# IPC Project README

**Project Overview**

The IPC (Inter-Process Communication) project is a robust and efficient C++ application designed to demonstrate various IPC mechanisms, including message queues, shared memory, and semaphores. This project showcases key principles of process communication and synchronization, making it an ideal demonstration of practical client-server interaction.

**Scope**

This project focuses on implementing and validating IPC mechanisms that allow multiple processes to communicate and synchronize their actions effectively. The project includes a main application demonstrating these functionalities and unit tests to ensure correctness.

**Functional Requirements**

1. **Message Queues**:
   * The system shall allow processes to send and receive messages using message queues.
2. **Shared Memory**:
   * The system shall allow processes to share data using shared memory segments.
3. **Semaphores**:
   * The system shall provide synchronization mechanisms using semaphores to control access to shared resources.
4. **Unit Testing**:
   * The system shall include unit tests to validate the functionality of message queues, shared memory, and semaphores.

**Non-functional Requirements**

1. **Performance**: The IPC mechanisms should operate efficiently with minimal latency.
2. **Reliability**: The system should handle errors gracefully and ensure that resources are managed correctly.
3. **Maintainability**: The code should be well-structured and documented to facilitate future maintenance and enhancements.
4. **Portability**: The implementation should be portable across Unix-like operating systems.

**User Roles**

* **Developers**: Responsible for implementing the IPC mechanisms and writing unit tests.
* **Testers**: Responsible for validating the functionality and performance of the IPC mechanisms.
* **End Users**: Users of applications that leverage IPC for communication and synchronization.

Input/Output Specifications

* **Input**:
  + Messages to be sent via message queues.
  + Data to be written to shared memory.
  + Semaphore operations (wait and signal).
* **Output**:
  + Confirmation messages indicating successful sending and receiving of messages.
  + Data read from shared memory.
  + Status messages indicating semaphore acquisition and release.

Challenges

1. **Concurrency Issues**: Managing concurrent access to shared resources and ensuring proper synchronization using semaphores.
2. **Error Handling**: Implementing robust error handling for system calls and ensuring that resources are cleaned up in case of failures.
3. **Testing**: Creating effective unit tests that accurately validate the functionality of IPC mechanisms.

**Skeleton Code Structure**

text

/ipc\_project/

│

├── CMakeLists.txt # CMake build configuration

├── IPC.h # Header file for IPC class

├── IPC.cpp # Implementation of IPC class

├── ipc\_main.cpp # Main application demonstrating IPC

└── ipc\_test.cpp # Unit tests for IPC functionality

Installation and Setup

1. **Clone the Repository**:

git clone <repository-url>

cd ipc\_project

1. **Install Dependencies**:
   * For Linux:

sudo apt-get update

sudo apt-get install build-essential libboost-all-dev

1. **Compile the Project**:

bash

mkdir build

cd build

cmake ..

make

Usage

1. **Running the Main Application**:  
   After building the project, you can run the main application:

./ipc\_example

1. **Running Tests**:  
   To run the unit tests, execute the following command:

./ipc\_test

**Testing and Validation**

The project includes unit tests written using Google Test to validate the functionality of the IPC mechanisms. The tests cover:

* **Message Queue Functionality**: Verifying that messages are sent and received correctly.
* **Shared Memory Functionality**: Ensuring that data can be shared between processes.
* **Semaphore Functionality**: Confirming that semaphores correctly manage access to shared resources.

All tests were executed successfully, confirming the correctness of the implementation.

Documentation

Documentation is provided within the code through comments and method descriptions. The code is structured to facilitate understanding and maintenance.

**Milestones**

**1**: Research and design IPC mechanisms.

**2**: Implement message queues and shared memory.

**3**: Implement semaphores and integrate all components.

**4**: Write unit tests and validate functionality.

**5**: Finalize documentation and prepare for submission.

License

This project is licensed under the MIT License. See the LICENSE file for more details.

Acknowledgments

* Special thanks to the authors of the books and resources that provided guidance on IPC mechanisms and Google Test.