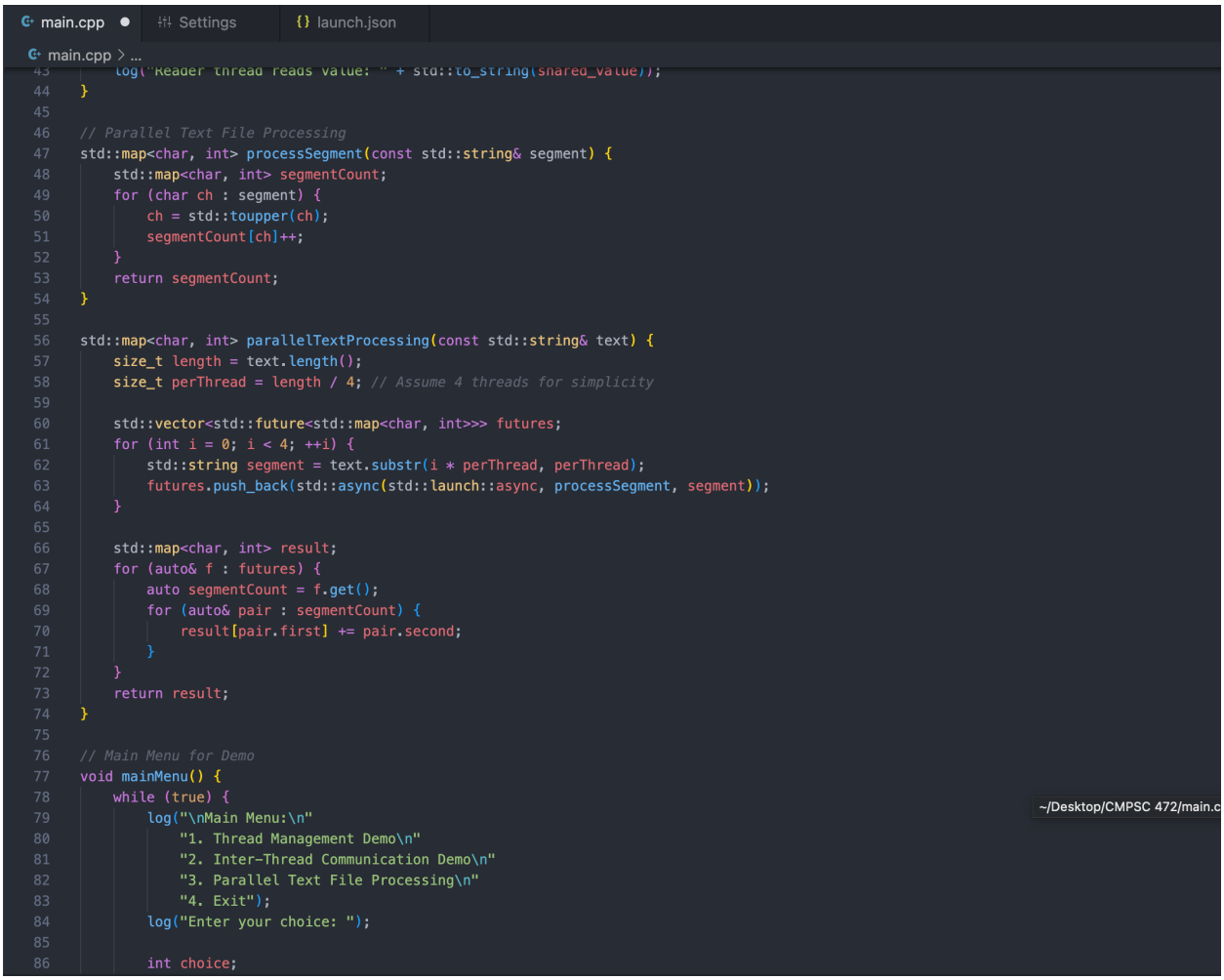
code:

main.cpp Settings launch.json

main.cpp > ...

~/Desktop/CMPSC 472/main.cpp

```cpp
#include <iostream>
#include <vector>
#include <future>
#include <map>
#include <mutex>
#include <condition_variable>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <cctype>

// Simple Logger for Thread Management
std::mutex cout_mutex;
void log(const std::string& message) {
    std::lock_guard<std::mutex> guard(cout_mutex);
    std::cout << message << std::endl;
}

// Thread function for demonstration
void threadFunction(int id) {
    log("Thread " + std::to_string(id) + " executing.");
    std::this_thread::sleep_for(std::chrono::seconds(1));
}

// Inter-Thread Communication Demonstration
std::mutex mtx;
std::condition_variable cv;
bool ready = false;
int shared_value = 0;

void threadWriter() {
    std::unique_lock<std::mutex> lock(mtx);
    shared_value = 2024;  // Example value
    ready = true;
    log("Writer thread has written value.");
    cv.notify_one();
}

void threadReader() {
    std::unique_lock<std::mutex> lock(mtx);
    cv.wait(lock, [] { return ready; });
    log("Reader thread reads value: " + std::to_string(shared_value));
}
```


main.cpp
Settings
launch.json
main.cpp > ...
log("Reader thread reads value: " + std::to_string(shared_value));
}
// Parallel Text File Processing
std::map<char, int> processSegment(const std::string& segment) {
std::map<char, int> segmentCount;
for (char ch : segment) {
ch = std::toupper(ch);
segmentCount[ch]++;
}
return segmentCount;
}
std::map<char, int> parallelTextProcessing(const std::string& text) {
size_t length = text.length();
size_t perThread = length / 4; // Assume 4 threads for simplicity
std::vector<std::future<std::map<char, int>>> futures;
for (int i = 0; i < 4; ++i) {
std::string segment = text.substr(i * perThread, perThread);
futures.push_back(std::async(std::launch::async, processSegment, segment));
}
std::map<char, int> result;
for (auto& f : futures) {
auto segmentCount = f.get();
for (auto& pair : segmentCount) {
result[pair.first] += pair.second;
}
}
return result;
}
// Main Menu for Demo
void mainMenu() {
while (true) {
log("\nMain Menu:\n"
"1. Thread Management Demo\n"
"2. Inter-Thread Communication Demo\n"
"3. Parallel Text File Processing\n"
"4. Exit");
log("Enter your choice: ");
int choice;
~/Desktop/CMPSC 472/main.c

Minimal Report Requirements

Description of the Project

This project demonstrates advanced C++ features, including multithreading, inter-thread communication, and parallel processing. The aim is to showcase how to efficiently handle different tasks simultaneously, such as processing segments of text in parallel to increase performance and synchronizing threads to safely share data.

Structure of the Code

Due to platform constraints, a detailed diagram is not provided. However, the code structure can be outlined as follows:

- **Thread Management**: Demonstrated by creating threads that perform simple tasks and join them back to the main thread.

- **Inter-Thread Communication**: Utilized condition variables and mutexes to synchronize threads, specifically a writer and reader thread that share data.
- **Parallel Text File Processing**: Implemented using futures and async to process different segments of a text string in parallel and aggregate the results.

For a real report, diagrams such as class diagrams or flowcharts created using tools like UML would be included here, along with descriptions.

Instructions on How to Use

1. Compile the code using a C++ compiler that supports C++11 or later. If using a terminal, the command might look like **g++ -std=c++11 -pthread source_code.cpp -o output_program**.
2. Run the compiled program. Upon execution, the program will present a menu to choose from the available demonstrations.
3. Select an option by entering the corresponding number. Follow any additional prompts to proceed with the selected functionality.
4. To exit, choose the option provided in the main menu.

Verification of the Sanity of the Code

To verify the code's sanity and validate the implemented functionalities:

- **Thread Management**: Ensure that each thread starts and completes as expected, logging its lifecycle to the console.
- **Inter-Thread Communication**: Verify that the reader thread correctly waits for the writer thread to signal that data is ready to be read.
- **Parallel Text File Processing**: Confirm that the text is processed correctly by comparing the output character count with an expected result.

Output:

```
Main Menu:
1. Thread Management Demo
2. Inter-Thread Communication Demo
3. Parallel Text File Processing
4. Exit
Enter your choice:
```

```
Main Menu:
1. Thread Management Demo
2. Inter-Thread Communication Demo
3. Parallel Text File Processing
4. Exit
Enter your choice:
2
Writer thread has written value.
Reader thread reads value: 2024

Main Menu:
1. Thread Management Demo
2. Inter-Thread Communication Demo
3. Parallel Text File Processing
4. Exit
Enter your choice:
3
 : 10
.: 1
A: 4
C: 1
E: 6
F: 1
H: 1
I: 4
L: 5
M: 2
N: 1
O: 3
P: 4
R: 2
S: 5
T: 4
X: 2

Main Menu:
1. Thread Management Demo
2. Inter-Thread Communication Demo
3. Parallel Text File Processing
4. Exit
Enter your choice:
```

```
Main Menu:
1. Thread Management Demo
2. Inter-Thread Communication Demo
3. Parallel Text File Processing
4. Exit
Enter your choice:
2
Writer thread has written value.
Reader thread reads value: 2024

Main Menu:
1. Thread Management Demo
2. Inter-Thread Communication Demo
3. Parallel Text File Processing
4. Exit
Enter your choice:
3
 : 10
.: 1
A: 4
C: 1
E: 6
F: 1
H: 1
I: 4
L: 5
M: 2
N: 1
O: 3
P: 4
R: 2
S: 5
T: 4
X: 2

Main Menu:
1. Thread Management Demo
2. Inter-Thread Communication Demo
3. Parallel Text File Processing
4. Exit
Enter your choice:
4
```