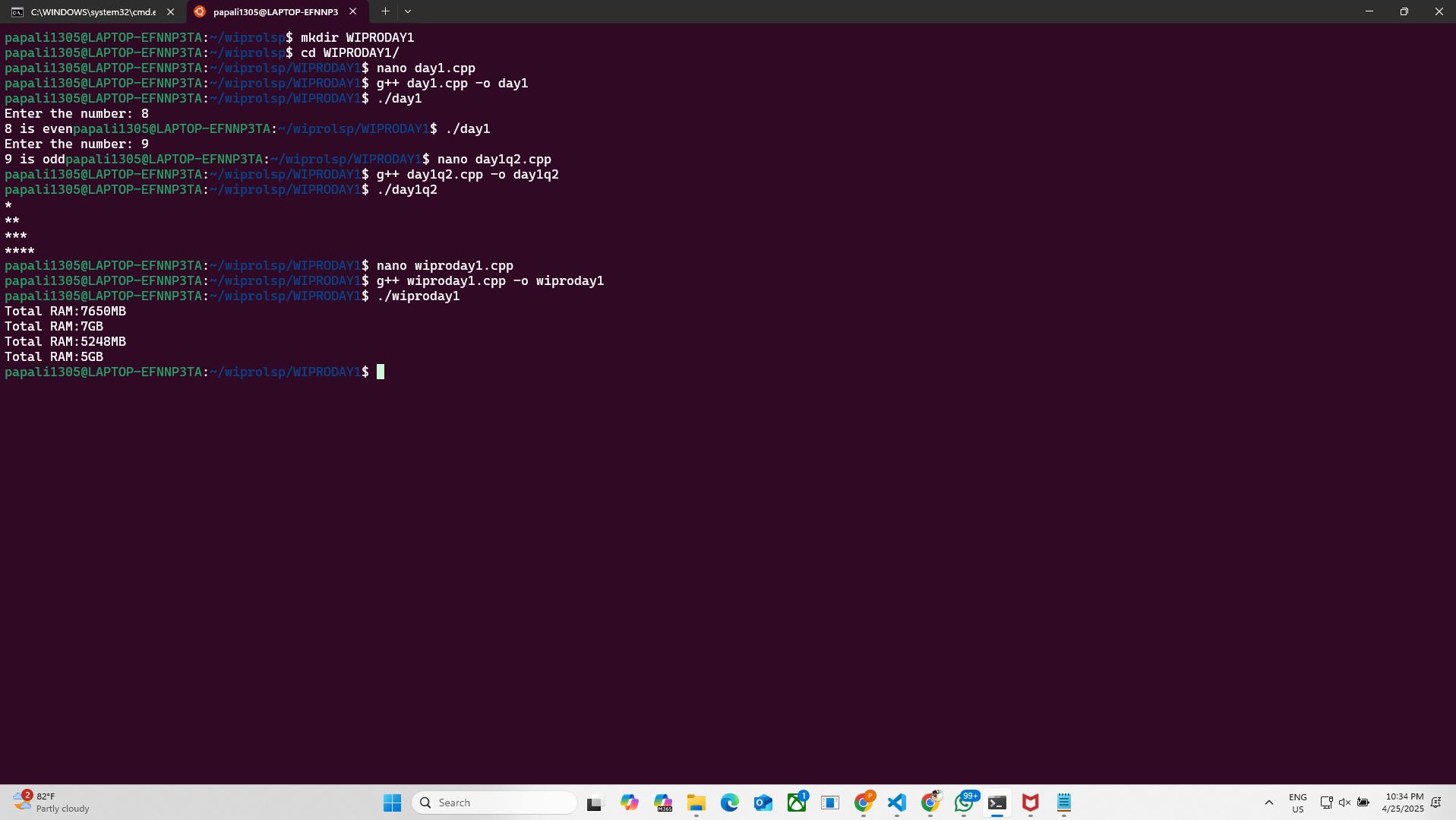
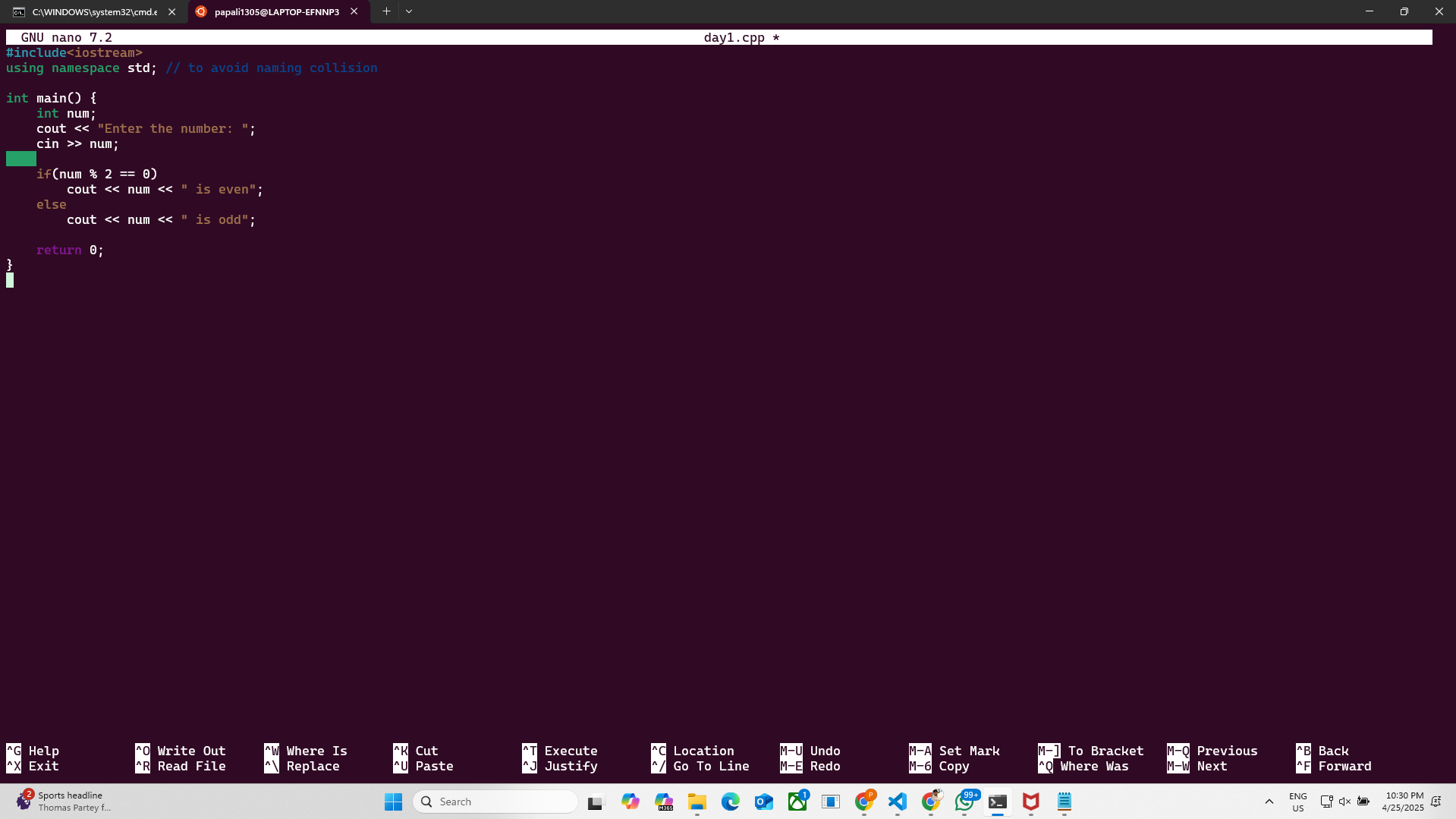
**WIPRO DAY – 1**

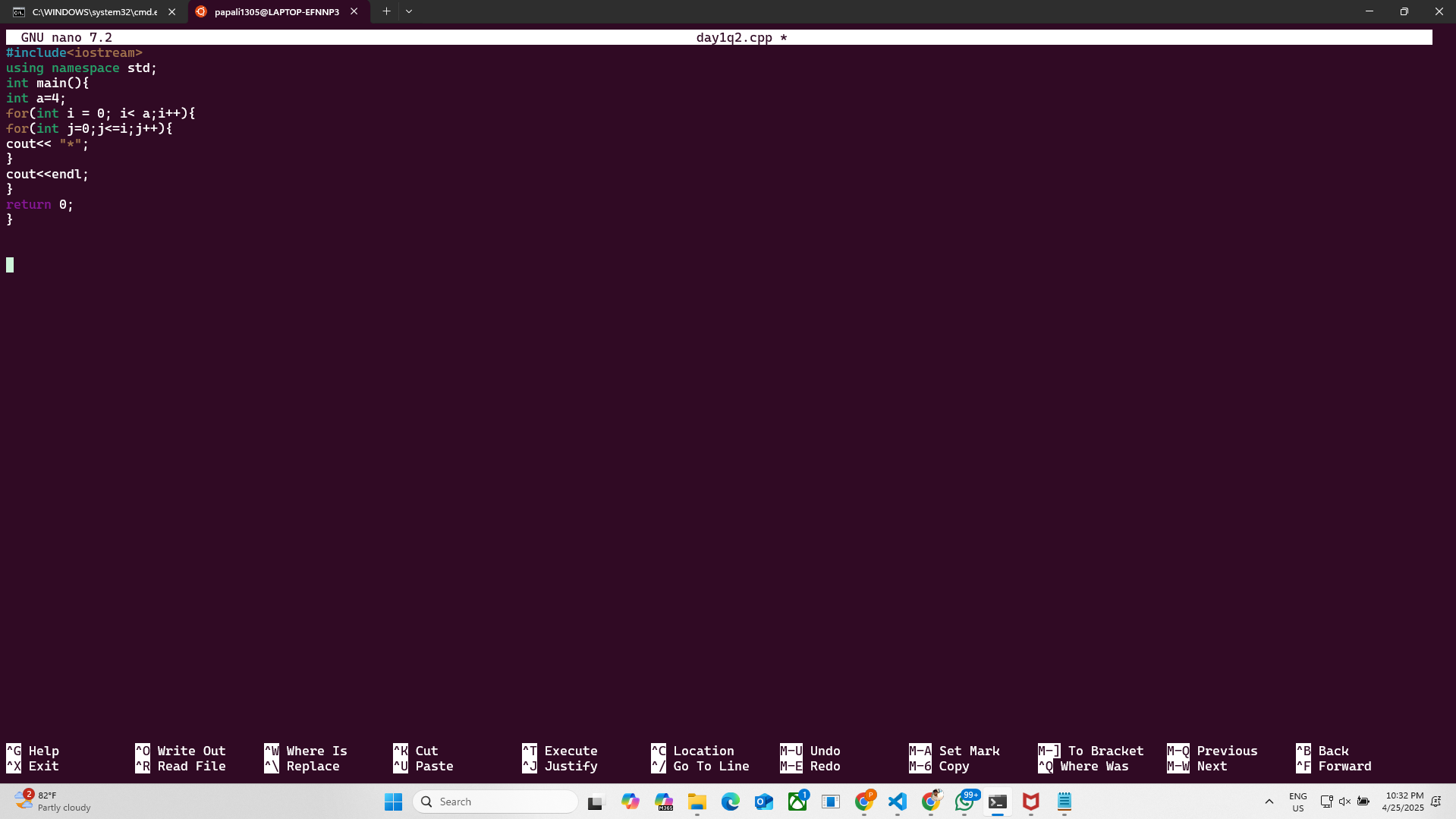
**STEP-1**



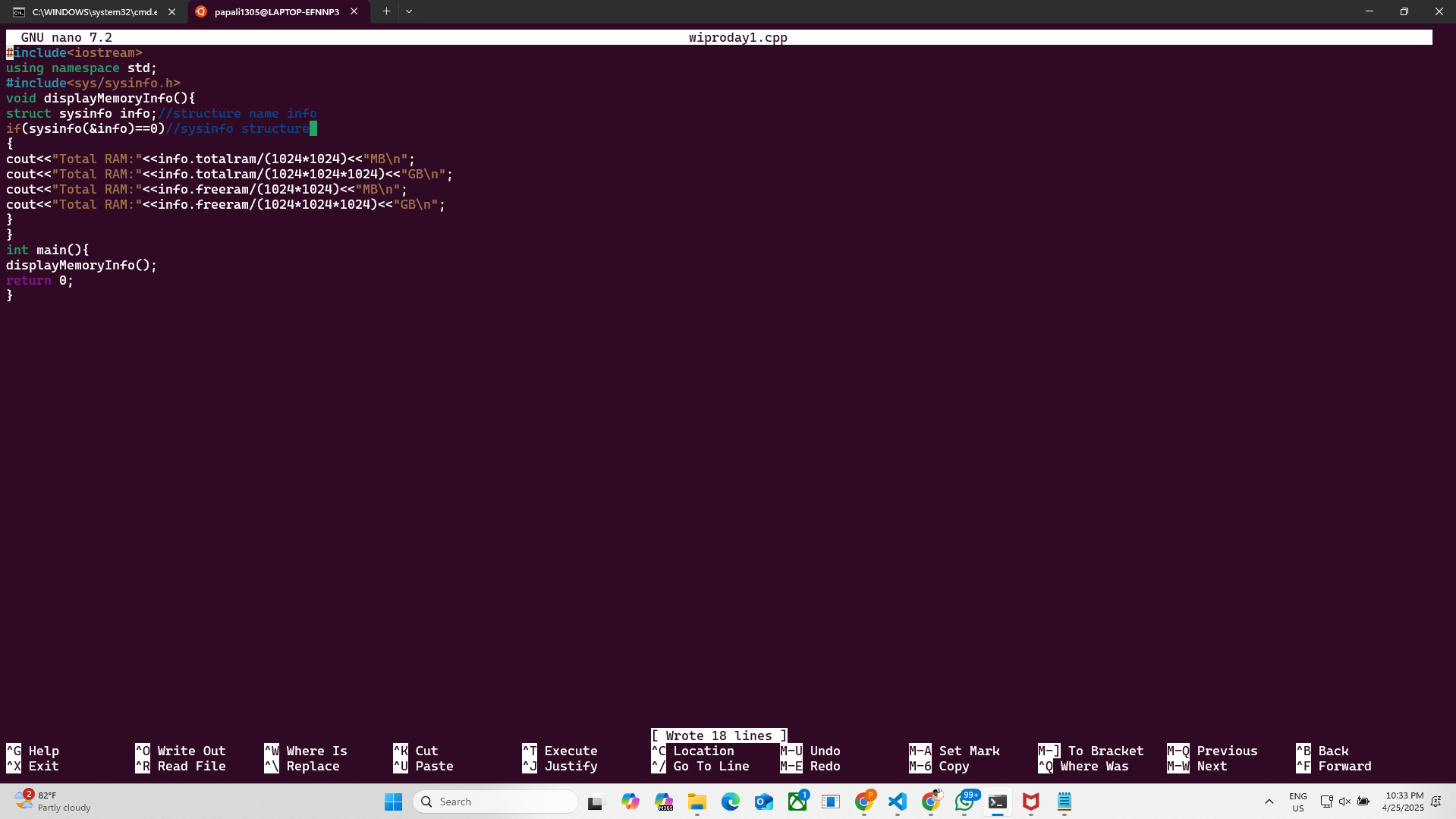
**STEP – 2**



**STEP – 3**



**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(){

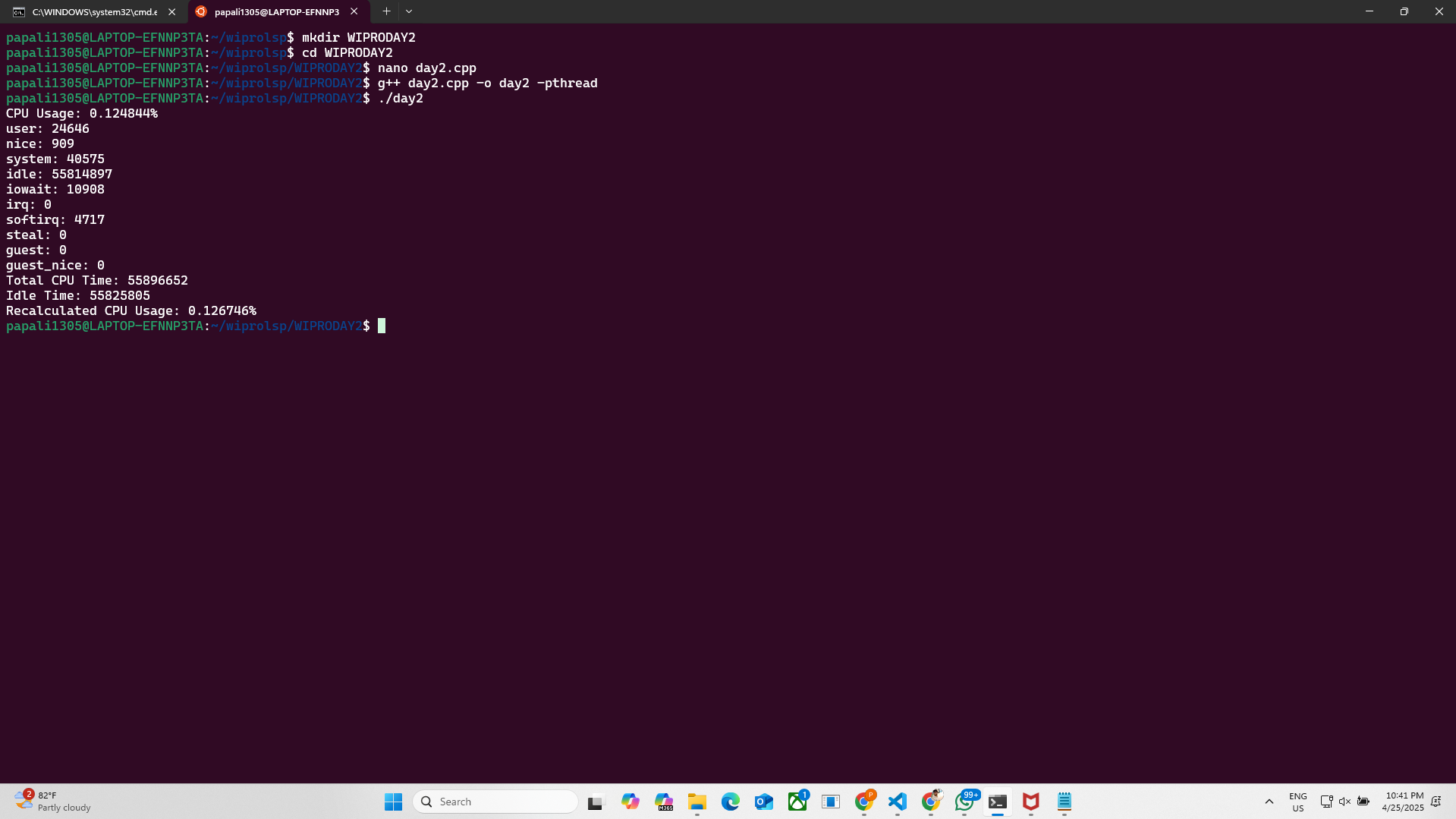
displayMemoryInfo();

return 0;

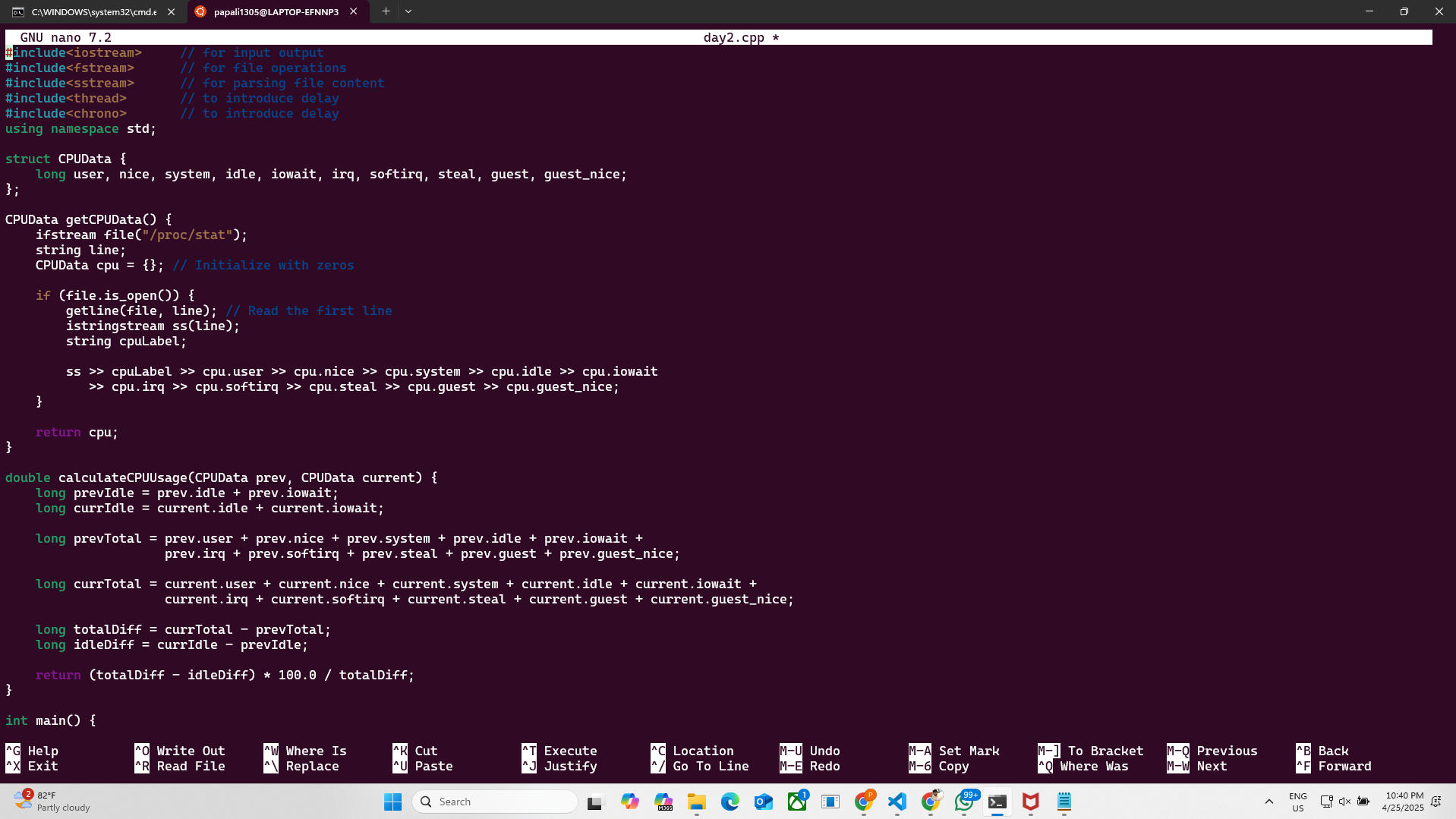
}

**WIPRO DAY – 2**

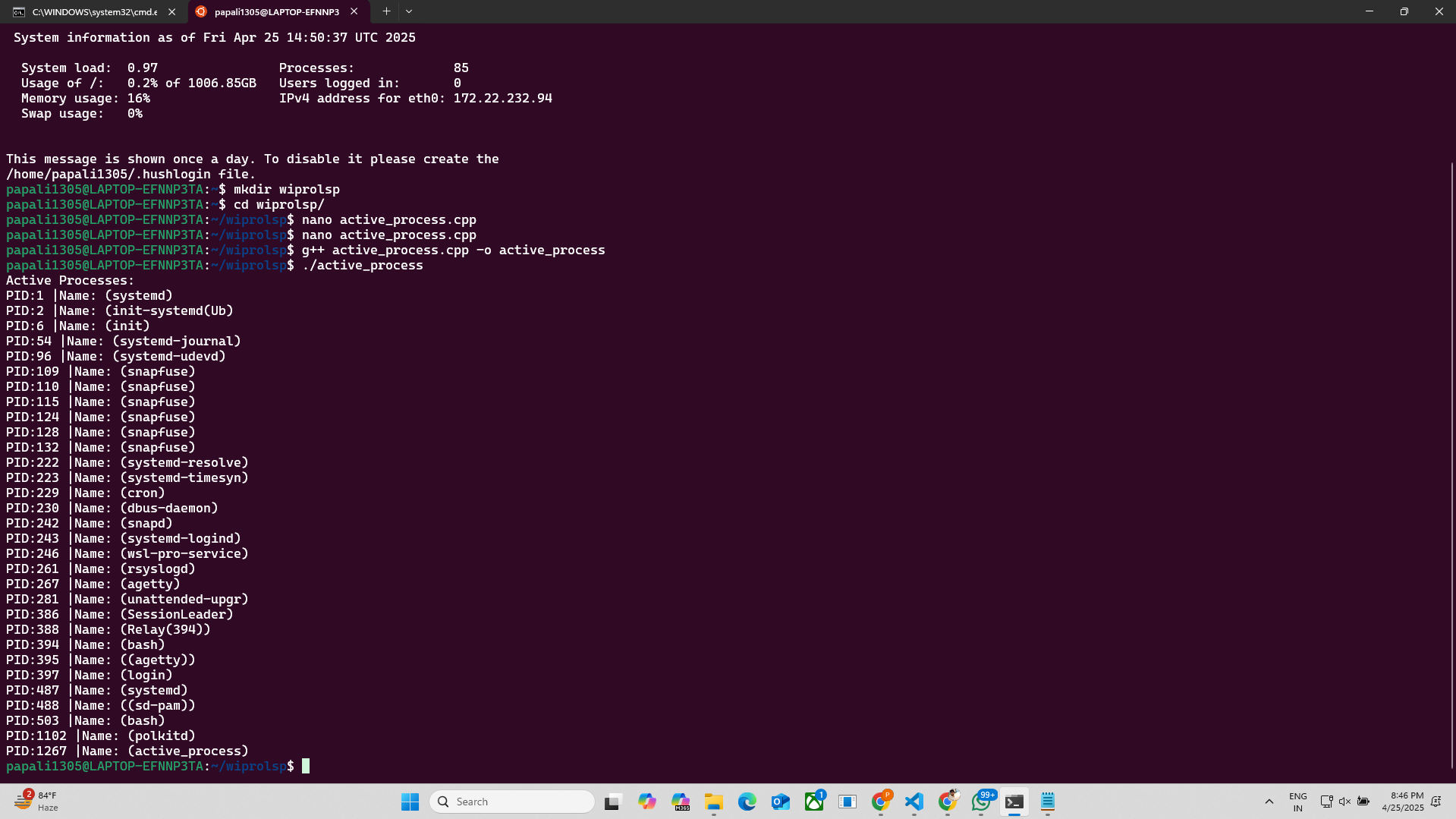
**STEP – 1**



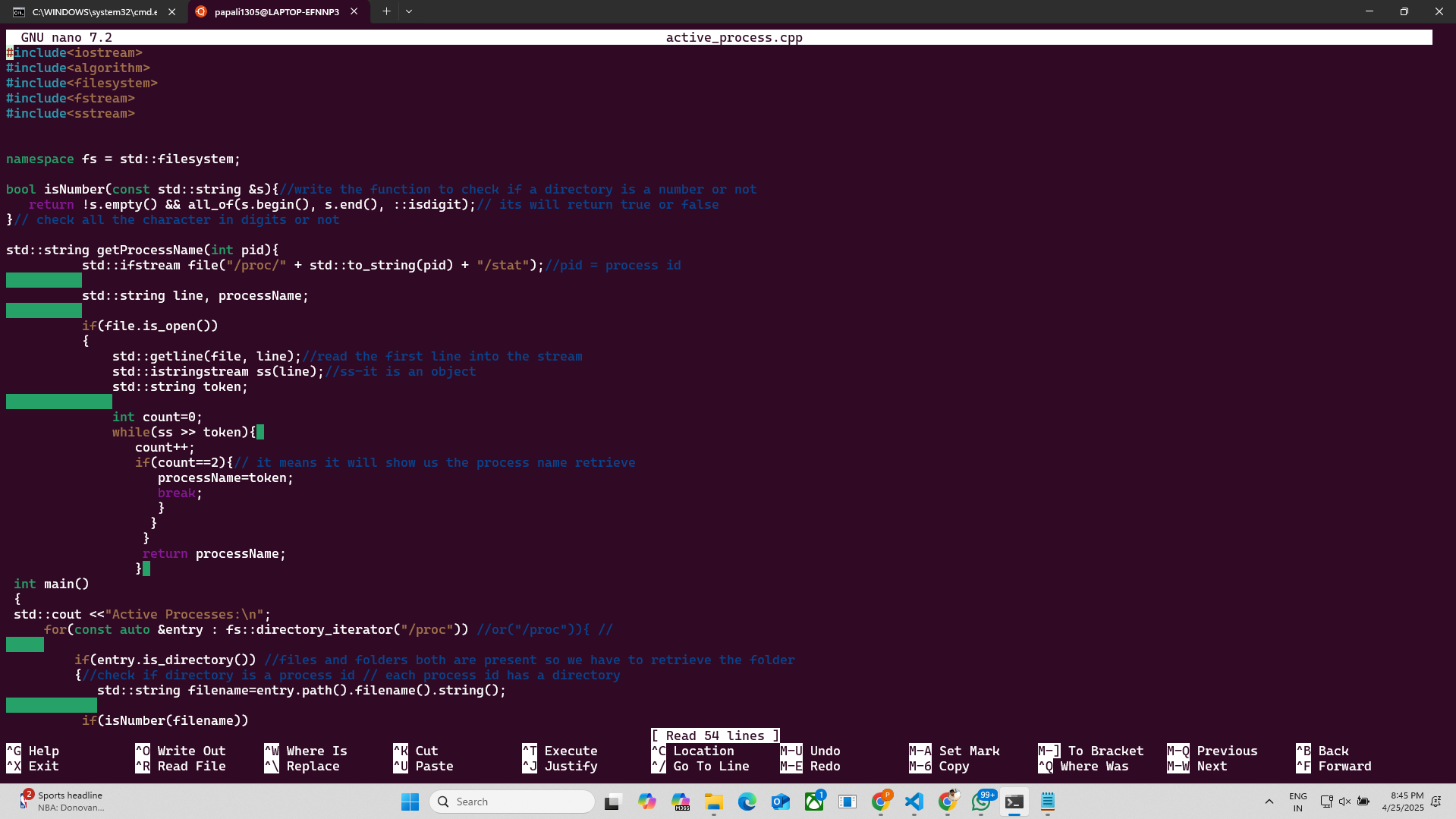
**STEP – 2**



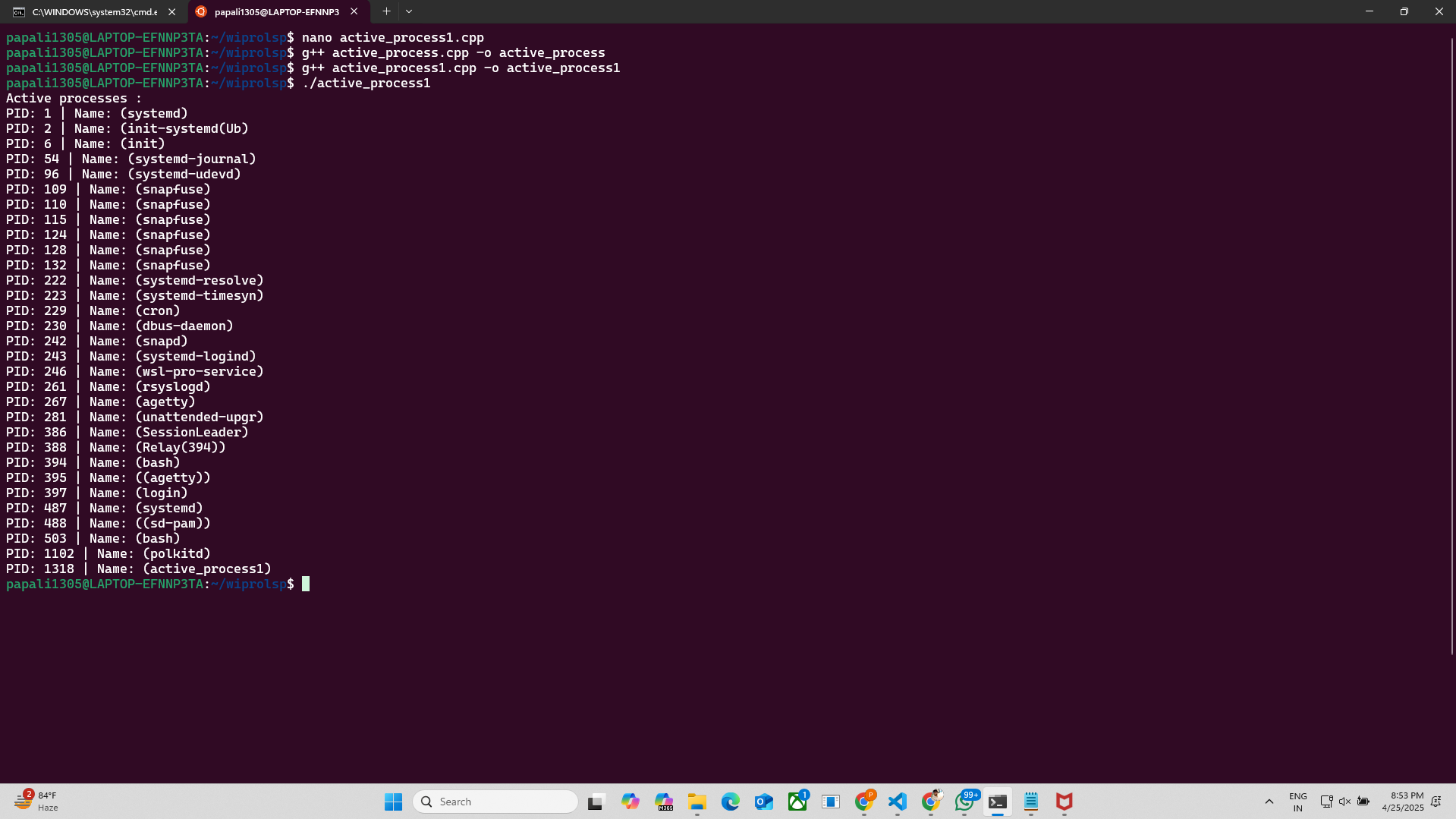
**STEP -3(ACTIVE)**



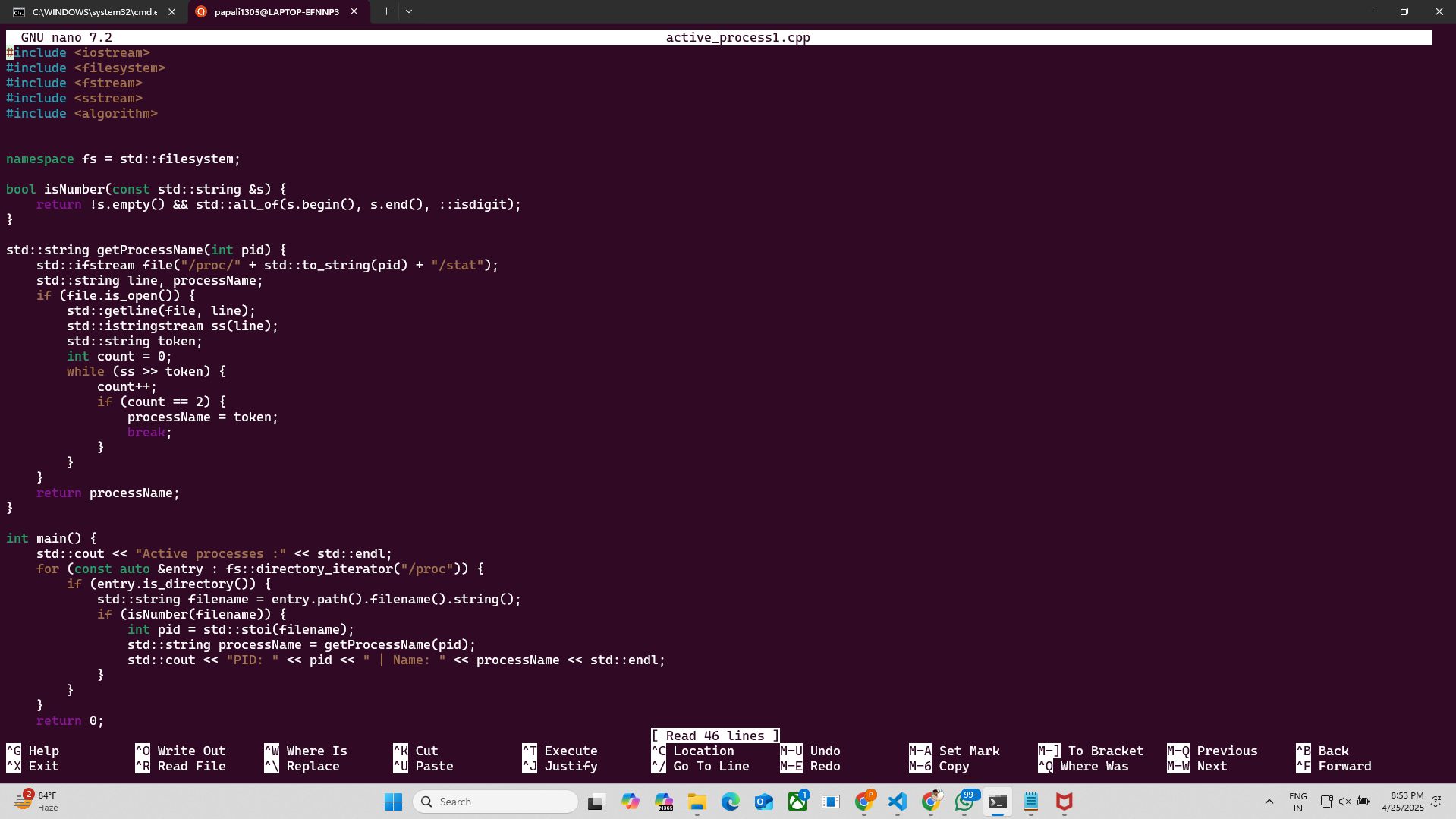
**STEP – 4**



**STEP – 5 (ACTIVE1)**



**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;**

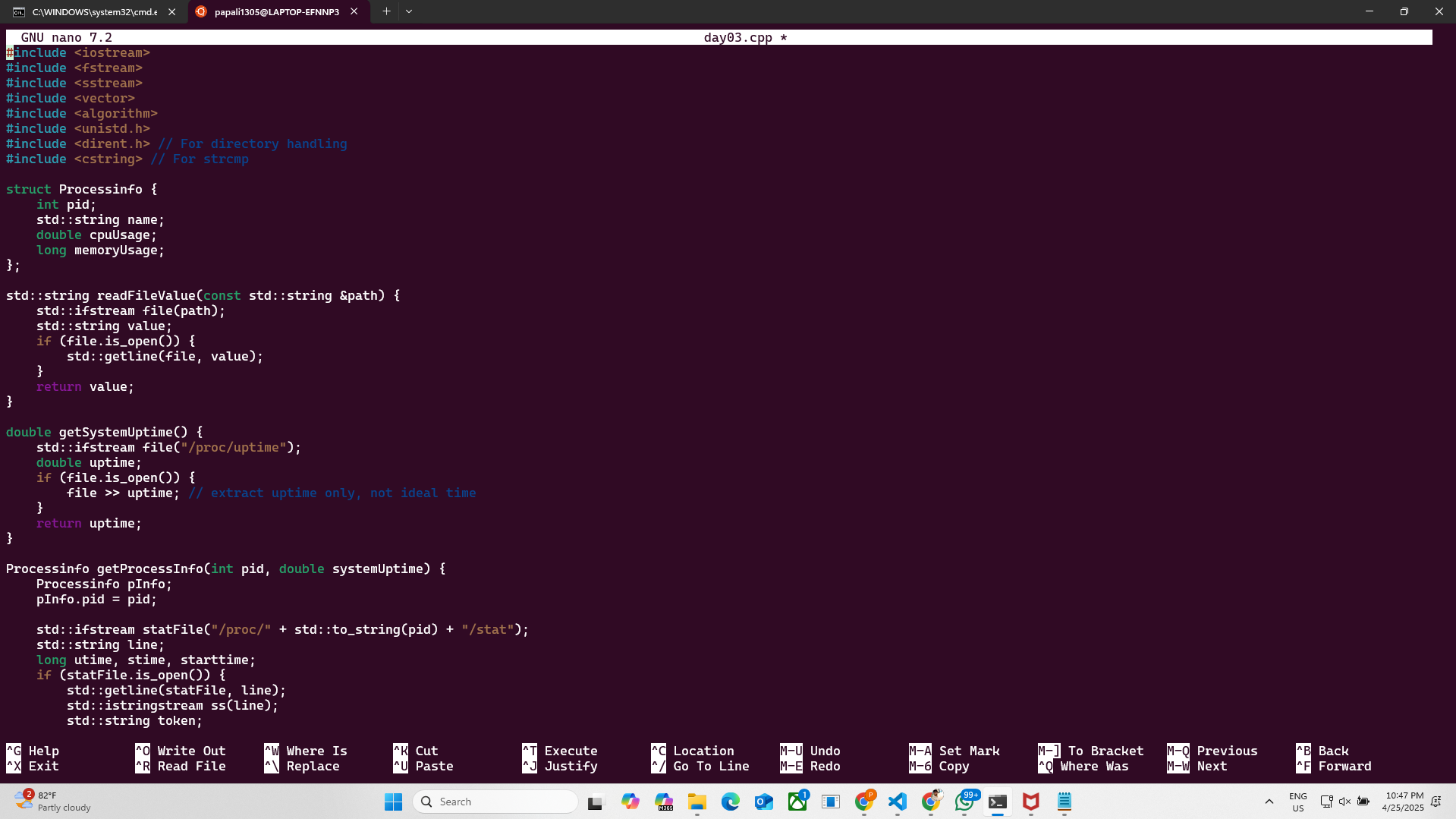
**}**

**WIPRO DAY – 3**

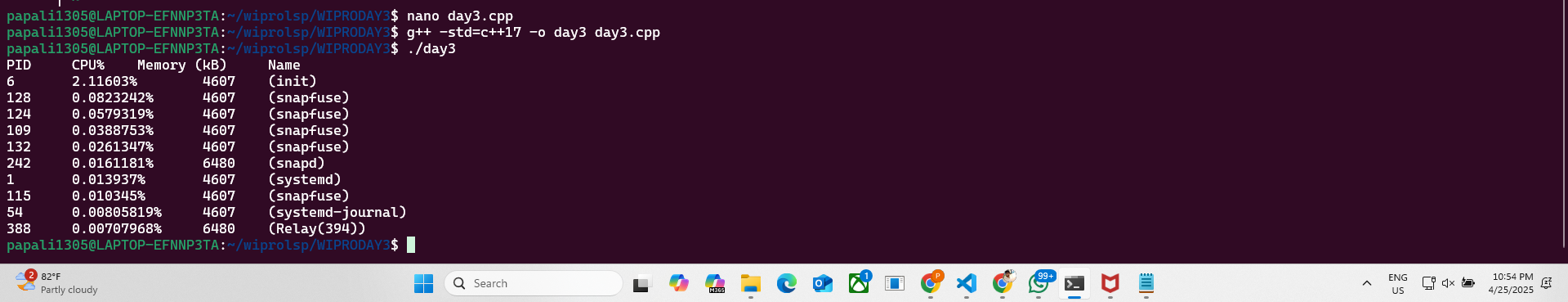
**STEP – 1**



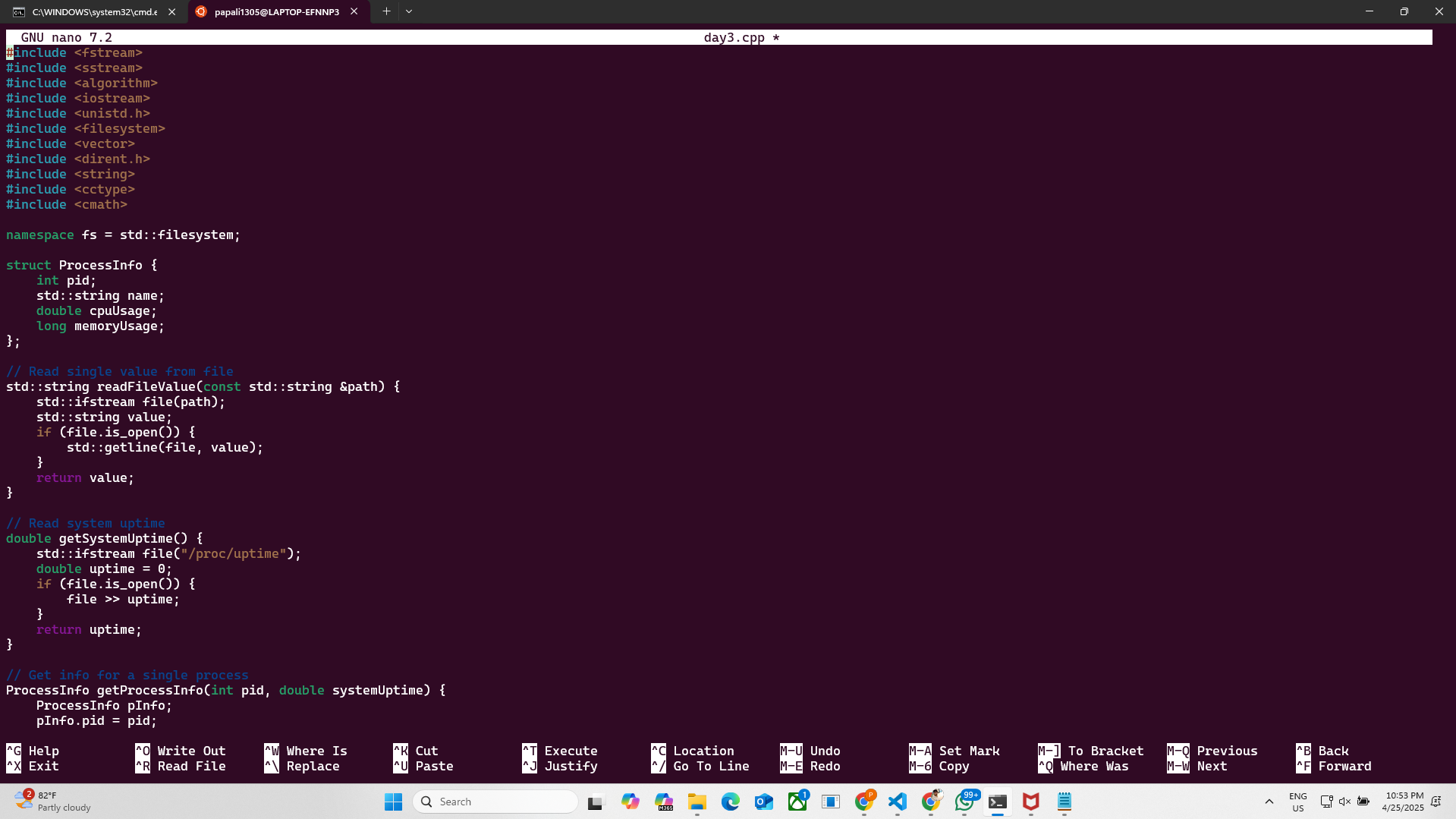
**STEP – 2**



**STEP -3**



**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";

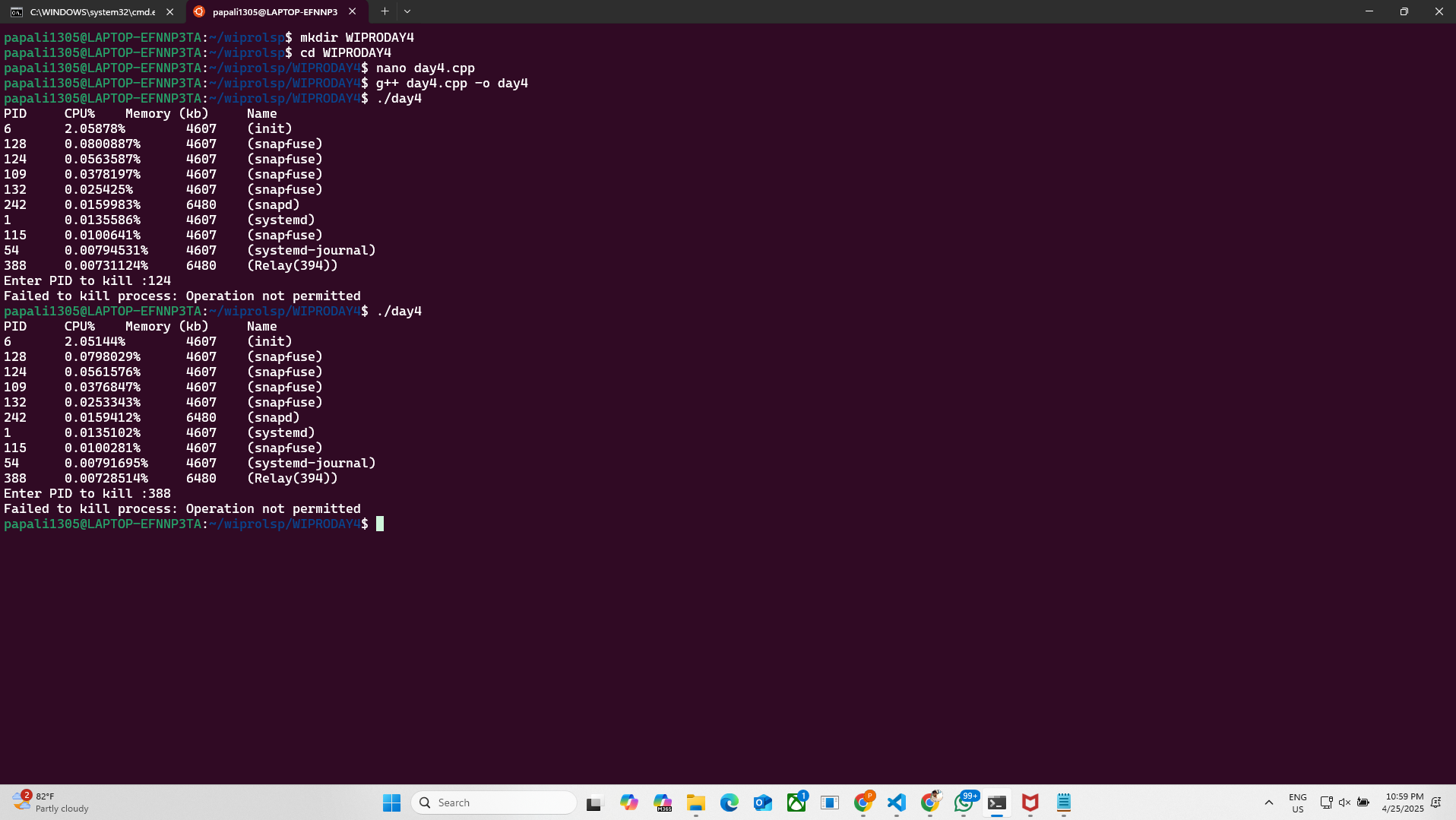
    }

    return 0;

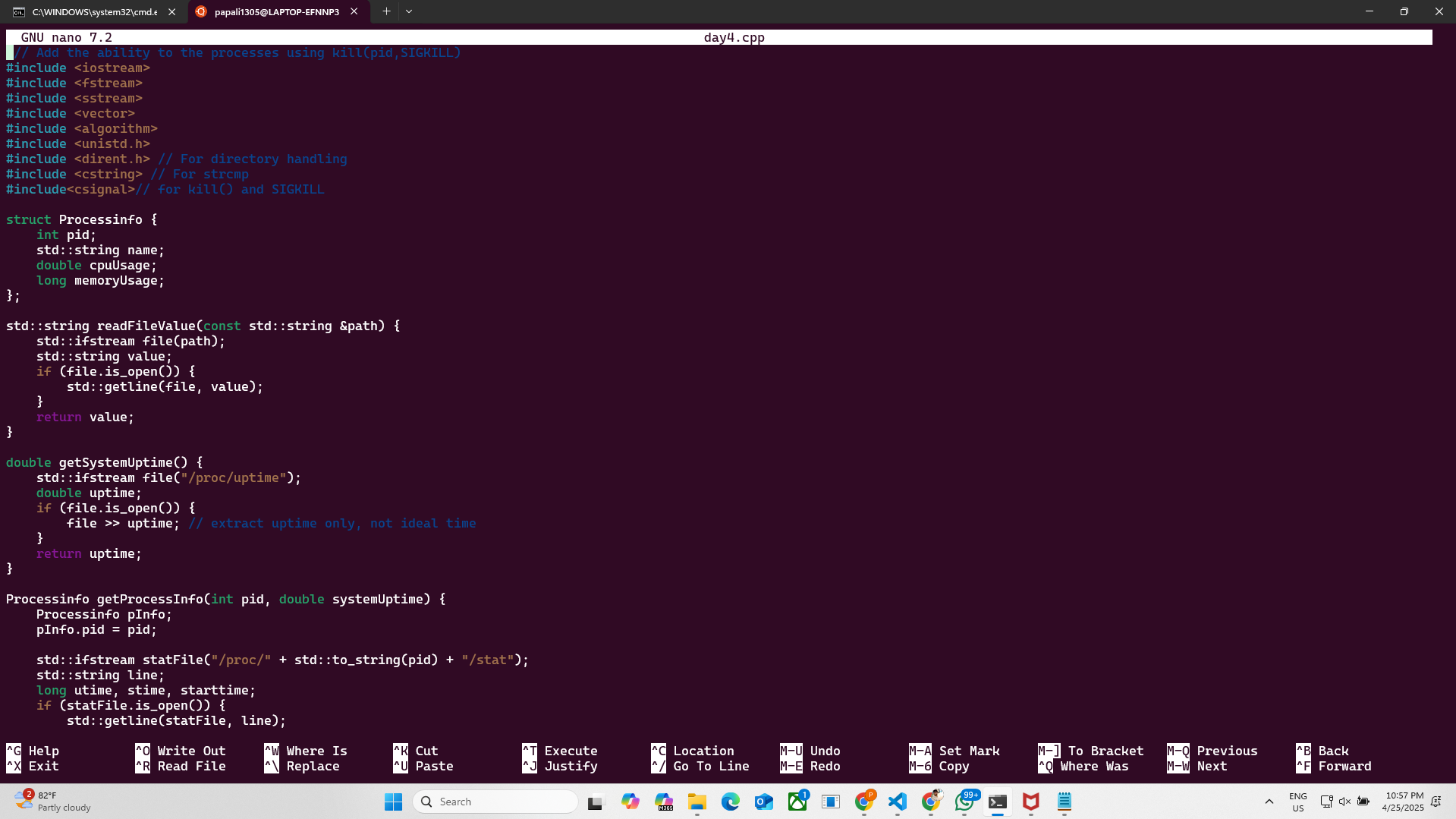
}

**WIPRO DAY – 4**

STEP – 1



STEP – 2



 // 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");

}

}

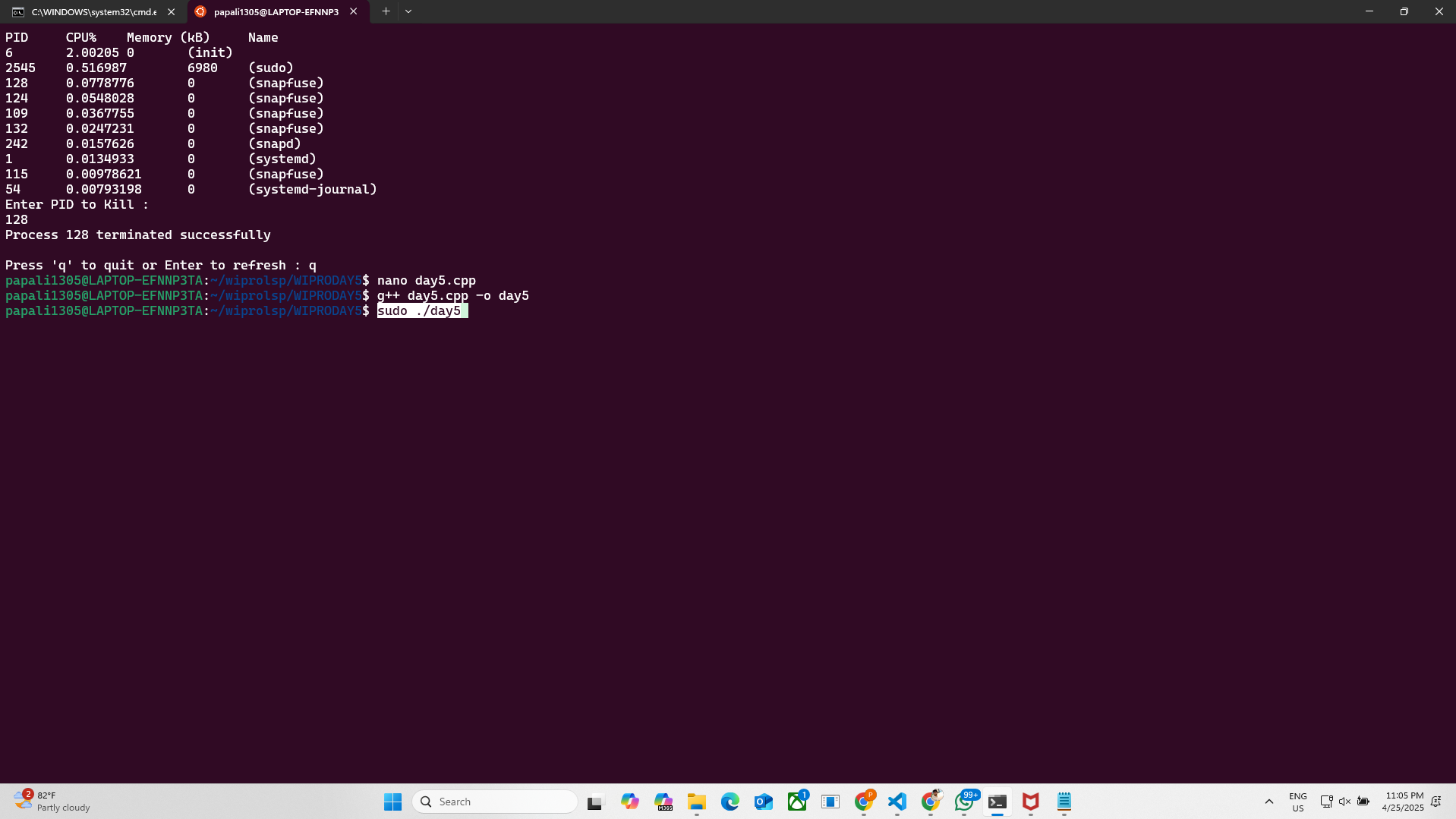
    // size\_t

    return 0;

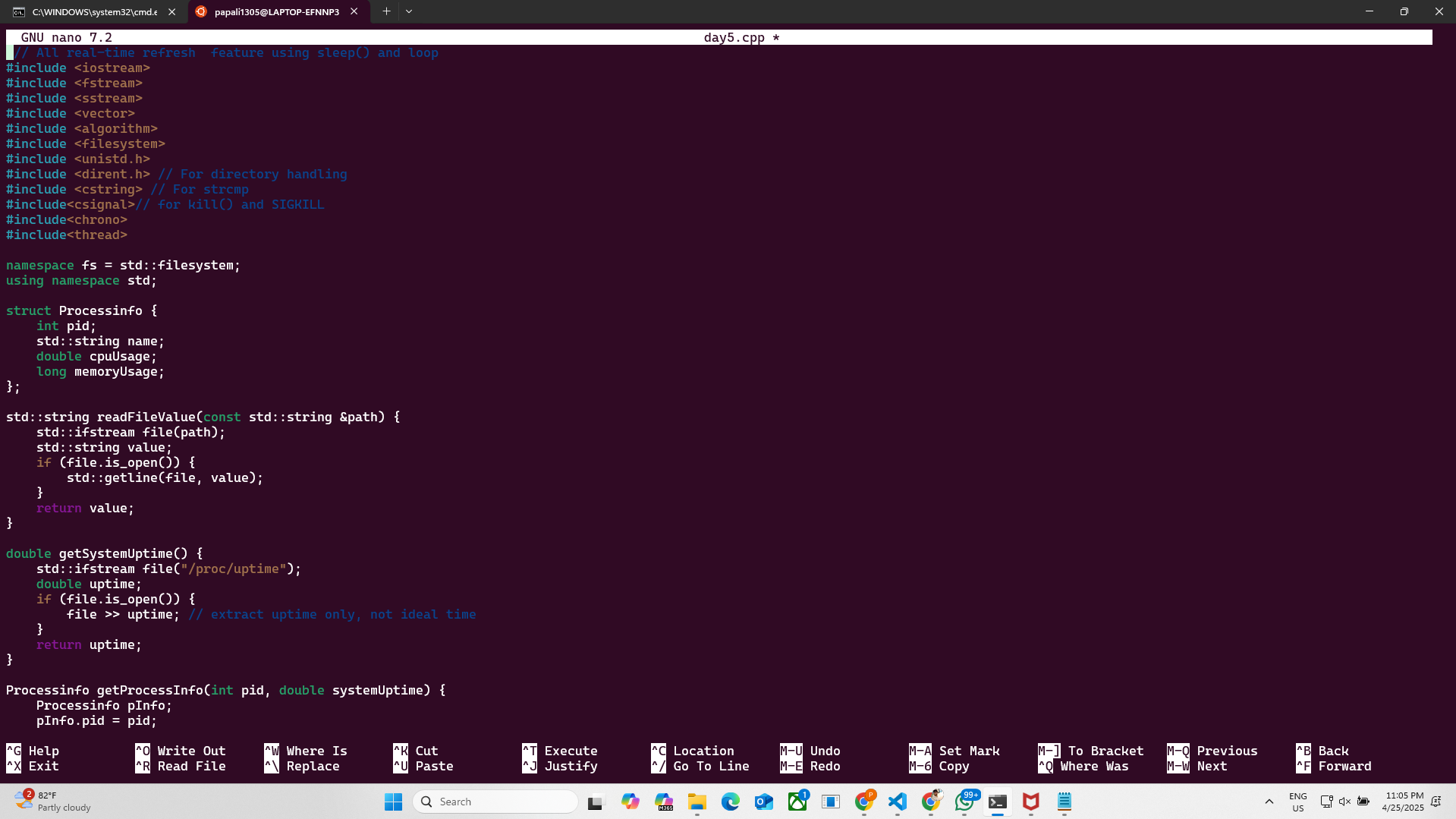
}

**WIPRO DAY – 5**

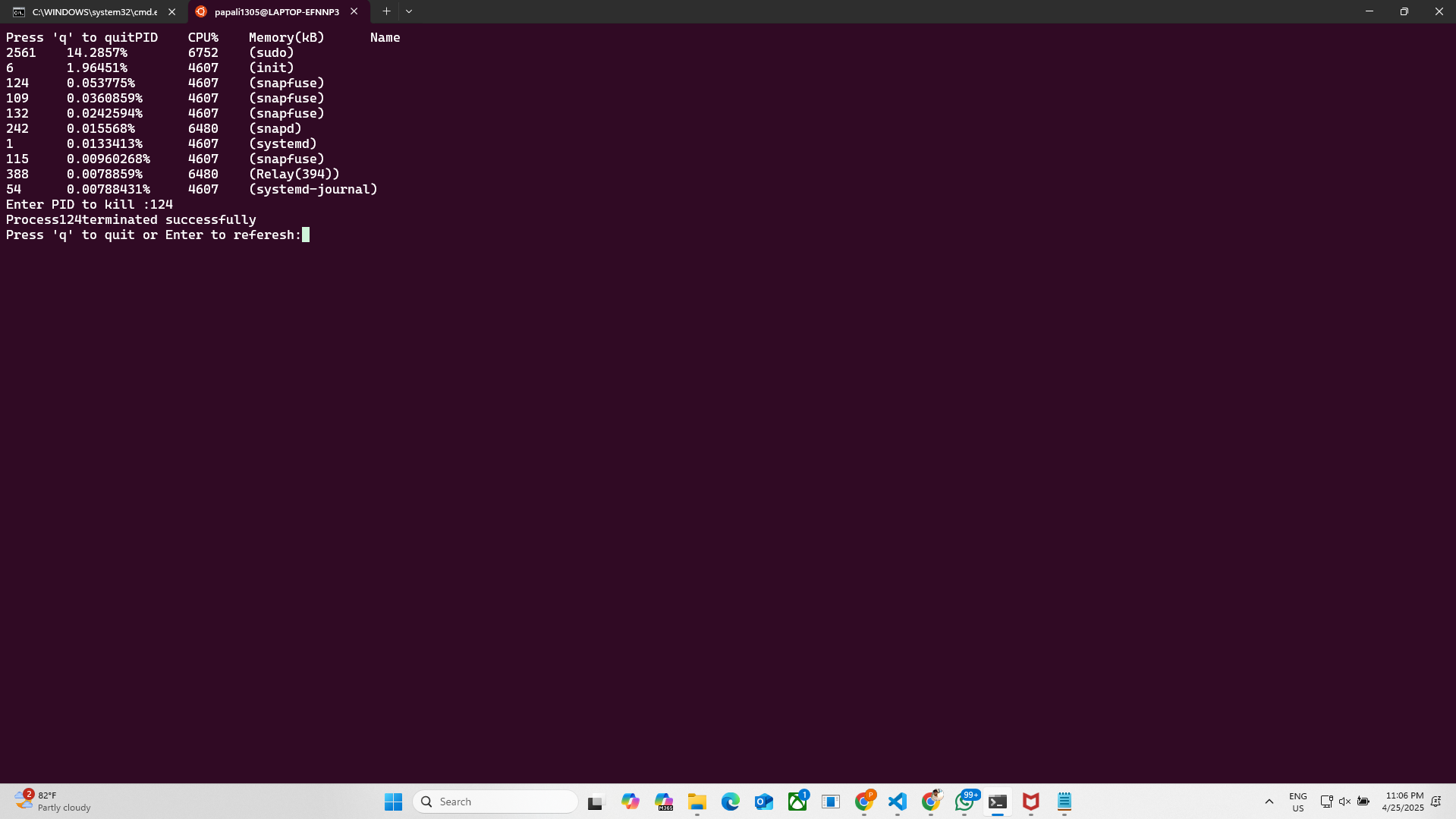
STEP – 1



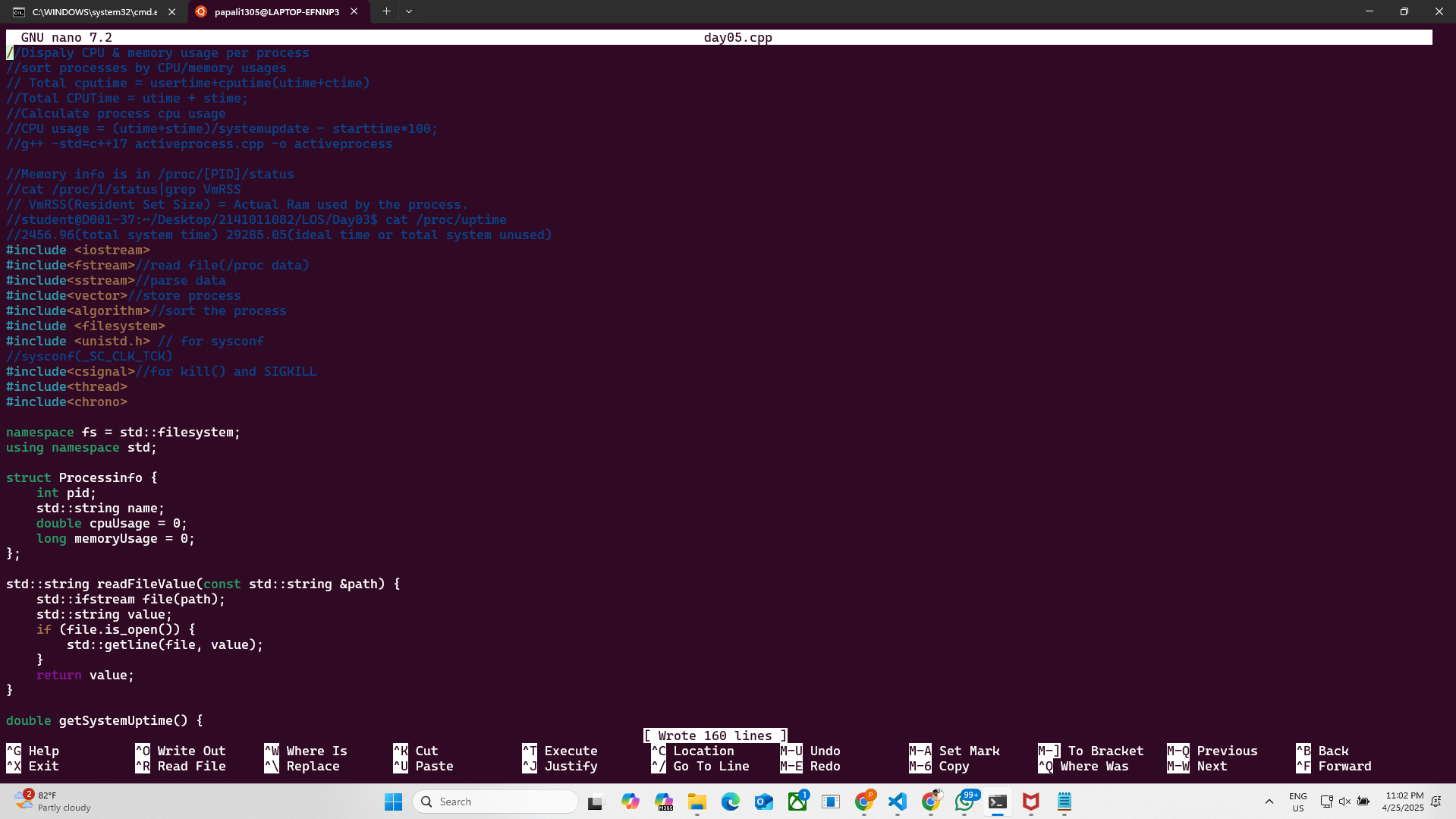
STEP – 2



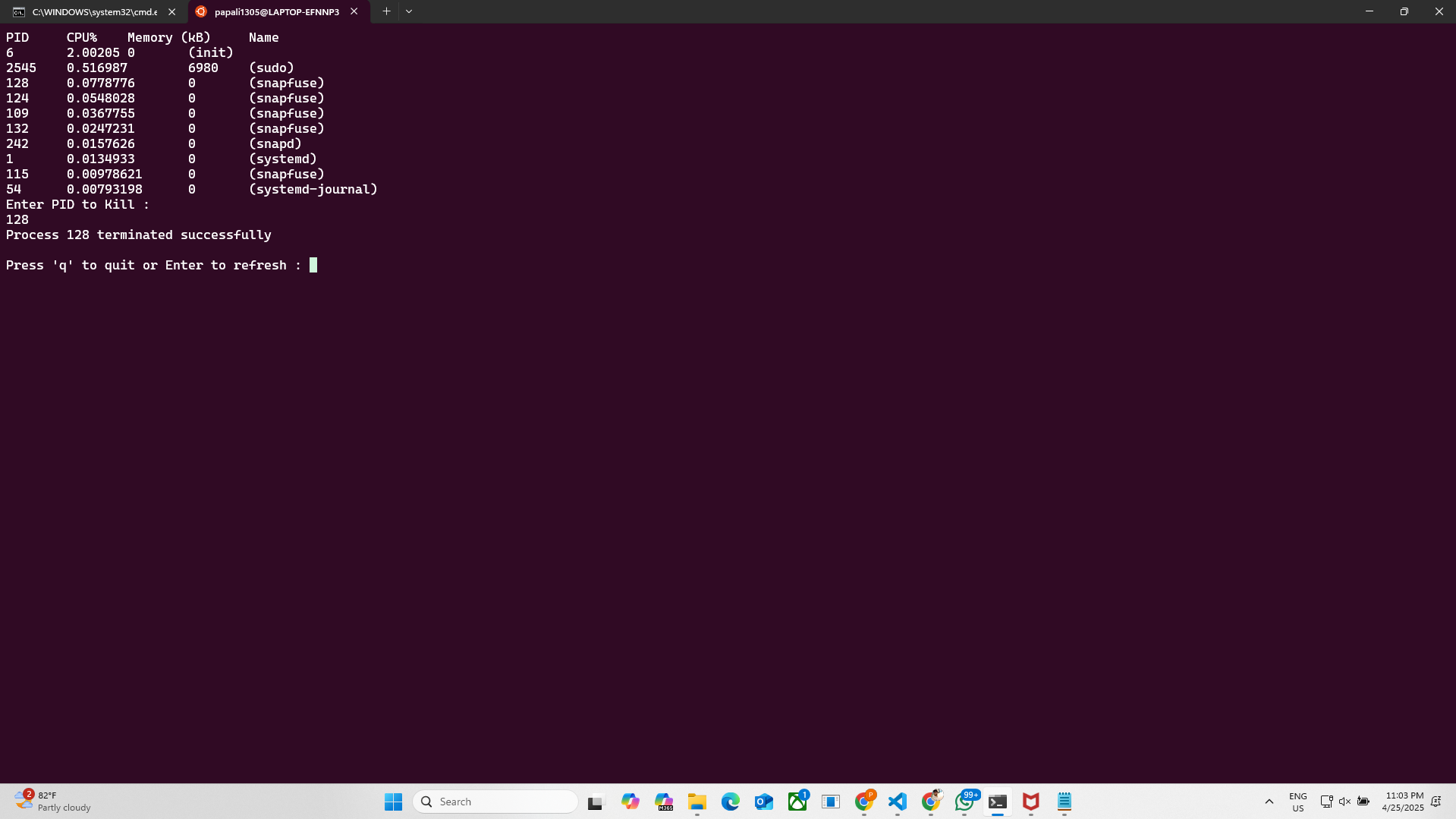
STEP – 3



STEP – 4



STEP 5



//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;

}