Operating Systems
Project #2*

소프트웨어학부

2020년 6월 1일

Designing a Thread Pool

Thread pools were introduced in Section 4.5.1. When thread pools are used, a task is submitted to the pool and executed by a thread from the pool. Work is submitted to the pool using a queue, and an available thread removes work from the queue. If there are no available threads, the work remains queued until one becomes available. If there is no work, threads await notification until a task becomes available. This project involves creating and managing a thread pool, and it can be completed using Pthreds and POSIX synchronization.

The Client

Users of the thread pool will utilize the following API:

- `void pool_init()` – Initializes the thread pool.
- `int pool_submit(void (*f)(void *p), void *p)` – where `f` is a pointer to the function that will be executed by a thread from the pool and `p` is a parameter passed to the function.
- `void pool_shutdown(void)` – Shuts down the thread pool once all tasks have completed.

We provide an example program `client.c` in the source code download that illustrates how to use the thread pool using these functions.

The Thread Pool

In the source code download we provide the C source file `threadpool.c` as a partial implementation of the thread pool. You will need to implement the functions that are called by client users, as well as several additional functions that support the internals of the thread pool. Implementation will involve the following activities:

1. The `pool_init()` function will create the threads at startup as well as initialize mutual-exclusion locks and semaphores.

*이 과제는 Operating System Concepts 10판 7장 프로그래밍 프로젝트와 같음

2. The `pool_submit()` function is partially implemented and currently places the function to be executed - as well as its data - into a task struct. The task struct represents work that will be completed by a thread in the pool. `pool_submit()` will add these tasks to the queue by invoking the `enqueue()` function, and worker threads will call `dequeue()` to retrieve work from the queue.

   The `pool_submit()` function has an int return value that is used to indicate if the task was successfully submitted to the pool (0 indicates success, 1 indicates failure). `pool_submit()` will return 1 if there is an attempt to submit work and the queue is full.

3. The `worker()` function is executed by each thread in the pool, where each thread will wait for available work. Once work becomes available, the thread will remove it from the queue and run the specified function. A semaphore can be used for notifying a waiting thread when work is submitted to the thread pool. Either named or unnamed semaphores may be used.

4. A mutex lock is necessary to avoid race conditions when accessing or modifying the queue.

5. The `pool_shutdown()` function will cancel each worker thread and then wait for each thread to terminate by calling `pthread_join()`. The semaphore operation `sem_wait()` is a cancellation point that allows a thread waiting on a semaphore to be cancelled.

Refer to the source-code download for additional details on this project. In particular, the `README` file describes the source and header files, as well as the `Makefile` for building the project.

threadpool.c

이번 프로젝트에서 학생이 새로 작성한 모든 코드는 `threadpool.c`에 들어가야 한다. 대기열의 길이는 10, worker 스레드의 갯수는 3으로 가정한다. 따라서 세 개의 스레드가 길이가 최대 10개인 대기열에 있는 작업을 하나씩 꺼내서 실행시키는 구조이다. `client.c`는 스레드 풀이 잘 동작하는지 검증하기 위한 것으로 수정 없이 그대로 사용해야 한다.

macOS

macOS는 unnamed 세마포를 더 이상 지원하지 않는다. 맥 사용자는 named 세마포를 사용하기 바란다. Ubuntu나 다른 Linux 사용자는 unnamed 세마포를 여전히 사용할 수 있다.

제출물

스레드 풀이 잘 설계되고 구현되었다는 것을 보여주는 자료를 각자가 판단하여 PDF로 묶어서 `이름_학번_PROJ2.pdf`로 제출한다. 여기에는 다음과 같은 것이 반드시 포함되어야 한다.

- 본인이 설계한 스레드 풀 알고리즘 (1쪽 분량)
- 프로그램 소스파일
- 컴파일 과정을 보여주는 화면 캡처
- 실행 결과물의 주요 장면과 그에 대한 설명 등

HK