

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY WAKNAGHAT, SOLAN(H.P.)



OPERATING SYSTEM PROJECT REPORT INTER-THREAD COMMUNICATION

SUBMITTED BY:
ANJALI BABELE
ID:221030449
MUSKAN SHARMA
ID:221030330
ANSHUL BHARADWAJ
ID:221030394
BATCH:CS411

SUBMITTED TO:
Dr.DEEPAK GUPTA

Purpose of Inter-Thread communication

Inter-thread communication in an operating system (OS) is essential for coordinating the actions of multiple threads within a process. The primary purposes of inter-thread communication include:

1. **Data Sharing:** Threads often need to share data with each other. Inter-thread communication mechanisms ensure that data is shared safely and efficiently, preventing race conditions and ensuring data consistency.
2. **Synchronization:** Threads may need to synchronize their actions to ensure they occur in a specific order or to coordinate access to shared resources. Synchronization primitives like mutexes, semaphores, and condition variables are used for this purpose.
3. **Task Coordination:** In complex applications, different threads may be responsible for different tasks. Inter-thread communication allows these threads to coordinate their work, ensuring that tasks are completed in the correct sequence and that dependent tasks are aware of the status of other tasks.
4. **Load Balancing:** Threads can distribute work among themselves to balance the load and improve the performance of the application. Communication mechanisms help threads share information about their workloads and redistribute tasks as needed.
5. **Event Notification:** Threads can notify each other of events that occur, such as the completion of a task or the availability of new data. This is often done using mechanisms like condition variables or message queues.
6. **Resource Management:** Threads often need to manage shared resources like memory, files, or network connections. Inter-thread communication ensures that these resources are used efficiently and without conflicts.

Motive of project

User Interaction: The code is designed to interact with the user by asking different questions and responding based on the user's answers.

1. **Question-Answer Mechanism**: It implements a system where various predefined questions can be asked, and corresponding answers are provided.

2. **Multiple Question Options**: The system has multiple questions to choose from, covering different topics to engage the user.

3. **User Choice**: The user selects a question from a list of options provided, ensuring an interactive and dynamic experience.

4. **Error Handling**: The system checks for invalid inputs and handles errors gracefully to ensure smooth interaction.

5. **Synchronization**: The code ensures that the process of asking questions and processing answers happens in a coordinated way, even if it involves multiple steps.

6. **Asynchronous Processing**: By using threads, the system can handle asking questions and processing answers simultaneously, making the interaction more efficient.

7. **Real-time Feedback**: The system provides immediate feedback to the user's input, simulating a real-time chat experience.

Software Requirements Specification (SRS) for Chat-like Interaction System

1. Introduction

1.1 Purpose

- The purpose of the software is to create an interactive console-based application for asking questions and providing corresponding answers.

1.2 Scope

- The software aims to provide a chat-like interaction where users can select from a list of questions and receive responses.

2. Overall Description

2.1 Product Perspective

- The software is a standalone console application for user interaction.

2.2 Product Functions

- Display a list of questions.
- Accept user input to select a question.
- Ask the selected question.
- Accept and validate user responses.
- Provide appropriate feedback based on the response.
- Ensure synchronized operations using threading.

3. Specific Requirements

3.1 Functional Requirements

- **FR1:** Display Question List
 - Display a list of 15 questions when the application starts.
- **FR2:** Select Question
 - Prompt the user to select a question by entering a number between 1 and 15.
- **FR3:** Ask Question
 - Display the selected question along with multiple-choice options.
- **FR4:** Accept and Validate Answer
 - Accept the user's answer and validate that it is within the provided options.
- **FR5:** Provide Response
 - Provide an appropriate response based on the user's input.

- **FR6:** Synchronize Operations

- Use mutexes and condition variables to ensure synchronized processing of questions and answers.

3.2 Non-Functional Requirements

- **NFR1:** Performance

- Respond to user inputs within 1 second.

- **NFR2:** Usability

- Provide clear instructions and handle invalid inputs gracefully.

- **NFR3:** Reliability

- Ensure consistent and correct operations through proper synchronization mechanisms.

- **NFR4:** Maintainability

- Maintain modular code for easy updates and maintenance.

4. Interface Requirements

4.1 User Interfaces

- The user interface is text-based and operates in a console window.

5. Other Requirements

5.1 Safety Requirements

- Ensure the software does not perform any actions that could harm the user's system.
- Handle errors gracefully to prevent crashes or unexpected behavior.

5.2 Security Requirements

- No specific security requirements mentioned in the provided code.

5.3 Software Quality Attributes

- Robustness: Handle unexpected inputs and errors gracefully.
- Maintainability: Ensure code modularity and readability for easy maintenance.

Low-Level Design (LLD)

The Low-Level Design (LLD) of the chat-like interaction system elaborates on the internal logic and interactions of individual components, providing a detailed understanding of the system's design

1. User Interface (UI):

• Responsibilities:

- Display questions and options to the user.
- Accept user input for selecting a question.
- Receive and display responses.

• Methods:

- `displayQuestionList()`: Displays the list of questions to the user.
- `displayOptions(const vector<string>& options)`: Displays the options for the selected question.
- `getUserInput()`: Accepts and validates user input.
- `displayResponse(const string& response)`: Displays the response to the user.

2. Question Manager:

• Responsibilities:

- Manages the list of available questions.
- Retrieves selected question and its options.
- Initiates the asking process for the selected question.

• Methods:

- `getQuestion(int questionNumber)`: Retrieves the question object based on the selected question number.
- `askQuestion(QuestionBase* question)`: Initiates the asking process for the selected question.

3. Answer Processor:

• Responsibilities:

- Processes user answers based on the selected question.
- Provides appropriate responses.

• Methods:

- `processAnswer(const string& answer, AnswerBase* answerHandler)`: Processes the user's answer and provides a response using the appropriate answer handler.

4. Threading Module:

- **Responsibilities:**

- Manages multi-threading for concurrent execution of question asking and answer processing.
- Ensures synchronized communication between question asking and answer processing.

- **Methods:**

- `askAndAnswerThread()`: Thread function for question asking and answer processing.
- `waitForResponse()`: Waits for the user's response before proceeding with answer processing.

5. Exception Handling:

- **Responsibilities:**

- Catches and handles exceptions to ensure the smooth operation of the system.
- Provides informative error messages to the user in case of invalid inputs or errors.

- **Methods:**

- `handleException(const exception& e)`: Handles exceptions gracefully and provides error messages to the user.

Interaction Details:

1. User Interaction Flow:

- User Interface prompts the user to select a question.
- User selects a question and provides an answer.
- User Interface displays the response to the user.

2. System Interaction Flow:

- User Interface interacts with the Question Manager to retrieve the selected question and options.
- Threading Module spawns threads for concurrent execution of question asking and answer processing.
- Question Manager initiates the asking process for the selected question.
- Answer Processor processes the user's answer and provides a response.
- Threading Module ensures synchronized communication between question asking and answer processing.
- Exception Handling catches and handles any errors or exceptions, ensuring the system's stability.

Methodology

Included Libraries:

1. `iostream`:

- Provides input and output stream functionality for console-based interaction.

2. `string`:

- Facilitates string manipulation and handling.

3. `exception`:

- Supports exception handling for error management.

4. `thread`:

- Enables multithreading functionality for concurrent execution.

5. `future`:

- Allows asynchronous execution and retrieval of results from threads.

6. `vector`:

- Provides dynamic array functionality for storing lists of questions and options.

7. `mutex` and `condition_variable`:

- Facilitate synchronization and mutual exclusion to prevent data races in multithreaded environments.

8. `memory`:

- Provides smart pointer functionality for managing dynamic memory allocation.

Explanation of the project:

1. Question and Answer System:

- The code implements a system where users can choose from a list of predefined questions and receive corresponding answers.

2. Modular Design:

- It employs separate classes for questions (`QuestionBase`) and answers (`AnswerBase`), allowing easy extension with new question types.

3. User Interaction:

- Users are prompted to select a question from a list. Their responses are then processed to provide the appropriate answers.

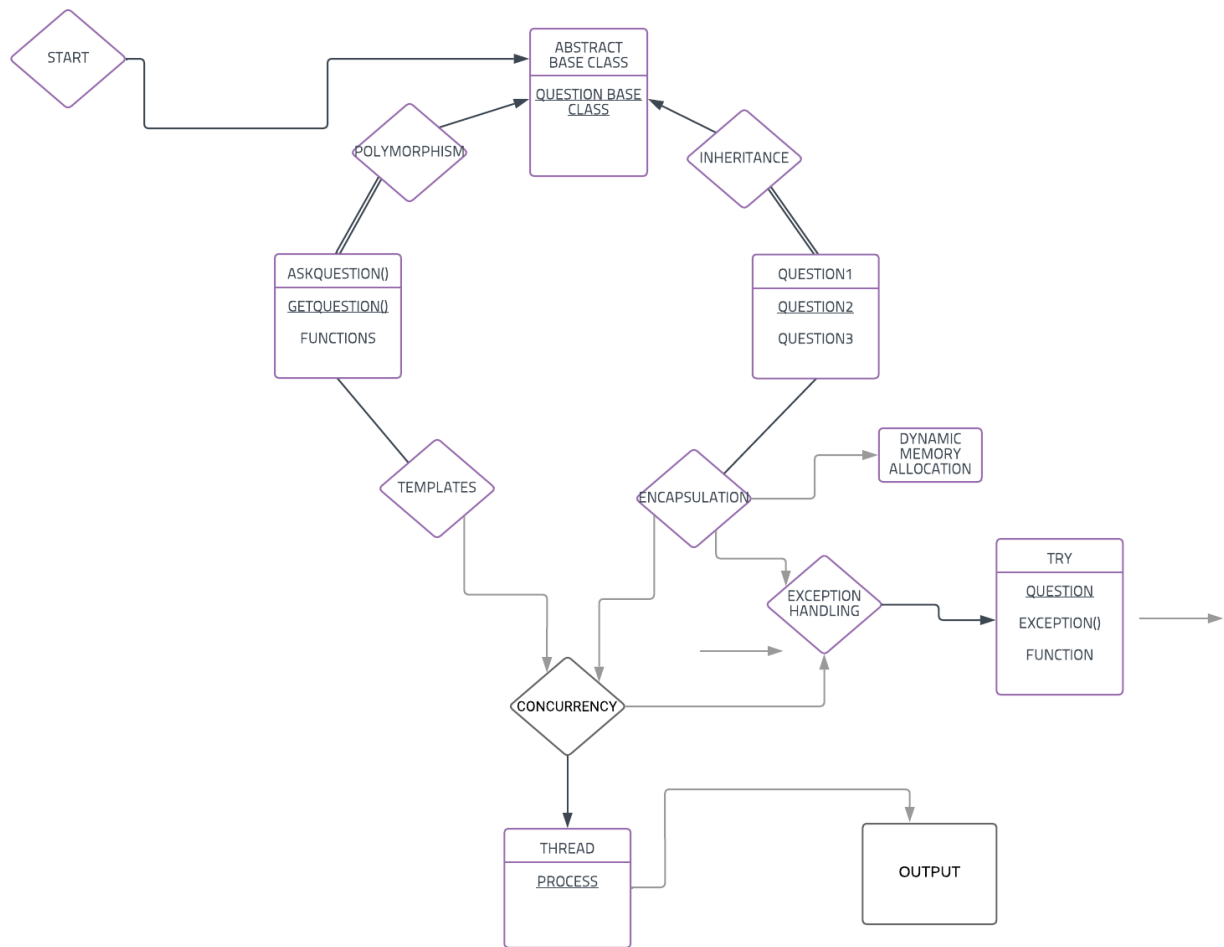
4. **Concurrency:**

- Multithreading is used for concurrent execution of asking questions and processing answers, enhancing responsiveness.

5. **Exception Handling:**

- Exception handling is implemented to manage errors gracefully, ensuring the stability of the program.

Unified Modeling Language daigram



Source code:

```
#include <iostream>
#include <string>
#include <exception>
#include <thread>
#include <future>
#include <vector>
#include <mutex>
#include <condition_variable>
#include <memory>

using namespace std;

class QuestionBase {
public:
    virtual void askQuestion() = 0;
    virtual vector<string> getOptions() = 0;
    virtual ~QuestionBase() {}
};

template <typename T>
string getFunctionName() {
    return typeid(T).name();
}

class Question1 : public QuestionBase {
public:
    void askQuestion() override {
        cout << "How are you feeling? Options: 1) bad, 2) happy, 3) sad" << endl;
    }
    vector<string> getOptions() override {
        return {"bad", "happy", "sad"};
    }
};

class Question2 : public QuestionBase {
public:
    void askQuestion() override {
        cout << "What is the capital of Japan, India, and the US? Options: 1) Tokyo, 2) New Delhi, 3) Washington D.C." << endl;
    }
    vector<string> getOptions() override {
        return {"Tokyo", "New Delhi", "Washington D.C."};
    }
};

class Question3 : public QuestionBase {
public:
    void askQuestion() override {
```

```
cout << "What is the most common technology? Options: 1) AI, 2) ML, 3) Information  
Technology" << endl;  
}  
vector<string> getOptions() override {  
return {"AI", "ML", "Information Technology"};  
}  
};
```

```
class Question4 : public QuestionBase {  
public:  
void askQuestion() override {  
cout << "What is the tallest mountain in the world? Options: 1) K2, 2) Everest, 3)  
Kangchenjunga" << endl;  
}  
vector<string> getOptions() override {  
return {"K2", "Everest", "Kangchenjunga"};  
}  
};
```

```
class Question5 : public QuestionBase {  
public:  
void askQuestion() override {  
cout << "What is the fastest land animal? Options: 1) Cheetah, 2) Lion, 3) Gazelle" <<  
endl;  
}  
vector<string> getOptions() override {  
return {"Cheetah", "Lion", "Gazelle"};  
}  
};
```

```
class Question6 : public QuestionBase {  
public:  
void askQuestion() override {  
cout << "What is the largest ocean? Options: 1) Atlantic, 2) Indian, 3) Pacific" << endl;  
}  
vector<string> getOptions() override {  
return {"Atlantic", "Indian", "Pacific"};  
}  
};
```

```
class Question7 : public QuestionBase {  
public:  
void askQuestion() override {  
cout << "Who wrote 'Hamlet'? Options: 1) Chaucer, 2) Shakespeare, 3) Dickens" << endl;  
}  
vector<string> getOptions() override {  
return {"Chaucer", "Shakespeare", "Dickens"};  
}  
};
```

```
class Question8 : public QuestionBase {
```

```
public:
void askQuestion() override {
cout << "What is the speed of light? Options: 1) 300,000 km/s, 2) 150,000 km/s, 3)
450,000 km/s" << endl;
}
vector<string> getOptions() override {
return {"300,000 km/s", "150,000 km/s", "450,000 km/s"};
}
};
```

```
class Question9 : public QuestionBase {
public:
void askQuestion() override {
cout << "What is the smallest planet in our solar system? Options: 1) Earth, 2) Mars, 3)
Mercury" << endl;
}
vector<string> getOptions() override {
return {"Earth", "Mars", "Mercury"};
}
};
```

```
class Question10 : public QuestionBase {
public:
void askQuestion() override {
cout << "What is the largest mammal? Options: 1) Elephant, 2) Blue Whale, 3) Giraffe" <<
endl;
}
vector<string> getOptions() override {
return {"Elephant", "Blue Whale", "Giraffe"};
}
};
```

```
class Question11 : public QuestionBase {
public:
void askQuestion() override {
cout << "What is the hardest natural substance? Options: 1) Gold, 2) Diamond, 3) Silver"
<< endl;
}
vector<string> getOptions() override {
return {"Gold", "Diamond", "Silver"};
}
};
```

```
class Question12 : public QuestionBase {
public:
void askQuestion() override {
cout << "What is the smallest country in the world? Options: 1) Monaco, 2) Vatican City, 3)
San Marino" << endl;
}
vector<string> getOptions() override {
return {"Monaco", "Vatican City", "San Marino"};
};
```

```
}  
};
```

```
class Question13 : public QuestionBase {  
public:  
void askQuestion() override {  
cout << "What is the largest desert in the world? Options: 1) Sahara, 2) Gobi, 3) Antarctic"  
<< endl;  
}  
vector<string> getOptions() override {  
return {"Sahara", "Gobi", "Antarctic"};  
}  
};
```

```
class Question14 : public QuestionBase {  
public:  
void askQuestion() override {  
cout << "What is the longest river in the world? Options: 1) Nile, 2) Amazon, 3) Yangtze"  
<< endl;  
}  
vector<string> getOptions() override {  
return {"Nile", "Amazon", "Yangtze"};  
}  
};
```

```
class Question15 : public QuestionBase {  
public:  
void askQuestion() override {  
cout << "What is the main ingredient in guacamole? Options: 1) Tomato, 2) Avocado, 3) Pepper" << endl;  
}  
vector<string> getOptions() override {  
return {"Tomato", "Avocado", "Pepper"};  
}  
};
```

```
class AnswerBase {  
public:  
virtual void giveAnswer(const string& answer) = 0;  
virtual ~AnswerBase() {}  
};
```

```
class Answer1 : public AnswerBase {  
public:  
void giveAnswer(const string& answer) override {  
cout << "You chose: " << answer << endl;  
}  
};
```

```
class Answer2 : public AnswerBase {  
public:
```

```
void giveAnswer(const string& answer) override {  
    cout << "The capital of Japan is Tokyo, India is New Delhi, and the US is Washington D.C."  
    << endl;  
}  
};
```

```
class Answer3 : public AnswerBase {  
public:  
    void giveAnswer(const string& answer) override {  
        cout << "The most common technology is " << answer << endl;  
    }  
};
```

```
class Answer4 : public AnswerBase {  
public:  
    void giveAnswer(const string& answer) override {  
        cout << "The tallest mountain in the world is " << answer << endl;  
    }  
};
```

```
class Answer5 : public AnswerBase {  
public:  
    void giveAnswer(const string& answer) override {  
        cout << "The fastest land animal is " << answer << endl;  
    }  
};
```

```
class Answer6 : public AnswerBase {  
public:  
    void giveAnswer(const string& answer) override {  
        cout << "The largest ocean is the " << answer << endl;  
    }  
};
```

```
class Answer7 : public AnswerBase {  
public:  
    void giveAnswer(const string& answer) override {  
        cout << "Hamlet was written by " << answer << endl;  
    }  
};
```

```
class Answer8 : public AnswerBase {  
public:  
    void giveAnswer(const string& answer) override {  
        cout << "The speed of light is " << answer << endl;  
    }  
};
```

```
class Answer9 : public AnswerBase {  
public:  
    void giveAnswer(const string& answer) override {
```

```
cout << "The smallest planet in our solar system is " << answer << endl;
}
};
```

```
class Answer10 : public AnswerBase {
public:
void giveAnswer(const string& answer) override {
cout << "The largest mammal is the " << answer << endl;
}
};
```

```
class Answer11 : public AnswerBase {
public:
void giveAnswer(const string& answer) override {
cout << "The hardest natural substance is " << answer << endl;
}
};
```

```
class Answer12 : public AnswerBase {
public:
void giveAnswer(const string& answer) override {
cout << "The smallest country in the world is " << answer << endl;
}
};
```

```
class Answer13 : public AnswerBase {
public:
void giveAnswer(const string& answer) override {
cout << "The largest desert in the world is the " << answer << endl;
}
};
```

```
class Answer14 : public AnswerBase {
public:
void giveAnswer(const string& answer) override {
cout << "The longest river in the world is the " << answer << endl;
}
};
```

```
class Answer15 : public AnswerBase {
public:
void giveAnswer(const string& answer) override {
cout << "The main ingredient in guacamole is " << answer << endl;
}
};
```

```
class QuestionException : public exception {
string message;
public:
QuestionException(const string& msg) : message(msg) {}
const char* what() const noexcept override {
```



```

return message.c_str();
}
};

```

```

void askAndAnswer(promise<string> && p, QuestionBase* question, condition_variable&
cv, mutex& mtx, bool& ready) {

```

```

try {
question->askQuestion();
vector<string> options = question->getOptions();
int choice;
cin >> choice;

```

```

if (choice < 1 || choice > options.size()) {
throw QuestionException("Invalid choice");
}

```

```

string answer = options[choice - 1];

```

```

{
lock_guard<mutex> lock(mtx);
p.set_value(answer);
ready = true;
}

```

```

cv.notify_one();
} catch (const exception & e) {
p.set_exception(make_exception_ptr(QuestionException("Invalid question asked")));
cv.notify_one();
}
}

```

```

void processAnswer(future<string> && f, AnswerBase* answer, condition_variable& cv,
mutex& mtx, bool& ready) {

```

```

try {
unique_lock<mutex> lock(mtx);
cv.wait(lock, [&ready] { return ready; });
string result = f.get();
cout << "Processing answer: " << result << endl;
answer->giveAnswer(result);
} catch (const exception & e) {
cerr << e.what() << endl;
}
}

```

```

int main() {
vector<unique_ptr<QuestionBase>> questions;
questions.emplace_back(make_unique<Question1>());
questions.emplace_back(make_unique<Question2>());
questions.emplace_back(make_unique<Question3>());
questions.emplace_back(make_unique<Question4>());
questions.emplace_back(make_unique<Question5>());

```

```
questions.emplace_back(make_unique<Question6>());
questions.emplace_back(make_unique<Question7>());
questions.emplace_back(make_unique<Question8>());
questions.emplace_back(make_unique<Question9>());
questions.emplace_back(make_unique<Question10>());
questions.emplace_back(make_unique<Question11>());
questions.emplace_back(make_unique<Question12>());
questions.emplace_back(make_unique<Question13>());
questions.emplace_back(make_unique<Question14>());
questions.emplace_back(make_unique<Question15>());
```

```
vector<unique_ptr<AnswerBase>> answers;
answers.emplace_back(make_unique<Answer1>());
answers.emplace_back(make_unique<Answer2>());
answers.emplace_back(make_unique<Answer3>());
answers.emplace_back(make_unique<Answer4>());
answers.emplace_back(make_unique<Answer5>());
answers.emplace_back(make_unique<Answer6>());
answers.emplace_back(make_unique<Answer7>());
answers.emplace_back(make_unique<Answer8>());
answers.emplace_back(make_unique<Answer9>());
answers.emplace_back(make_unique<Answer10>());
answers.emplace_back(make_unique<Answer11>());
answers.emplace_back(make_unique<Answer12>());
answers.emplace_back(make_unique<Answer13>());
answers.emplace_back(make_unique<Answer14>());
answers.emplace_back(make_unique<Answer15>());
```

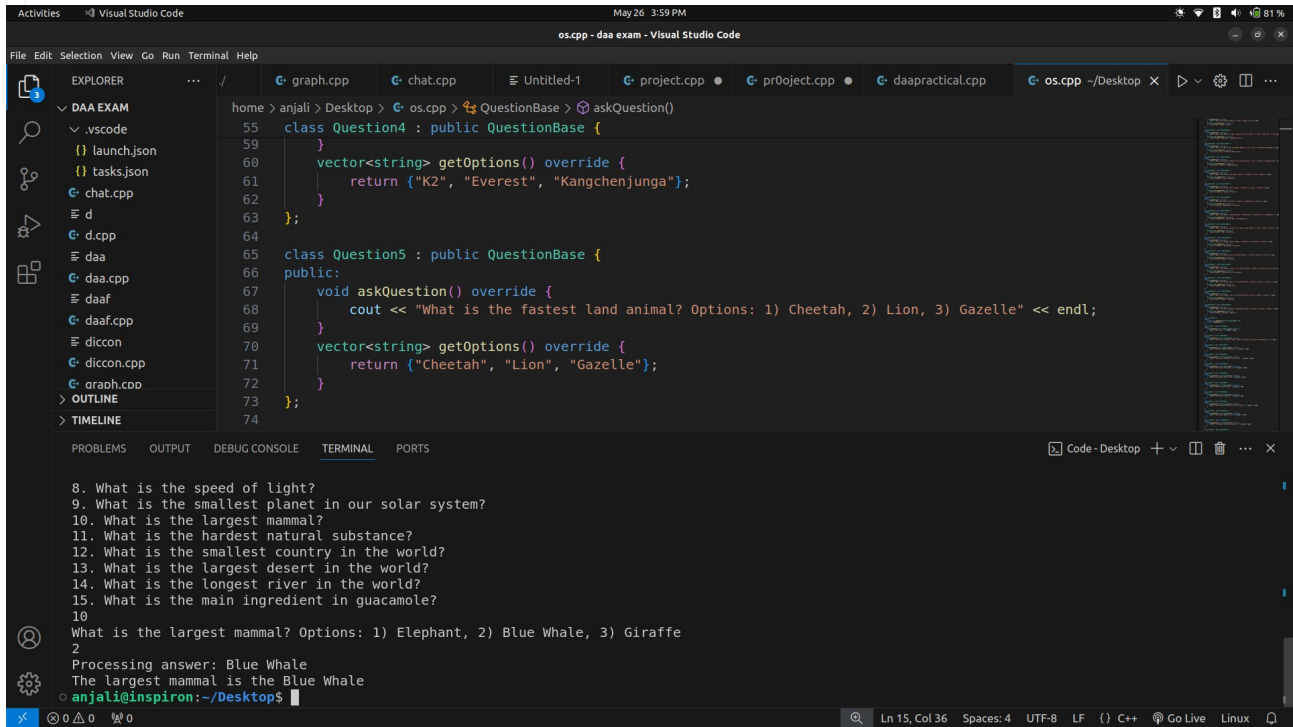
```
mutex mtx;
condition_variable cv;
bool ready = false;
```

```
int userChoice;
```

```
cout << "Choose a question to answer (1 to 15):" << endl;
cout << "1. How are you feeling?" << endl;
cout << "2. What is the capital of Japan, India, and the US?" << endl;
cout << "3. What is the most common technology?" << endl;
cout << "4. What is the tallest mountain in the world?" << endl;
cout << "5. What is the fastest land animal?" << endl;
cout << "6. What is the largest ocean?" << endl;
cout << "7. Who wrote 'Hamlet'?" << endl;
cout << "8. What is the speed of light?" << endl;
cout << "9. What is the smallest planet in our solar system?" << endl;
cout << "10. What is the largest mammal?" << endl;
cout << "11. What is the hardest natural substance?" << endl;
cout << "12. What is the smallest country in the world?" << endl;
cout << "13. What is the largest desert in the world?" << endl;
cout << "14. What is the longest river in the world?" << endl;
cout << "15. What is the main ingredient in guacamole?" << endl;
cin >> userChoice;
```

```
if (userChoice >= 1 && userChoice <= 15) {  
    promise<string> p;  
    future<string> f = p.get_future();  
  
    thread qThread(askAndAnswer, move(p), questions[userChoice - 1].get(), ref(cv), ref(mtx),  
        ref(ready));  
    thread aThread(processAnswer, move(f), answers[userChoice - 1].get(), ref(cv), ref(mtx),  
        ref(ready));  
  
    qThread.join();  
    aThread.join();  
  
    ready = false;  
} else {  
    cout << "Invalid choice" << endl;  
}  
  
return 0;  
}
```

OUTPUT:



The screenshot displays the Visual Studio Code interface. The Explorer sidebar on the left shows a project named 'DAA EXAM' with files like .vscode, launch.json, tasks.json, chat.cpp, d, d.cpp, daa, daa.cpp, daaf.cpp, diccon, diccon.cpp, and araph.cpp. The main editor window shows the code for 'os.cpp' at the path '~/Desktop'. The code defines a 'QuestionBase' class and two subclasses, 'Question4' and 'Question5'. 'Question4' overrides 'getOptions()' to return {'K2', 'Everest', 'Kangchenjunga'}. 'Question5' overrides 'askQuestion()' to print 'What is the fastest land animal? Options: 1) Cheetah, 2) Lion, 3) Gazelle' and 'getOptions()' to return {'Cheetah', 'Lion', 'Gazelle'}. The bottom panel shows the 'TERMINAL' tab with the following output:

```
8. What is the speed of light?
9. What is the smallest planet in our solar system?
10. What is the largest mammal?
11. What is the hardest natural substance?
12. What is the smallest country in the world?
13. What is the largest desert in the world?
14. What is the longest river in the world?
15. What is the main ingredient in guacamole?
10
What is the largest mammal? Options: 1) Elephant, 2) Blue Whale, 3) Giraffe
2
Processing answer: Blue Whale
The largest mammal is the Blue Whale
anjali@inspiron:~/Desktop$
```

The status bar at the bottom indicates the current position is Line 15, Column 36, with 4 spaces, UTF-8 encoding, LF line endings, and C++ language.

