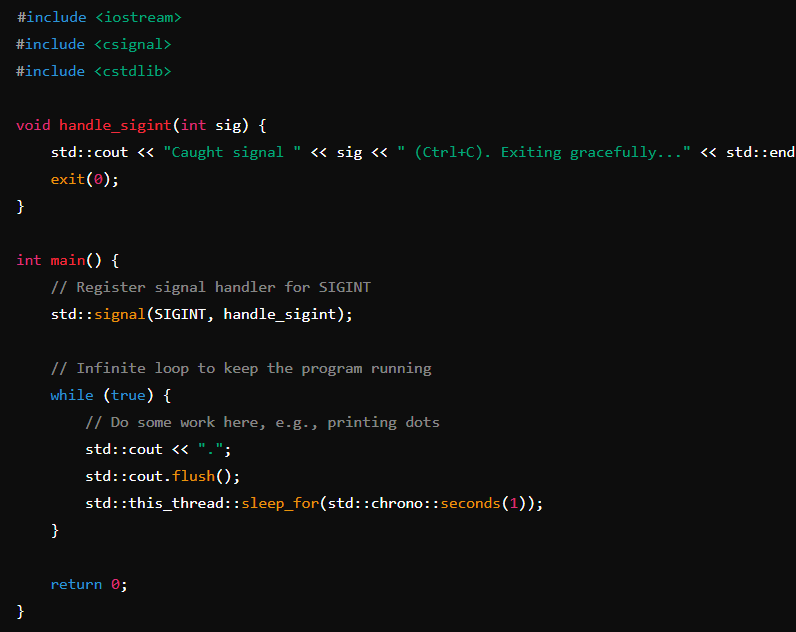
1. Simple Signal Handler: Write a C++ program that handles the SIGINT signal (Ctrl+C) gracefully by printing a custom message before exiting.

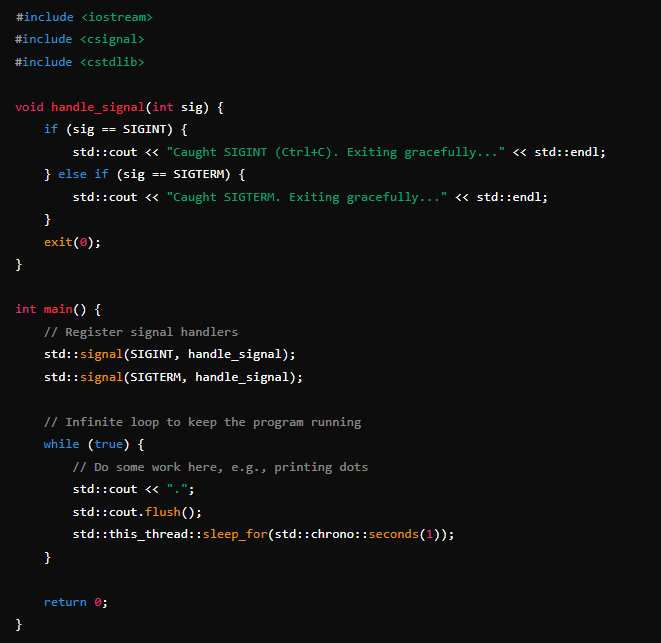
Multiple Signal Handling: Create a program that handles both SIGINT and SIGTERM signals, printing a different message for each.

Ignoring Signals: Develop a program that ignores the SIGTERM signal and continues execution even after it's sent.

Simple Signal Handler :



Multiple Signal Handling :

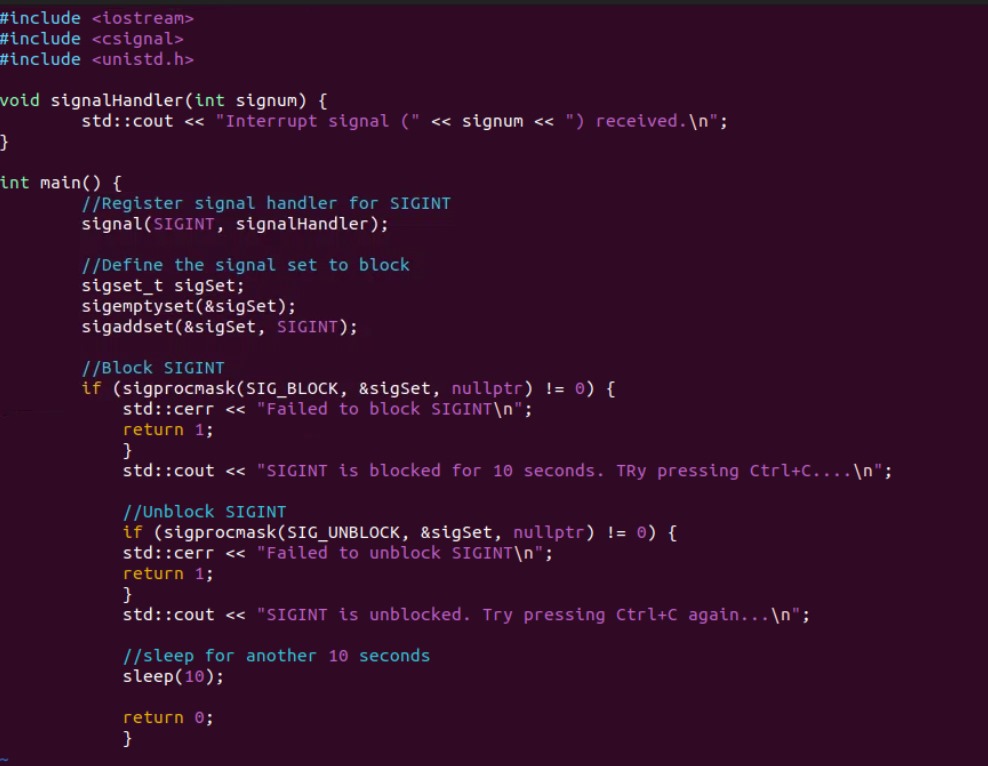


Ignoring Signals :





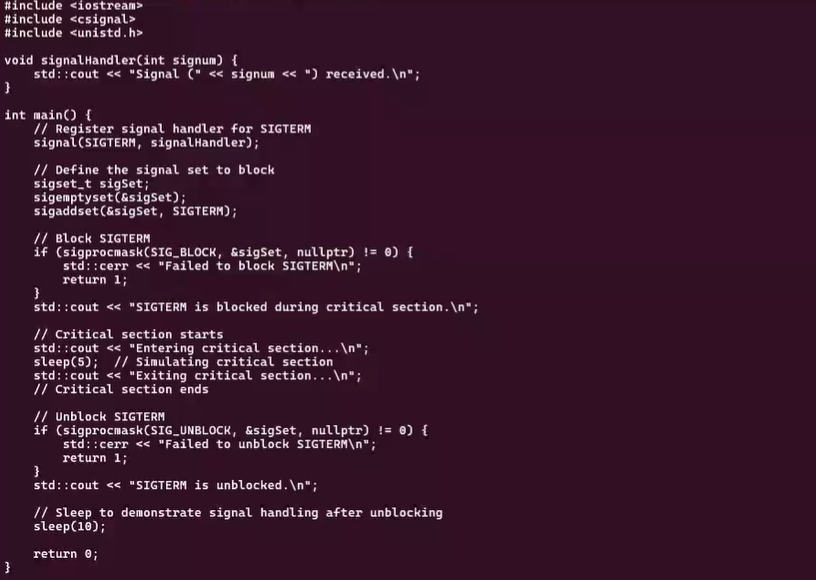
Example :



Example :



Example :



2.Signal Masking and Unmasking for Graceful Shutdown

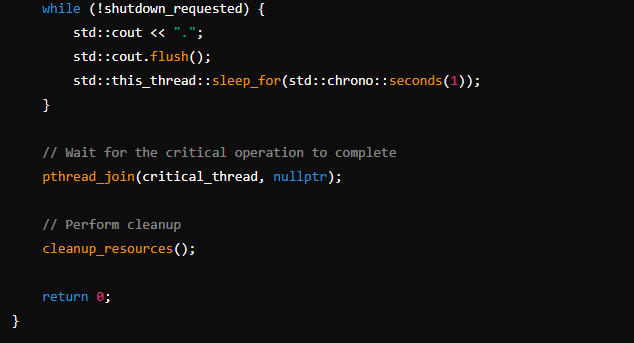
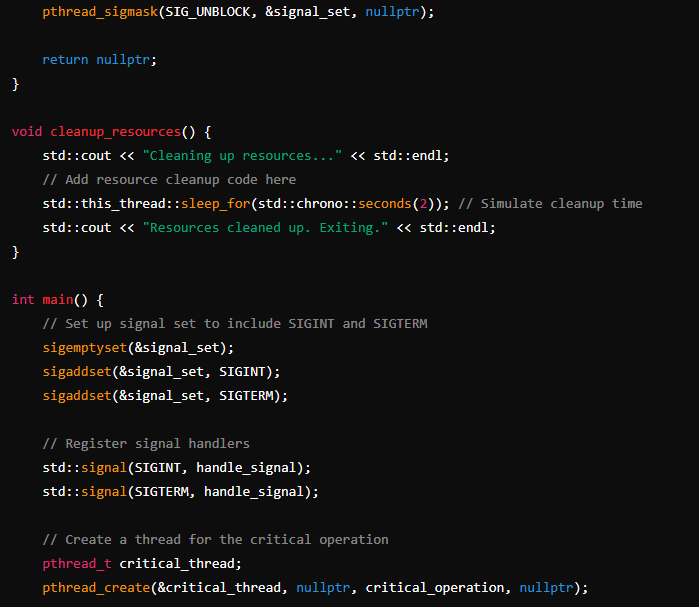
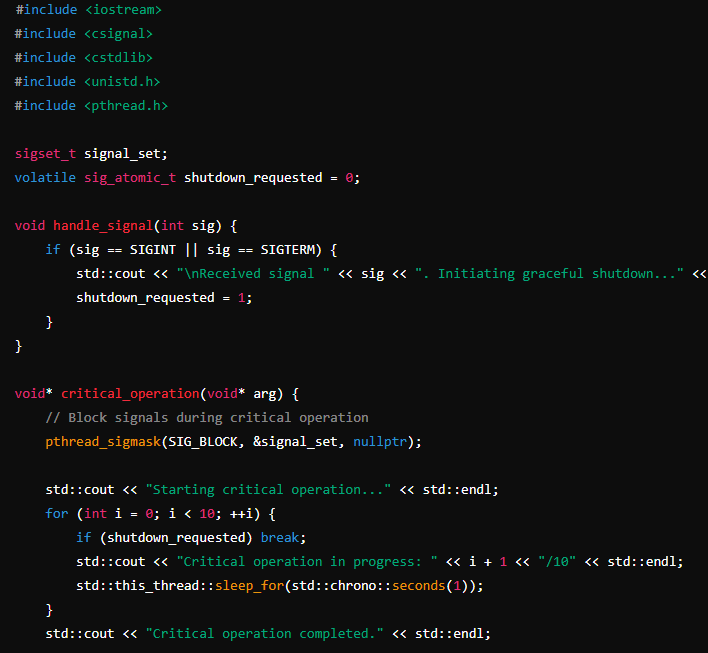
Problem: Develop a C++ application that gracefully handles termination signals (e.g., SIGTERM, SIGINT) by masking specific signals during critical operations and unmasking them afterwards. Implement a clean shutdown procedure that ensures all resources are released before the process exits.

Key Challenges:

Determining the appropriate signals to mask during critical operations.

Ensuring timely unmasking of signals to avoid process hangs.

Implementing a robust shutdown mechanism that handles unexpected interruptions.



3. Signal Masking and Unmasking for Error Handling

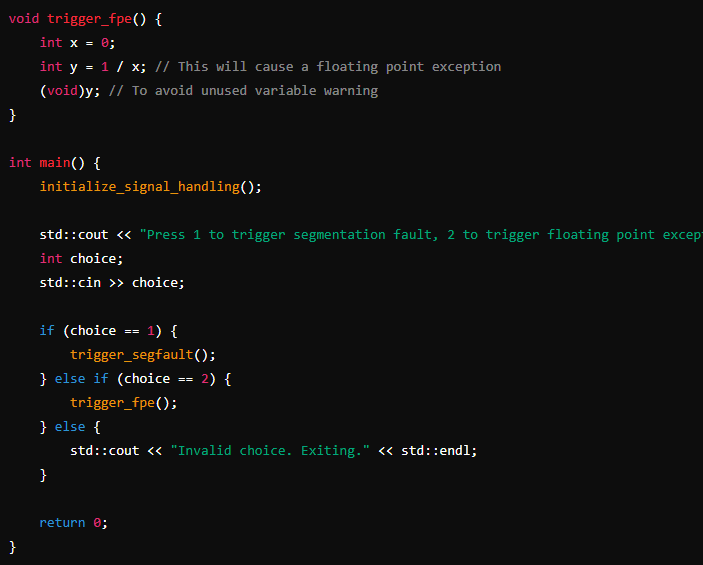
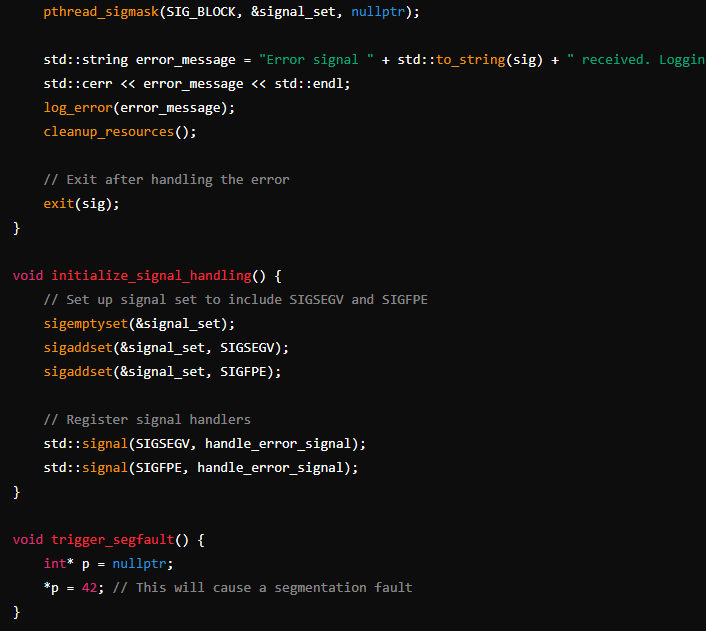
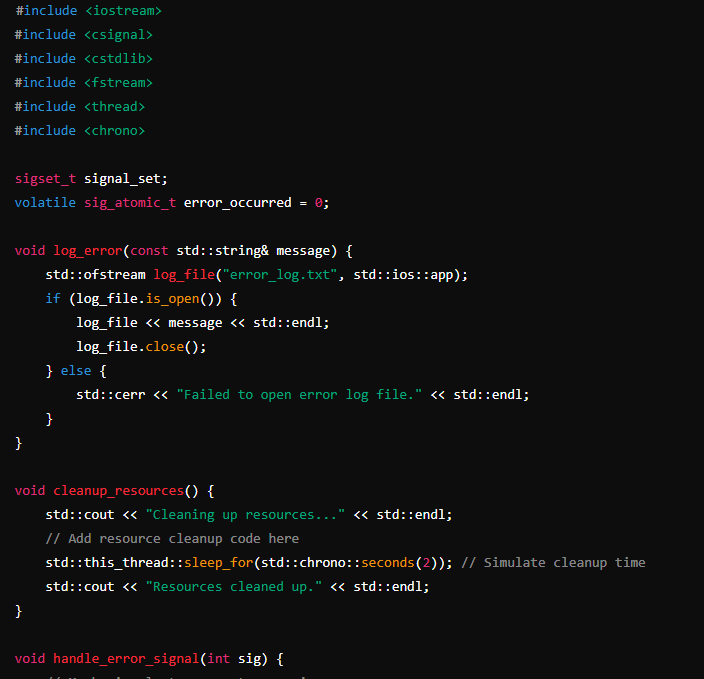
Problem: Create a C++ application that uses signal masking and unmasking to handle errors gracefully. Mask specific signals during error handling routines to prevent recursive signal delivery. Implement a mechanism to log error details and perform necessary cleanup actions before re-enabling the masked signals.

Key Challenges:

Identifying the appropriate signals to mask during error handling.

Preventing infinite recursion of signal handlers.

Ensuring proper error logging and resource cleanup.



Example :

