Embedded Software Engineer Test Project: Memory Manager

September 16, 2024

Project Overview

You are tasked with developing a **Bluetooth-enabled Memory Manager** for an embedded system. This project involves creating a memory management system that can efficiently handle dynamic memory allocation and deallocation, with additional capabilities to communicate memory usage statistics and error logs via Bluetooth to a simulated smartphone app. The project aims to evaluate the candidate's understanding of memory management, communication protocols, system design, and optimization techniques. This project should take you around 3 to 4 hours to complete, and you should document your design choices as you make them.

Project Objectives

1. **System Architecture**: Design a modular architecture combining memory management and Bluetooth communication.
2. **Memory Allocation and Deallocation**: Implement custom algorithms for managing dynamic memory.
3. **Fragmentation Management**: Address fragmentation issues to optimize memory usage.
4. **Bluetooth Communication**: Simulate pairing, data exchange, and command handling with a simulated smartphone app.
5. **Error Handling and Logging**: Implement robust error handling and logging mechanisms.
6. **Testing**: Write unit and integration tests to ensure system reliability.

Requirements

1. **Core Features**:
   - **Memory Management**:
     - **Memory Pool Initialization**: Implement a mechanism to initialize a fixed-size memory pool.
     - **Custom Allocation and Deallocation**: Develop algorithms for allocating and deallocating memory blocks from the pool.

  - **Fragmentation Handling**: Address internal and external fragmentation to optimize memory usage.
  - **Memory Utilization Reporting**: Provide statistics on memory usage, including allocated and free memory.
- **Bluetooth Communication**:
  - **Bluetooth Pairing**: Implement functionality simulate the pairing with a simulated smartphone app.
  - **Command Handling**: Implement a basic command protocol for receiving commands from the app (e.g., memory statistics request).
  - **Data Transmission**: Simulate sending back data to the app, such as memory usage statistics and error logs.
  - **Persistent Settings**: Save paired device information and settings persistently.

2. **Architecture**:
   - Use a modular approach, separating the memory manager from other system components.
   - Employ design patterns (e.g., Singleton, Factory) where appropriate for managing memory resources.
3. **Error Handling and Logging**:
   - Detect and handle common memory issues such as out-of-memory conditions and invalid deallocations, as well as Bluetooth issues.
   - Implement a logging system to record memory operations, errors, and system events.
4. **Testing**:
   - Provide a suite of unit tests for key components.
   - Write integration tests to ensure proper interaction between memory management and Bluetooth communication components.

Instructions

1. **Setup**:
   - Use C/C++ as the primary programming language.
   - Use a build system of your choice (e.g., Make, CMake).
2. **Development**:
   - Begin by designing the memory manager architecture and document your design decisions.
   - Adhere to best practices for code readability, maintainability, and documentation.
3. **Testing**:
   - Develop unit tests for individual components, focusing on various scenarios and edge cases.
   - Implement integration tests to verify interactions between components and simulate a real device environment.
4. **Submission**:
   - Ensure that your code compiles and runs correctly.
   - Include a README file with instructions for building and running the project, as well as any additional notes or considerations.
   - Compress all files into a .zip file and submit.

Evaluation Criteria

1. **Design and Architecture** (60%):
   - Clarity and modularity of the system design.
   - Use of design patterns and abstraction.
   - Scalability and flexibility of the architecture.
2. **Implementation** (20%):
   - Correctness and functionality of the implemented features.
   - Code quality, readability, and documentation.
   - Error handling and logging mechanisms.
3. **Testing** (10%):
   - Coverage and quality of unit and integration tests.
   - Testing methodology and framework choice.
4. **Documentation** (10%):
   - Quality of project documentation.
   - Clarity of README and code comments.

Detailed Feature Breakdown

Memory Pool Initialization

- **Fixed-Size Pool**: Implement a memory pool with a fixed size, allocating a contiguous block of memory to be managed.
- **Initialization Function**: Create a function to initialize the memory pool, setting up necessary data structures for management.

Custom Allocation and Deallocation

- **Allocation Algorithm**: Implement a custom algorithm to allocate memory blocks from the pool. Consider strategies such as first fit, best fit, or next fit.
- **Deallocation Algorithm**: Implement a method to free memory blocks, updating the memory pool's internal structures accordingly.

Fragmentation Handling

- **Internal Fragmentation**: Implement techniques to minimize internal fragmentation, such as block splitting.
- **External Fragmentation**: Implement compaction or coalescing techniques to reduce external fragmentation and improve memory utilization.

Memory Utilization Reporting

- **Statistics Collection**: Implement functionality to collect and report statistics on memory usage, such as the total allocated memory, free memory, and fragmentation metrics.
- **Reporting Interface**: Provide an interface for users to query memory usage statistics and optimize memory management strategies.

Simulated Bluetooth Communication

- **Pairing and Communication**: Simulate the pairing and communication process between the memory manager and a simulated smartphone app. Assume that all API's are pre-defined and accessible - you do not have to implement these functions yourself.

Error Handling and Logging

- **Memory Errors**: Implement error detection for common memory issues, such as out-of-memory conditions and invalid deallocations.
- **Logging System**: Design a logging system to capture memory operations, errors, and events for debugging and analysis.

Testing

- **Unit Tests**: Develop unit tests for individual components, covering different allocation and deallocation scenarios.
- **Integration Tests**: Implement integration tests to ensure proper interaction between memory management components and overall system stability.