Mini Project Operating Systems

Implementation Of a Thread Pool in C using POSIX Threads

BY

PRAGYA DAGA - 1PI12CS117

SURAJ KOHLI - 1PI12CS185

VIVEK BHARADWAJ - 1PI12CS200

Under the guidance of
N.S. Kumar

August 2014 – December 2014

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
PES INSTITUTE OF TECHNOLOGY
100 FEET RING ROAD, BANASHANKARI III STAGE
BANGALORE-560085

Abstract

- The aim of our project is to create a thread pool which is getting initialised beforehand to save time, for the user.

- A thread pool has a certain number of active threads which have been assigned some tasks.

- These tasks are queued up in a special queue maintained for the pool.

- Threads which are idle, pickup the task from the queue and execute them.

- It also contains some threads which are idle and will be terminated after some period of time(linger period).

- POSIX threads or Pthreads is a POSIX Standard for threads. The implementation of the API is available on most *nix platforms.

- The standard is realised in C using a pthread.h interface and a libpthread library

How Is It Done ?

Our project is essentially divided into 2 main modules

- Handling Tasks, Queue to feed the pool
    - Task submodule
    - Queue submodule

- Thread pool itself

Tasks

- Provide an abstraction of the computation the thread performs.

- They are functions that take and return a ptr to void. These can be cast appropriately in the function / at the caller respectively.

- Has a simple constructor mk_task which returns a ptr to the task created as per the specifications in the mk_task call.

- It also has a run_task method, that runs the task.

Queue

- Provides an abstraction of a FirstInFirstOut data structure.

- Contains all the pending tasks for the pool

- A thread might request for a task from the queue

- A task may be added to the queue.

Thread pool

- The actual pool of threads is implemented here.

- The pool may at any given point be in 3 states :
  - working

    This is the normal functioning of the pool.
    This is the start state of its life.

  - graceful_shutdown

    This is a state wherein a call to shutdown the pool has been made by the user.
    Now, all the pending tasks in the queue are finished.
    No more tasks may be added to the pool.
    This is the normal way to shutdown the pool.

  - immediate_shutdown

    This is similar to graceful shutdown.
    Instead, of finishing the pending tasks, any idle thread will be killed and those executing a task shall finish their respective task and then get killed.
    This should only be used in case of an emergency.

- The functionalities provided by the pool are in the form of the following functions.
  - mk_threadpool(n_threads)

  - threadpool_add_task(pool, task)

  - threadpool_destroy(pool, shutdown_type)

- Inspired by Pthreads, we also provide the pool functionality with threadpool.h interface along with a libthreadpool library(static library with .a extension)

<u>How To Use It ?</u>

1. Change to the root directory of the project.

2. Run configure.sh to setup all required directories.
   ```
   $ ./configure.sh
   ```

3. Compile the library. Library will be present in the lib directory of the root directory of the project
   ```
   $ make lib
   ```

4. Include threadpool.h and task.h in your source files. These are in the include/ directory of the project.
   You may want to use the -I path/to/include (upper case i) while compiling your sources

5. Link against this library using the -lthreadpool option.
   You may want to use the -L path/to/lib option to tell the linker about the path of the libraries.

6. You must also link against the pthread library, since libthreadpool is an extension
   Use the -lpthread option for this.

7. A more thorough list of instructions is present in the README file for the project.