Embedded Software Engineer Technical Assignment

Objective:
Demonstrate your ability to work in a Linux multithreaded environment by developing two processes that communicate with each other. Each process should have at least two threads, and synchronization mechanisms should be implemented to ensure proper coordination.

Task:
Create a C++ program that consists of two separate processes, each containing at least two threads. The processes should communicate with each other using inter-process communication (IPC) techniques. Employ synchronization mechanisms to ensure thread safety and proper communication between the processes.

Requirements:

1. **Producer Process:**
    - Implement two threads within the Producer process.
    - The first thread should generate random numbers at regular intervals and send them to the Consumer process.
    - The second thread should receive acknowledgments from the Consumer process, confirming that the numbers have been received successfully.
2. **Consumer Process:**
    - Implement two threads within the Consumer process.
    - The first thread should receive the random numbers sent by the Producer process and perform a simple computation (e.g., summing the numbers).
    - The second thread should periodically send acknowledgments back to the Producer process after the computation is completed.
3. **Inter-Process Communication (IPC):**
    - Use an appropriate IPC mechanism to facilitate communication between the Producer and Consumer processes. Possible IPC mechanisms include:
        - Named pipes (FIFO)
        - Shared memory
        - Message queues
        - Sockets

4. **Synchronization:**

- Implement synchronization mechanisms (e.g., mutexes, semaphores) to ensure that threads within each process coordinate properly and that shared resources (e.g., IPC channels) are accessed safely.

5. **Error Handling:**
    - Include error handling to manage unexpected conditions, such as communication failures or resource allocation errors.
6. **Project Organization:**
    - Use CMake to organize and build the project.
    - Provide a CMakeLists.txt file to support the building process.
7. **Compilation and Execution:**
    - Provide clear instructions on how to compile and execute your code on a Linux system.
    - Ensure that your code can be compiled using g++.
8. **Documentation:**
    - Include a README.md file with:
        - Instructions on how to set up and run the program.
        - An explanation of the chosen IPC mechanism and its suitability for this task.
        - A description of the synchronization mechanisms used and how they ensure thread safety.
9. **Submission:**
    - Don’t make the project public. Keep it private.
    - Share the project via a GitHub repository and provide access to valeriyi@pointgrab.com.
    - Provide a commit history to show the development process.