**File 1**

#include<chrono>

#include<iostream>

#include<stdio.h>

#include <iomanip>

#include<unistd.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include <string>

#include <fstream>

using namespace std;

using namespace std::chrono;

char arr[3][3]; // For Board printing

int record[3]; // To store win/loss/draw record of players

pthread\_cond\_t c = PTHREAD\_COND\_INITIALIZER;

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

int done = 0;

class TicTacToe { // TicTacToe class implementation

public:

TicTacToe();

static void\* playgame(void\*);

static void drawboard();

static void playermove();

static void\* checkwin(void\*);

static void\* checkdraw(void\*);

static void\* checkresult(void\*);

};

TicTacToe::TicTacToe() {

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

arr[i][j] = '-';

}

}

void\* TicTacToe::playgame(void\* arg)

{

pthread\_mutex\_lock(&m); // Mutex locked to block access to global variables

system("clear");

drawboard();

playermove();

done = 1;

pthread\_cond\_signal(&c); // Signal sent to checkresult thread to wake up from sleep.

pthread\_mutex\_unlock(&m);

cout << endl;

pthread\_exit(NULL);

}

void TicTacToe::drawboard()

{

cout << "HUMAN (X) - COMPUTER (O)" << endl << endl

<< "Terminal: HUMAN" << endl << endl;

for (int i = 0; i < 3; i++)

{

cout << " ";

for (int j = 0; j < 3; j++)

{

cout << arr[i][j];

if (j == 2)

continue;

cout << " | ";

}

if (i == 2)

continue;

cout << endl;

cout << "\_\_\_\_|\_\_\_\_\_|\_\_\_\_" << endl

<< " | | " << endl;

}

}

void TicTacToe::playermove()

{

int row, col;

bool correctmove = false;

cout << endl << endl;

cout << "Make your Move!" << endl;

while (!correctmove)

{

cout << "Enter Row(0-2): ";

cin >> row;

cout << "Enter Col(0-2): ";

cin >> col;

if ((row < 3 && row > -1) || (col > -1 && col < 3))

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (arr[row][col] == '-')

{

arr[row][col] = 'X';

correctmove = true;

}

}

}

}

if (!correctmove)

cout << "Wrong Inputs, please enter again" << endl;

}

}

void\* TicTacToe::checkresult(void\* arg)

{

pthread\_mutex\_lock(&m);

while (done == 0) // Condition Variable used.

pthread\_cond\_wait(&c, &m); // Waiting for the playgame thread to complete execution.

char\* temp = (char\*)arg;

char c = \*temp;

pthread\_t t\_draw, t\_win;

void\* first, \* second;

pthread\_create(&t\_win, NULL, TicTacToe::checkwin, &c);

pthread\_create(&t\_draw, NULL, TicTacToe::checkdraw, NULL);

pthread\_join(t\_win, &first);

pthread\_join(t\_draw, &second);

bool\* win\_check = (bool\*)first;

bool\* draw\_check = (bool\*)second;

int\* result = new int;

if (\*win\_check)

\*result = 1;

else if (\*draw\_check)

\*result = 2;

else

\*result = 0;

pthread\_mutex\_unlock(&m);

return result;

}

void\* TicTacToe::checkwin(void\* arg)

{

char\* c = (char\*)arg;

bool\* ptr = new bool;

\*ptr = false;

for (int i = 0; i < 3; i++)

{

if (arr[i][0] == (\*c) && arr[i][1] == (\*c) && arr[i][2] == (\*c))

\*ptr = true;

if (arr[0][i] == (\*c) && arr[1][i] == (\*c) && arr[2][i] == (\*c))

\*ptr = true;

}

if (arr[0][0] == (\*c) && arr[1][1] == (\*c) && arr[2][2] == (\*c))

\*ptr = true;

if (arr[0][2] == (\*c) && arr[1][1] == (\*c) && arr[2][0] == (\*c))

\*ptr = true;

return ptr;

}

void\* TicTacToe::checkdraw(void\* arg)

{

bool\* ptr = new bool;

\*ptr = true;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (arr[i][j] == '-')

\*ptr = false;

}

}

return ptr;

}

void\* save\_score(void\* arg)

{

fstream in;

in.open("Score.txt", ios::out | ios::trunc);

for (int i = 0; i < 3; i++)

in << record[i] << endl;

}

bool time\_condition = true;

float turn\_time;

void\* time\_taken(void\*)

{

auto start = high\_resolution\_clock::now();

while (time\_condition);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(stop - start);

turn\_time = duration.count();

}

const char pipe\_file[] = "/tmp/program";

void main\_program()

{

int status;

char ch;

void\* r\_temp;

bool fh = true;

pid\_t pid;

pthread\_t tid, time\_thread, r\_thread, p\_thread;

TicTacToe obj;

mkfifo(pipe\_file, 0666);

while (fh)

{

done = 0; // Condition variable for Synchronization.

ch = 'X';

time\_condition = true;

pthread\_create(&time\_thread, NULL, time\_taken, NULL);

pthread\_create(&p\_thread, NULL, TicTacToe::playgame, NULL);

//obj.playgame();

pthread\_create(&r\_thread, NULL, TicTacToe::checkresult, &ch);

pthread\_join(p\_thread, NULL);

time\_condition = false;

pthread\_join(r\_thread, &r\_temp);

int\* result = (int\*)r\_temp;

if (\*result != 0)

{

fh = false;

pid = fork(); // Creating another process to calculate the score

if (pid == 0)

{

if (\*result == 1)

record[0]++;

if (\*result == 2)

record[2]++;

pthread\_create(&tid, NULL, save\_score, NULL); // Updated scores have been stored.

pthread\_join(tid, NULL);

exit(0); // Child process terminated.

}

wait(NULL); // Parent Process will wait for child to execute.

if (\*result == 1)

{

system("clear");

obj.drawboard();

cout << endl << endl << "Congratulations! You have won the game!" << endl;

}

else if (\*result == 2)

{

system("clear");

obj.drawboard();

cout << endl << endl << "The Game has been Drawn!" << endl;

}

}

status = open(pipe\_file, O\_WRONLY);

write(status, arr, 9 \* sizeof(char));

close(status);

if (fh)

{

system("clear");

obj.drawboard();

cout << "\n\nTime Taken for turn: " << turn\_time << " milliseconds" << endl;

cout << endl << endl << "Waiting for opponent's turn...." << endl;

status = open(pipe\_file, O\_RDONLY);

read(status, arr, 9 \* sizeof(char));

close(status);

ch = 'O';

pthread\_create(&r\_thread, NULL, TicTacToe::checkresult, &ch);

done = 1;

pthread\_cond\_signal(&c); // Signal sent to checkresult thread to wake up from sleep.

pthread\_join(r\_thread, &r\_temp);

int\* opp\_result = new int;

opp\_result = (int\*)r\_temp;

if (\*opp\_result != 0)

{

fh = false;

pid = fork(); // Creating another process to calculate the score

if (pid == 0)

{

if (\*opp\_result == 1)

record[1]++;

if (\*opp\_result == 2)

record[2]++;

pthread\_create(&tid, NULL, save\_score, NULL); // Updated scores have been stored.

pthread\_join(tid, NULL);

exit(0); // Child process terminated.

}

wait(NULL);

if (\*opp\_result == 1)

{

system("clear");

obj.drawboard();

cout << endl << endl << "Sorry, You have lost the game!" << endl;

}

else if (\*opp\_result == 2)

{

system("clear");

obj.drawboard();

cout << endl << endl << "The Game has been Drawn!" << endl;

}

}

}

}

}

void load\_file() // This will load all the data from High Score file into the array.

{

fstream in;

in.open("Score.txt", ios::in);

for (int i = 0; i < 3; i++)

in >> record[i];

in.close();

}

int main()

{

// FILE 1

fstream in;

int status;

bool fh = true;

//bool check = true;

while (fh)

{

load\_file();

system("clear");

cout << endl << "\t\t\tWELCOME TO TIC-TAC-TOE!" << endl << endl

<< "Please select from below" << endl

<< "1) Start Game (Player vs Computer)" << endl

<< "2) Display High Scores" << endl

<< "3) Exit Game" << endl

<< "Your Option: ";

int choice;

cin >> choice;

switch (choice)

{

case 1:

status = open(pipe\_file, O\_WRONLY);

write(status, &fh, sizeof(bool));

close(status);

main\_program();

break;

case 2:

cout << setw(15) << "Player wins: " << setw(5) << " " << record[0] << endl;

cout << setw(15) << "Computer wins: " << setw(5) << " " << record[1] << endl;

cout << setw(15) << "Draws played: " << setw(5) << " " << record[2] << endl << endl;

break;

case 3:

fh = false;

status = open(pipe\_file, O\_WRONLY);

write(status, &fh, sizeof(bool));

close(status);

cout << "Thank you for playing!!" << endl << endl;

exit(0);

break;

default:

cout << "Invalid choice entered. Please try again... " << endl << endl;

}

cin.ignore();

string temp;

cout << "Press Enter to continue...." << endl;

getline(cin, temp);

}

return 0;

}

**File 2**

#include<iostream>

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include<string>

#include<chrono>

using namespace std;

using namespace std::chrono;

char arr[3][3];

pthread\_cond\_t c = PTHREAD\_COND\_INITIALIZER;

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

int done = 0;

class TicTacToe {

public:

TicTacToe();

static void\* playgame(void\*);

static void drawboard();

static void playermove();

static void\* checkwin(void\*);

static void\* checkdraw(void\*);

static void\* checkresult(void\*);

};

TicTacToe::TicTacToe() {

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

arr[i][j] = '-';

}

}

void\* TicTacToe::playgame(void\* arg)

{

pthread\_mutex\_lock(&m); // Mutex locked to block access to global variables

system("clear");

drawboard();

playermove();

done = 1;

pthread\_cond\_signal(&c); // Signal sent to checkresult thread to wake up from sleep.

pthread\_mutex\_unlock(&m);

cout << endl;

pthread\_exit(NULL);

}

void TicTacToe::drawboard()

{

cout << "HUMAN (X) - COMPUTER (O)" << endl << endl

<< "Terminal: COMPUTER" << endl << endl;

for (int i = 0; i < 3; i++)

{

cout << " ";

for (int j = 0; j < 3; j++)

{

cout << arr[i][j];

if (j == 2)

continue;

cout << " | ";

}

if (i == 2)

continue;

cout << endl;

cout << "\_\_\_\_|\_\_\_\_\_|\_\_\_\_" << endl

<< " | | " << endl;

}

}

void TicTacToe::playermove()

{

int row = 0, col = 0;

bool correctmove = false;

cout << endl << endl;

cout << "Computer is contemplating its move..." << endl;

sleep(2);

cout << endl;

do

{

if ((row < 3 && row > -1) || (col > -1 && col < 3))

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (arr[row][col] == '-')

{

arr[row][col] = 'O';

correctmove = true;

}

}

}

}

row++;

if (row > 2)

{

row = 0;

col++;

}

} while (!correctmove);

}

void\* TicTacToe::checkresult(void\* arg)

{

pthread\_mutex\_lock(&m);

while (done == 0) // Condition Variable used.

pthread\_cond\_wait(&c, &m); // Waiting for the playgame thread to complete execution.

char\* temp = (char\*)arg;

char c = \*temp;

pthread\_t t\_draw, t\_win;

void\* first, \* second;

pthread\_create(&t\_win, NULL, TicTacToe::checkwin, &c);

pthread\_create(&t\_draw, NULL, TicTacToe::checkdraw, NULL);

pthread\_join(t\_win, &first);

pthread\_join(t\_draw, &second);

bool\* win\_check = (bool\*)first;

bool\* draw\_check = (bool\*)second;

int\* result = new int;

if (\*win\_check)

\*result = 1;

else if (\*draw\_check)

\*result = 2;

else

\*result = 0;

pthread\_mutex\_unlock(&m);

return result;

}

void\* TicTacToe::checkwin(void\* arg)

{

char\* c = (char\*)arg;

bool\* ptr = new bool;

\*ptr = false;

for (int i = 0; i < 3; i++)

{

if (arr[i][0] == (\*c) && arr[i][1] == (\*c) && arr[i][2] == (\*c))

\*ptr = true;

if (arr[0][i] == (\*c) && arr[1][i] == (\*c) && arr[2][i] == (\*c))

\*ptr = true;

}

if (arr[0][0] == (\*c) && arr[1][1] == (\*c) && arr[2][2] == (\*c))

\*ptr = true;

if (arr[0][2] == (\*c) && arr[1][1] == (\*c) && arr[2][0] == (\*c))

\*ptr = true;

return ptr;

}

void\* TicTacToe::checkdraw(void\* arg)

{

bool\* ptr = new bool;

\*ptr = true;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (arr[i][j] == '-')

\*ptr = false;

}

}

return ptr;

}

// ---------------- PROGRAM 2 -------------------------

bool time\_condition = true;

void\* time\_taken(void\* arg)

{

float\* turn\_time = new float;

auto start = high\_resolution\_clock::now();

while (time\_condition);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(stop - start);

\*turn\_time = duration.count();

return turn\_time; // Returing the Time to the main function.

}

int main()

{

system("clear");

int status;

char ch;

const char pipe\_file[] = "/tmp/program";

mkfifo(pipe\_file, 0666);

bool check = true;

float\* turn\_time;

cout << endl << endl << "Waiting for instruction...." << endl;

status = open(pipe\_file, O\_RDONLY);

read(status, &check, sizeof(bool));

close(status);

pthread\_t tid, time\_thread, r\_thread, p\_thread;

while (check)

{

TicTacToe obj;

bool fh = true, first\_turn = true;

while (fh)

{

void\* r\_temp;

system("clear");

obj.drawboard();

if (!first\_turn)

cout << "\n\nTime Taken for the turn: " << \*turn\_time << " milliseconds" << endl << endl;

first\_turn = false;

cout << endl << endl << "Waiting for opponent's turn...." << endl;

status = open(pipe\_file, O\_RDONLY);

read(status, arr, 9 \* sizeof(char));

close(status);

ch = 'X';

pthread\_create(&r\_thread, NULL, TicTacToe::checkresult, &ch);

done = 1;

pthread\_cond\_signal(&c); // Signal sent to checkresult thread to wake up from sleep.

pthread\_join(r\_thread, &r\_temp);

int\* opp\_result = (int\*)r\_temp;

if (\*opp\_result == 1)

{

fh = false;

system("clear");

obj.drawboard();

cout << endl << endl << "Sorry, You have lost the game!" << endl;

}

else if (\*opp\_result == 2)

{

fh = false;

system("clear");

obj.drawboard();

cout << endl << endl << "The Game has been Drawn!" << endl;

}

else

{

done = 0;

void\* temp;

time\_condition = true;

ch = 'O';

pthread\_create(&time\_thread, NULL, time\_taken, NULL);

pthread\_create(&p\_thread, NULL, TicTacToe::playgame, NULL);

pthread\_create(&r\_thread, NULL, TicTacToe::checkresult, &ch);

pthread\_join(p\_thread, NULL);

time\_condition = false;

pthread\_join(r\_thread, &r\_temp);

int\* result = (int\*)r\_temp;

pthread\_join(time\_thread, &temp);

turn\_time = (float\*)temp;

if (\*result == 1)

{

fh = false;

system("clear");

obj.drawboard();

cout << endl << endl << "Congratulations! You have won the game!" << endl;

}

else if (\*result == 2)

{

fh = false;

system("clear");

obj.drawboard();

cout << endl << endl << "The Game has been Drawn!" << endl;

}

status = open(pipe\_file, O\_WRONLY);

write(status, arr, 9 \* sizeof(char));

close(status);

}

}

cout << endl << endl << "Waiting for next instruction...." << endl;

status = open(pipe\_file, O\_RDONLY);

read(status, &check, sizeof(bool));

close(status);

}

cout << endl << "Thank you for Playing!" << endl << endl;

exit(0);

}