PROJECT REPORT

ON

SYSTEM MONITER TOOL

Name – Rimjhim Mohanty

Regno – 2241013268

Brach – Cse (Ds)

Institute Name - Institute of Technical Education and Research

Introduction

The System Monitor Tool is a Linux-based project developed in C++ to display real-time information about system processes, CPU usage, and memory utilization. Similar to the functionality of the top command, this tool collects data from the /proc file system and presents it in a clear and user-friendly terminal interface. It helps users monitor active processes, understand system resource consumption, and identify high-usage tasks. The project enhances understanding of Linux process management, CPU scheduling, and memory handling. It also provides hands-on experience with file handling, system calls, and real-time data updates in C++.

Objectives

- To develop a C++-based system monitoring application that displays real-time information such as running processes, CPU usage, and memory usage.
- To understand and utilize the Linux /proc filesystem for extracting process and system-level details like PID, process name, CPU time, and RAM usage.
- To implement features similar to the top command, including sorting processes based on CPU or memory usage and refreshing data at regular intervals.
- To improve knowledge of system programming, process management, and resource monitoring in Linux-based operating systems.
- To provide a user-friendly terminal interface for efficient system performance monitoring and analysis.

System Requirements

1.Hardware Requirements

- Minimum **2 GB RAM**
- **Dual-core processor** or higher
- At least **200 MB of free disk space**
- A system capable of running **Linux (Ubuntu or similar)**

2. Software Requirements

- **Operating System:** Ubuntu/Linux (recommended: Ubuntu 20.04 or later)

- **Compiler:** g++ with **C++17 support**
- **Libraries Used:**
    - <filesystem> (for process directory access)
    - Standard C++ Libraries (iostream, fstream, unistd.h, etc.)
- **Tools Required:**
    - Terminal / Command Line
    - Text Editor (VS Code, gedit, nano, etc.)
- **Access to /proc filesystem** for process and system data

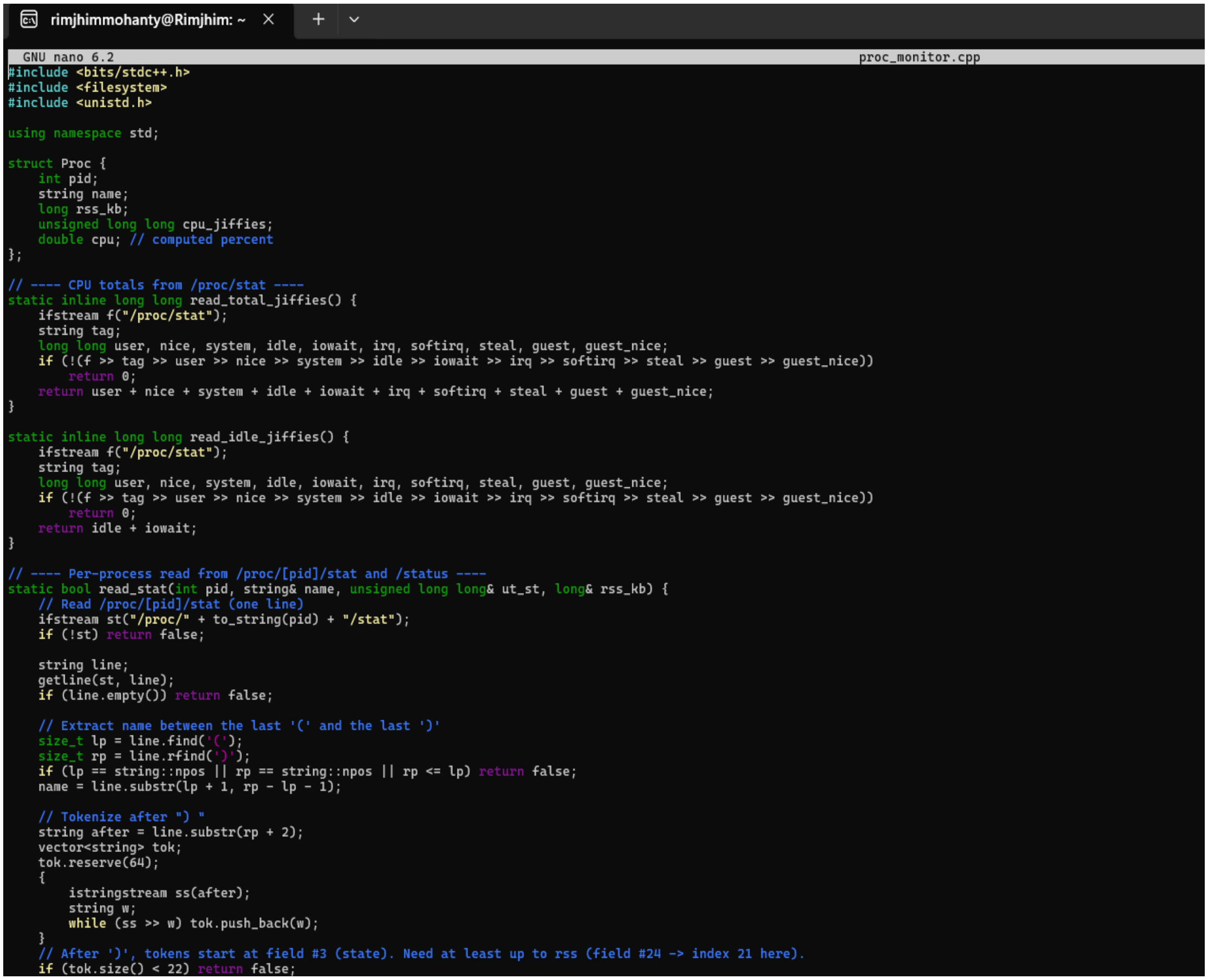
Source Code / Implementation

```
#include <bits/stdc++.h>
#include <filesystem>
#include <unistd.h>
using namespace std;
struct Proc {
int pid;
string name;
long rss_kb;
unsigned long long cpu_jiffies;
double cpu;
};
long long read_total_jiffies() {
ifstream f("/proc/stat");
string tag;
long long a,b,c,d,e,fv,g,h,i,j;
```

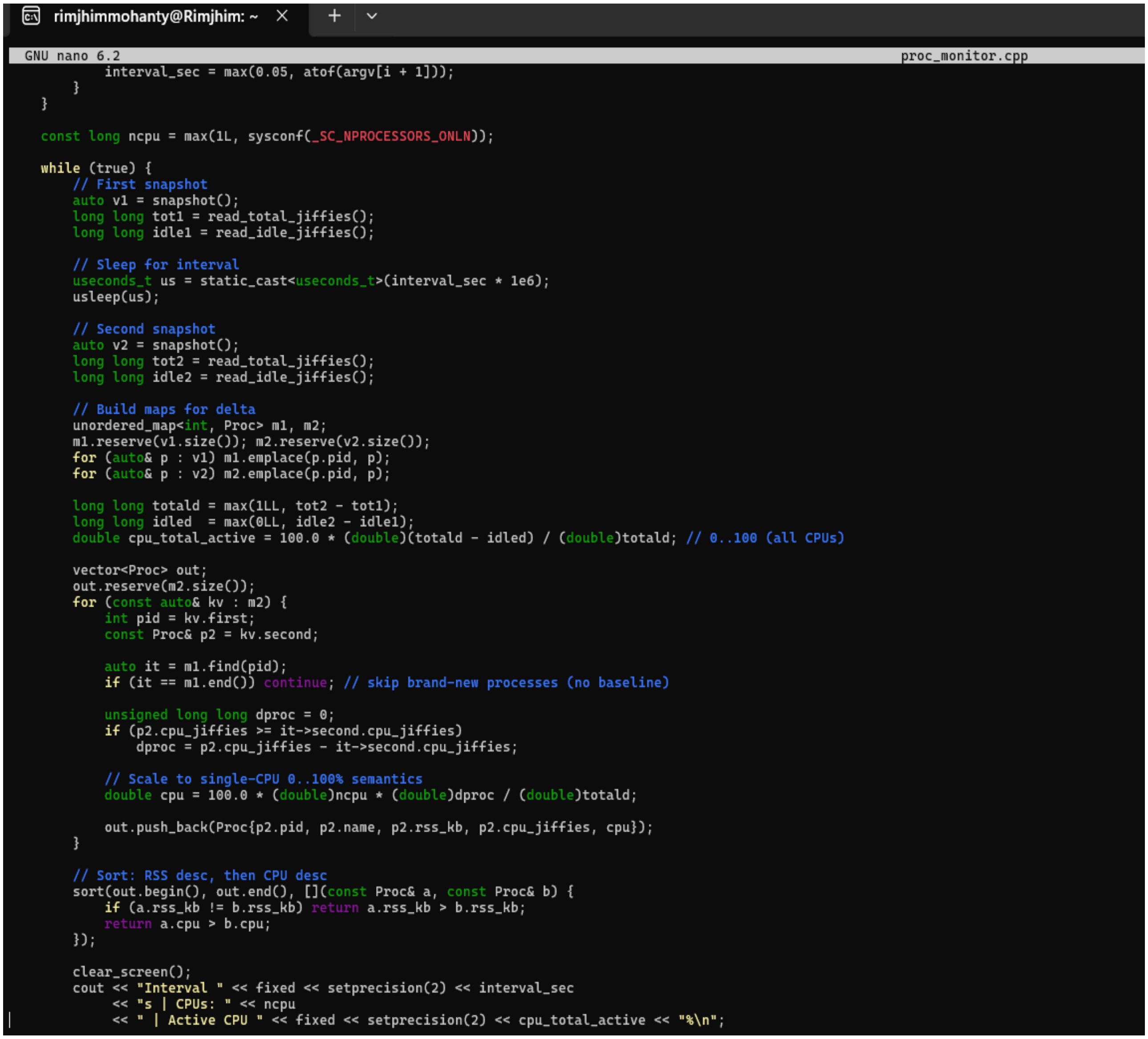
```
f >> tag >> a >> b >> c >> d >> e >> fv >> g >> h >> i >> j;

return a + b + c + d + e + fv + g + h + i + j;

}

long long read_idle_jiffies() {

ifstream f("/proc/stat");

string tag;

long long user, nice, system, idle, iowait, irq, softirq, steal, guest, guest_nice;

f >> tag >> user >> nice >> system >> idle >> iowait >> irq >> softirq >> steal >> guest >> guest_nice;

return idle + iowait;

}

bool read_stat(int pid, string &name, unsigned long long &ut_st, long &rss_kb) {

ifstream st("/proc/" + to_string(pid) + "/stat");

if (!st) return false;

string line;

getline(st, line);

auto lp = line.find('(');

auto rp = line.rfind(')');

if (lp == string::npos || rp == string::npos || rp <= lp) return false;

name = line.substr(lp + 1, rp - lp - 1);

string data = line.substr(rp + 2);

istringstream ss(data);

unsigned long long ut = 0, stt = 0;

long rss_pages = 0;

string skip;

for (int i = 0; i < 11; i++) ss >> skip;

ss >> ut >> stt;
```

```
for (int i = 0; i < 7; i++) ss >> skip;
ss >> rss_pages;
ut_st = ut + stt;
long page_kb = sysconf(_SC_PAGESIZE) / 1024;
rss_kb = rss_pages * page_kb;
return true;
}
vector<Proc> snapshot() {
vector<Proc> v;
for (auto &entry : filesystem::directory_iterator("/proc")) {
string name = entry.path().filename().string();
if (!all_of(name.begin(), name.end(), ::isdigit)) continue;
int pid = stoi(name);
string pname;
unsigned long long cpu;
long rss;
if (read_stat(pid, pname, cpu, rss)) {
v.push_back({pid, pname, rss, cpu, 0.0});
}
}
return v;
}
int main() {
int ncpu = max(1L, sysconf(_SC_NPROCESSORS_ONLN));
while (true) {
auto snap1 = snapshot();
long long t1 = read_total_jiffies();
long long idle1 = read_idle_jiffies();
usleep(1500000);
auto snap2 = snapshot();
long long t2 = read_total_jiffies();
long long idle2 = read_idle_jiffies();
unordered_map<int, Proc> map1, map2;
for (auto &p : snap1) map1[p.pid] = p;
for (auto &p : snap2) map2[p.pid] = p;
long long total_delta = t2 - t1;
double cpu_total = 100.0 * (double)(total_delta - (idle2 - idle1)) / total_delta;
vector<Proc> result;
for (auto &[pid, p2] : map2) {
if (map1.find(pid) == map1.end()) continue;
unsigned long long delta = p2.cpu_jiffies - map1[pid].cpu_jiffies;
double cpu_usage = (double) delta * 100.0 * ncpu / total_delta;
result.push_back({pid, p2.name, p2.rss_kb, p2.cpu_jiffies, cpu_usage});
}
sort(result.begin(), result.end(), [](Proc a, Proc b) {
if (a.rss_kb != b.rss_kb) return a.rss_kb > b.rss_kb;
return a.cpu > b.cpu;
});
cout << "\033[H\033[J";
cout << "CPU Usage: " << fixed << setprecision(2) << cpu_total << "%\n";
cout << left << setw(8) << "PID" << setw(25) << "Process"
<< setw(12) << "Memory(KB)" << setw(8) << "CPU%\n";

for (int i = 0; i < min((int)result.size(), 20); i++) {
```

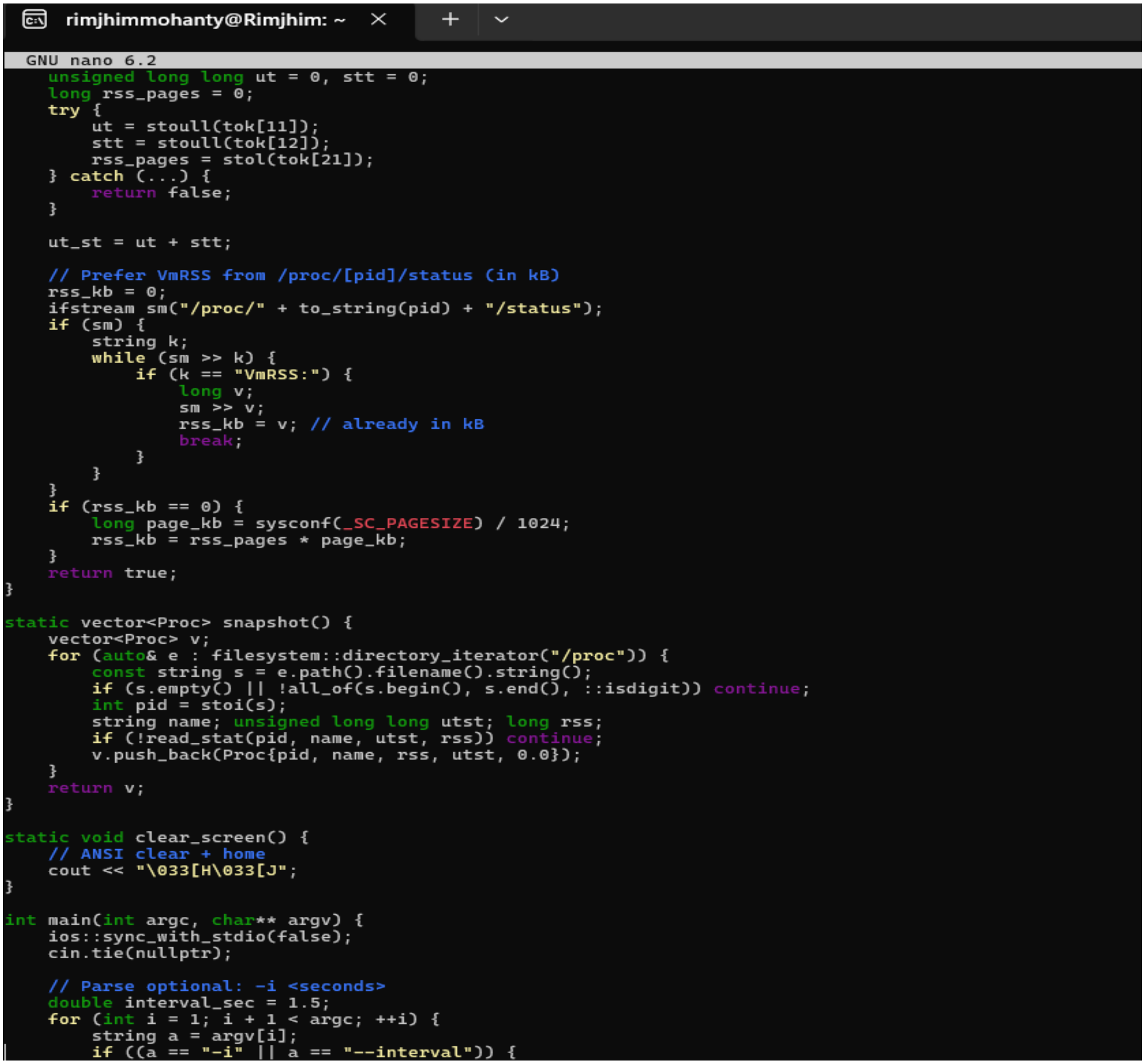
```
cout << left << setw(8) << result[i].pid
<< setw(25) << result[i].name.substr(0, 24)
<< setw(12) << result[i].rss_kb
          << setw(8) << fixed << setprecision(2) << result[i].cpu << endl;
     }
   }
}
```



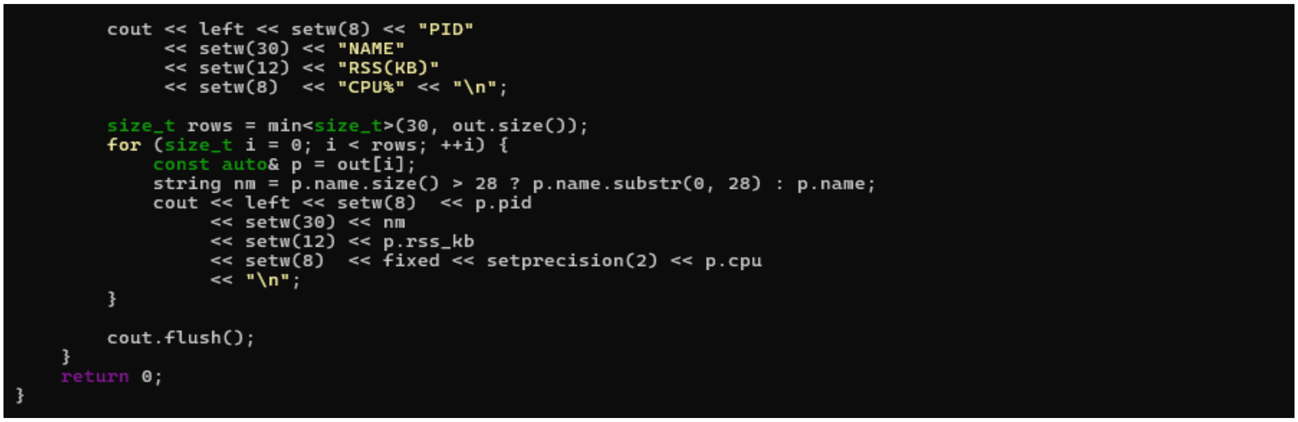
```
  GNU nano 6.2                                                              proc_monitor.cpp
#include <bits/stdc++.h>
#include <filesystem>
#include <unistd.h>

using namespace std;

struct Proc {
    int pid;
    string name;
    long rss_kb;
    unsigned long long cpu_jiffies;
    double cpu; // computed percent
};

// ---- CPU totals from /proc/stat ----
static inline long long read_total_jiffies() {
    ifstream f("/proc/stat");
    string tag;
    long long user, nice, system, idle, iowait, irq, softirq, steal, guest, guest_nice;
    if (!(f >> tag >> user >> nice >> system >> idle >> iowait >> irq >> softirq >> steal >> guest >> guest_nice))
        return 0;
    return user + nice + system + idle + iowait + irq + softirq + steal + guest + guest_nice;
}

static inline long long read_idle_jiffies() {
    ifstream f("/proc/stat");
    string tag;
    long long user, nice, system, idle, iowait, irq, softirq, steal, guest, guest_nice;
    if (!(f >> tag >> user >> nice >> system >> idle >> iowait >> irq >> softirq >> steal >> guest >> guest_nice))
        return 0;
    return idle + iowait;
}

// ---- Per-process read from /proc/[pid]/stat and /status ----
static bool read_stat(int pid, string& name, unsigned long long& ut_st, long& rss_kb) {
    // Read /proc/[pid]/stat (one line)
    ifstream st("/proc/" + to_string(pid) + "/stat");
    if (!st) return false;

    string line;
    getline(st, line);
    if (line.empty()) return false;

    // Extract name between the last '(' and the last ')'
    size_t lp = line.find('(');
    size_t rp = line.rfind(')');
    if (lp == string::npos || rp == string::npos || rp <= lp) return false;
    name = line.substr(lp + 1, rp - lp - 1);

    // Tokenize after ") "
    string after = line.substr(rp + 2);
    vector<string> tok;
    tok.reserve(64);
    {
        istringstream ss(after);
        string w;
        while (ss >> w) tok.push_back(w);
    }
    // After ')', tokens start at field #3 (state). Need at least up to rss (field #24 -> index 21 here).
    if (tok.size() < 22) return false;
```



```
            interval_sec = max(0.05, atof(argv[i + 1]));
        }
    }

    const long ncpu = max(1L, sysconf(_SC_NPROCESSORS_ONLN));

    while (true) {
        // First snapshot
        auto v1 = snapshot();
        long long tot1 = read_total_jiffies();
        long long idle1 = read_idle_jiffies();

        // Sleep for interval
        useconds_t us = static_cast<useconds_t>(interval_sec * 1e6);
        usleep(us);

        // Second snapshot
        auto v2 = snapshot();
        long long tot2 = read_total_jiffies();
        long long idle2 = read_idle_jiffies();

        // Build maps for delta
        unordered_map<int, Proc> m1, m2;
        m1.reserve(v1.size()); m2.reserve(v2.size());
        for (auto& p : v1) m1.emplace(p.pid, p);
        for (auto& p : v2) m2.emplace(p.pid, p);

        long long totald = max(1LL, tot2 - tot1);
        long long idled  = max(0LL, idle2 - idle1);
        double cpu_total_active = 100.0 * (double)(totald - idled) / (double)totald; // 0..100 (all CPUs)

        vector<Proc> out;
        out.reserve(m2.size());
        for (const auto& kv : m2) {
            int pid = kv.first;
            const Proc& p2 = kv.second;

            auto it = m1.find(pid);
            if (it == m1.end()) continue; // skip brand-new processes (no baseline)

            unsigned long long dproc = 0;
            if (p2.cpu_jiffies >= it->second.cpu_jiffies)
                dproc = p2.cpu_jiffies - it->second.cpu_jiffies;

            // Scale to single-CPU 0..100% semantics
            double cpu = 100.0 * (double)ncpu * (double)dproc / (double)totald;

            out.push_back(Proc{p2.pid, p2.name, p2.rss_kb, p2.cpu_jiffies, cpu});
        }

        // Sort: RSS desc, then CPU desc
        sort(out.begin(), out.end(), [](const Proc& a, const Proc& b) {
            if (a.rss_kb != b.rss_kb) return a.rss_kb > b.rss_kb;
            return a.cpu > b.cpu;
        });

        clear_screen();
        cout << "Interval " << fixed << setprecision(2) << interval_sec
             << "s | CPUs: " << ncpu
             << " | Active CPU " << fixed << setprecision(2) << cpu_total_active << "%\n";
```



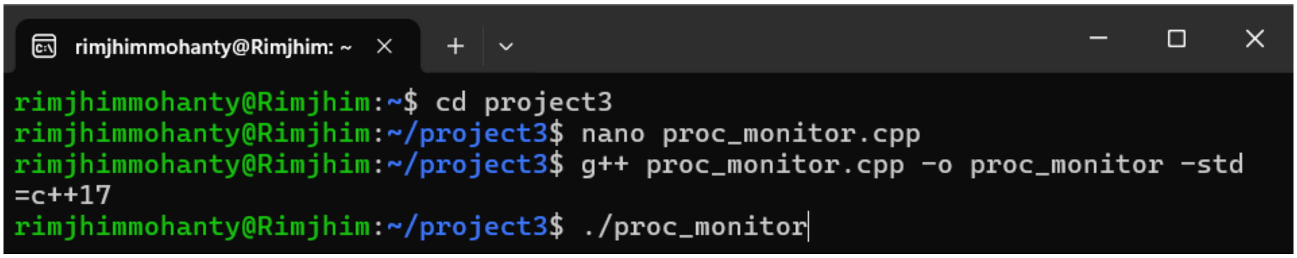
```
    unsigned long long ut = 0, stt = 0;
    long rss_pages = 0;
    try {
        ut = stoull(tok[11]);
        stt = stoull(tok[12]);
        rss_pages = stol(tok[21]);
    } catch (...) {
        return false;
    }

    ut_st = ut + stt;

    // Prefer VmRSS from /proc/[pid]/status (in kB)
    rss_kb = 0;
    ifstream sm("/proc/" + to_string(pid) + "/status");
    if (sm) {
        string k;
        while (sm >> k) {
            if (k == "VmRSS:") {
                long v;
                sm >> v;
                rss_kb = v; // already in kB
                break;
            }
        }
    }
    if (rss_kb == 0) {
        long page_kb = sysconf(_SC_PAGESIZE) / 1024;
        rss_kb = rss_pages * page_kb;
    }
    return true;
}

static vector<Proc> snapshot() {
    vector<Proc> v;
    for (auto& e : filesystem::directory_iterator("/proc")) {
        const string s = e.path().filename().string();
        if (s.empty() || !all_of(s.begin(), s.end(), ::isdigit)) continue;
        int pid = stoi(s);
        string name; unsigned long long utst; long rss;
        if (!read_stat(pid, name, utst, rss)) continue;
        v.push_back(Proc{pid, name, rss, utst, 0.0});
    }
    return v;
}

static void clear_screen() {
    // ANSI clear + home
    cout << "\033[H\033[J";
}

int main(int argc, char** argv) {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // Parse optional: -i <seconds>
    double interval_sec = 1.5;
    for (int i = 1; i + 1 < argc; ++i) {
        string a = argv[i];
        if ((a == "-i" || a == "--interval")) {
```



```
        cout << left << setw(8) << "PID"
             << setw(30) << "NAME"
             << setw(12) << "RSS(KB)"
             << setw(8)  << "CPU%" << "\n";

        size_t rows = min<size_t>(30, out.size());
        for (size_t i = 0; i < rows; ++i) {
            const auto& p = out[i];
            string nm = p.name.size() > 28 ? p.name.substr(0, 28) : p.name;
            cout << left << setw(8)  << p.pid
                 << setw(30) << nm
                 << setw(12) << p.rss_kb
                 << setw(8)  << fixed << setprecision(2) << p.cpu
                 << "\n";
        }

        cout.flush();
    }
    return 0;
}
```

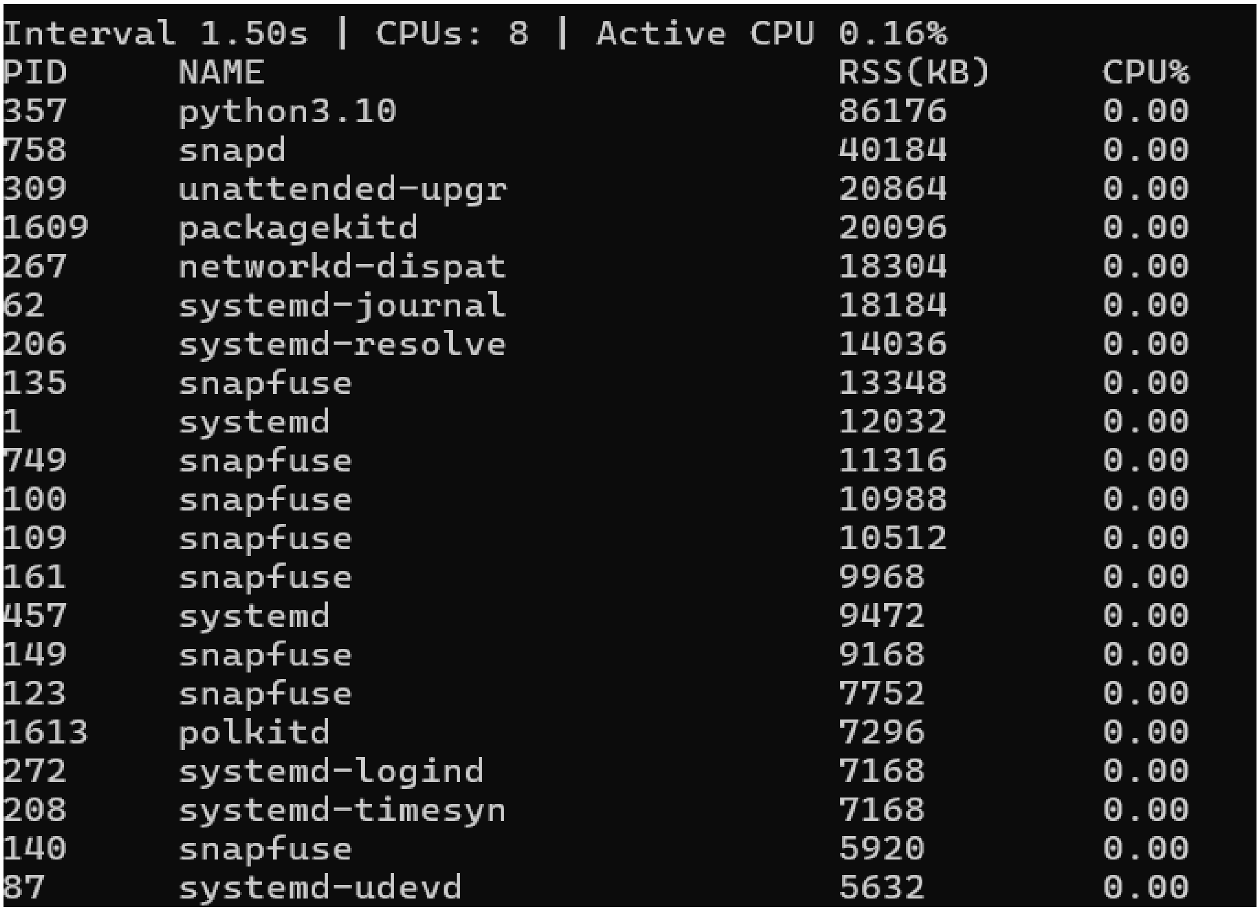
OUTPUT

Successful Compilation of the System Monitor Program



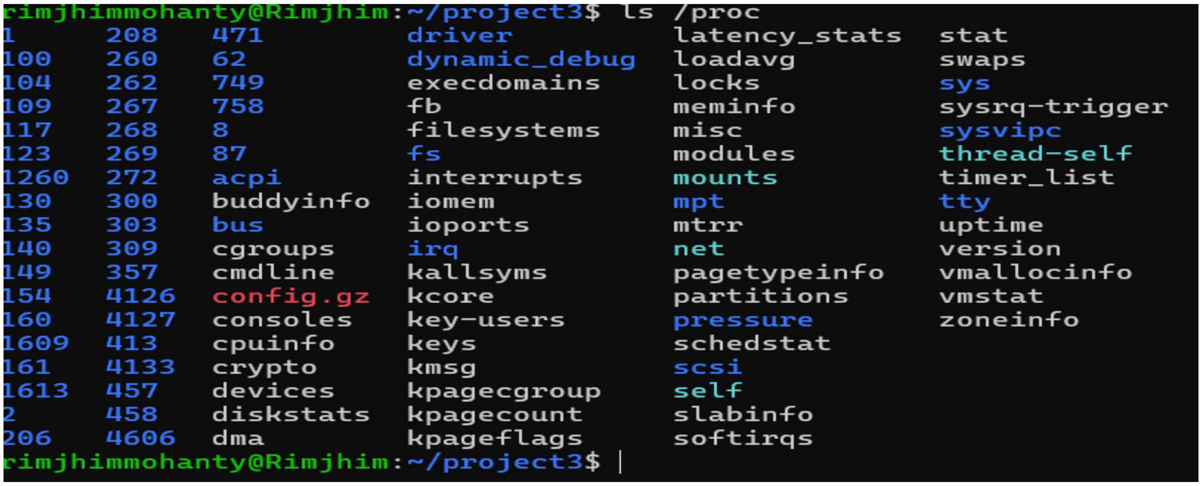
```
rimjhimmohanty@Rimjhim:~$ cd project3
rimjhimmohanty@Rimjhim:~/project3$ nano proc_monitor.cpp
rimjhimmohanty@Rimjhim:~/project3$ g++ proc_monitor.cpp -o proc_monitor -std
=c++17
rimjhimmohanty@Rimjhim:~/project3$ ./proc_monitor
```

Execution of the System Monitor Tool



```
Interval 1.50s | CPUs: 8 | Active CPU 0.16%
PID     NAME                          RSS(KB)     CPU%
357     python3.10                    86176       0.00
758     snapd                         40184       0.00
309     unattended-upgr               20864       0.00
1609    packagekitd                   20096       0.00
267     networkd-dispat               18304       0.00
62      systemd-journal               18184       0.00
206     systemd-resolve               14036       0.00
135     snapfuse                      13348       0.00
1       systemd                       12032       0.00
749     snapfuse                      11316       0.00
100     snapfuse                      10988       0.00
109     snapfuse                      10512       0.00
161     snapfuse                      9968        0.00
457     systemd                       9472        0.00
149     snapfuse                      9168        0.00
123     snapfuse                      7752        0.00
1613    polkitd                       7296        0.00
272     systemd-logind                7168        0.00
208     systemd-timesyn               7168        0.00
140     snapfuse                      5920        0.00
87      systemd-udevd                 5632        0.00
```

Viewing the /proc Directory in Linux



```
rimjhimmohanty@Rimjhim:~/project3$ ls /proc
1     208   471        driver         latency_stats  stat
100   260   62         dynamic_debug  loadavg        swaps
104   262   749        execdomains    locks          sys
109   267   758        fb             meminfo        sysrq-trigger
117   268   8          filesystems    misc           sysvipc
123   269   87         fs             modules        thread-self
1260  272   acpi       interrupts     mounts         timer_list
130   300   buddyinfo  iomem          mpt            tty
135   303   bus        ioports        mtrr           uptime
140   309   cgroups    irq            net            version
149   357   cmdline    kallsyms       pagetypeinfo   vmallocinfo
154   4126  config.gz  kcore          partitions     vmstat
160   4127  consoles   key-users      pressure       zoneinfo
1609  413   cpuinfo    keys           schedstat
161   4133  crypto     kmsg           scsi
1613  457   devices    kpagecgroup    self
2     458   diskstats  kpagecount     slabinfo
206   4606  dma        kpageflags     softirqs
rimjhimmohanty@Rimjhim:~/project3$ |
```

Conclusion

The System Monitor Tool effectively tracks real-time CPU and memory usage of running processes using data from the Linux /proc filesystem. It functions as a lightweight alternative to the top command while enhancing practical knowledge of system programming, process management, and C++ file handling. Overall, the project is efficient, educational, and provides a strong foundation for future enhancements.