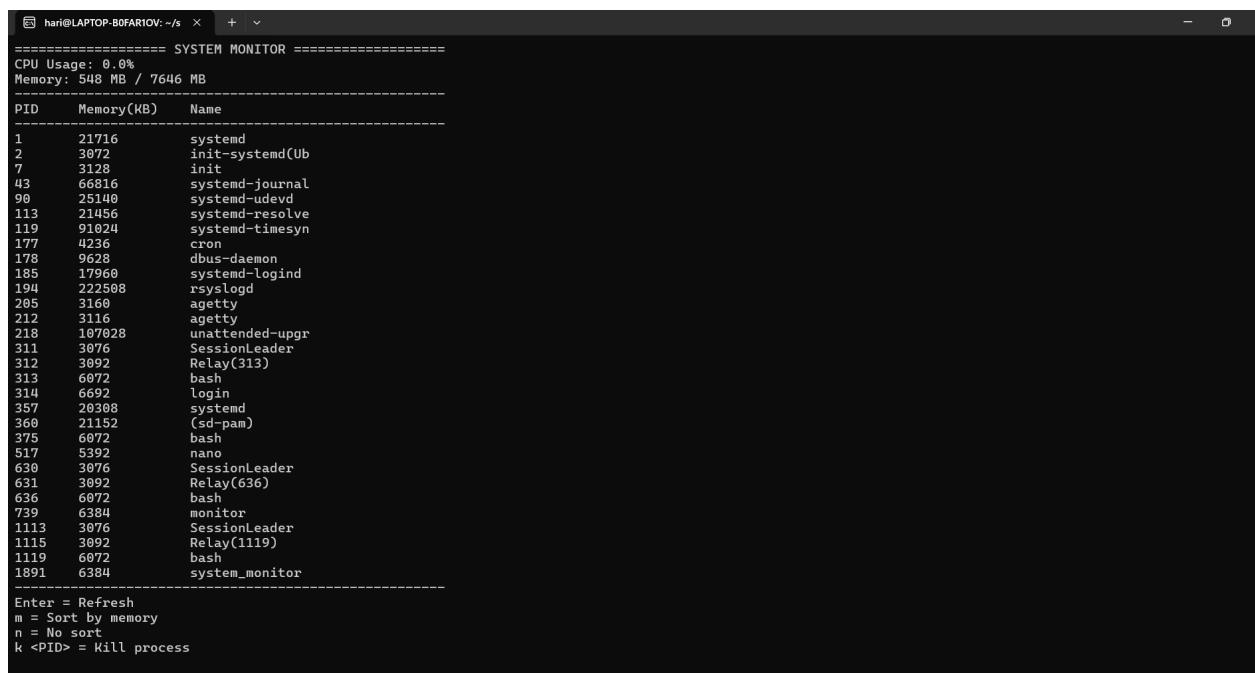
Organization: Wipro

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.

Sample Output Screenshot



A screenshot of a terminal window titled "SYSTEM MONITOR". The window shows system statistics and a list of running processes. The statistics include CPU Usage (0.0%) and Memory usage (548 MB / 7646 MB). The process list is sorted by memory usage, showing various system daemons and user sessions. At the bottom of the window, there are keyboard shortcuts for Refresh, Sort by memory, No sort, and Kill process.

PID	Memory(KB)	Name
1	21716	systemd
2	3072	init-systemd(Up
7	3128	init
43	66816	systemd-journal
90	25140	systemd-udevd
113	21456	systemd-resolve
119	91024	systemd-timesync
177	4236	cron
178	9628	dbus-daemon
185	17968	systemd-logind
194	222568	rsyslogd
205	3160	agetty
212	3116	agetty
218	107028	unattended-upgr
311	3076	SessionLeader
312	3092	Relay(313)
313	6072	bash
314	6692	login
357	29388	systemd
360	21152	(sd-pam)
375	6072	bash
517	5392	nano
630	3076	SessionLeader
631	3092	Relay(636)
636	6072	bash
739	6384	monitor
1113	3076	SessionLeader
1115	3092	Relay(1119)
1119	6072	bash
1891	6384	system_monitor

Enter = Refresh
m = Sort by memory
n = No sort
k <PID> = Kill process

Complete C++ Source Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <dirent.h>
#include <unistd.h>
#include <algorithm>
#include <signal.h>
#include <cstring>
```

```

#include <iomanip>
using namespace std;

/*
MEMORY READING FROM /proc/meminfo
-----
Reads MemTotal and MemAvailable
*/
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 = 0;
    if (totalDiff != 0)
        cpuPercent = (float)(totalDiff - idleDiff) * 100.0 / totalDiff;

    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<Process> getProcesses() {
    vector<Process> 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;

        string path1 = "/proc/" + dirname + "/comm";
        ifstream f1(path1);
        if (f1.good()) getline(f1, pname);

        long mem = 0;
        string path2 = "/proc/" + dirname + "/statm";

```

```

        ifstream f2(path2);
        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 <PID> = 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;
}
```