# WIPRO DAY – 1

## STEP-1

```
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ mkdir WIPRODAY1
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ cd WIPRODAY1/
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ nano day1.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ g++ day1.cpp -o day1
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ ./day1
Enter the number: 8
8 is evenpapali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ ./day1
Enter the number: 9
9 is oddpapali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ nano day1q2.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ g++ day1q2.cpp -o day1q2
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ ./day1q2
*
**
***
****
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ nano wiproday1.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ g++ wiproday1.cpp -o wiproday1
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$ ./wiproday1
Total RAM:7650MB
Total RAM:7GB
Total RAM:5248MB
Total RAM:5GB
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY1$
```

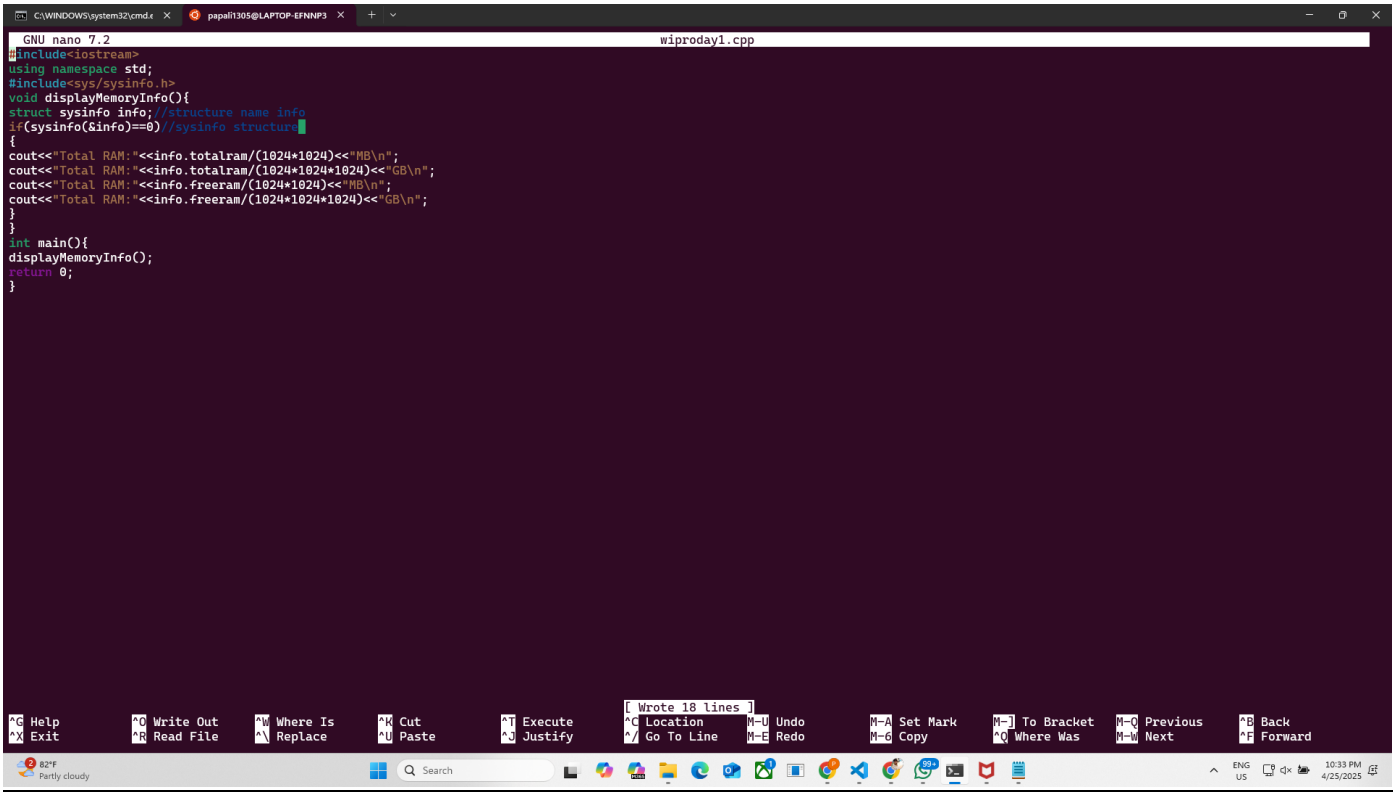
## STEP – 2

```
GNU nano 7.2                                                    day1.cpp *
#include<iostream>
using namespace std; // to avoid naming collision

int main() {
    int num;
    cout << "Enter the number: ";
    cin >> num;

    if(num % 2 == 0)
        cout << num << " is even";
    else
        cout << num << " is odd";

    return 0;
}
```

## STEP – 3

```
GNU nano 7.2                                                    day1q2.cpp *
#include<iostream>
using namespace std;
int main(){
int a=4;
for(int i = 0; i< a;i++){
for(int j=0;j<=i;j++){
cout<< "*";
}
cout<<endl;
}
return 0;
}
```

## STEP – 4




## CODE

```
#include<iostream>
using namespace std;
#include<sys/sysinfo.h>
void displayMemoryInfo(){
struct sysinfo info;//structure name info
if(sysinfo(&info)==0)//sysinfo structure
{
cout<<"Total RAM:"<<info.totalram/(1024*1024)<<"MB\n";
cout<<"Total RAM:"<<info.totalram/(1024*1024*1024)<<"GB\n";
cout<<"Total RAM:"<<info.freeram/(1024*1024)<<"MB\n";
cout<<"Total RAM:"<<info.freeram/(1024*1024*1024)<<"GB\n";
}
}
int main(){
displayMemoryInfo();
return 0;
}
```

# WIPRO DAY – 2

## STEP – 1

```
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ mkdir WIPRODAY2
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ cd WIPRODAY2
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY2$ nano day2.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY2$ g++ day2.cpp -o day2 -pthread
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY2$ ./day2
CPU Usage: 0.124844%
user: 24646
nice: 909
system: 40575
idle: 55814897
iowait: 10908
irq: 0
softirq: 4717
steal: 0
guest: 0
guest_nice: 0
Total CPU Time: 55896652
Idle Time: 55825805
Recalculated CPU Usage: 0.126746%
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY2$
```

## STEP – 2

```
GNU nano 7.2                                                    day2.cpp *
#include<iostream>     // for input output
#include<fstream>      // for file operations
#include<sstream>      // for parsing file content
#include<thread>       // to introduce delay
#include<chrono>       // to introduce delay
using namespace std;

struct CPUData {
    long user, nice, system, idle, iowait, irq, softirq, steal, guest, guest_nice;
};

CPUData getCPUData() {
    ifstream file("/proc/stat");
    string line;
    CPUData cpu = {}; // Initialize with zeros

    if (file.is_open()) {
        getline(file, line); // Read the first line
        istringstream ss(line);
        string cpuLabel;

        ss >> cpuLabel >> cpu.user >> cpu.nice >> cpu.system >> cpu.idle >> cpu.iowait
           >> cpu.irq >> cpu.softirq >> cpu.steal >> cpu.guest >> cpu.guest_nice;
    }

    return cpu;
}

double calculateCPUUsage(CPUData prev, CPUData current) {
    long prevIdle = prev.idle + prev.iowait;
    long currIdle = current.idle + current.iowait;

    long prevTotal = prev.user + prev.nice + prev.system + prev.idle + prev.iowait +
                     prev.irq + prev.softirq + prev.steal + prev.guest + prev.guest_nice;

    long currTotal = current.user + current.nice + current.system + current.idle + current.iowait +
                     current.irq + current.softirq + current.steal + current.guest + current.guest_nice;

    long totalDiff = currTotal - prevTotal;
    long idleDiff = currIdle - prevIdle;

    return (totalDiff - idleDiff) * 100.0 / totalDiff;
}

int main() {

^G Help      ^O Write Out   ^W Where Is    ^K Cut       ^T Execute   ^C Location   M-U Undo     M-A Set Mark   M-] To Bracket  M-Q Previous   ^B Back
^X Exit      ^R Read File   ^\ Replace     ^U Paste     ^J Justify   ^/ Go To Line M-E Redo     M-6 Copy       ^Q Where Was    M-W Next       ^F Forward
```

## STEP -3(ACTIVE)

```
System information as of Fri Apr 25 14:50:37 UTC 2025

  System load:  0.97               Processes:             85
  Usage of /:   0.2% of 1006.85GB  Users logged in:       0
  Memory usage: 16%                IPv4 address for eth0: 172.22.232.94
  Swap usage:   0%


This message is shown once a day. To disable it please create the
/home/papali1305/.hushlogin file.
papali1305@LAPTOP-EFNNP3TA:~$ mkdir wiprolsp
papali1305@LAPTOP-EFNNP3TA:~$ cd wiprolsp/
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ nano active_process.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ nano active_process.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ g++ active_process.cpp -o active_process
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ ./active_process
Active Processes:
PID:1 |Name: (systemd)
PID:2 |Name: (init-systemd(Ub)
PID:6 |Name: (init)
PID:54 |Name: (systemd-journal)
PID:96 |Name: (systemd-udevd)
PID:109 |Name: (snapfuse)
PID:110 |Name: (snapfuse)
PID:115 |Name: (snapfuse)
PID:124 |Name: (snapfuse)
PID:128 |Name: (snapfuse)
PID:132 |Name: (snapfuse)
PID:222 |Name: (systemd-resolve)
PID:223 |Name: (systemd-timesyn)
PID:229 |Name: (cron)
PID:230 |Name: (dbus-daemon)
PID:242 |Name: (snapd)
PID:243 |Name: (systemd-logind)
PID:246 |Name: (wsl-pro-service)
PID:261 |Name: (rsyslogd)
PID:267 |Name: (agetty)
PID:281 |Name: (unattended-upgr)
PID:386 |Name: (SessionLeader)
PID:388 |Name: (Relay(394))
PID:394 |Name: (bash)
PID:395 |Name: ((agetty))
PID:397 |Name: (login)
PID:487 |Name: (systemd)
PID:488 |Name: ((sd-pam))
PID:503 |Name: (bash)
PID:1102 |Name: (polkitd)
PID:1267 |Name: (active_process)
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$
```

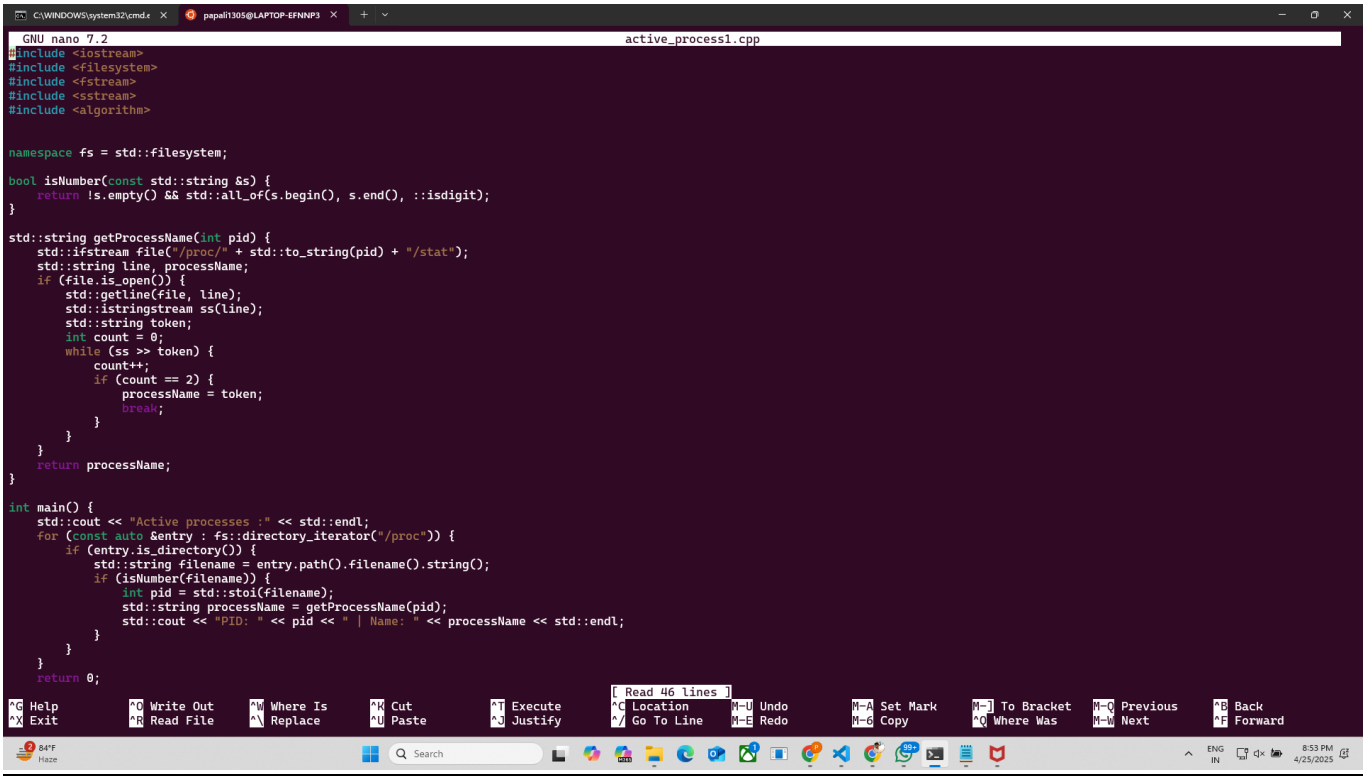
## STEP – 4

```
GNU nano 7.2                                                    active_process.cpp
#include<iostream>
#include<algorithm>
#include<filesystem>
#include<fstream>
#include<sstream>


namespace fs = std::filesystem;

bool isNumber(const std::string &s){//write the function to check if a directory is a number or not
   return !s.empty() && all_of(s.begin(), s.end(), ::isdigit);// its will return true or false
}// check all the character in digits or not

std::string getProcessName(int pid){
        std::ifstream file("/proc/" + std::to_string(pid) + "/stat");//pid = process id

        std::string line, processName;

         if(file.is_open())
         {
             std::getline(file, line);//read the first line into the stream
             std::istringstream ss(line);//ss-it is an object
             std::string token;

             int count=0;
             while(ss >> token){
                count++;
                if(count==2){// it means it will show us the process name retrieve
                   processName=token;
                   break;
                  }
                 }
                }
                return processName;
               }
 int main()
 {
 std::cout <<"Active Processes:\n";
     for(const auto &entry : fs::directory_iterator("/proc")) //or("/proc")){ //

         if(entry.is_directory()) //files and folders both are present so we have to retrieve the folder
         {//check if directory is a process id // each process id has a directory
            std::string filename=entry.path().filename().string();

          if(isNumber(filename))
                                                    [ Read 54 lines ]
^G Help     ^O Write Out  ^W Where Is   ^K Cut      ^T Execute    ^C Location   M-U Undo     M-A Set Mark  M-] To Bracket  M-Q Previous  ^B Back
^X Exit     ^R Read File  ^\ Replace    ^U Paste    ^J Justify    ^/ Go To Line M-E Redo     M-6 Copy      ^Q Where Was    M-W Next      ^F Forward
```

## STEP – 5 (ACTIVE1)

```
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ nano active_process1.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ g++ active_process.cpp -o active_process
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ g++ active_process1.cpp -o active_process1
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ ./active_process1
Active processes :
PID: 1 | Name: (systemd)
PID: 2 | Name: (init-systemd(Ub)
PID: 6 | Name: (init)
PID: 54 | Name: (systemd-journal)
PID: 96 | Name: (systemd-udevd)
PID: 109 | Name: (snapfuse)
PID: 110 | Name: (snapfuse)
PID: 115 | Name: (snapfuse)
PID: 124 | Name: (snapfuse)
PID: 128 | Name: (snapfuse)
PID: 132 | Name: (snapfuse)
PID: 222 | Name: (systemd-resolve)
PID: 223 | Name: (systemd-timesyn)
PID: 229 | Name: (cron)
PID: 230 | Name: (dbus-daemon)
PID: 242 | Name: (snapd)
PID: 243 | Name: (systemd-logind)
PID: 246 | Name: (wsl-pro-service)
PID: 261 | Name: (rsyslogd)
PID: 267 | Name: (agetty)
PID: 281 | Name: (unattended-upgr)
PID: 386 | Name: (SessionLeader)
PID: 388 | Name: (Relay(394))
PID: 394 | Name: (bash)
PID: 395 | Name: ((agetty))
PID: 397 | Name: (login)
PID: 487 | Name: (systemd)
PID: 488 | Name: ((sd-pam))
PID: 503 | Name: (bash)
PID: 1102 | Name: (polkitd)
PID: 1318 | Name: (active_process1)
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$
```

## STEP – 6




## CODE

```
#include<iostream>    // for input output

#include<fstream>     // for file operations

#include<sstream>     // for parsing file content

#include<thread>      // to introduce delay

#include<chrono>      // to introduce delay

using namespace std;


struct CPUData {

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

};
```

```
CPUData getCPUData() {
   ifstream file("/proc/stat");
   string line;
   CPUData cpu = {}; // Initialize with zeros

   if (file.is_open()) {
      getline(file, line); // Read the first line
      istringstream ss(line);
      string cpuLabel;

      ss >> cpuLabel >> cpu.user >> cpu.nice >> cpu.system >> cpu.idle >> cpu.iowait
         >> cpu.irq >> cpu.softirq >> cpu.steal >> cpu.guest >> cpu.guest_nice;
   }

   return cpu;
}

double calculateCPUUsage(CPUData prev, CPUData current) {
   long prevIdle = prev.idle + prev.iowait;
   long currIdle = current.idle + current.iowait;

   long prevTotal = prev.user + prev.nice + prev.system + prev.idle + prev.iowait +
                    prev.irq + prev.softirq + prev.steal + prev.guest + prev.guest_nice;
```

```
    long currTotal = current.user + current.nice + current.system + current.idle + current.iowait
+
                    current.irq + current.softirq + current.steal + current.guest + current.guest_nice;

    long totalDiff = currTotal - prevTotal;
    long idleDiff = currIdle - prevIdle;

    return (totalDiff - idleDiff) * 100.0 / totalDiff;
}

int main() {
    CPUData prevData = getCPUData();
    this_thread::sleep_for(chrono::seconds(1)); // Wait for 1 second
    CPUData currData = getCPUData();

    double cpuUsage = calculateCPUUsage(prevData, currData);

    cout << "CPU Usage: " << cpuUsage << "%\n";
    cout << "user: " << currData.user << "\n";
    cout << "nice: " << currData.nice << "\n";
    cout << "system: " << currData.system << "\n";
    cout << "idle: " << currData.idle << "\n";
    cout << "iowait: " << currData.iowait << "\n";
    cout << "irq: " << currData.irq << "\n";
```

```
    cout << "softirq: " << currData.softirq << "\n";
    cout << "steal: " << currData.steal << "\n";
    cout << "guest: " << currData.guest << "\n";
    cout << "guest_nice: " << currData.guest_nice << "\n";

    long total_cpu_time = currData.user + currData.nice + currData.system + currData.idle +
                          currData.iowait + currData.irq + currData.softirq + currData.steal +
                          currData.guest + currData.guest_nice;

    int idle_time = currData.idle + currData.iowait;
    cout << "Total CPU Time: " << total_cpu_time << "\n";
    cout << "Idle Time: " << idle_time << "\n";

    double cpu_usage = ((total_cpu_time - idle_time) / (double)total_cpu_time) * 100;
    cout << "Recalculated CPU Usage: " << cpu_usage << "%\n";

    return 0;
}
```

# WIPRO DAY – 3

## STEP – 1

```
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ mkdir WIPRODAY3
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ cd WIPRODAY3
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY3$ nano day03.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY3$ g++ day03.cpp -o day03
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY3$ ./day03
PID     CPU%    Memory (kb)     Name
2472    7.14286%        9192    (day03)
6       2.20412%        4607    (init)
128     0.0857518%      4607    (snapfuse)
124     0.0603439%      4607    (snapfuse)
109     0.0404939%      4607    (snapfuse)
132     0.0272228%      4607    (snapfuse)
242     0.016449%       6480    (snapd)
1       0.0140635%      4607    (systemd)
115     0.0107757%      4607    (snapfuse)
54      0.00816684%     4607    (systemd-journal)
```

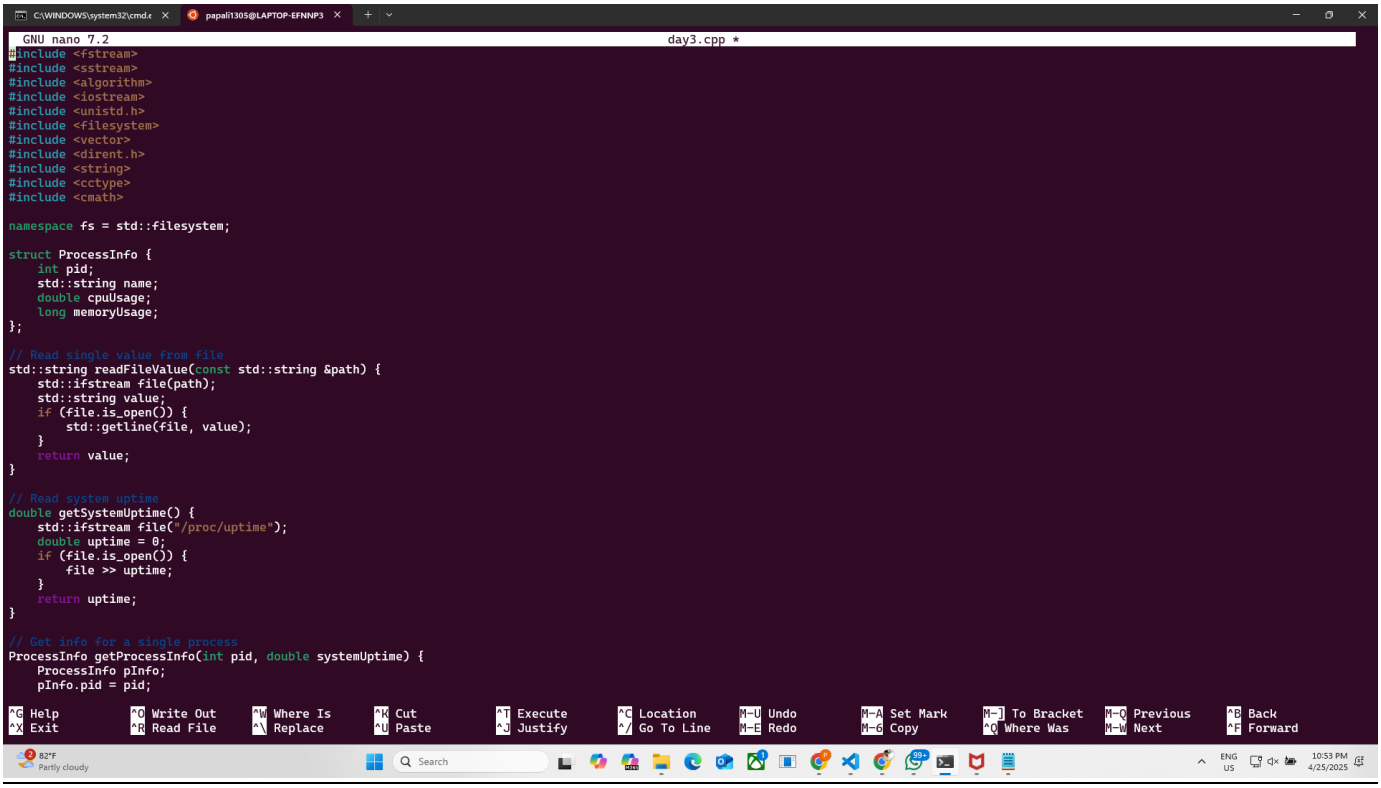
## STEP – 2

```
  GNU nano 7.2                                                    day03.cpp *
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <unistd.h>
#include <dirent.h> // For directory handling
#include <cstring> // For strcmp

struct Processinfo {
    int pid;
    std::string name;
    double cpuUsage;
    long memoryUsage;
};

std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
        std::getline(file, value);
    }
    return value;
}

double getSystemUptime() {
    std::ifstream file("/proc/uptime");
    double uptime;
    if (file.is_open()) {
        file >> uptime; // extract uptime only, not ideal time
    }
    return uptime;
}

Processinfo getProcessInfo(int pid, double systemUptime) {
    Processinfo pInfo;
    pInfo.pid = pid;

    std::ifstream statFile("/proc/" + std::to_string(pid) + "/stat");
    std::string line;
    long utime, stime, starttime;
    if (statFile.is_open()) {
        std::getline(statFile, line);
        std::istringstream ss(line);
        std::string token;

^G Help      ^O Write Out   ^W Where Is    ^K Cut       ^T Execute   ^C Location  M-U Undo      M-A Set Mark   M-] To Bracket  M-Q Previous   ^B Back
^X Exit      ^R Read File   ^\ Replace     ^U Paste     ^J Justify   ^/ Go To Line M-E Redo      M-6 Copy       ^Q Where Was    M-W Next       ^F Forward
```

## STEP -3

```
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY3$ nano day3.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY3$ g++ -std=c++17 -o day3 day3.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY3$ ./day3
PID     CPU%    Memory (kB)     Name
6       2.11603%        4607    (init)
128     0.0823242%      4607    (snapfuse)
124     0.0579319%      4607    (snapfuse)
109     0.0388753%      4607    (snapfuse)
132     0.0261347%      4607    (snapfuse)
242     0.0161181%      6480    (snapd)
1       0.013937%       4607    (systemd)
115     0.010345%       4607    (snapfuse)
54      0.00805819%     4607    (systemd-journal)
388     0.00707968%     6480    (Relay(394))
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY3$
```

## STEP – 4




## CODE

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <unistd.h>
#include <dirent.h> // For directory handling
#include <cstring> // For strcmp

struct Processinfo {
    int pid;
    std::string name;
    double cpuUsage;
    long memoryUsage;
};

std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
```

```
        std::getline(file, value);
    }
    return value;
}

double getSystemUptime() {
    std::ifstream file("/proc/uptime");
    double uptime;
    if (file.is_open()) {
        file >> uptime; // extract uptime only, not ideal time
    }
    return uptime;
}

Processinfo getProcessInfo(int pid, double systemUptime) {
    Processinfo pInfo;
    pInfo.pid = pid;

    std::ifstream statFile("/proc/" + std::to_string(pid) + "/stat");
    std::string line;
    long utime, stime, starttime;
    if (statFile.is_open()) {
        std::getline(statFile, line);
        std::istringstream ss(line);
        std::string token;
        int count = 0;
        while (ss >> token) {
            count++;
            if (count == 2) pInfo.name = token;
            else if (count == 14) utime = std::stol(token);
            else if (count == 15) stime = std::stol(token);
            else if (count == 22) starttime = std::stol(token);
        }
    }

    std::ifstream memFile("/proc/" + std::to_string(pid) + "/status");
    if (memFile.is_open()) {
        std::string key, value, unit;
        while (memFile >> key >> value >> unit) {
            if (key == "VmRSS:") {
                pInfo.memoryUsage = std::stol(value);
                break;
            }
        }
    }
```

```
    long total_time = utime + stime;
    double seconds = systemUptime - (starttime / sysconf(_SC_CLK_TCK));
    pInfo.cpuUsage = (static_cast<double>(total_time) / sysconf(_SC_CLK_TCK)) /
seconds * 100;

    return pInfo;
}

std::vector<Processinfo> getAllProcesses() {
    std::vector<Processinfo> processes;
    double systemUptime = getSystemUptime();

    DIR *dir = opendir("/proc");
    struct dirent *entry;
    if (dir != nullptr) {
        while ((entry = readdir(dir)) != nullptr) {
            if (entry->d_type == DT_DIR) {
                // Check if the directory name is a number (PID)
                if (std::all_of(entry->d_name, entry->d_name + strlen(entry-
>d_name), ::isdigit)) {
                    int pid = std::stoi(entry->d_name);
                    processes.push_back(getProcessInfo(pid, systemUptime));
                }
            }
        }
        closedir(dir);
    }
    return processes;
}

void sortProcesses(std::vector<Processinfo> &processes, bool sortByCPU) {
    if (sortByCPU) {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.cpuUsage > b.cpuUsage;
        });
    } else {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.memoryUsage > b.memoryUsage;
        });
    }
}
```

```
int main() {
    std::vector<Processinfo> processes = getAllProcesses();
    sortProcesses(processes, true);
    std::cout << "PID\tCPU%\tMemory (kb)\tName\n";
    for (size_t i = 0; i < std::min(processes.size(), size_t(10)); i++) {
        std::cout << processes[i].pid << "\t"
                  << processes[i].cpuUsage << "%\t"
                  << processes[i].memoryUsage << "\t"
                  << processes[i].name << "\n";
    }

    // size_t

    return 0;
}
```

```
#include <fstream>
#include <sstream>
#include <algorithm>
#include <iostream>
#include <unistd.h>
#include <filesystem>
#include <vector>
#include <dirent.h>
#include <string>
#include <cctype>
#include <cmath>

namespace fs = std::filesystem;

struct ProcessInfo {
    int pid;
    std::string name;
    double cpuUsage;
    long memoryUsage;
};

// Read single value from file
std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
        std::getline(file, value);
```

```
    }
    return value;
}

// Read system uptime
double getSystemUptime() {
    std::ifstream file("/proc/uptime");
    double uptime = 0;
    if (file.is_open()) {
        file >> uptime;
    }
    return uptime;
}

// Get info for a single process
ProcessInfo getProcessInfo(int pid, double systemUptime) {
    ProcessInfo pInfo;
    pInfo.pid = pid;

    std::ifstream file("/proc/" + std::to_string(pid) + "/stat");
    std::string line;
    long utime = 0, stime = 0, starttime = 0;

    if (file.is_open()) {
        std::getline(file, line);
        std::istringstream ss(line);
        std::string token;
        int count = 0;

        while (ss >> token) {
            count++;
            if (count == 2)
                pInfo.name = token;
            else if (count == 14)
                utime = std::stol(token);
            else if (count == 15)
                stime = std::stol(token);
            else if (count == 22)
                starttime = std::stol(token);
        }

        // Get memory usage
        std::ifstream memFile("/proc/" + std::to_string(pid) + "/status");
        if (memFile.is_open()) {
            std::string key, value, unit;
```

```
            while (memFile >> key >> value >> unit) {
                if (key == "VmRSS:") {
                    pInfo.memoryUsage = std::stol(value);
                    break;
                }
            }
        }

        long total_time = utime + stime;
        double seconds = systemUptime - (starttime / sysconf(_SC_CLK_TCK));

        if (seconds > 0)
            pInfo.cpuUsage = (total_time /
static_cast<double>(sysconf(_SC_CLK_TCK))) / seconds * 100.0;
        else
            pInfo.cpuUsage = 0.0;
    }

    return pInfo;
}

// Get all processes
std::vector<ProcessInfo> getAllProcesses() {
    std::vector<ProcessInfo> processes;
    double systemUptime = getSystemUptime();

    for (const auto &entry : fs::directory_iterator("/proc")) {
        if (entry.is_directory()) {
            std::string filename = entry.path().filename().string();
            if (std::all_of(filename.begin(), filename.end(), ::isdigit)) {
                int pid = std::stoi(filename);
                processes.push_back(getProcessInfo(pid, systemUptime));
            }
        }
    }
    return processes;
}

// Sort processes
void sortProcesses(std::vector<ProcessInfo> &processes, bool sortByCPU) {
    if (sortByCPU) {
        std::sort(processes.begin(), processes.end(), [](const ProcessInfo &a,
const ProcessInfo &b) {
            return a.cpuUsage > b.cpuUsage;
        });
```

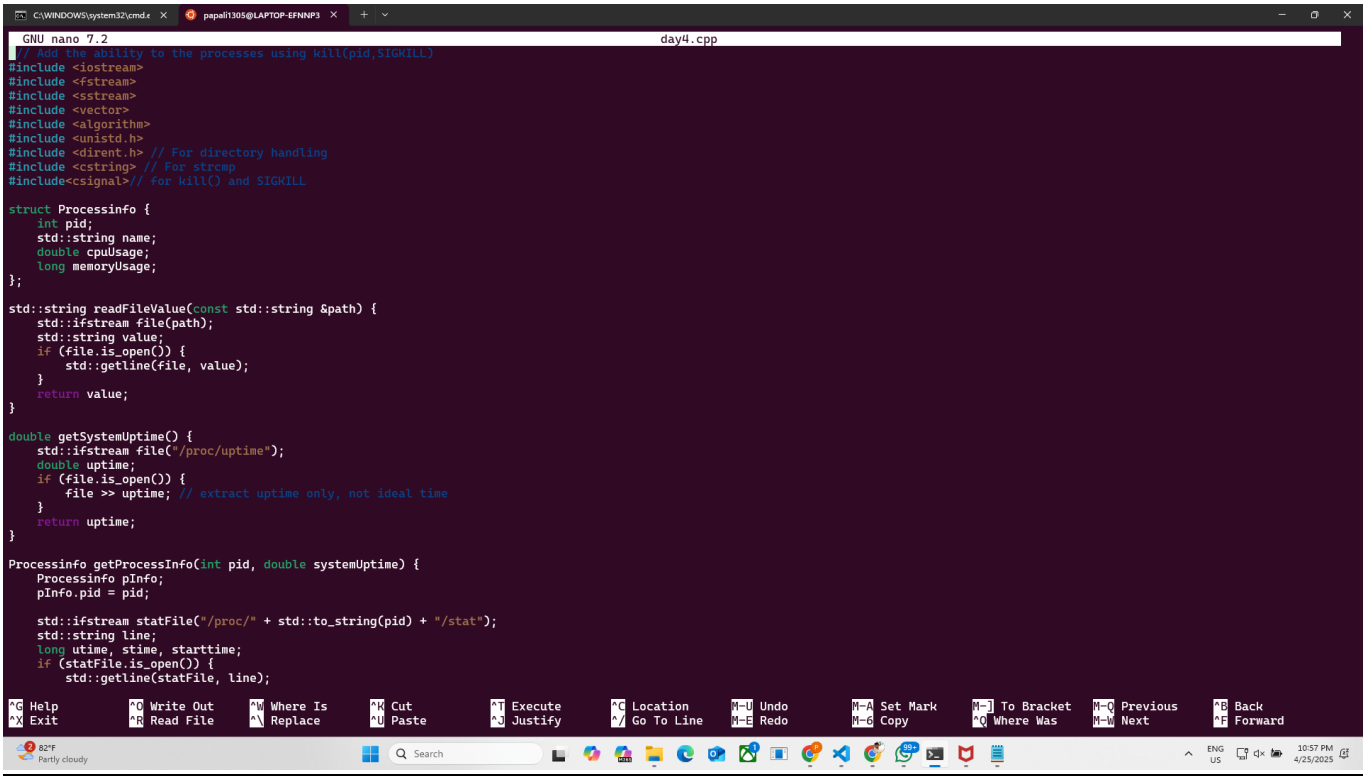
```
    } else {
        std::sort(processes.begin(), processes.end(), [](const ProcessInfo &a,
const ProcessInfo &b) {
            return a.memoryUsage > b.memoryUsage;
        });
    }
}

// Main
int main() {
    std::vector<ProcessInfo> processes = getAllProcesses();
    sortProcesses(processes, true);  // true = sort by CPU

    std::cout << "PID\tCPU%\tMemory (kB)\tName\n";
    for (size_t i = 0; i < std::min(processes.size(), size_t(10)); ++i) {
        std::cout << processes[i].pid << "\t"
                  << processes[i].cpuUsage << "%\t"
                  << processes[i].memoryUsage << "\t"
                  << processes[i].name << "\n";
    }

    return 0;
}
```

# WIPRO DAY – 4

## STEP – 1

```
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ mkdir WIPRODAY4
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp$ cd WIPRODAY4
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY4$ nano day4.cpp
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY4$ g++ day4.cpp -o day4
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY4$ ./day4
PID     CPU%    Memory (kb)     Name
6       2.05878%        4607    (init)
128     0.0800887%      4607    (snapfuse)
124     0.0563587%      4607    (snapfuse)
109     0.0378197%      4607    (snapfuse)
132     0.025425%       4607    (snapfuse)
242     0.0159983%      6480    (snapd)
1       0.0135586%      4607    (systemd)
115     0.0100641%      4607    (snapfuse)
54      0.00794531%     4607    (systemd-journal)
388     0.00731124%     6480    (Relay(394))
Enter PID to kill :124
Failed to kill process: Operation not permitted
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY4$ ./day4
PID     CPU%    Memory (kb)     Name
6       2.05144%        4607    (init)
128     0.0798029%      4607    (snapfuse)
124     0.0561576%      4607    (snapfuse)
109     0.0376847%      4607    (snapfuse)
132     0.025343%       4607    (snapfuse)
242     0.0159412%      6480    (snapd)
1       0.0135102%      4607    (systemd)
115     0.0100281%      4607    (snapfuse)
54      0.00791695%     4607    (systemd-journal)
388     0.00728514%     6480    (Relay(394))
Enter PID to kill :388
Failed to kill process: Operation not permitted
papali1305@LAPTOP-EFNNP3TA:~/wiprolsp/WIPRODAY4$
```

## STEP – 2




```cpp
// Add the ability to the processes using kill(pid,SIGKILL)
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <unistd.h>
#include <dirent.h> // For directory handling
#include <cstring> // For strcmp
#include<csignal>// for kill() and SIGKILL

struct Processinfo {
    int pid;
    std::string name;
    double cpuUsage;
    long memoryUsage;
};

std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
        std::getline(file, value);
    }
```

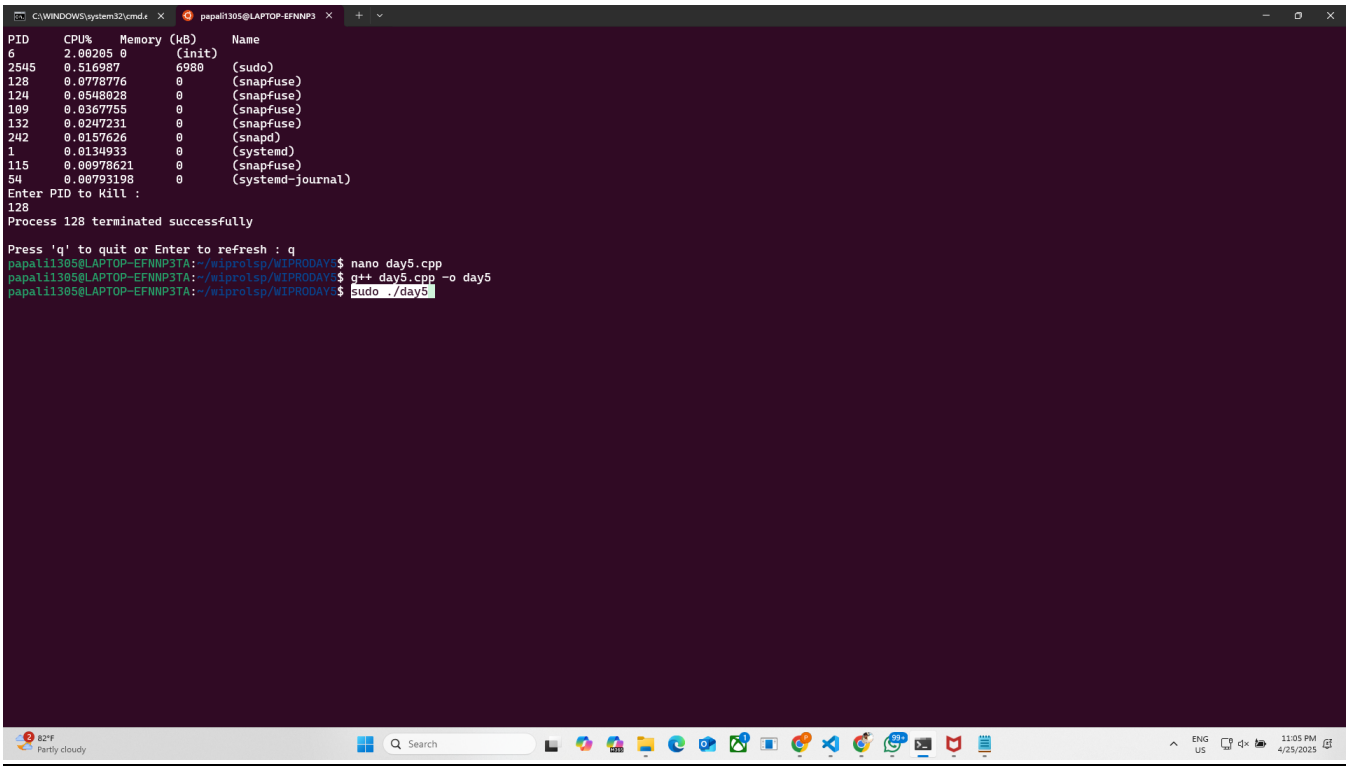
```
    return value;
}

double getSystemUptime() {
    std::ifstream file("/proc/uptime");
    double uptime;
    if (file.is_open()) {
        file >> uptime; // extract uptime only, not ideal time
    }
    return uptime;
}

Processinfo getProcessInfo(int pid, double systemUptime) {
    Processinfo pInfo;
    pInfo.pid = pid;

    std::ifstream statFile("/proc/" + std::to_string(pid) + "/stat");
    std::string line;
    long utime, stime, starttime;
    if (statFile.is_open()) {
        std::getline(statFile, line);
        std::istringstream ss(line);
        std::string token;
        int count = 0;
        while (ss >> token) {
            count++;
            if (count == 2) pInfo.name = token;
            else if (count == 14) utime = std::stol(token);
            else if (count == 15) stime = std::stol(token);
            else if (count == 22) starttime = std::stol(token);
        }
    }

    std::ifstream memFile("/proc/" + std::to_string(pid) + "/status");
    if (memFile.is_open()) {
        std::string key, value, unit;
        while (memFile >> key >> value >> unit) {
            if (key == "VmRSS:") {
                pInfo.memoryUsage = std::stol(value);
                break;
            }
        }
    }

    long total_time = utime + stime;
```

```
    double seconds = systemUptime - (starttime / sysconf(_SC_CLK_TCK));
    pInfo.cpuUsage = (static_cast<double>(total_time) / sysconf(_SC_CLK_TCK)) /
seconds * 100;

    return pInfo;
}

std::vector<Processinfo> getAllProcesses() {
    std::vector<Processinfo> processes;
    double systemUptime = getSystemUptime();

    DIR *dir = opendir("/proc");
    struct dirent *entry;
    if (dir != nullptr) {
        while ((entry = readdir(dir)) != nullptr) {
            if (entry->d_type == DT_DIR) {
                // Check if the directory name is a number (PID)
                if (std::all_of(entry->d_name, entry->d_name + strlen(entry-
>d_name), ::isdigit)) {
                    int pid = std::stoi(entry->d_name);
                    processes.push_back(getProcessInfo(pid, systemUptime));
                }
            }
        }
        closedir(dir);
    }
    return processes;
}

void sortProcesses(std::vector<Processinfo> &processes, bool sortByCPU) {
    if (sortByCPU) {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.cpuUsage > b.cpuUsage;
        });
    } else {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.memoryUsage > b.memoryUsage;
        });
    }
}

int main() {
```

```
    std::vector<Processinfo> processes = getAllProcesses();
    sortProcesses(processes, true);
    std::cout << "PID\tCPU%\tMemory (kb)\tName\n";
    for (size_t i = 0; i < std::min(processes.size(), size_t(10)); i++) {
        std::cout << processes[i].pid << "\t"
                  << processes[i].cpuUsage << "%\t"
                  << processes[i].memoryUsage << "\t"
                  << processes[i].name << "\n";
    }
    int targetPid;

std::cout <<"Enter PID to kill :";
std::cin >> targetPid;

// use kill() function

//signature : int kil(pid_t pid, int sig);

// kill return integer type data
//pid : Process ID
//ig : Signal to send (SIGKILL, SIGTERM, etc.)

if (targetPid>0)
{
if(kill(targetPid, SIGKILL)==0)
{
std::cout << "Process" << targetPid << "terminated successfully";
}
else
{
perror("Failed to kill process");
}
}

    // size_t

    return 0;
}
```

# WIPRO DAY – 5

## STEP – 1




## STEP – 2

```
GNU nano 7.2                                                    day5.cpp *
// All real-time refresh  feature using sleep() and loop
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <filesystem>
#include <unistd.h>
#include <dirent.h> // For directory handling
#include <cstring> // For strcmp
#include<csignal>// for kill() and SIGKILL
#include<chrono>
#include<thread>

namespace fs = std::filesystem;
using namespace std;

struct Processinfo {
    int pid;
    std::string name;
    double cpuUsage;
    long memoryUsage;
};

std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
        std::getline(file, value);
    }
    return value;
}

double getSystemUptime() {
    std::ifstream file("/proc/uptime");
    double uptime;
    if (file.is_open()) {
        file >> uptime; // extract uptime only, not ideal time
    }
    return uptime;
}

Processinfo getProcessInfo(int pid, double systemUptime) {
    Processinfo pInfo;
    pInfo.pid = pid;

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket M-Q Previous   ^B Back
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^Q Where Was   M-W Next       ^F Forward
```

## STEP – 3

```
Press 'q' to quitPID    CPU%    Memory(kB)      Name
2561    14.2857%        6752    (sudo)
6       1.96451%        4607    (init)
124     0.053775%       4607    (snapfuse)
109     0.0360859%      4607    (snapfuse)
132     0.0242594%      4607    (snapfuse)
242     0.015568%       6480    (snapd)
1       0.0133413%      4607    (systemd)
115     0.00960268%     4607    (snapfuse)
388     0.0078859%      6480    (Relay(394))
54      0.00788431%     4607    (systemd-journal)
Enter PID to kill :124
Process124terminated successfully
Press 'q' to quit or Enter to referesh:
```

## STEP – 4

```
GNU nano 7.2                                                    day05.cpp
/*Dispaly CPU & memory usage per process
//sort processes by CPU/memory usages
// Total cputime = usertime+cputime(utime+ctime)
//Total CPUTime = utime + stime;
//Calculate process cpu usage
//CPU usage = (utime+stime)/systemupdate - starttime*100;
//g++ -std=c++17 activeprocess.cpp -o activeprocess

//Memory info is in /proc/[PID]/status
//cat /proc/1/status|grep VmRSS
// VmRSS(Resident Set Size) = Actual Ram used by the process.
//student@D001-37:~/Desktop/2141011082/LOS/Day03$ cat /proc/uptime
//2456.96(total system time) 29285.05(ideal time or total system unused)
#include <iostream>
#include<fstream>//read file(/proc data)
#include<sstream>//parse data
#include<vector>//store process
#include<algorithm>//sort the process
#include <filesystem>
#include <unistd.h> // for sysconf
//sysconf(_SC_CLK_TCK)
#include<csignal>//for kill() and SIGKILL
#include<thread>
#include<chrono>

namespace fs = std::filesystem;
using namespace std;

struct Processinfo {
    int pid;
    std::string name;
    double cpuUsage = 0;
    long memoryUsage = 0;
};

std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
        std::getline(file, value);
    }
    return value;
}

double getSystemUptime() {
                                                [ Wrote 160 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut       ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket  M-Q Previous  ^B Back
^X Exit      ^R Read File  ^\ Replace    ^U Paste     ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^Q Where Was    M-W Next      ^F Forward
```

## STEP 5

```
PID     CPU%    Memory (kB)     Name
6       2.00205 0       (init)
2545    0.516987        6980    (sudo)
128     0.0778776       0       (snapfuse)
124     0.0548028       0       (snapfuse)
109     0.0367755       0       (snapfuse)
132     0.0247231       0       (snapfuse)
242     0.0157626       0       (snapd)
1       0.0134933       0       (systemd)
115     0.00978621      0       (snapfuse)
54      0.00793198      0       (systemd-journal)
Enter PID to Kill :
128
Process 128 terminated successfully

Press 'q' to quit or Enter to refresh :
```

```
//Dispaly CPU & memory usage per process
//sort processes by CPU/memory usages
// Total cputime = usertime+cputime(utime+ctime)
//Total CPUTime = utime + stime;
//Calculate process cpu usage
//CPU usage = (utime+stime)/systemupdate - starttime*100;
//g++ -std=c++17 activeprocess.cpp -o activeprocess

//Memory info is in /proc/[PID]/status
//cat /proc/1/status|grep VmRSS
// VmRSS(Resident Set Size) = Actual Ram used by the process.
//student@D001-37:~/Desktop/2141011082/LOS/Day03$ cat /proc/uptime
//2456.96(total system time) 29285.05(ideal time or total system unused)
#include <iostream>
#include<fstream>//read file(/proc data)
#include<sstream>//parse data
#include<vector>//store process
#include<algorithm>//sort the process
#include <filesystem>
#include <unistd.h> // for sysconf
//sysconf(_SC_CLK_TCK)
#include<csignal>//for kill() and SIGKILL
#include<thread>
#include<chrono>

namespace fs = std::filesystem;
using namespace std;

struct Processinfo {
    int pid;
    std::string name;
    double cpuUsage = 0;
    long memoryUsage = 0;
};

std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
        std::getline(file, value);
    }
    return value;
}

double getSystemUptime() {
```

```
    std::ifstream file("/proc/uptime");
    double uptime = 0;
    if (file.is_open()) {
        file >> uptime;
    }
    return uptime;
}

Processinfo getProcessInfo(int pid, double systemUptime) {
    Processinfo proc;
    proc.pid = pid;

    std::ifstream statFile("/proc/" + std::to_string(pid) + "/stat");
    std::string line;

    long utime = 0, stime = 0, starttime = 0;

    if (statFile.is_open()) {
        std::getline(statFile, line);
        std::istringstream ss(line);
        std::string token;
        for (int i = 1; ss >> token; ++i) {
            if (i == 2) proc.name = token;
            else if (i == 14) utime = std::stol(token);
            else if (i == 15) stime = std::stol(token);
            else if (i == 22) starttime = std::stol(token);
        }
    }

    std::ifstream memFile("/proc/" + std::to_string(pid) + "/status");
    if (memFile.is_open()) {
        std::string key, value, unit;
        while (memFile >> key >> value >> unit) {
            if (key == "VmRSS:") {
                proc.memoryUsage = std::stol(value);
                break;
            }
        }
    }

    long total_time = utime + stime;
    double seconds = systemUptime - (starttime / sysconf(_SC_CLK_TCK));
    if (seconds > 0) {
        proc.cpuUsage = ((total_time / (double)sysconf(_SC_CLK_TCK)) / seconds) *
100.0;
```

```
    }

    return proc;
}

std::vector<Processinfo> getAllprocess() {
    std::vector<Processinfo> processes;
    double systemUptime = getSystemUptime();

    for (const auto &entry : fs::directory_iterator("/proc")) {
        if (entry.is_directory()) {
            std::string filename = entry.path().filename().string();
            if (std::all_of(filename.begin(), filename.end(), ::isdigit)) {
                int pid = std::stoi(filename);
                processes.push_back(getProcessInfo(pid, systemUptime));
            }
        }
    }
    return processes;
}

void sortProcesses(std::vector<Processinfo> &processes, bool sortByCPU) {
    if (sortByCPU) {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.cpuUsage > b.cpuUsage;
        });
    } else {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.memoryUsage > b.memoryUsage;
        });
    }
}

int main() {
    char input;
    while(true){
        system("clear");
        vector<Processinfo> processes = getAllprocess();
        sortProcesses(processes, true); // Change to false to sort by memory
usage

        cout << "PID\tCPU%\tMemory (kB)\tName\n";
        for (size_t i = 0; i < min(processes.size(), size_t(10)); ++i) {
```

```
            cout << processes[i].pid << "\t"
            << processes[i].cpuUsage << "\t"
            << processes[i].memoryUsage << "\t"
            << processes[i].name << "\n";
        }
        int targetPid;
        cout <<"Enter PID to Kill : " << endl;
        cin >> targetPid;
        //Signature : int kill(pid_t,int sig)
        if(targetPid){
            if(kill(targetPid,SIGKILL)==0){
                cout << "Process " << targetPid << " terminated successfully" <<
endl;
            }
            else{
                perror("Failed to kill Process");
            }
        }
        cout <<"\nPress 'q' to quit or Enter to refresh : ";
        cin.ignore();
        input = getchar();
        if(input == 'q' || input == 'Q')
            break;
        std::this_thread::sleep_for(std::chrono::seconds(2));
    }
    return 0;
}




 // All real-time refresh  feature using sleep() and loop
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <filesystem>
#include <unistd.h>
#include <dirent.h> // For directory handling
#include <cstring> // For strcmp
#include<csignal>// for kill() and SIGKILL
#include<chrono>
#include<thread>
```

```
namespace fs = std::filesystem;
using namespace std;

struct Processinfo {
    int pid;
    std::string name;
    double cpuUsage;
    long memoryUsage;
};

std::string readFileValue(const std::string &path) {
    std::ifstream file(path);
    std::string value;
    if (file.is_open()) {
        std::getline(file, value);
    }
    return value;
}

double getSystemUptime() {
    std::ifstream file("/proc/uptime");
    double uptime;
    if (file.is_open()) {
        file >> uptime; // extract uptime only, not ideal time
    }
    return uptime;
}

Processinfo getProcessInfo(int pid, double systemUptime) {
    Processinfo pInfo;
    pInfo.pid = pid;

    std::ifstream statFile("/proc/" + std::to_string(pid) + "/stat");
    std::string line;
    long utime, stime, starttime;
    if (statFile.is_open()) {
        std::getline(statFile, line);
        std::istringstream ss(line);
        std::string token;
        int count = 0;
        while (ss >> token) {
            count++;
            if (count == 2) pInfo.name = token;
            else if (count == 14) utime = std::stol(token);
            else if (count == 15) stime = std::stol(token);
```

```
            else if (count == 22) starttime = std::stol(token);
        }
    }

    std::ifstream memFile("/proc/" + std::to_string(pid) + "/status");
    if (memFile.is_open()) {
        std::string key, value, unit;
        while (memFile >> key >> value >> unit) {
            if (key == "VmRSS:") {
                pInfo.memoryUsage = std::stol(value);
                break;
            }
        }
    }

    long total_time = utime + stime;
    double seconds = systemUptime - (starttime / sysconf(_SC_CLK_TCK));
    pInfo.cpuUsage = (static_cast<double>(total_time) / sysconf(_SC_CLK_TCK)) /
seconds * 100;

    return pInfo;
}

std::vector<Processinfo> getAllProcesses() {
    std::vector<Processinfo> processes;
    double systemUptime = getSystemUptime();

    DIR *dir = opendir("/proc");
    struct dirent *entry;
    if (dir != nullptr) {
        while ((entry = readdir(dir)) != nullptr) {
            if (entry->d_type == DT_DIR) {
                // Check if the directory name is a number (PID)
                if (std::all_of(entry->d_name, entry->d_name + strlen(entry-
>d_name), ::isdigit)) {
                    int pid = std::stoi(entry->d_name);
                    processes.push_back(getProcessInfo(pid, systemUptime));
                }
            }
        }
        closedir(dir);
    }
    return processes;
}
```

```
void sortProcesses(std::vector<Processinfo> &processes, bool sortByCPU) {
    if (sortByCPU) {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.cpuUsage > b.cpuUsage;
        });
    } else {
        std::sort(processes.begin(), processes.end(), [](const Processinfo &a,
const Processinfo &b) {
            return a.memoryUsage > b.memoryUsage;
        });
    }
}

int main() {
char input;
while(true)
{
system("clear");

vector<Processinfo> processes = getAllProcesses();
       sortProcesses(processes, true);

       std::cout<<"Press 'q' to quit";

       cout <<"PID\tCPU%\tMemory(kB)\tName\n";
       for (size_t i = 0; i < std::min(processes.size(), size_t(10)); i++) {
        std::cout << processes[i].pid << "\t"
                  << processes[i].cpuUsage << "%\t"
                  << processes[i].memoryUsage << "\t"
                  << processes[i].name << "\n";
 }
    int targetPid;

std::cout <<"Enter PID to kill :";
std::cin >> targetPid;

// use kill() function

//signature : int kil(pid_t pid, int sig);

// kill return integer type data
//pid : Process ID
//ig : Signal to send (SIGKILL, SIGTERM, etc.)
```

```
if (targetPid>0)
{
if(kill(targetPid, SIGKILL)==0)
{
std::cout << "Process" << targetPid << "terminated successfully";
}
else
{
perror("Failed to kill process");
}
}

cout <<"\nPress 'q' to quit or Enter to referesh:";
cin.ignore();

input = getchar();

if(input == 'q' || input == 'Q')
break;

std::this_thread::sleep_for(std::chrono::seconds(0));
}

    // size_t

    return 0;
}
```