Assignment 3 (LSP)

System Monitor Tool

Student Name: Dinesh Kumar Panda

Department: Computer Science and Information Technology

Tools and Technologies Used:

C++, Linux Environment, g++ Compiler, System Calls, /proc File System

Abstract:

This project focuses on building a lightweight system monitoring tool in C++ that mimics the functionality of the Linux 'top' command. The tool gathers and displays real-time information about system processes, CPU utilization, and memory usage by reading from the /proc file system. It provides a command-line interface for users to sort, view, and manage processes effectively.

Objective:

To design and implement a system monitor tool that provides real-time insights into running processes, CPU usage, and memory consumption, allowing users to manage and analyze system performance efficiently.

Problem Statement:

Monitoring and managing system resources is crucial for performance optimization. Existing tools like 'top' are complex to understand for beginners. This project aims to create a simplified version using C++ that can display system process information, sort processes, and allow users to terminate processes directly from the interface.

Technologies Used:

- Programming Language: C++
- Operating System: Linux
- Compiler: g++
- Libraries: <dirent.h>, <signal.h>, <chrono>, <thread>
- Data Source: /proc file system

Code: system_monitor.cpp

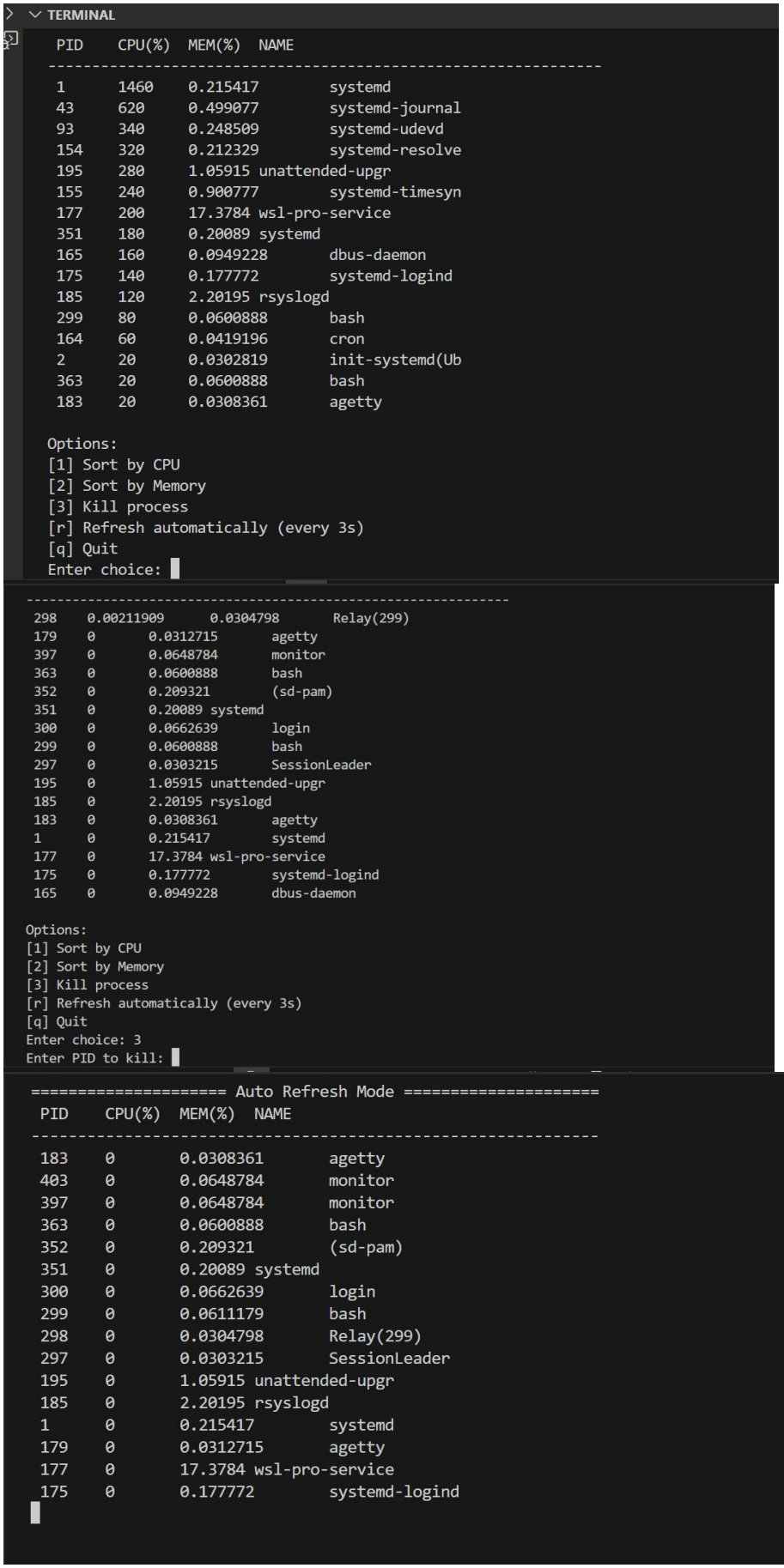
```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <algorithm>
#include <unistd.h>
#include <dirent.h>
#include <sys/types.h>
#include <signal.h>
#include <chrono>
```

```
#include <thread>

using namespace std;

struct Process {
   int pid;
   string name;
   double cpuUsage;
   double memUsage;
};

// Function to read total system memory
long getTotalMemory() {
   ifstream meminfo("/proc/meminfo");
   string key;
   long value;
   string unit;
   while (meminfo >> key >> value >> unit) {
      if (key == "MemTotal:") {
         return value;
      }
   }
   return 1; // prevent divide by zero
}

// Function to get CPU total time from /proc/stat
long getTotalCPUTime() {
   ifstream file("/proc/stat");
   string cpu;
   long user, nice, system, idle, iowait, irq, softirq, steal;
   file >> cpu >> user >> nice >> system >> idle >> iowait >> irq >> softirq >> steal;
   return user + nice + system + idle + iowait + irq + softirq + steal;
}

// Function to get process CPU time
long getProcessCPUTime(int pid) {
   string path = "/proc/" + to_string(pid) + "/stat";
   ifstream file(path);
   if (!file) return 0;
   string tmp;
   long utime, stime;
   for (int i = 0; i < 13; i++) file >> tmp; // skip first 13 fields
   file >> utime >> stime;
   return utime + stime;
}

// Function to get process name
```

```
string getProcessName(int pid) {
   string path = "/proc/" + to_string(pid) + "/comm";
   ifstream file(path);
   string name;
   getline(file, name);
   return name;
}

// Function to get memory usage of process
double getMemoryUsage(int pid, long totalMem) {
   string path = "/proc/" + to_string(pid) + "/statm";
   ifstream file(path);
   long mem;
   file >> mem;
   return (double)mem * getpagesize() / 1024.0 / totalMem * 100.0;
}

// Gather process data
vector<Process> getProcesses(long totalMem, long totalCPUTimePrev, vector<long>
&prevTimes) {
   vector<Process> processes;
   DIR *dir = opendir("/proc");
   if (!dir) return processes;

   struct dirent *entry;
   long totalCPUTimeNow = getTotalCPUTime();

   while ((entry = readdir(dir)) != NULL) {
      if (isdigit(entry->d_name[0])) {
         int pid = stoi(entry->d_name);
         string name = getProcessName(pid);
         long procTimeNow = getProcessCPUTime(pid);
         long procTimePrev = (pid < (int)prevTimes.size()) ? prevTimes[pid] : 0;
         double cpuUsage = 100.0 * (procTimeNow - procTimePrev) /
(double)(totalCPUTimeNow - totalCPUTimePrev + 1);
         double memUsage = getMemoryUsage(pid, totalMem);
         processes.push_back({pid, name, cpuUsage, memUsage});
         if (pid >= (int)prevTimes.size()) prevTimes.resize(pid + 1);
         prevTimes[pid] = procTimeNow;
      }
   }
   closedir(dir);
   return processes;
}

int main() {
   long totalMem = getTotalMemory();
```

```
    long totalCPUPrev = getTotalCPUTime();
    vector<long> prevTimes(10000, 0);
    vector<Process> processes;

    char choice;
    while (true) {
        system("clear");
        cout << "====================== System Monitor Tool
======================\n";
        cout << " PID\tCPU(%)\tMEM(%)\tNAME\n";
        cout << "------------------------------------------------------------\n";

        processes = getProcesses(totalMem, totalCPUPrev, prevTimes);
        totalCPUPrev = getTotalCPUTime();

        sort(processes.begin(), processes.end(), [](const Process &a, const Process &b) {
            return a.cpuUsage > b.cpuUsage;
        });

        int count = 0;
        for (auto &p : processes) {
            if (count++ > 15) break;
            cout << " " << p.pid << "\t" << p.cpuUsage << "\t" << p.memUsage << "\t" << p.name
<< endl;
        }

        cout << "\nOptions:\n"
             << "[1] Sort by CPU\n"
             << "[2] Sort by Memory\n"
             << "[3] Kill process\n"
             << "[r] Refresh automatically (every 3s)\n"
             << "[q] Quit\n"
             << "Enter choice: ";
        cin >> choice;

        if (choice == '1') {
            sort(processes.begin(), processes.end(), [](const Process &a, const Process &b) {
                return a.cpuUsage > b.cpuUsage;
            });
        } else if (choice == '2') {
            sort(processes.begin(), processes.end(), [](const Process &a, const Process &b) {
                return a.memUsage > b.memUsage;
            });
        } else if (choice == '3') {
            int pid;
            cout << "Enter PID to kill: ";
            cin >> pid;
```

```
            if (kill(pid, SIGTERM) == 0)
                cout << "Process " << pid << " terminated.\n";
            else
                perror("Error killing process");
            this_thread::sleep_for(chrono::seconds(2));
        } else if (choice == 'r') {
            for (int i = 0; i < 5; i++) { // refresh 5 times
                system("clear");
                processes = getProcesses(totalMem, totalCPUPrev, prevTimes);
                totalCPUPrev = getTotalCPUTime();
                sort(processes.begin(), processes.end(), [](const Process &a, const Process &b) {
                    return a.cpuUsage > b.cpuUsage;
                });
                cout << "====================== Auto Refresh Mode
====================\n";
                cout << " PID\tCPU(%)\tMEM(%)\tNAME\n";
                cout << "------------------------------------------------------------\n";
                int count = 0;
                for (auto &p : processes) {
                    if (count++ > 15) break;
                    cout << " " << p.pid << "\t" << p.cpuUsage << "\t" << p.memUsage << "\t" <<
p.name << endl;
                }
                this_thread::sleep_for(chrono::seconds(3));
            }
        } else if (choice == 'q') break;
    }

    return 0;
}
```


TERMINAL
PID CPU(%) MEM(%) NAME
1 1460 0.215417 systemd
43 620 0.499077 systemd-journal
93 340 0.248509 systemd-udevd
154 320 0.212329 systemd-resolve
195 280 1.05915 unattended-upgr
155 240 0.900777 systemd-timesyn
177 200 17.3784 wsl-pro-service
351 180 0.20089 systemd
165 160 0.0949228 dbus-daemon
175 140 0.177772 systemd-logind
185 120 2.20195 rsyslogd
299 80 0.0600888 bash
164 60 0.0419196 cron
2 20 0.0302819 init-systemd(Ub
363 20 0.0600888 bash
183 20 0.0308361 agetty
Options:
[1] Sort by CPU
[2] Sort by Memory
[3] Kill process
[r] Refresh automatically (every 3s)
[q] Quit
Enter choice:
298 0.00211909 0.0304798 Relay(299)
179 0 0.0312715 agetty
397 0 0.0648784 monitor
363 0 0.0600888 bash
352 0 0.209321 (sd-pam)
351 0 0.20089 systemd
300 0 0.0662639 login
299 0 0.0600888 bash
297 0 0.0303215 SessionLeader
195 0 1.05915 unattended-upgr
185 0 2.20195 rsyslogd
183 0 0.0308361 agetty
1 0 0.215417 systemd
177 0 17.3784 wsl-pro-service
175 0 0.177772 systemd-logind
165 0 0.0949228 dbus-daemon
Options:
[1] Sort by CPU
[2] Sort by Memory
[3] Kill process
[r] Refresh automatically (every 3s)
[q] Quit
Enter choice: 3
Enter PID to kill:
Auto Refresh Mode
PID CPU(%) MEM(%) NAME
183 0 0.0308361 agetty
403 0 0.0648784 monitor
397 0 0.0648784 monitor
363 0 0.0600888 bash
352 0 0.209321 (sd-pam)
351 0 0.20089 systemd
300 0 0.0662639 login
299 0 0.0611179 bash
298 0 0.0304798 Relay(299)
297 0 0.0303215 SessionLeader
195 0 1.05915 unattended-upgr
185 0 2.20195 rsyslogd
1 0 0.215417 systemd
179 0 0.0312715 agetty
177 0 17.3784 wsl-pro-service
175 0 0.177772 systemd-logind