****

**OPERATING SYSTEM LAB**

**FINAL PROJECT**

**PROJECT BY:**

**NAME: Aman Imran**

**ROLL NO: 23f-0738**

**HOW TO RUN?**

**Execute the following commands:**

1. Sudo apt install libgtk-3-dev

2. cd ~/Desktop/OS-Project (Make sure the project directory is on Desktop)

3. ./build.sh 4 8 256

**CODES:**

**armstrong.cpp:**

#include <iostream>

using namespace std;

int main() {

int num, originalNum, remainder, result = 0;

cout << "Enter a three-digit integer: ";

cin >> num;

originalNum = num;

while (originalNum != 0) {

remainder = originalNum % 10;

result += remainder \* remainder \* remainder;

originalNum /= 10;

}

if (result == num)

cout << num << " is an Armstrong number.";

else

cout << num << " is not an Armstrong number.";

return 0;

}

**bubblesort.cpp:**

#include <iostream>

#include <vector>

void bubbleSort(std::vector<int>& arr) {

int n = arr.size();

bool swapped;

do {

swapped = false;

for (int i = 1; i < n; i++) {

if (arr[i - 1] > arr[i]) {

std::swap(arr[i - 1], arr[i]);

swapped = true;

}

}

n--;

} while (swapped);

}

int main() {

int size;

std::cout << "Enter the number of elements: ";

std::cin >> size;

std::vector<int> arr(size);

std::cout << "Enter " << size << " elements:\n";

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

std::cin >> arr[i];

}

bubbleSort(arr);

std::cout << "Sorted array: ";

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

return 0;

}

**calculator.cpp:**

# include <iostream>

using namespace std;

int main()

{

char op;

float num1, num2;

system("clear");

cout << "Enter operator: \n +, -, \*, /: ";

cin >> op;

cout << "Enter two operands: ";

cin >> num1 >> num2;

switch(op) {

case '+':

cout << num1 << " + " << num2 << " = " << num1 + num2;

break;

case '-':

cout << num1 << " - " << num2 << " = " << num1 - num2;

break;

case '\*':

cout << num1 << " \* " << num2 << " = " << num1 \* num2;

break;

case '/':

cout << num1 << " / " << num2 << " = " << num1 / num2;

break;

default:

// If the operator is other than +, -, \* or /, error message is shown

cout << "Error! operator is not correct";

break;

}

cout << "\n\n";

return 0;

}

**conversion.cpp:**

#include <iostream>

#include <bitset>

#include <iomanip>

int main() {

int decimalNumber;

std::cout << "Enter a decimal number: ";

std::cin >> decimalNumber;

std::bitset<sizeof(int) \* 8> binaryNumber(decimalNumber);

std::cout << "Decimal number is: " << decimalNumber << std::endl;

std::cout << "Binary: " << binaryNumber << std::endl;

std::cout << "Octal: " << std::oct << decimalNumber << std::endl;

std::cout << "Hexadecimal: 0x" << std::hex << std::uppercase << decimalNumber << std::endl;

return 0;

}

**factorial.cpp:**

#include <iostream>

using namespace std;

int factorial(int n)

{

if (n == 0)

return 1;

return n \* factorial(n - 1);

}

int main()

{

cout<<"Enter number : ";

int num ;

cin>>num;

cout << "Factorial of "<< num << " is " << factorial(num) << endl;

return 0;

}

**fibonacci.cpp:**

#include <iostream>

using namespace std;

// 0 1 1 2 3 5 8 13

// sum of two previous consequtive numbers

int main() {

int end\_number, num1 = 0, num2 = 1, nextTerm = 0;

cout << "Enter the number of terms: ";

cin >> end\_number;

cout << "----------------- Fibonacci Series till given number ----------\n ";

for (int i = 1; i <= end\_number; i++) {

// Prints the first two terms.

if(i == 1) {

cout <<num1<< ", ";

continue;

}

if(i == 2) {

cout << num2 << ", ";

continue;

}

nextTerm = num1 + num2;

num1 = num2;

num2 = nextTerm;

cout << nextTerm << ", ";

}

return 0;

}

**num\_guess\_game.cpp:**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main()

{

int number\_to\_guess, guess, no\_of\_tries = 0;

srand(time(0));

number\_to\_guess = rand() % 1000 + 1;

cout << "--------------- Number Guess Game --------------- \n\n";

do

{

cout << "Enter a guess between 1 and 1000 : ";

cin >> guess;

no\_of\_tries++;

if (guess > number\_to\_guess)

cout << "Your guess is above the actual number !\n\n";

else if (guess < number\_to\_guess)

cout << "Your guess is below the actual number !\n\n";

else

cout << "\nHURRAHH! You got it in " << no\_of\_tries << " guesses!\n";

} while (guess != number\_to\_guess);

return 0;

}

**os\_notepad.cpp:**

#include<iostream>

#include <fstream>

using namespace std;

int main()

{

char text[900];

fstream new\_file;

new\_file.open("new\_file.txt",ios::out);

if(!new\_file)

{

cout<<"File creation failed";

}

else

{

cout<<"New file created\n\n";

cin.getline(text, sizeof(text));

new\_file<<text<<endl;

new\_file.close(); // Step 4: Closing file

}

return 0;

}

**palindrome.cpp:**

#include <iostream>

using namespace std;

int main()

{

int n, num, digit, rev = 0;

cout << "Enter a positive number: ";

cin >> num;

n = num;

do

{

digit = num % 10;

rev = (rev \* 10) + digit;

num = num / 10;

} while (num != 0);

cout << " The reverse of the number is: " << rev << endl;

if (n == rev)

cout << " The number is a palindrome.";

else

cout << " The number is not a palindrome.";

return 0;

}

**prime.cpp:**

#include <iostream>

using namespace std;

int main() {

int i, n;

bool isPrime = true;

cout << "Enter a positive integer: ";

cin >> n;

// 0 and 1 are not prime numbers

if (n == 0 || n == 1) {

isPrime = false;

}

else {

for (i = 2; i <= n / 2; ++i) {

if (n % i == 0) {

isPrime = false;

break;

}

}

}

if (isPrime)

cout << n << " is a prime number";

else

cout << n << " is not a prime number";

return 0;

}

**sys\_time.cpp:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

using namespace std;

int main()

{

time\_t t = time(NULL);

tm \*tPtr = localtime(&t);

cout << "\n\n Display the Current Date and Time :\n";

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

cout << " seconds = " << (tPtr->tm\_sec) << endl;

cout << " minutes = " << (tPtr->tm\_min) << endl;

cout << " hours = " << (tPtr->tm\_hour) << endl;

cout << " day of month = " << (tPtr->tm\_mday) << endl;

cout << " month of year = " << (tPtr->tm\_mon) + 1 << endl;

cout << " year = " << (tPtr->tm\_year) + 1900 << endl;

cout << " weekday = " << (tPtr->tm\_wday) << endl;

cout << " day of year = " << (tPtr->tm\_yday) << endl;

cout << " daylight savings = " << (tPtr->tm\_isdst) << endl;

cout << endl;

cout << endl;

cout << " Current Date: " << (tPtr->tm\_mday) << "/" << (tPtr->tm\_mon) + 1 << "/" << (tPtr->tm\_year) + 1900 << endl;

cout << " Current Time: " << (tPtr->tm\_hour) << ":" << (tPtr->tm\_min) << ":" << (tPtr->tm\_sec) << endl;

cout << endl;

return 0;

}

**song.cpp:**

#include<iostream>

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<pthread.h>

void \*fun(void \*arg)

{

system("mpg123 ~/Desktop/OS-Project/prgs/song2.mp3");

pthread\_exit(NULL);

return NULL;

}

using namespace std;

int main()

{

pthread\_t p1;

pthread\_create(&p1,NULL,fun,NULL);

pthread\_join(p1,NULL);

return 0;

}

**Main.cpp:**

#include <unistd.h>

#include <sys/wait.h>

#include <sstream>

#include <iostream>

#include <string>

#include <cstring>

#include <fstream>

#include <vector>

#include <sys/shm.h>

#include <pthread.h>

#include <stdlib.h>

#include <algorithm>

#include <semaphore.h>

#include <signal.h>

#include <fstream>

#include <queue>

using namespace std;

struct dict

{

string key;

string value;

};

class prg\_info

{

public:

string name;

int req\_ram;

int req\_mem;

prg\_info(){}

prg\_info(string n, int r, int m)

{

name = n;

req\_mem = m;

req\_ram = r;

}

};

class proccess

{

public:

int pid;

prg\_info\* req\_resources;

proccess(){}

proccess(int id, prg\_info\* resources)

{

pid = id;

req\_resources = resources;

}

};

// \* Globals

int disk = 0, memory = 0, cores = 0;

dict \*commands;

pthread\_t last\_thread\_id;

bool running = false;

string userin;

int Count = 0; // Commands Count

prg\_info\* run\_prg;

vector<prg\_info>prg\_infos;

queue<proccess> FIFO\_Queue;

vector<int> processes;

pthread\_mutex\_t sync\_Process\_io;

pthread\_mutex\_t schedularMutex;

int Ramid, Coresid, Memid;

int \*remaining\_mem, \*remaining\_cores, \*remaining\_disk;

// \* -------

// \* Function Prototypes

void removeWhitespaces(string &str);

bool userLogin();

void readspecs();

prg\_info\* findProgram(string);

void readPrograms();

void readCommands();

void \*processCommand(void \*args);

void creatingProcess(prg\_info\*);

void writeProcesses();

void readProcesses();

void \*keepProcessesUpdated(void \*args);

void tot\_mem\_usage();

void \*scheduler(void \*args);

void manualinput(int);

// \* -------

void releaseResources()

{

cout << "\n\nResources Released!\n\n";

\*remaining\_cores = \*remaining\_cores + 1;

\*remaining\_disk = \*remaining\_disk + run\_prg->req\_mem;

\*remaining\_mem = \*remaining\_mem + run\_prg->req\_ram;

running = false;

}

void CTRL\_C\_Handler(int signum)

{

}

void Terminal\_Handler(int signum)

{

if(running == true)

{

releaseResources();

}

exit(signum);

}

int main(int argc, char \*\*argv)

{

// -- OS INITIALIZATIONS

// chdir("..");

signal(SIGINT, CTRL\_C\_Handler);

signal(SIGHUP, Terminal\_Handler);

Ramid = shmget((key\_t)2345, sizeof(int), 0666 | IPC\_CREAT);

Coresid = shmget((key\_t)2346, sizeof(int), 0666 | IPC\_CREAT);

Memid = shmget((key\_t)2347, sizeof(int), 0666 | IPC\_CREAT);

remaining\_mem = (int \*)shmat(Ramid, NULL, 0);

remaining\_cores = (int \*)shmat(Coresid, NULL, 0);

remaining\_disk = (int \*)shmat(Memid, NULL, 0);

// cout << \*remaining\_cores << endl;

// cout << \*remaining\_mem << endl;

// cout << \*remaining\_disk << endl;

pthread\_mutex\_init(&sync\_Process\_io, NULL);

pthread\_mutex\_init(&schedularMutex, NULL);

readspecs();

readPrograms();

readCommands();

writeProcesses();

if (pthread\_create(&last\_thread\_id, NULL, keepProcessesUpdated, NULL))

{

cout << "Process Log Init Failed.\nShutting Down Xaxis.\n";

exit(-1);

}

if (pthread\_create(&last\_thread\_id, NULL, scheduler, NULL))

{

cout << "Scheduler Init Failed.\nShutting Down Xaxis.\n";

exit(-2);

}

// -- IDR SE APNA START KR

if (argc != 0)

{

userin = "";

for (int i = 1; i < argc; i++)

{

userin += argv[i];

userin += " ";

}

}

manualinput(argc);

// -- Destroying

pthread\_mutex\_destroy(&sync\_Process\_io);

pthread\_mutex\_destroy(&schedularMutex);

}

/\*

\*/

void manualinput(int args)

{

// while(!userLogin());

// cin.ignore();

while (1)

{

usleep(10000);

if(running == false)

{

if (args == 1)

{

cout << "user@FantasyOs : ";

getline(cin, userin);

cin.clear();

}

else

{

args = 1;

}

if(\*remaining\_cores == 0)

{

cout << "[!]ERROR: No cores are available.\n";

}

else

{

if(userin == "exit")

{

exit(0);

}

else if (userin == "resusage")

{

cout << "\nCORES IN USE : " << cores - \*remaining\_cores << endl;

cout << "DISK IN USE : " << disk - \*remaining\_disk << " MBs\n";

cout << "MEMORY IN USE : " << memory - \*remaining\_mem << " MBs\n\n";

//userin.clear();

}

else if (userin == "specs")

{

cout << "\nCPU Cores : " << cores << endl;

cout << "MEMORY : " << memory << " MBs\n";

cout << "STORAGE : " << disk << " MBs" << " MBs\n\n";

//userin.clear();

}

else

{

if (pthread\_create(&last\_thread\_id, NULL, processCommand, NULL))

{

cout << "Thread Creation Failed, Couldn't Process Command.\n";

}

pthread\_join(last\_thread\_id, NULL);

usleep(20000);

}

}

}

}

}

/\*

\*/

void removeWhitespaces(string &str)

{

for (int i = 0; i < str.size(); i++)

{

if(str[i] == ' ' || str[i] == '\n')

{

str.erase(str.begin()+i);

i=0;

}

}

}

void readspecs()

{

ifstream reader("./osspecs");

reader >> cores >> memory >> disk;

}

/\*

\*/

void readCommands()

{

string comm\_tmp;

ifstream reader;

reader.open("commands");

// Count Commands

while (getline(reader, comm\_tmp))

{

Count++;

}

//------

commands = new dict[Count];

reader.close();

reader.open("commands");

for (int i = 0; i < Count; i++)

{

reader >> commands[i].key;

getline(reader, commands[i].value, '\n');

}

reader.close();

}

/\*

\*/

/\*

\*/

void \*processCommand(void \*args)

{

bool found = false;

string arg, comm;

stringstream split(userin);

split >> arg;

getline(split, comm);

if (arg == "run")

{

found = true;

removeWhitespaces(comm);

transform(comm.begin(), comm.end(), comm.begin(), ::tolower);

run\_prg = findProgram(comm);

if (run\_prg != NULL)

{

creatingProcess(run\_prg);

}

else

{

cout << "[!]ERROR: Invalid Program\n";

}

}

else

{

for (int j = 0; j < Count; j++)

{

if (arg == commands[j].key)

{

found = true;

string val = commands[j].value;

if (val.rfind("internal:", 0) == 0)

{

// Handle internal logic by rerouting userin

userin = val.substr(9);

pthread\_exit(0); // Let manualinput re-handle it

}

else if (val.substr(0, 2) == "cd")

{

chdir(&comm[0]);

}

else

{

system((val + comm).c\_str());

}

}

}

}

if (!found)

{

cout << "[!]ERROR: Invalid Command\n";

}

pthread\_exit(0);

}

void stopProcess(int pid)

{

string cmd = "kill -STOP ";

cmd += to\_string(pid);

system(&cmd[0]);

}

void removeProcessFromQueue(int pid)

{

readProcesses();

for (int i = 0; i < processes.size(); i++)

{

if (processes[i] == pid)

{

processes.erase(processes.begin() + i);

break;

}

}

writeProcesses();

}

void creatingProcess(prg\_info\* prg)

{

int pid = fork();

if (pid != 0)

{

FIFO\_Queue.push(proccess(pid, prg));

// processes.push\_back(pid);

writeProcesses();

usleep(20000);

wait(NULL);

running = false;

//userin.clear();

}

else

{

// stops the current proccess and wait for the scheduler to run it.

stopProcess(getpid());

string arg = "./prgs/" + prg->name;

system(&arg[0]);

removeProcessFromQueue(getpid());

\*remaining\_cores = \*remaining\_cores + 1;

\*remaining\_disk = \*remaining\_disk + prg->req\_mem;

\*remaining\_mem = \*remaining\_mem + prg->req\_ram;

exit(0);

}

return;

}

void writeProcesses()

{

pthread\_mutex\_lock(&sync\_Process\_io);

ofstream writer("./processes");

for(int i=0;i<processes.size(); i++){

writer << processes[i] << endl;

}

writer.close();

pthread\_mutex\_unlock(&sync\_Process\_io);

return;

}

void readProcesses()

{

pthread\_mutex\_lock(&sync\_Process\_io);

processes.clear();

int id;

string name;

ifstream reader;

reader.open("./processes");

while (reader >> id)

{

processes.push\_back(id);

}

reader.close();

pthread\_mutex\_unlock(&sync\_Process\_io);

}

void readPrograms()

{

string name;

int ram, mem;

ifstream reader;

reader.open("./programs");

do

{

reader >> name;

reader >> ram >> mem;

prg\_infos.push\_back(prg\_info(name, ram, mem));

} while (!reader.eof());

reader.close();

}

prg\_info\* findProgram(string name)

{

for (int i = 0; i < prg\_infos.size(); i++)

{

if(prg\_infos[i].name == name)

{

return &prg\_infos[i];

}

}

return NULL;

}

void \*keepProcessesUpdated(void \*args)

{

while (1)

{

readProcesses();

usleep(10000);

}

}

void \*scheduler(void \*args)

{

while (1)

{

pthread\_mutex\_lock(&schedularMutex);

int numDots = 1;

//busy waiting

while (\*remaining\_cores == 0 && running == false);

if (FIFO\_Queue.size())

{

proccess \*p;

p = &FIFO\_Queue.front();

while(\*remaining\_mem < p->req\_resources->req\_ram && running == false);

while(\*remaining\_disk < p->req\_resources->req\_mem && running == false);

processes.push\_back(p->pid);

writeProcesses();

string reviver = "kill -CONT ";

reviver += to\_string(p->pid);

system(&reviver[0]);

\*remaining\_cores = \*remaining\_cores - 1;

\*remaining\_disk = \*remaining\_disk - p->req\_resources->req\_mem;

\*remaining\_mem = \*remaining\_mem - p->req\_resources->req\_ram;

running = true;

usleep(2000);

FIFO\_Queue.pop();

}

pthread\_mutex\_unlock(&schedularMutex);

}

}

**Run.cpp:**

#include <sys/shm.h>

#include <cstdlib>

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

int Ramid = shmget((key\_t)2345, sizeof(int), 0666 | IPC\_CREAT);

int Coresid = shmget((key\_t)2346, sizeof(int), 0666 | IPC\_CREAT);

int Diskid = shmget((key\_t)2347, sizeof(int), 0666 | IPC\_CREAT);

int\* ram = (int \*)shmat(Ramid, NULL, 0);

int\* cores = (int \*)shmat(Coresid, NULL, 0);

int\* disk = (int \*)shmat(Diskid, NULL, 0);

if (argc == 4) {

\*cores = atoi(argv[1]);

\*ram = atoi(argv[2]);

\*disk = atoi(argv[3]);

}

return 0;

}

**Frontend.c:**

#include <stdlib.h>

#include <stdio.h>

#include <sys/types.h>

#include <signal.h>

#include <unistd.h>

#include <string.h>

#include <gtk/gtk.h>

#include <gtk/gtkx.h>

#include <math.h>

#include <ctype.h>

#include <pthread.h>

#include <stdbool.h>

// Global variables

GtkBuilder \*builder;

GtkWidget \*GTK\_Window;

GtkWidget \*Clock\_Widget;

GtkWidget \*quitButton;

// Thread control variables

volatile bool running = true;

volatile bool pressed = false;

pthread\_mutex\_t pressed\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_t time\_thread;

// ----- Safe command execution wrapper -----

char\* exec\_cmd(const char\* cmd) {

FILE \*fp;

char result[1000];

char \*output = NULL;

fp = popen(cmd, "r");

if (!fp) {

return strdup("error");

}

if (fgets(result, sizeof(result), fp) == NULL) {

pclose(fp);

return strdup("error");

}

pclose(fp);

// Remove trailing newline if present

result[strcspn(result, "\n")] = 0;

output = strdup(result);

if (!output) {

return strdup("error");

}

return output;

}

// ----- Safe GTK clock update -----

gboolean update\_clock(gpointer data) {

if (Clock\_Widget != NULL && data != NULL) {

gtk\_label\_set\_text(GTK\_LABEL(Clock\_Widget), (const char\*)data);

}

free(data); // Free the duplicated time string

return FALSE; // Run once

}

// ----- Background time thread -----

void\* update\_time(void \*args) {

while (running) {

char\* time\_str = exec\_cmd("date +%T"); // use 24-hr format

if (time\_str != NULL) {

g\_idle\_add(update\_clock, time\_str); // pass to main thread

}

sleep(1);

}

return NULL;

}

// ----- Quit button handler -----

void quit\_pressed(GtkButton \*button) {

running = false; // Signal thread to stop

pthread\_join(time\_thread, NULL);

gtk\_main\_quit();

}

// ----- Safe program launcher -----

void run\_program(const char \*program) {

pthread\_mutex\_lock(&pressed\_mutex);

if (!pressed) {

pid\_t pid = fork();

if (pid == 0) { // Child process

execlp("alacritty", "alacritty", "-e", "./FantasyOS", "run", program, NULL);

// If we get here, exec failed

fprintf(stderr, "Failed to launch program: %s\n", program);

exit(EXIT\_FAILURE);

} else if (pid < 0) {

fprintf(stderr, "Failed to fork for program: %s\n", program);

}

pressed = true;

} else {

pressed = false;

}

pthread\_mutex\_unlock(&pressed\_mutex);

}

// ----- Button callbacks -----

void cal\_pressed(GtkButton \*button) { run\_program("calculator"); }

void game\_pressed(GtkButton \*button) { run\_program("num\_guess\_game"); }

void conversion\_pressed(GtkButton \*button) { run\_program("conversion"); }

void song\_pressed(GtkButton \*button) { run\_program("song"); }

void sys\_time\_pressed(GtkButton \*button) { run\_program("sys\_time"); }

void pass\_pressed(GtkButton \*button) { run\_program("password\_setting"); }

void fac\_pressed(GtkButton \*button) { run\_program("factorial"); }

void palindrome\_pressed(GtkButton \*button) { run\_program("palindrome"); }

void prime\_pressed(GtkButton \*button) { run\_program("prime"); }

void notepad\_pressed(GtkButton \*button) { run\_program("os\_notepad"); }

void bubble\_pressed(GtkButton \*button) { run\_program("bubblesort"); }

void armstrong\_pressed(GtkButton \*button) { run\_program("armstrong"); }

void fib\_pressed(GtkButton \*button) { run\_program("fibonacci"); }

void terminal\_pressed(GtkButton \*button) { run\_program(""); } // opens FantasyOS

// ----- Widget connector macro -----

#define CONNECT(id, func) \

do { \

GtkWidget\* btn = GTK\_WIDGET(gtk\_builder\_get\_object(builder, id)); \

if (btn) { \

g\_signal\_connect(G\_OBJECT(btn), "clicked", G\_CALLBACK(func), NULL); \

} else { \

g\_printerr("WARNING: Button %s not found.\n", id); \

} \

} while(0)

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

// Initialize GTK

gtk\_init(&argc, &argv);

// Check for glade file

if (!g\_file\_test("frontend.glade", G\_FILE\_TEST\_EXISTS)) {

g\_printerr("ERROR: frontend.glade not found.\n");

return EXIT\_FAILURE;

}

// Load UI from glade file

builder = gtk\_builder\_new\_from\_file("frontend.glade");

if (!builder) {

g\_printerr("ERROR: Failed to load Glade file.\n");

return EXIT\_FAILURE;

}

// Get main widgets

GTK\_Window = GTK\_WIDGET(gtk\_builder\_get\_object(builder, "window"));

Clock\_Widget = GTK\_WIDGET(gtk\_builder\_get\_object(builder, "Clock\_Widget"));

quitButton = GTK\_WIDGET(gtk\_builder\_get\_object(builder, "shutdown"));

// Validate widgets

if (!GTK\_Window || !Clock\_Widget || !quitButton) {

g\_printerr("ERROR: One or more essential widgets not found in Glade.\n");

g\_object\_unref(builder);

return EXIT\_FAILURE;

}

// Connect quit button

g\_signal\_connect(G\_OBJECT(quitButton), "clicked", G\_CALLBACK(quit\_pressed), NULL);

// Connect all application buttons

CONNECT("cal\_button", cal\_pressed);

CONNECT("game\_button", game\_pressed);

CONNECT("conversion\_button", conversion\_pressed);

CONNECT("song\_button", song\_pressed);

CONNECT("sys\_time\_button", sys\_time\_pressed);

CONNECT("fac\_button", fac\_pressed);

CONNECT("palindrome\_button", palindrome\_pressed);

CONNECT("prime\_button", prime\_pressed);

CONNECT("notepad\_button", notepad\_pressed);

CONNECT("bubble\_button", bubble\_pressed);

CONNECT("armstrong\_button", armstrong\_pressed);

CONNECT("fib\_button", fib\_pressed);

CONNECT("terminal\_button", terminal\_pressed);

// Initialize mutex

if (pthread\_mutex\_init(&pressed\_mutex, NULL) != 0) {

g\_printerr("ERROR: Mutex initialization failed\n");

g\_object\_unref(builder);

return EXIT\_FAILURE;

}

// Start time thread

if (pthread\_create(&time\_thread, NULL, update\_time, NULL) != 0) {

g\_printerr("ERROR: Failed to create clock thread\n");

pthread\_mutex\_destroy(&pressed\_mutex);

g\_object\_unref(builder);

return EXIT\_FAILURE;

}

// Show window and start main loop

gtk\_widget\_show\_all(GTK\_Window);

gtk\_main();

// Cleanup

pthread\_mutex\_destroy(&pressed\_mutex);

g\_object\_unref(builder);

return EXIT\_SUCCESS;

}

**Make.sh:**

gcc `pkg-config --cflags gtk+-3.0` -o frontend frontend.c `pkg-config --libs gtk+-3.0` -rdynamic

**Build.sh:**

#!/bin/bash

export LIBGL\_ALWAYS\_SOFTWARE=1

unset GSK\_RENDERER

cores=$1

ram=$(($2 \* 1024))

disk=$(($3 \* 1024))

echo $cores > osspecs

echo $ram >> osspecs

echo $disk >> osspecs

g++ -o run run.cpp

./run $cores $ram $disk

g++ -o FantasyOS main.cpp -lpthread

mkdir -p prgs

# Compile .cpp source files from prgs/ into prgs/

g++ -o prgs/calculator prgs/calculator.cpp

g++ -o prgs/bubblesort prgs/bubblesort.cpp

g++ -o prgs/factorial prgs/factorial.cpp

g++ -o prgs/armstrong prgs/armstrong.cpp

g++ -o prgs/prime prgs/prime.cpp

g++ -o prgs/palindrome prgs/palindrome.cpp

g++ -o prgs/os\_notepad prgs/os\_notepad.cpp

g++ -o prgs/sys\_time prgs/sys\_time.cpp

g++ -o prgs/num\_guess\_game prgs/num\_guess\_game.cpp

g++ -o prgs/conversion prgs/conversion.cpp

g++ -o prgs/fibonacci prgs/fibonacci.cpp

g++ -o prgs/song prgs/song.cpp

./make.sh

./frontend

**Programs:**

armstrong 200 50

bubblesort 200 100

calculator 1000 500

conversion 100 50

factorial 500 200

fibonacci 300 100

num\_guess\_game 200 100

os\_notepad 500 400

palindrome 200 100

prime 500 400

sys\_time 400 300

song 1000 500

**COMMANDS:**

goback cd ..

open env -u GSK\_RENDERER xdg-open

create touch

newfolder mkdir

delete rm

loc pwd

delfolder rmdir

plist cat ~/Desktop/OS-Project/processes

clear clear

list ls

move mv

copy cp

date date

show cat

help cat cmd

systeminfo echo "FantasyOS | RAM: 2048MB | Disk: 256GB | Cores: 8"

resusage echo "2 cores in use, 240MB RAM used"

**glade:**

goback cd ..

open env -u GSK\_RENDERER xdg-open

create touch

newfolder mkdir

delete rm

loc pwd

delfolder rmdir

plist cat ~/Desktop/OS-Project/processes

clear clear

list ls

move mv

copy cp

date date

show cat

help cat cmd

systeminfo echo "FantasyOS | RAM: 2048MB | Disk: 256GB | Cores: 8"

resusage echo "2 cores in use, 240MB RAM used"

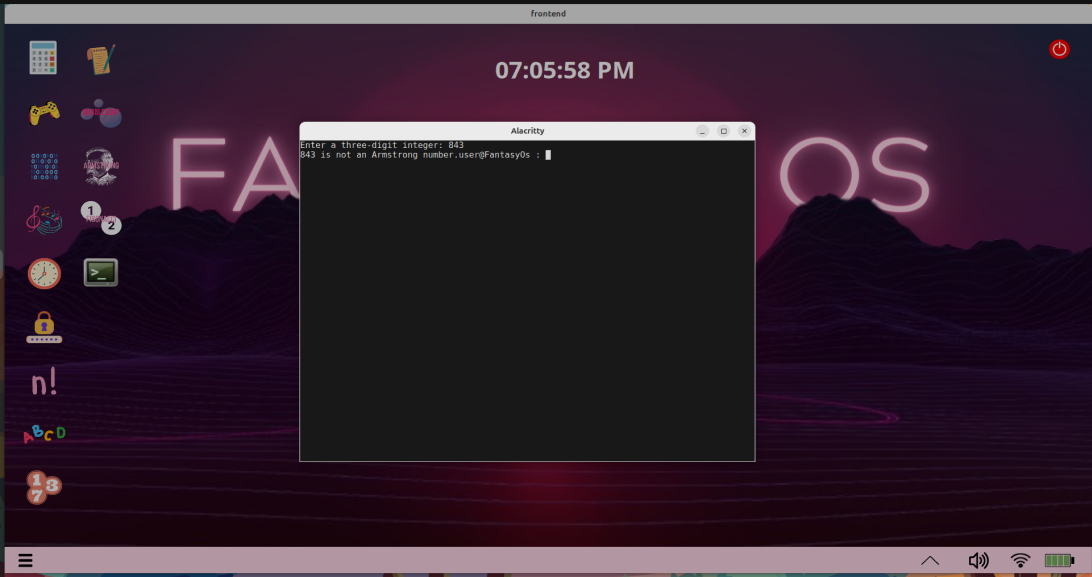
**OUTPUTS:**



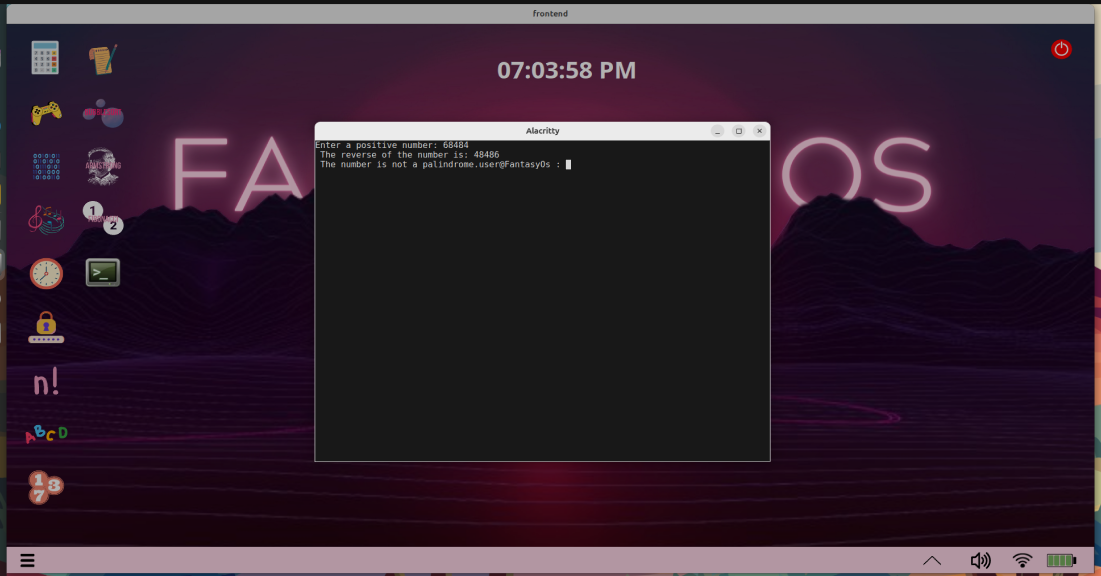


**CALCULATOR:**

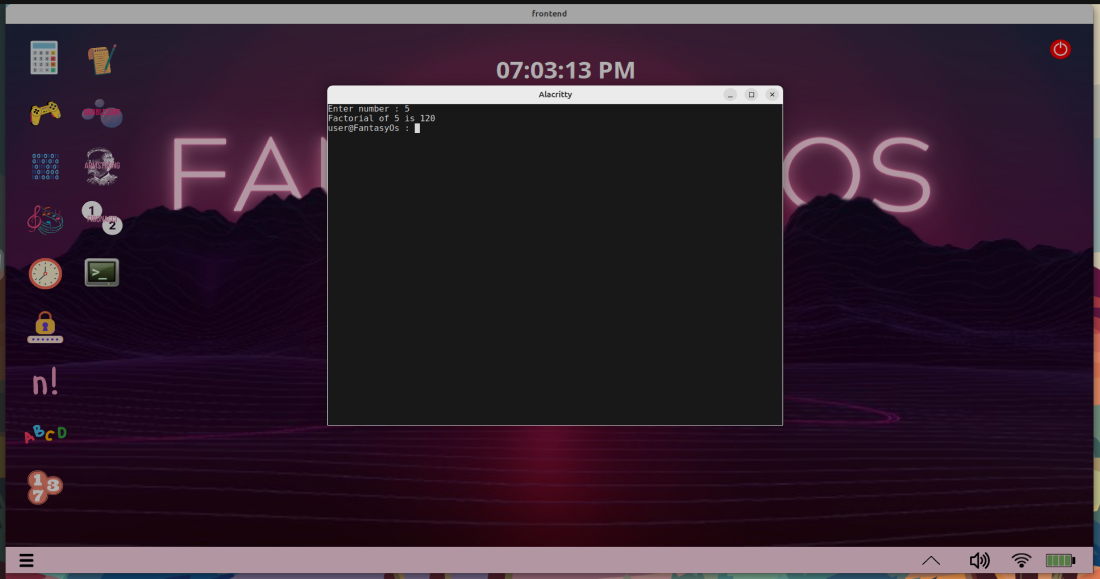
**ARMSTRONG:**



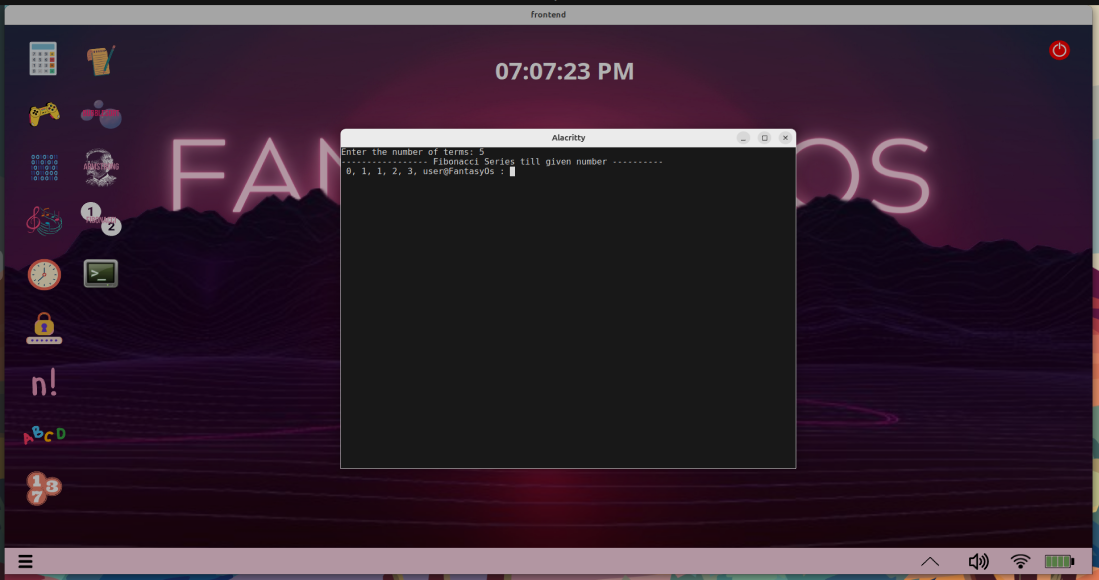
**PALINDROME:**

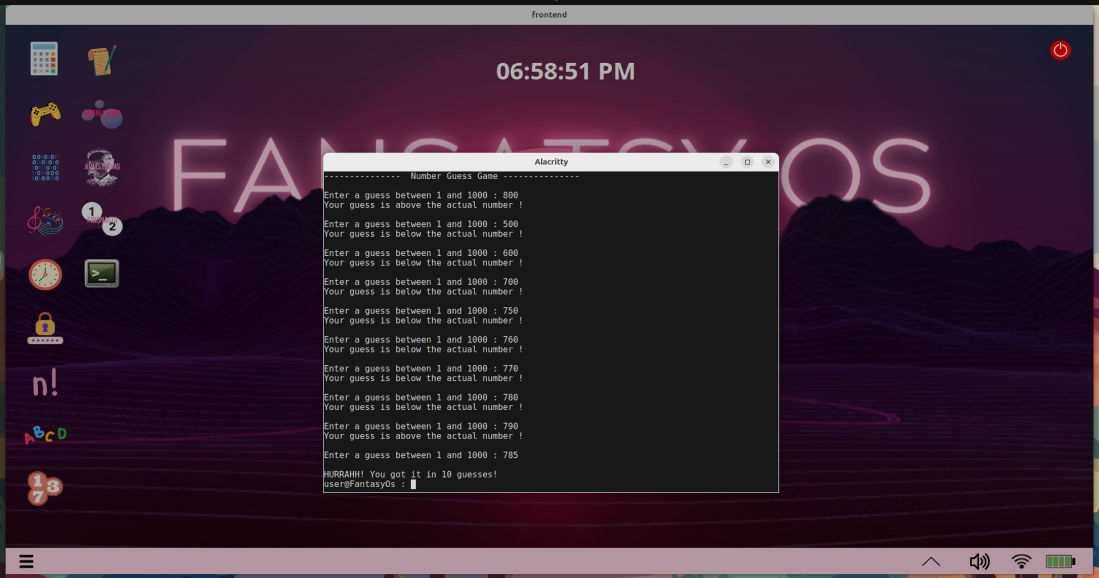


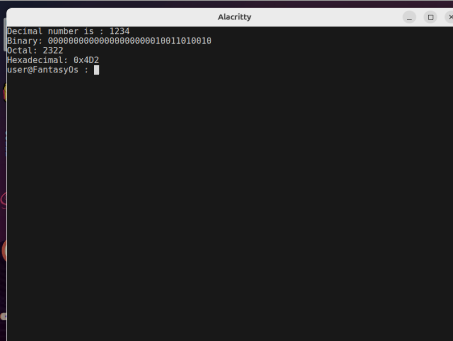
**FACTORIAL:**

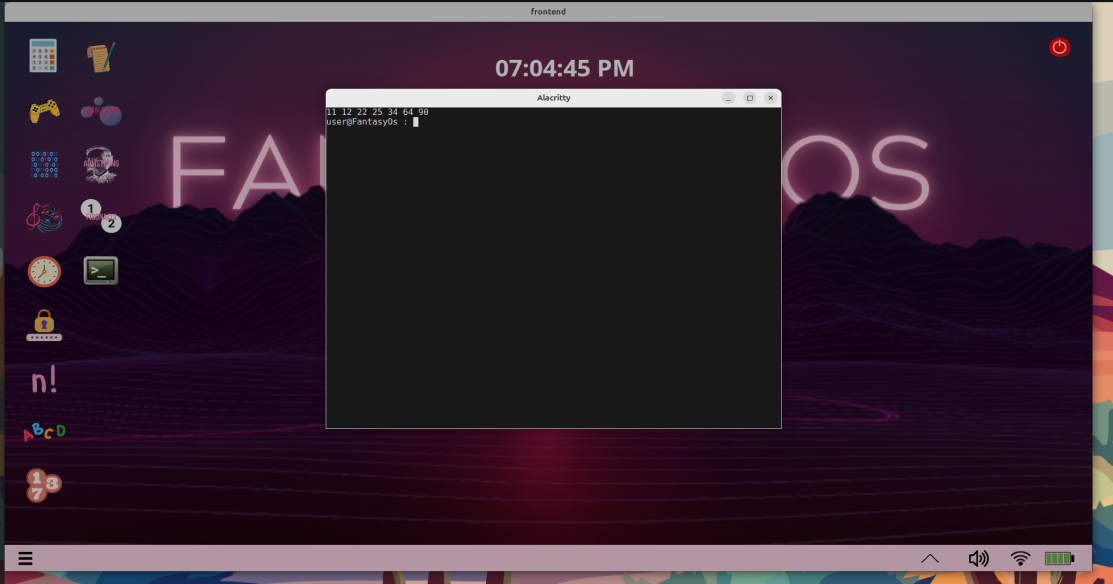


**FIBONACCI:**



**NUM\_GUESS\_GAME:**

**CONVERSION:**

**BUBBLE\_SORT:**

**SYS\_TIME:**