**Group Members:**

M. Tayyab Khan 22F-3440

M. Ahmad 22F-8814

Taha Saadat 22F-3439

May 5, 2024

MINI Operating system project

operating system – lab

Contents

[Welcome Screen: 1](#_Toc165826312)

[Kernel Mode: 2](#_Toc165826313)

[User Mode: 4](#_Toc165826314)

[Application running: 6](#_Toc165826315)

[Operating System Exits: 7](#_Toc165826316)

[Different Applications: 7](#_Toc165826317)

[File Management/Text Editor: 7](#_Toc165826318)

[Games: 10](#_Toc165826319)

[System Defaults: 12](#_Toc165826320)

[Process Synchronization: 14](#_Toc165826321)

[Resource Allocation 15](#_Toc165826322)

[Multitasking 15](#_Toc165826323)

[Process Scheduling 16](#_Toc165826324)

[Code Implementation 16](#_Toc165826325)

# 

# Welcome Screen:

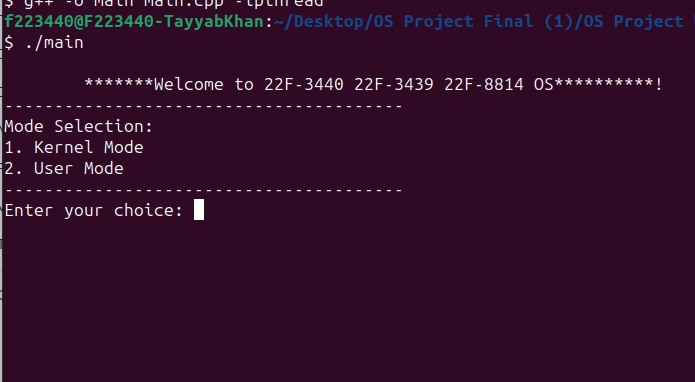
the ./main file is run on the console to log into the operating system. Upon running we get two options to select from.

1. User Mode
2. Kernel Mode

**User Mode:**This mode allows us to run the operating system features like running basic tasks as we do in an operating system.

**Kernel Mode:**  
This will allow the user to change the system hardware requirements.

**ScreenShot:**



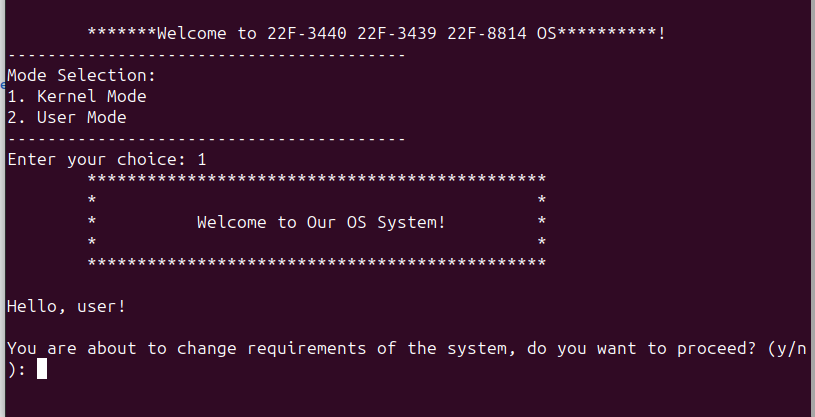
## Kernel Mode:

By default if we move to User mode the system will have the requirements as followed:  
default\_ram = 8;

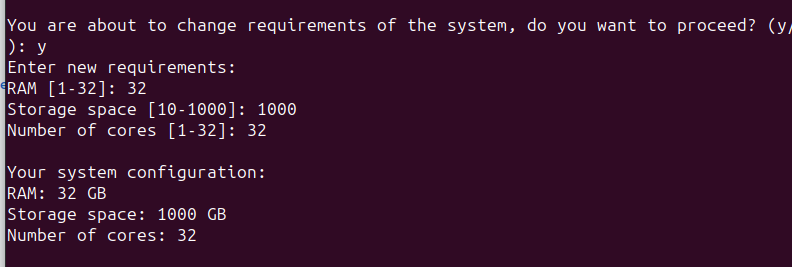
default\_space = 256;

default\_core = 8;

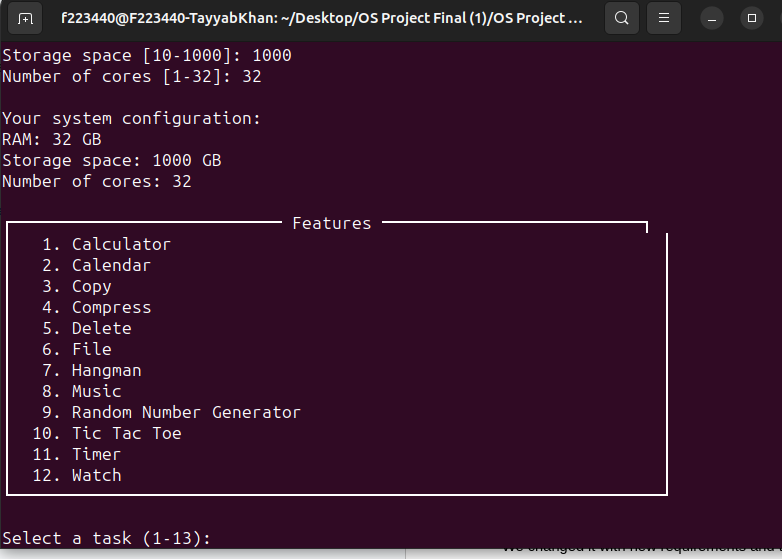
But we can edit these by going to kernal mode like this:



User is prompted if he wants to proceed, upon entering y we get:

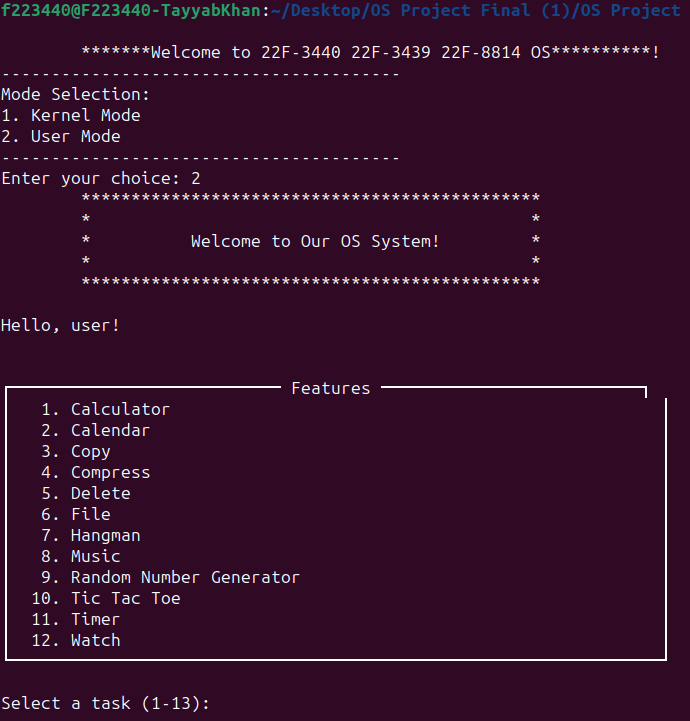


We changed it with new requirements and the message of new system configuration is shown on the screen.

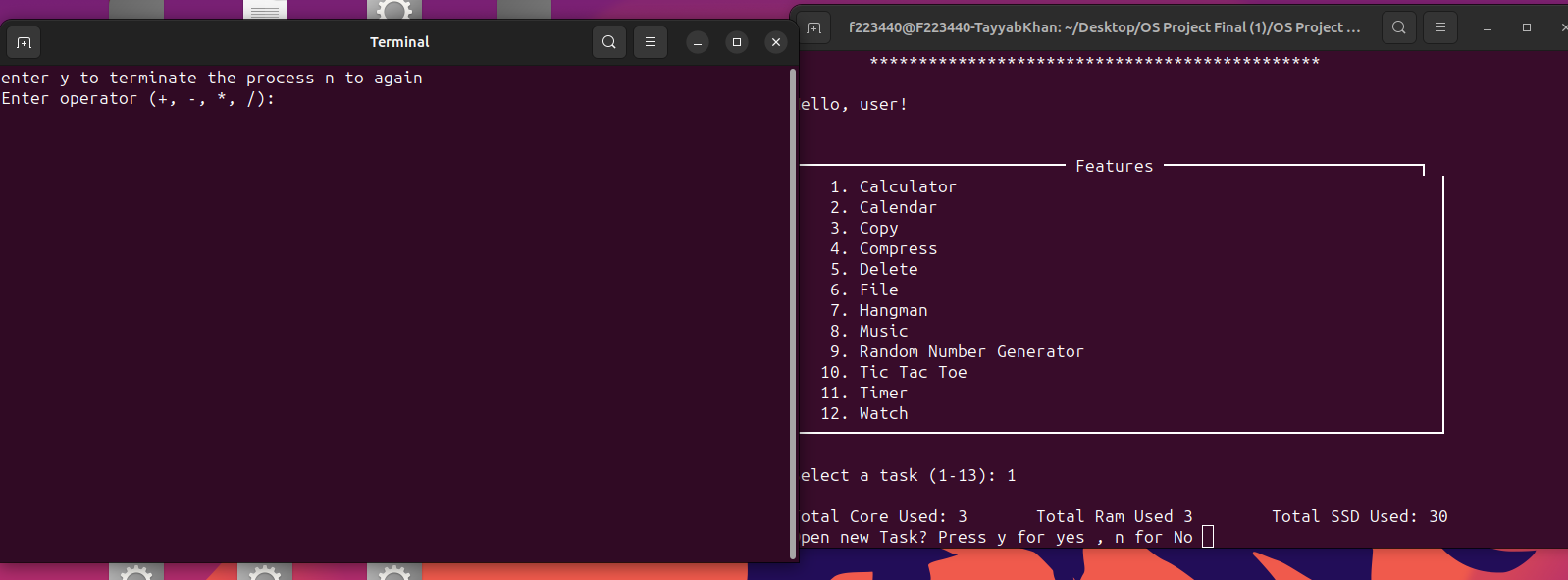
After this it proceeds to User Mode:  


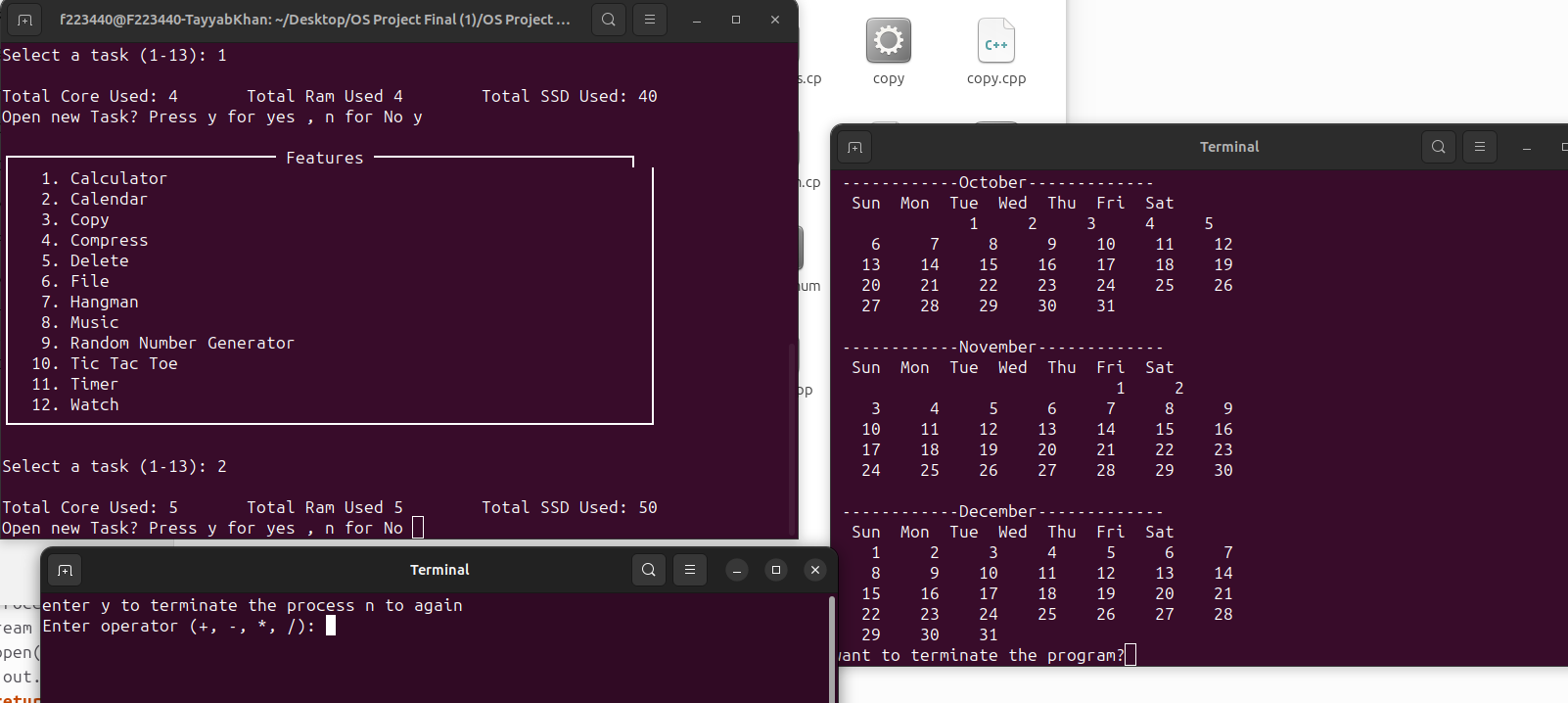
## User Mode:

When we enter User mode, the following features console is displayed.   
Here user can select new tasks which will open in the new terminal.



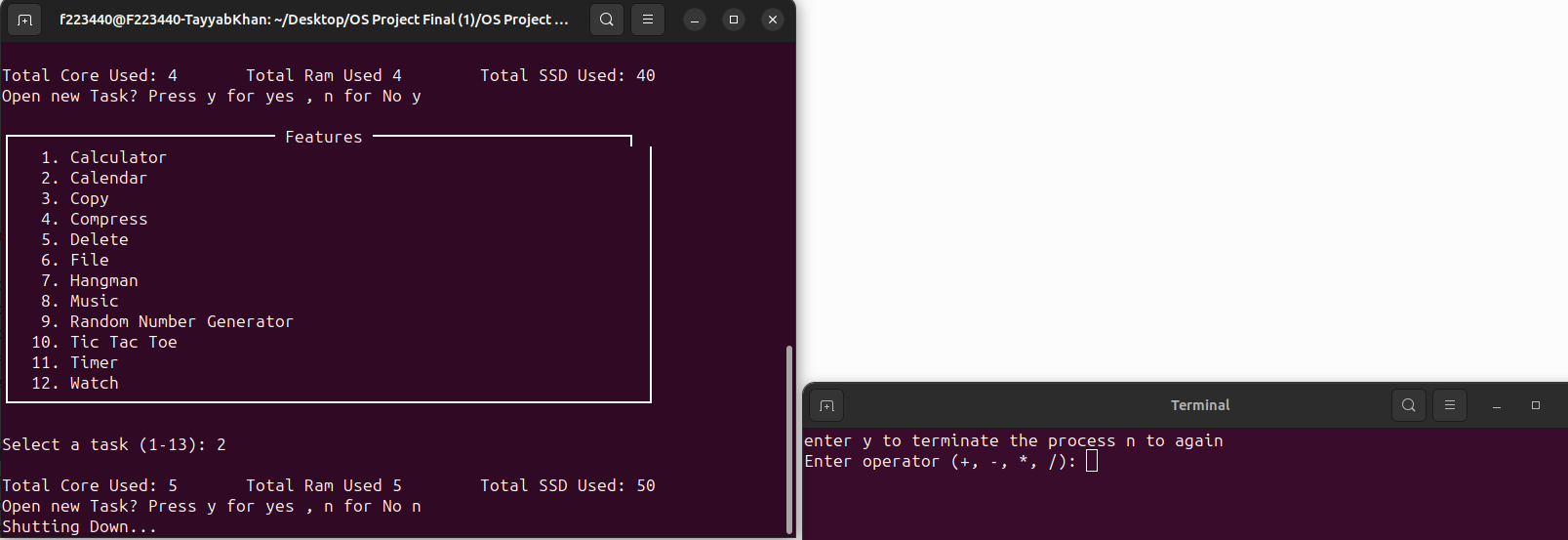
### Application running:

Whenever we run a new task, it displays the RAM, CORES and SPACE being used by that application:  


If multiple processes are opened it displays the cores and ram and space:  


We can use the processes on their sepreate console windows noramlly how we would use.

### Operating System Exits:

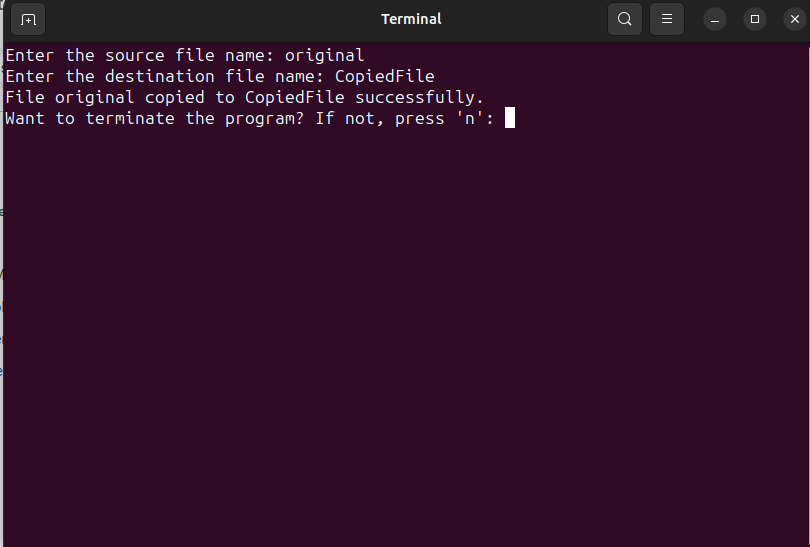
When we press ‘n’ it will close the Operating system and the kill command for terminals will kill all the terminals as well:  


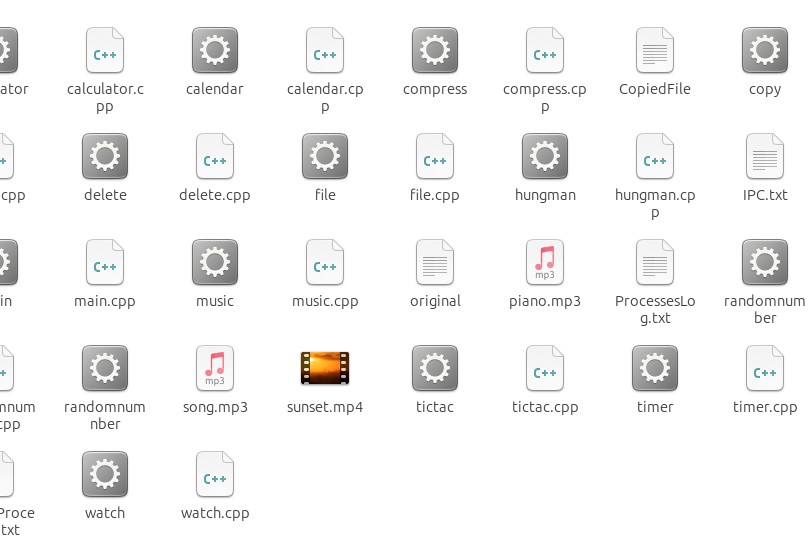
## Different Applications:

### File Management/Text Editor:

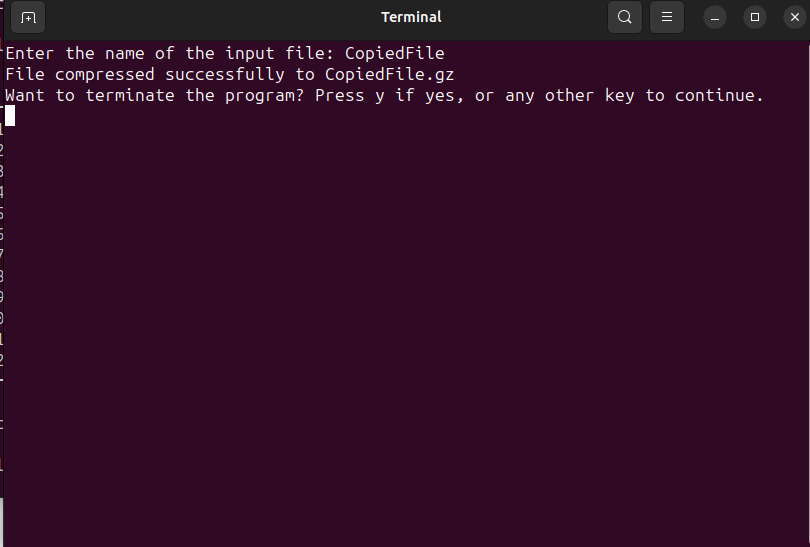
1. Copy: It makes a copy of the source file to the folder where OS exists.

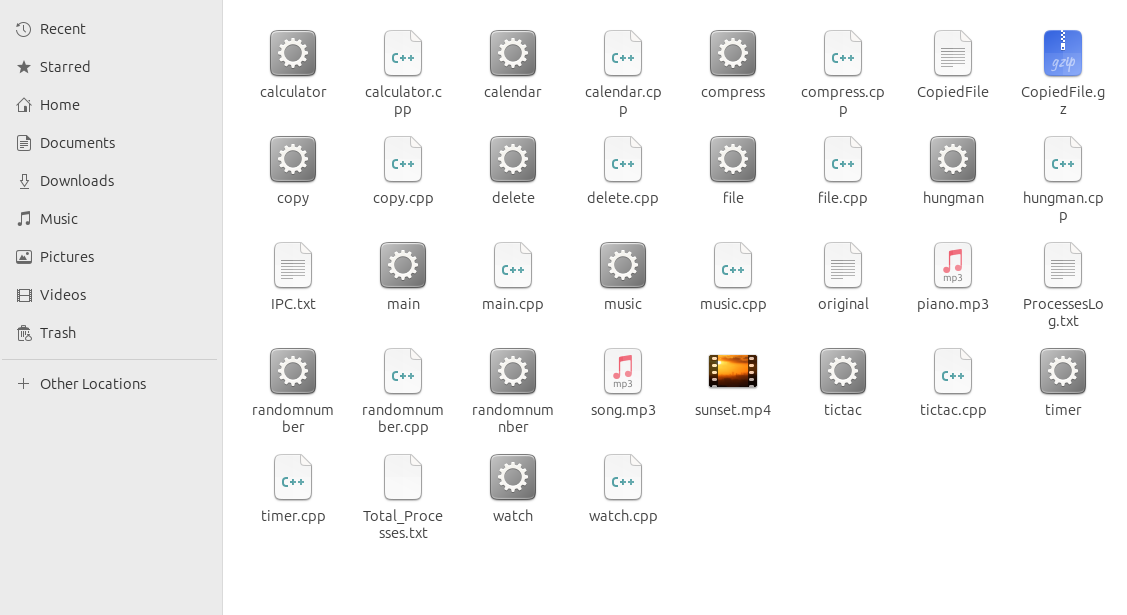
Working: We will make a copy of **Original.txt** file with the name “CopiedFile.txt”



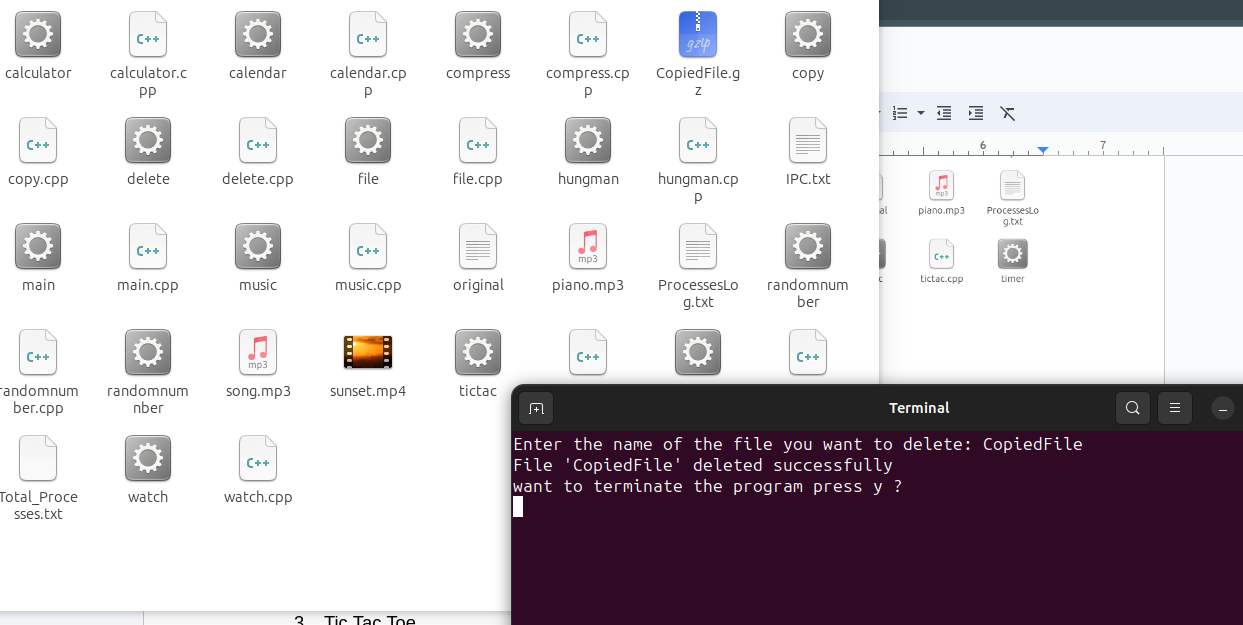


1. Compress: This helps compresses file that we give the name of, for example we will compress the file “CopiedFile” and then see the output

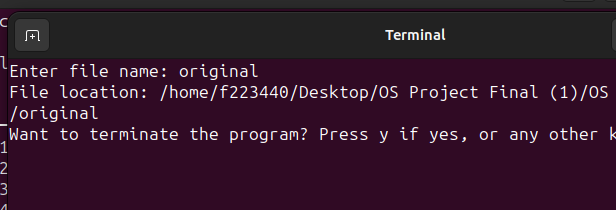




1. Delete: This will delete the file that we give the name of file. We will delete the **CopiedFile.txt:**

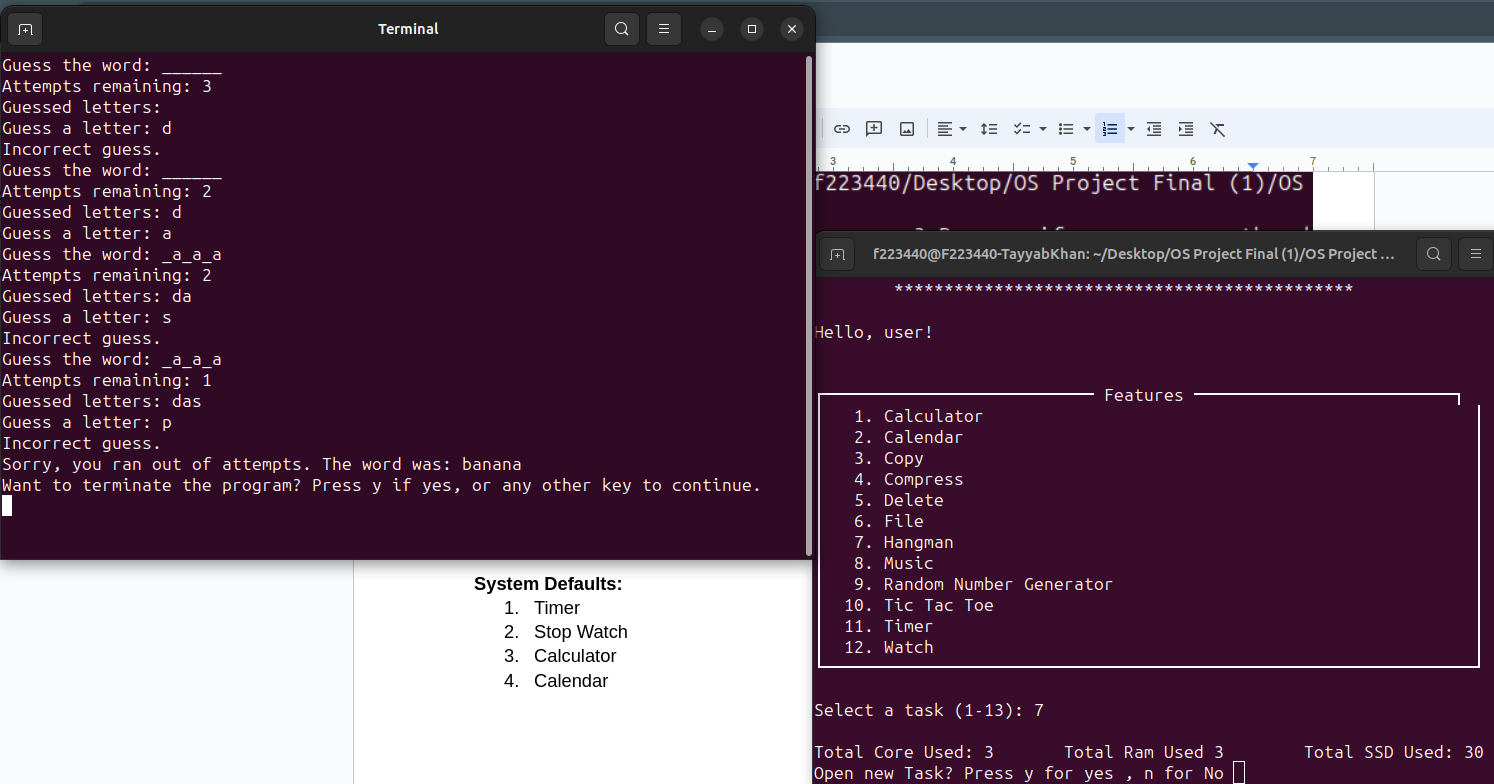


1. File: This will give us the location of the file name we give it. It works just like -ls command we have in Linux: Suppose we want to find the location of the **original.txt** file

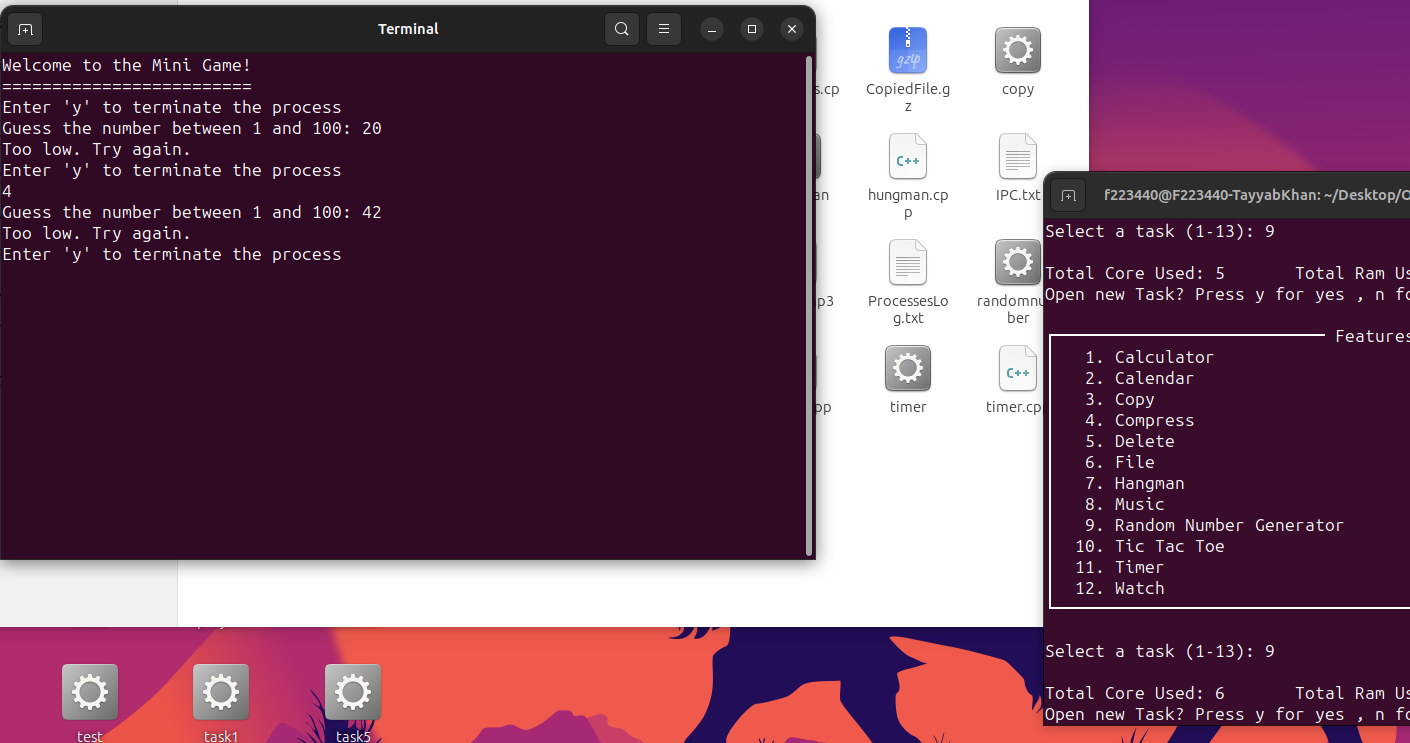


### **Games:**

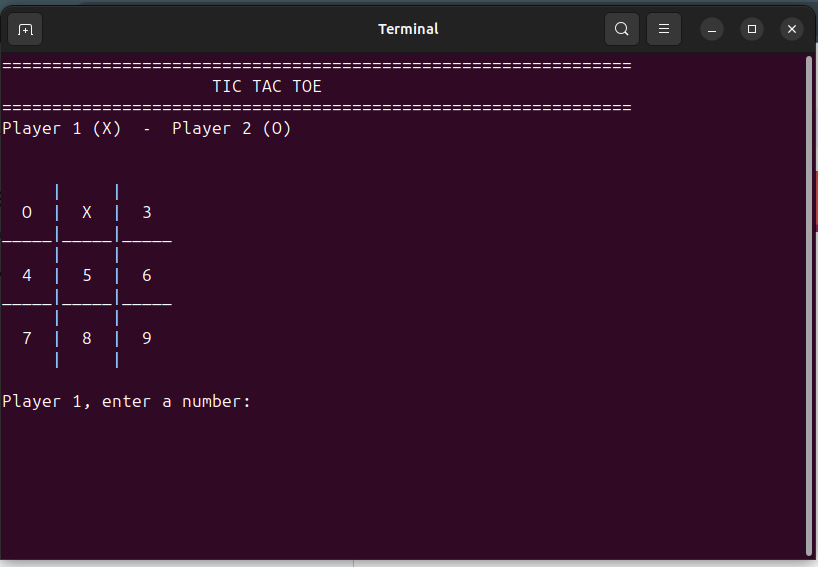
1. Hangman



1. Random Number Game



1. Tic Tac Toe

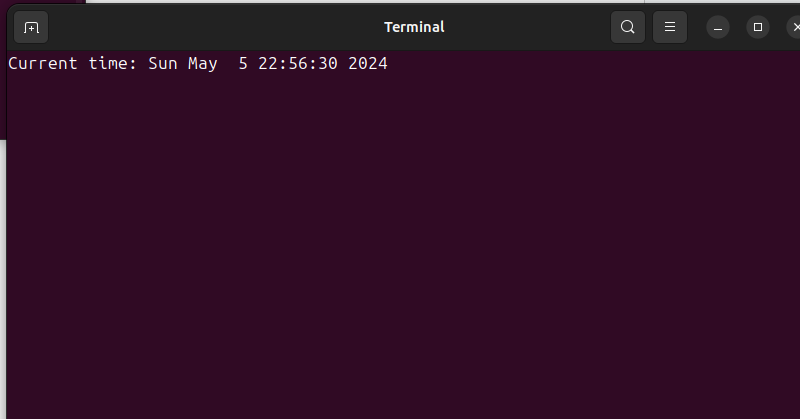


**Music:**

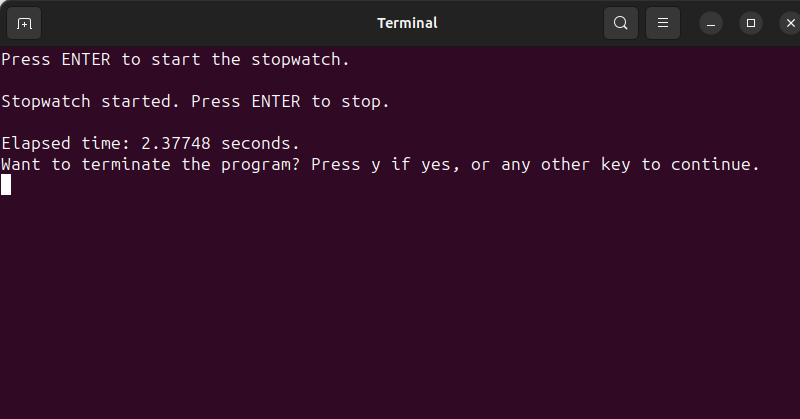
1. Music Player

### **System Defaults:**

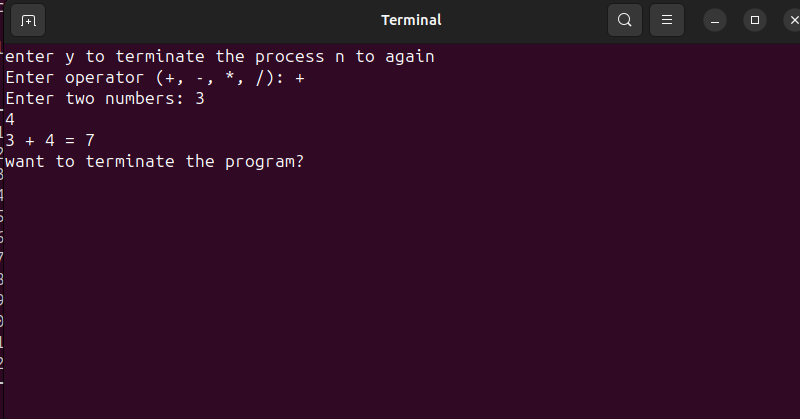
1. **Time:** displays the current time on the new terminal, behaves like a clock.



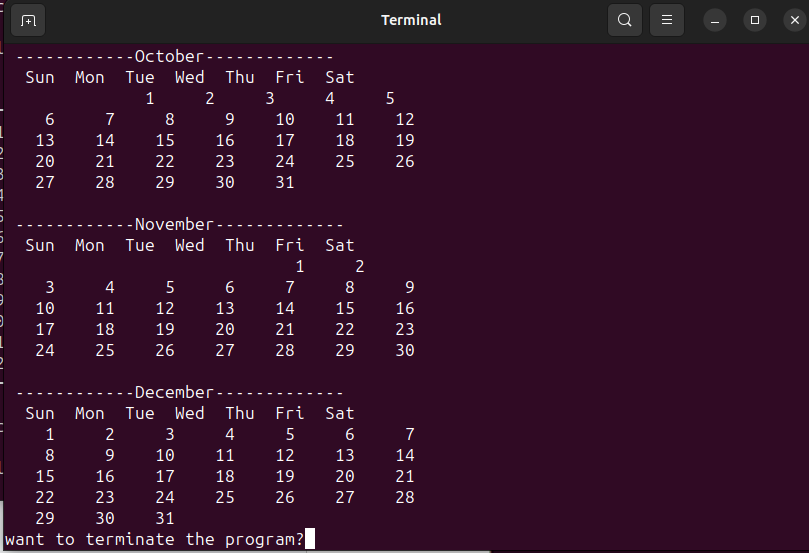
1. **Stop Watch**



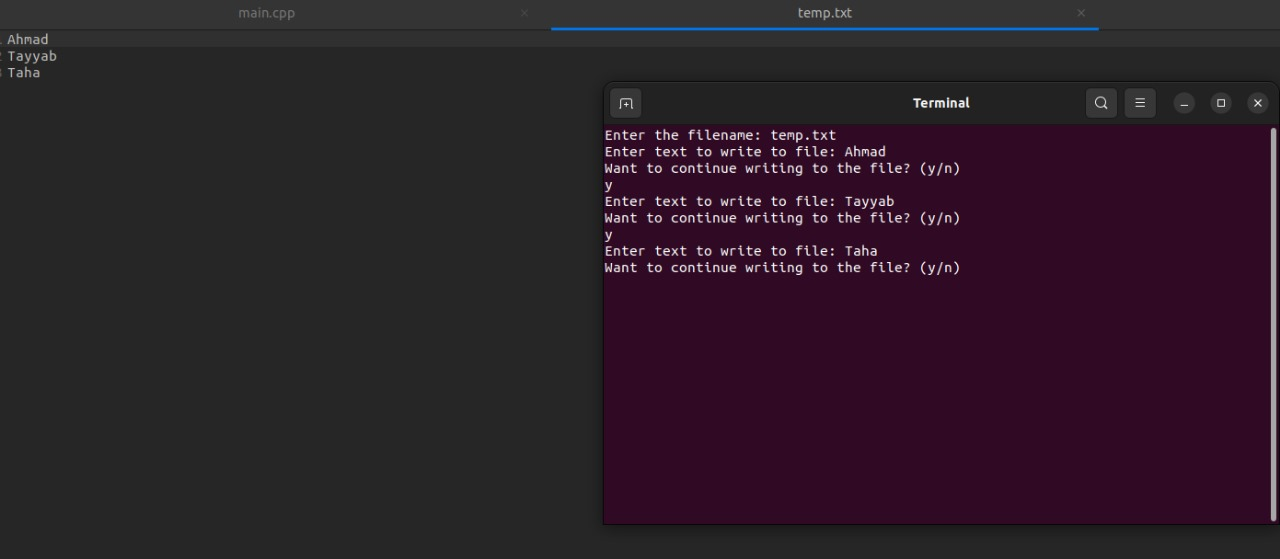
1. **Calculator**



1. **Calendar**



**Text Editor:** this is a basic text editor that will auto save as well when we write to it after every interval.



Code Working

### **Process Synchronization:**

**Semaphore Usage**

1. Semaphore Initialization
   * A semaphore named semaphore is initialized with an initial value of 1 using sem\_init(&semaphore, 0, 1);.
   * This semaphore is used to protect shared resources and ensure mutual exclusion.
2. Acquiring and Releasing Semaphores
   * Before accessing shared resources, a thread acquires the semaphore using sem\_wait(&semaphore);.
   * After accessing the shared resources, the thread releases the semaphore using sem\_post(&semaphore);.
3. Shared Resource Protection
   * The following variables are shared among threads and need protection:
     + ram\_check
     + space\_check
     + core\_check
4. Protecting Access to Shared Resources
   * The semaphore ensures that only one thread can access the shared resources at a time.

runCommands function:

void \*runCommands(void \*arg) {

sem\_wait(&semaphore); // Acquire the semaphore before accessing shared resources

core\_check++;

ram\_check++;

space\_check += 10;

cout << "\nTotal Core Used: " << core\_check << "\t Total Ram Used " << ram\_check << "\t Total SSD Used: " << space\_check << endl;

sem\_post(&semaphore); // Release the semaphore after accessing shared resources

// Rest of the function

}

**Example**

Suppose multiple threads are executing runCommands function concurrently. Here's how the process synchronization works:

1. Thread A Enters runCommands
   * Thread A acquires the semaphore.
   * Thread A increments core\_check, ram\_check, and updates space\_check.
   * Thread A releases the semaphore.
2. Thread B Enters runCommands
   * Thread B tries to acquire the semaphore but is blocked because Thread A holds it.
   * Thread B waits until Thread A releases the semaphore.
3. Thread A Exits
   * Thread A releases the semaphore.
   * Semaphore value becomes 1 (indicating it's available).
4. Thread B Continues
   * Thread B acquires the semaphore.
   * Thread B increments core\_check, ram\_check, and updates space\_check.
   * Thread B releases the semaphore.

### **Resource Allocation**

* How it's Managed:
  + Resources like RAM, storage space, and CPU cores are allocated based on user requirements.
  + Users specify resource requirements, which are then managed by the system.
  + For example, users specify RAM, storage space, and CPU core numbers interactively.
  + Resource allocation is controlled by semaphores to ensure mutual exclusion.

### **Multitasking**

How it's Managed:

* + Multitasking is achieved by running multiple threads concurrently.
  + The system can perform multiple tasks simultaneousl on seperate terminals

### **Process Scheduling**

1. Task Queuing:
   * Tasks chosen by the user are enqueued in a ready queue.
   * If the system is at maximum capacity (i.e., all CPU cores are busy), the tasks are queued until resources become available.
2. Task Execution:
   * Tasks are executed based on their priority in the ready queue.
   * Each task is executed in a separate thread.
3. Resource Availability:
   * The system checks if resources like RAM, CPU cores, and storage space are available.
   * If resources are available, the task is dequeued and executed.
   * If resources are not available, the task remains in the queue until resources become available.

### **Code Implementation**

* Ready Queue:
  + A queue<int> ready\_queue; is used to hold the IDs of tasks ready for execution.
* Task Enqueuing:
  + When a user selects a task, its ID is pushed into the ready queue.

ready\_queue.push(task);

* Task Dequeuing and Execution:
  + The main() function continuously checks the ready queue for tasks.
  + If resources are available, it dequeues a task and executes it.

if (!ready\_queue.empty()) {

// Dequeue the task

task = ready\_queue.front();

ready\_queue.pop();

// Execute the task

}

Resource Management:

* + Resources like RAM, CPU cores, and storage space are managed to ensure that only a certain number of tasks are executed concurrently.
  + Semaphore protects shared resource access and ensures mutual exclusion.

**Complete Code:**  
#include <iostream>

#include <fstream>

#include <unistd.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <dirent.h> // for opendir, readdir, closedir

#include <cstring> // for strcmp

#include <string>

#include <thread>

#include <mutex>

#include <semaphore.h>

#include <chrono>

#include <queue>

#include <cstdio>

#include <iomanip>

#include <cstdlib> // for system(), atoi()

using namespace std;

const int max\_ram = 32, min\_ram = 1, default\_ram = 8;

const int max\_space = 1000, min\_space = 10, default\_space = 256;

const int max\_core = 32, min\_core = 1, default\_core = 8;

int ram = default\_ram;

int space = default\_space;

int core = default\_core;

// shared resource protected by a semaphore

sem\_t semaphore;

int ram\_check = 0;

int space\_check = 0;

int core\_check = 0;

bool AddProcessToLog() {

ofstream out;

out.open("ProcessesLog.txt", ios::app);

if (!out.is\_open())

return false;

out << 0 << endl;

out.close();

return true;

}

bool RemoveProcessFromLog() {

ifstream in;

in.open("ProcessesLog.txt");

if (!in.is\_open())

return false;

string input;

int count = 0;

while (!in.eof()) {

getline(in, input);

count++;

}

in.close();

ofstream out;

out.open("ProcessesLog.txt");

for (int i = 1; i < count - 1; i++) {

out << 0 << endl;

}

out.close();

return true;

}

int GetCountProcessLog() {

ifstream in;

in.open("ProcessesLog.txt");

if (!in.is\_open())

return -1;

string input;

int count = 0;

while (!in.eof()) {

getline(in, input);

count++;

}

in.close();

return count - 1;

}

void ClearFile() {

ofstream out;

out.open("Total\_Processes.txt");

cout << "";

out.close();

}

void \*runCommands(void \*arg) {

sem\_wait(&semaphore); // acquire the semaphore before accessing the shared resource

core\_check++;

ram\_check++;

space\_check += 10;

cout << "\nTotal Core Used: " << core\_check << "\t Total Ram Used " << ram\_check << "\t Total SSD Used: " << space\_check << endl;

sem\_post(&semaphore); // release the semaphore after accessing the shared resource

string filename = \*static\_cast<string\*>(arg);

string compileCmd = "g++ -o " + filename + " " + filename + ".cpp";

string runCmd = "gnome-terminal -- ./" + filename;

if (filename == "compress") {

compileCmd = "g++ -o " + filename + " " + filename + ".cpp -lz";

runCmd = "gnome-terminal -- ./" + filename;

} else if (filename == "music") {

compileCmd = "g++ -o " + filename + " " + filename + ".cpp -lao -lmpg123";

// Prompt the user to choose the music file

char choice;

cout << "Which music file do you want to play? Enter '1' for song.mp3 or '2' for piano.mp3: ";

cin >> choice;

// Construct the run command based on user's choice

if (choice == '1') {

runCmd = "./" + filename + " song.mp3";

} else if (choice == '2') {

runCmd = "./" + filename + " piano.mp3";

} else {

cerr << "Invalid choice. Exiting music player." << endl;

pthread\_exit(NULL);

RemoveProcessFromLog(); // Remove zeros when the process is terminated

}

}

system(compileCmd.c\_str());

system(runCmd.c\_str());

pthread\_exit(NULL);

RemoveProcessFromLog(); // Remove zeros when the process is terminated

}

void killTerminals() {

system("pkill -f gnome-terminal");

}

void startingMenu() {

cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\t\* \*\n";

cout << "\t\* Welcome to Our OS System! \*\n";

cout << "\t\* \*\n";

cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout << "Hello, user!\n\n";

}

void displayFeatures() {

cout << "\n┌─────────────────────────── Features ──────────────────────────┐\n";

cout << "│ 1. Calculator │\n";

cout << "│ 2. Calendar │\n";

cout << "│ 3. Copy │\n";

cout << "│ 4. Compress │\n";

cout << "│ 5. Delete │\n";

cout << "│ 6. File │\n";

cout << "│ 7. Hangman │\n";

cout << "│ 8. Music │\n";

cout << "│ 9. Random Number Generator │\n";

cout << "│ 10. Tic Tac Toe │\n";

cout << "│ 11. Timer │\n";

cout << "│ 12. Watch │\n";

cout << "└─────────────────────────────────────────────────────────────────┘\n";

}

int main(int argc, char \*argv[]) {

string os\_name = "22F-3440 22F-3439 22F-8814 OS";

string args;

char choice\_flag = 'y';

char choice\_flag\_inner = 'n';

char ans;

char choice = 'y';

int ready\_queue\_flag = 0;

int task = 0;

bool kernel\_check = false, user\_check = false;

int req\_space;

int req\_ram;

sem\_init(&semaphore, 0, 1);

queue<int> ready\_queue;

cout << "\n\t\*\*\*\*\*\*\*Welcome to " << os\_name << "\*\*\*\*\*\*\*\*\*\*!\n";

cout << "----------------------------------------\n";

cout << "Mode Selection:\n";

cout << "1. Kernel Mode\n";

cout << "2. User Mode\n";

cout << "----------------------------------------\n";

int input;

while (ready\_queue\_flag == 0) {

cout << "Enter your choice: ";

cin >> input;

if (input == 1 || input == 2) {

ready\_queue\_flag++;

} else {

cout << "Invalid choice. Please enter 1 or 2.\n";

}

}

if (input == 1) {

kernel\_check = true;

startingMenu();

} else if (input == 2) {

user\_check = true;

startingMenu();

}

while (choice == 'y') {

if (kernel\_check == true) {

cout << "You are about to change requirements of the system, do you want to proceed? (y/n): ";

cin >> ans;

if (ans == 'y') {

cout << "Enter new requirements:\n";

cout << "RAM [" << min\_ram << "-" << max\_ram << "]: ";

cin >> req\_ram;

if (req\_ram < min\_ram) {

cout << "RAM too low, using default value.\n";

} else if (req\_ram > max\_ram) {

cout << "RAM too high, using maximum value.\n";

ram = max\_ram;

} else {

ram = req\_ram;

}

cout << "Storage space [" << min\_space << "-" << max\_space << "]: ";

cin >> req\_space;

if (req\_space < min\_space) {

cout << "Storage space allocated is low, System will use default space!\n";

} else if (req\_space > max\_space) {

cout << "Storage space allocated is too much, System will use default space!\n";

space = max\_space;

} else {

space = req\_space;

}

cout << "Number of cores [" << min\_core << "-" << max\_core << "]: ";

int req\_core;

cin >> req\_core;

if (req\_core < min\_core) {

cout << "Number of cores too low, using default value.\n";

} else if (req\_core > max\_core) {

cout << "Number of cores too high, using maximum value.\n";

core = max\_core;

} else {

core = req\_core;

}

cout << "\nYour system configuration:\n";

cout << "RAM: " << ram << " GB\n";

cout << "Storage space: " << space << " GB\n";

cout << "Number of cores: " << core << "\n";

}

}

core\_check = GetCountProcessLog();

ram\_check = GetCountProcessLog();

space\_check = GetCountProcessLog() \* 10;

if (choice\_flag == 'y') {

if (GetCountProcessLog() == core) {

while (true) {

displayFeatures();

cout << "\nSelect a task (1-13): ";

cin >> task;

ready\_queue.push(task);

cout << "\nMAXIMUM SYSTEM CAPACITY REACHED! New Processes will wait in ready queue, until resources become free!\n";

cout << "Do you want to perform another task? (y/n) ";

cin >> choice;

choice\_flag\_inner = 'y';

if (choice == 'n') {

choice\_flag = 'n';

}

if (!ready\_queue.empty()) {

if (ready\_queue.front() > 0 && ready\_queue.front() < 14) {

task = ready\_queue.front();

core\_check = GetCountProcessLog();

ram\_check = GetCountProcessLog();

space\_check = GetCountProcessLog() \* 10;

}

ready\_queue.pop();

break;

}

if (GetCountProcessLog() != core) {

break;

}

}

} else {

if (ready\_queue.empty()) {

displayFeatures();

cout << "\nSelect a task (1-13): ";

cin >> task;

}

if (!ready\_queue.empty()) {

if (ready\_queue.front() > 0 && ready\_queue.front() < 14) {

task = ready\_queue.front();

core\_check = GetCountProcessLog();

ram\_check = GetCountProcessLog();

space\_check = GetCountProcessLog() \* 10;

}

ready\_queue.pop();

}

}

} else {

if (GetCountProcessLog() == core) {

while (true) {

if (GetCountProcessLog() != core) {

break;

}

}

}

if (GetCountProcessLog() < core) {

int check\_core\_loop = GetCountProcessLog();

while (true) {

if (GetCountProcessLog() < check\_core\_loop) {

break;

}

}

}

if (ready\_queue.empty()) {

break;

}

if (!ready\_queue.empty()) {

if (ready\_queue.front() > 0 && ready\_queue.front() < 14) {

task = ready\_queue.front();

core\_check = GetCountProcessLog();

ram\_check = GetCountProcessLog();

space\_check = GetCountProcessLog() \* 10;

}

ready\_queue.pop();

}

}

switch (task) {

case 1:

args = "calculator";

if (GetCountProcessLog() == core) {

ready\_queue.push(1);

} else {

AddProcessToLog();

pthread\_t thread\_1;

pthread\_create(&thread\_1, NULL, runCommands, &args);

pthread\_join(thread\_1, NULL);

}

break;

case 2:

args = "calendar";

if (GetCountProcessLog() == core) {

ready\_queue.push(2);

} else {

AddProcessToLog();

pthread\_t thread\_2;

pthread\_create(&thread\_2, NULL, runCommands, &args);

pthread\_join(thread\_2, NULL);

}

break;

case 3:

args = "copy";

if (GetCountProcessLog() == core) {

ready\_queue.push(3);

} else {

AddProcessToLog();

pthread\_t thread\_3;

pthread\_create(&thread\_3, NULL, runCommands, &args);

pthread\_join(thread\_3, NULL);

}

break;

case 4:

args = "compress";

if (GetCountProcessLog() == core) {

ready\_queue.push(4);

} else {

AddProcessToLog();

pthread\_t thread\_4;

pthread\_create(&thread\_4, NULL, runCommands, &args);

pthread\_join(thread\_4, NULL);

}

break;

case 5:

args = "delete";

if (GetCountProcessLog() == core) {

ready\_queue.push(5);

} else {

AddProcessToLog();

pthread\_t thread\_5;

pthread\_create(&thread\_5, NULL, runCommands, &args);

pthread\_join(thread\_5, NULL);

}

break;

case 6:

args = "file";

if (GetCountProcessLog() == core) {

ready\_queue.push(6);

} else {

AddProcessToLog();

pthread\_t thread\_6;

pthread\_create(&thread\_6, NULL, runCommands, &args);

pthread\_join(thread\_6, NULL);

}

break;

case 7:

args = "hungman";

if (GetCountProcessLog() == core) {

ready\_queue.push(7);

} else {

AddProcessToLog();

pthread\_t thread\_7;

pthread\_create(&thread\_7, NULL, runCommands, &args);

pthread\_join(thread\_7, NULL);

}

break;

case 8:

args = "music";

if (GetCountProcessLog() == core) {

ready\_queue.push(8);

} else {

AddProcessToLog();

pthread\_t thread\_8;

pthread\_create(&thread\_8, NULL, runCommands, &args);

pthread\_join(thread\_8, NULL);

}

break;

case 9:

args = "randomnumber";

if (GetCountProcessLog() == core) {

ready\_queue.push(9);

} else {

AddProcessToLog();

pthread\_t thread\_9;

pthread\_create(&thread\_9, NULL, runCommands, &args);

pthread\_join(thread\_9, NULL);

}

break;

case 10:

args = "tictac";

if (GetCountProcessLog() == core) {

ready\_queue.push(10);

} else {

AddProcessToLog();

pthread\_t thread\_10;

pthread\_create(&thread\_10, NULL, runCommands, &args);

pthread\_join(thread\_10, NULL);

}

break;

case 11:

args = "timer";

if (GetCountProcessLog() == core) {

ready\_queue.push(11);

} else {

AddProcessToLog();

pthread\_t thread\_11;

pthread\_create(&thread\_11, NULL, runCommands, &args);

pthread\_join(thread\_11, NULL);

}

break;

case 12:

args = "watch";

if (GetCountProcessLog() == core) {

ready\_queue.push(13);

} else {

AddProcessToLog();

pthread\_t thread\_13;

pthread\_create(&thread\_13, NULL, runCommands, &args);

pthread\_join(thread\_13, NULL);

}

break;

default:

cerr << "Invalid task number." << endl;

break;

}

if (choice\_flag == 'y' && ready\_queue.empty() && choice\_flag\_inner == 'n') { // flag to check if user does not want to perform another task

cout << "Open new Task? Press y for yes , n for No ";

cin >> choice;

}

choice\_flag\_inner = 'n';

if (choice == 'n') { // condition to check user does not want to perform another task

choice\_flag = 'n';

if (!ready\_queue.empty()) { // condition to check if queue is empty or not

choice = 'y';

continue;

}

if (ready\_queue.empty()) { // condition to check if queue is empty or not

choice = 'n';

}

}

}

ClearFile();

cout << "Shutting Down";

for (int i = 0; i < 5; i++) {

cout << ".";

cout.flush();

this\_thread::sleep\_for(chrono::seconds(1));

}

cout << endl;

// Kill all child processes' terminals

killTerminals();

return 0;

}