**Bug Report**

# **Title:** Thread Pool Implementation

# Description

The thread pool implementation in C++ has several issues, including improper handling of task queue overflow, potential race conditions, and inadequate error handling. These issues could lead to task mismanagement, program crashes, or unpredictable behaviour under concurrent load.

## Steps to Reproduce

1. **Compile the Code:**
   * Save the provided code as main.cpp.
   * Compile the code using: g++ -o thread\_pool main.cpp -pthread.
2. **Run the Executable:**
   * Execute the program with: ./thread\_pool.
3. **Test Queue Overflow Handling:**
   * Adjust the queue size and add a number of tasks that exceed the queue capacity to observe behavior.
4. **Stress Test Concurrent Task Addition:**
   * Modify the code to add tasks from multiple threads simultaneously to test thread safety and synchronization.

# Expected Behaviour

1. **Queue Overflow Handling:**
   * The system should handle task overflow gracefully, either by blocking or by rejecting new tasks when the queue is full.
2. **Thread Safety:**
   * Access to the shared task queue should be properly synchronized to prevent race conditions.
3. **Error Handling:**
   * Tasks should not be added if the thread pool is in a shutdown state, and appropriate exceptions should be thrown.

# Actual Behaviour

1. **Queue Overflow Handling:**
   * The queue might not handle overflow scenarios correctly, potentially causing crashes or undefined behaviour when the queue is full.
2. **Thread Safety Issues:**
   * There could be race conditions due to improper synchronization, leading to unpredictable behaviour when tasks are added or removed concurrently.
3. **Error Handling:**
   * The code may not correctly handle errors related to adding tasks after the thread pool has been shut down, leading to possible exceptions or program crashes.

# Code Affected

1. **TaskQueue Class:**
   * Methods: push and pop
   * Issue: Improper handling of queue capacity and synchronization.
2. **ThreadPool Class:**
   * Methods: add\_task and worker
   * Issue: Potential race conditions and improper handling of the shutdown state.

# Suggested Fixes

1. **Update TaskQueue::push:**
   * Ensure that the push method blocks or handles cases where the queue is full. Properly synchronize access to the queue to prevent race conditions.

void push(Task task) {

std::unique\_lock<std::mutex> lock(mutex);

// Wait until there is space in the queue

condition\_full.wait(lock, [this]() { return queue.size() < capacity; });

queue.push(task);

condition\_empty.notify\_one();

}

1. **Update ThreadPool::add\_task:**
   * Modify the add\_task method to handle errors when attempting to add tasks after shutdown has been initiated.

void add\_task(std::function<void()> function) {

{

std::unique\_lock<std::mutex> lock(mutex);

if (shutdown) {

throw std::runtime\_error("Cannot add task to a shutdown thread pool");

}

}

task\_queue.push({function});

condition.notify\_one();

}

1. **Update ThreadPool::worker:**
   * Improve the worker method to ensure correct handling of shutdown signals and proper processing of tasks.

void worker() {

while (true) {

Task task;

{

std::unique\_lock<std::mutex> lock(mutex);

condition.wait(lock, [this]() { return shutdown || !task\_queue.is\_empty(); });

if (shutdown && task\_queue.is\_empty()) {

return;

}

task = task\_queue.pop();

}

task.function();

}

}

# Testing After Fixes

1. **Recompile and Run:**
   * Ensure the code compiles without errors.
   * Run the executable and observe task handling, especially during high load and concurrent task addition.
2. **Test Overflow Handling:**
   * Validate that the queue correctly handles overflow situations by adding tasks beyond the queue’s capacity.
3. **Verify Thread Safety:**
   * Stress test with multiple concurrent task additions to ensure proper synchronization and prevent race conditions.
4. **Check Error Handling:**
   * Verify that attempting to add tasks after shutdown triggers appropriate exceptions and does not lead to undefined behaviour.