

TA-Student Interaction Simulation

CSCI-6100 Operating System

Programming Assignment

by Sai Kundan Suddapalli

Tools Used



Programming Language

is used for the simulation



C MAKE

Build system for the C++ code



Docker

Allows for independent execution of C++ code, isolating it from the host operating system

Project Structure

- **src folder:** Contains the C++ code for the simulation, including the main program (main.cxx).
- **CMakeLists.txt:** Configuration file for CMake, a build system.
- **Docker_tar_folder:** Stores a Docker image archive (programming_assignment.tar) that allows you to run the simulation without manual setup it is big file so gzip file using gunzip or use 7zip in windows.
- **Dockerfile:** Defines the instructions for building the Docker image

Code consists of 2 main functions

Sleep_TA function

Simulates the behavior of TA thread.

Student function

Simulates the behavior of student threads.

Main Code

The code initializes random number generation and creates N Student threads, detaching them to run independently. Each Student thread sleeps for a random amount of time before seeking assistance from the TA. A Sleep_TA thread is created, attached to the main thread, ensuring the main thread waits for the TA thread to complete. A try-catch block handles exceptions for robust error handling.

Predefined Macros

The code uses three predefined macros:

→ MAX_WAITING 2: The maximum time students wait in the waiting area (number of chairs).

→ HELP_TIME 5: Simulated time for the TA to address a student's question.

→ TA_GOING_TIME 10: The TA goes home if no students arrive for help within 10 seconds.

These macros control the simulation, realistically representing student interactions with a TA.

Two main code blocks of main code

The Student threads code block creates a loop that iterates through a specified number of student threads, each represented by a `student_thread` object. Each student thread is created using the `student` function, which simulates the behavior of a student seeking assistance from the TA. After creation, each student thread is detached to run independently. The code also incorporates a random wait time before the next student thread is created, simulating the arrival of students at different intervals. Finally, the code detaches all the student threads, ensuring that the main thread does not wait for all student threads to complete before proceeding.

```
for (int i = 0; i < num_students; i++) {
    student_threads.emplace_back(student, i + 1);
    this_thread::sleep_for(chrono::seconds(rand() % 2)); //
    Random wait time before next student arrives
}

// Join all student threads
for (auto& s : student_threads) {
    if (s.joinable()) {
        s.detach();
    }
}
```

The TA thread code block creates a `ta_thread` object using the `sleeping_ta` function (defined elsewhere in the code), simulating the TA's behavior. The TA thread is attached to the main thread, ensuring the main thread waits for the TA thread to complete. This ensures that the TA thread runs concurrently with the student threads, allowing for realistic interaction between them.

```
thread ta_thread(sleeping_ta); // Create TA thread
if (ta_thread.joinable()) {
    ta_thread.join();
}
```

Sleeping_TA Function

The `sleeping_ta` function is a core component of the student-TA interaction simulation. It models the TA's behavior, mimicking their actions within the virtual environment.

The function operates within an infinite loop, continually monitoring for student requests. If no students are waiting for assistance, the TA enters a "sleeping" state, patiently waiting for a student to arrive or for a predetermined time limit (10 seconds) to expire. This wait is managed using the `cv.wait_for()` function, which suspends the TA thread until either a student request wakes it up, or the 10-second time limit is reached.

The time limit ensures that the TA doesn't remain inactive indefinitely, preventing a potential deadlock if no students require help. If the TA remains in the sleeping state for 10 seconds without receiving a request, it decides to exit the simulation, simulating the TA going home for the day.

Upon receiving a student request, the TA awakens and assists the student. The `this_thread::sleep_for()` function simulates the time the TA spends addressing the student's question. After helping the student, the TA reduces the `waiting_students` count by one, reflecting the completion of the interaction. The TA then returns to the sleeping state, ready to assist another student.

```
void sleeping_ta() {
    while (true) {
        // Decrease waiting students count
        unique_lock<mutex> lock(mtx);
        auto start_idle_time = chrono::steady_clock::now();
        cv.wait_for(lock, 10s, [] { return !ta_sleeping; }); // Wait until TA is woken until some student call in 10 seconds or else
        // TA is helping a student
        if (waiting_students == 0) {
            cout << "[TA THREAD] No students waiting. TA goes back to sleep." << endl;
            auto end_idle_time = chrono::steady_clock::now();
            auto idle_duration = chrono::duration_cast<chrono::seconds>(end_idle_time - start_idle_time);
            //cout << "count :::" << idle_duration.count() ;
            if (idle_duration.count() >= 10) {
                cout << "[TA THREAD] No students for 10 seconds. TA is exiting." << endl;
                return; // Exit the TA thread
            }
            ta_sleeping = true; // TA goes back to sleep
        }
        else{
            cout << "[TA THREAD] TA is helping a student..." << endl;
            this_thread::sleep_for(chrono::seconds(HELP_TIME)); // Simulate helping time
            cout << "[TA THREAD] TA is done helping a student!" << endl;
            if (waiting_students > 0) {

                this_thread::sleep_for(chrono::seconds(HELP_TIME)); // Simulate helping time
                cout << "[TA THREAD] TA is done helping a student!" << endl;
                waiting_students--;
                cout << "[TA THREAD] Waiting students = " << waiting_students << endl;
            }
        }
    }
}
```

Student function

The `student()` function simulates a student arriving at the TA's office hours and requesting help. It takes the student's ID as an input and continuously attempts to get help from the TA.

```
if (ta_sleeping) {
    cout << "[STUDENT THREAD] Student " << student_id << " wakes up TA!" << endl;
    ta_sleeping = false; // TA is awake
    cv.notify_one(); // Signal the TA to help
    cout << "[STUDENT THREAD] student " << student_id << " done with work going bye" << endl;
    return; // Student is helped, exit the function
}
if (waiting_students < MAX_WAITING) {
    waiting_students++;
    cout << "[STUDENT THREAD] Student " << student_id << " is seated on a waiting chair. Waiting students = " <<
waiting_students << endl;
    return; // Student is seated, exit the function
}
else {
    cout << "[STUDENT THREAD] Student " << student_id << " finds no chairs available and will come back later." << endl;
    // this is for debugging
    //cout << "TA ::: " << ta_sleeping << "waiting_student_counter ::: " << waiting_students << endl ;
    lock.unlock(); // Unlock the mutex before sleeping

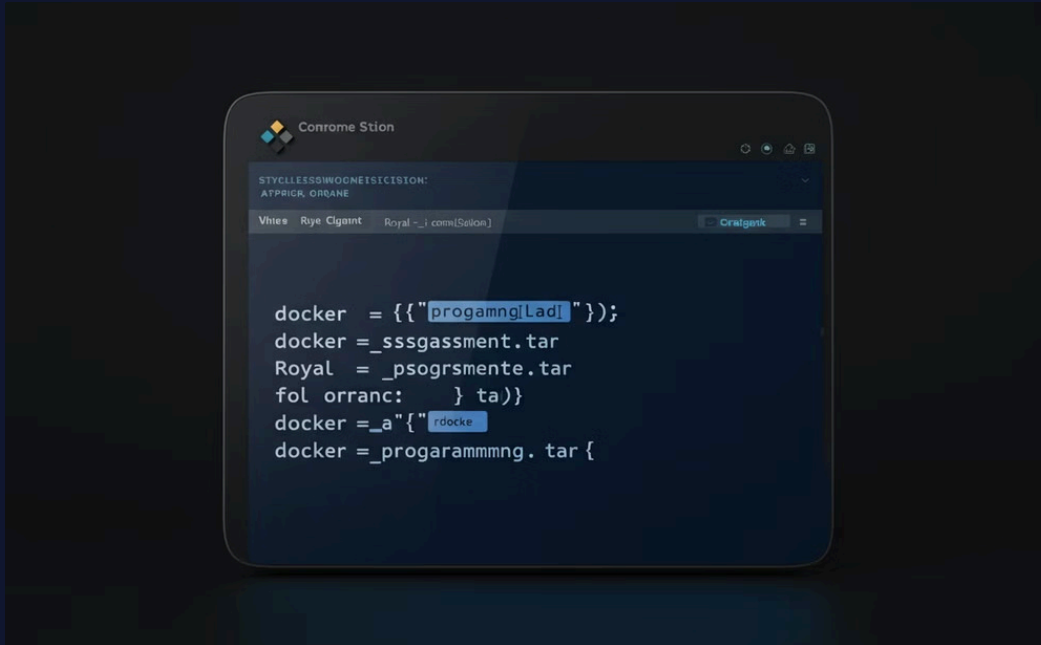
    // Wait for a random time before trying again
    this_thread::sleep_for(chrono::seconds(rand() % 8)); // Random wait between 1 to 3 seconds

    // Re-lock the mutex for the next attempt
    continue; //will start of the while loop to so it go to if condition
}
```

First, it checks if the TA is sleeping. If so, the student wakes up the TA and signals them to start helping.

If the TA is not sleeping, the student checks if there are available waiting chairs. If there are, the student sits down and waits for the TA to finish with the current student. If there are no chairs, the student leaves and will try again later.

To run C++ code using Docker to make it OS independent



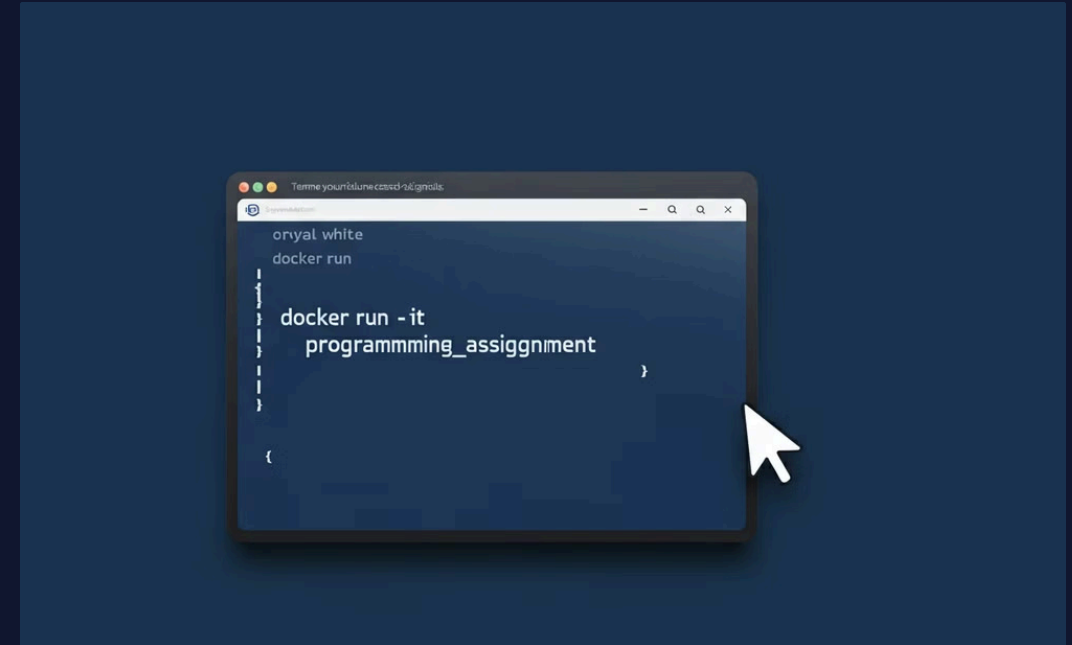
Load the Docker Image

Download the repository and navigate to the Docker_tar_folder directory: `cd Docker_tar_folder/`

Load the Docker image: `docker load -i programming_assignment.tar`

Benefits of Docker:

- No manual build environment setup.
- Simplified execution across operating systems.



Run the Simulation

Verify the image: `docker images`

Run the simulation in interactive mode: `docker run -it programming_assignment`

Compile C++ Code natively

Running the C++ File Directly

Requirements: A C++ compiler, such as Visual Studio (Windows) or g++ (Linux/macOS).

Instructions:

1. Open the `main.cxx` file in your preferred code editor.
2. Compile and run the code using your compiler:
 - Windows: Use Visual Studio or a command-line compiler.
 - Linux/macOS: `g++ -pthread -o pro main.cxx ./pro`
(Optional) Provide an argument to specify the number of students: `./pro argument={Number of Students}`

Building with CMake

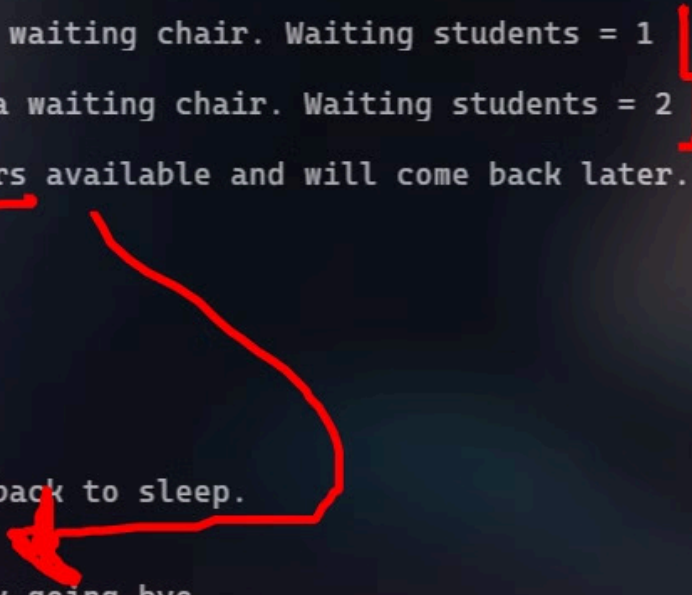
Requirements: CMake installed on your system.

Instructions:

1. Create a build directory: `mkdir build && cd build`
2. Configure the build with CMake: `cmake ..`
3. Build the project: `make`
4. Run the simulation: `./pro` (Optional) Provide an argument to specify the number of students: `./pro argument={Number of Students}`

Sample Simulation Output

```
[STUDENT THREAD] Student 5 is coming!
[STUDENT THREAD] Student 5 is seated on a waiting chair. Waiting students = 1
[STUDENT THREAD] Student 12 is coming!
[STUDENT THREAD] Student 12 is seated on a waiting chair. Waiting students = 2
[STUDENT THREAD] Student 10 is coming!
[STUDENT THREAD] Student 10 finds no chairs available and will come back later.
[TA THREAD] TA is helping a student
[TA THREAD] TA is done helping a student!
[TA THREAD] TA is done helping a student!
[TA THREAD] Waiting students = 1
[TA THREAD] TA is helping a student ...
[TA THREAD] TA is done helping a student!
[TA THREAD] TA is done helping a student!
[TA THREAD] Waiting students = 0
[TA THREAD] No students waiting. TA goes back to sleep.
[STUDENT THREAD] Student 10 is coming!
[STUDENT THREAD] Student 10 wakes up TA!
[STUDENT THREAD] student 10 done with work going bye
[STUDENT THREAD] Student 13 is coming!
[STUDENT THREAD] Student 13 is seated on a waiting chair. Waiting students = 1
[TA THREAD] TA is helping a student ...
[TA THREAD] TA is done helping a student!
[TA THREAD] TA is done helping a student!
[TA THREAD] Waiting students = 0
[TA THREAD] No students waiting. TA goes back to sleep.
[TA THREAD] No students waiting. TA goes back to sleep.
[TA THREAD] No students for 10 seconds. TA is exiting.
```



Sample run output

```
~/Documents/OS5
> docker run -it programming_assignment
Enter the number of students: 15
[STUDENT THREAD] Student 1 is coming!
[STUDENT THREAD] Student 1 wakes up TA!
[STUDENT THREAD] student 1 done with work going bye
[TA THREAD] No students waiting. TA goes back to sleep.
```

Docker run

Run the simulation using Docker.

```
~/Documents/OS5/OS-threading- main*
> cd build/

~/Documents/OS5/OS-threading-/build main*
> cmake ..
-- The C compiler identification is GNU 14.2.1
-- The CXX compiler identification is GNU 14.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.8s)
-- Generating done (0.0s)
-- Build files have been written to: /home/kundan/Documents/OS5/OS-threading-/build

~/Documents/OS5/OS-threading-/build main*
> make
[ 50%] Building CXX object CMakeFiles/pro.dir/src/main.cxx.o
[100%] Linking CXX executable pro
[100%] Built target pro

~/Documents/OS5/OS-threading-/build main*
> ./pro
Enter the number of students: 4
```

CMake build

Build the project using CMake.

```
~/Documents/OS5/OS-threading-/src main*
> g++ -pthread main.cxx -o pro

~/Documents/OS5/OS-threading-/src main*
> ./pro
Enter the number of students: 1
[STUDENT THREAD] Student 1 is coming!
[STUDENT THREAD] Student 1 wakes up TA!
[STUDENT THREAD] student 1 done with work going bye
[TA THREAD] No students waiting. TA goes back to sleep.
[TA THREAD] No students for 10 seconds. TA is exiting.
```

Compiling using G++

Compile the C++ code using G++.