Thread Pool Basic Implementation Solutions

Overview

- In the previous lecture, we said that a thread pool needs:
  - A fixed-size container of C++ thread objects
  - A queue of tasks, which are represented as callable objects
- How does our thread pool class implement these requirements?
  - It has std::vector<std::thread> as a data member, for the threads
  - It uses the concurrent queue from an earlier lecture
  - The tasks are represented as std::function objects
  - The concurrent queue stores these std::function objects

Interface

- Describe the interface of the thread pool class
  - The thread pool class has a public submit() member function
  - Users call this to send a task to the thread pool
  - The thread pool class also has a private worker() member function
  - This is the entry point function for the threads
  - The class also has a constructor and a destructor

Member functions

- Describe the member functions of the thread pool class
  - submit() takes an std::function object as argument (a task)
  - It pushes the std:function object onto the concurrent queue
  - The next thread which becomes available will invoke the task
  - worker() is an infinite loop
  - It pops a task off the queue, then invokes it
  - When the task is complete, it pops another task off the queue, and so on
  - The constructor populates the std::vector of std::thread objects
  - These std::thread objects have worker() as their entry point function
  - The destructor calls join() on the std::thread objects

thread_pool Usage

- How does other code use the thread pool class?
  - The code creates an object of the thread pool class
  - It then sends tasks to the thread pool
  - To do this, it calls the submit() member function
  - It passes the task as the argument to submit()
  - The task function must have the same signature as the elements of the concurrent queue