

Report-Project5

付昊源

517021910753

2019 年 6 月 11 日

1 Designing a Thread Pool

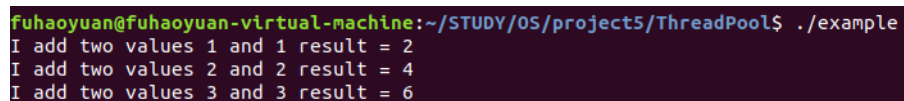
1.1 程序设计思想

本 project 的目的是实现一个线程池，每次一个 task 来时，若线程池中有空闲的线程，则直接将该任务分给线程池去执行，否则任务将会被存于任务队列中，直到有空闲的线程再去执行。留给用户的接口有 `pool_init()`, `pool_submit()`, `pool_shutdown()`，分别用于线程池的初始化，任务的分配和线程池的回收。

初始化时，要初始化线程池中的各个线程以及所需使用的信号量、互斥锁。接收一个线程时，首先将其排入队列，等到有空闲线程可用时，再执行该任务。此外，`worker()` 函数是每一个线程将会调用的函数：先在 queue 中找到一个线程，按照任务所给的函数执行，然后将自己的状态改为空闲，最后返回。

为了体现线程池无空闲线程的等待状况，我在定义任务函数时让该函数 `sleep` 三秒，从而可以在输出中展现等待。（如图 1、图 2、图 3 所示）

1.2 运行结果截图



```
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project5/ThreadPool$ ./example
I add two values 1 and 1 result = 2
I add two values 2 and 2 result = 4
I add two values 3 and 3 result = 6
```

图 1: Thread Pool-1

```
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project5/ThreadPool$ ./example
I add two values 1 and 1 result = 2
I add two values 2 and 2 result = 4
I add two values 3 and 3 result = 6
I add two values 5 and 5 result = 10
I add two values 4 and 4 result = 8
I add two values 6 and 6 result = 12
```

图 2: Thread Pool-2

```
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project5/ThreadPool$ ./example
I add two values 1 and 1 result = 2
I add two values 2 and 2 result = 4
I add two values 3 and 3 result = 6
I add two values 5 and 5 result = 10
I add two values 4 and 4 result = 8
I add two values 6 and 6 result = 12
I add two values 7 and 7 result = 14
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project5/ThreadPool$
```

图 3: Thread Pool-3

1.3 核心数据结构及代码解释

在本 project 的实现中, 我定义了结构体 `thread`, 包含 `pthread` 变量 `tid`, 整型变量: 下标 `index` 和工作状态 `working`。定义下标的作用是便于在线程执行的过程中改变线程的工作状态从而使其他任务可以使用该线程。信号量 `sem` 和互斥锁 `mutex` 用于控制线程的分配, 互斥锁 `mutex1` 用于 `queue` 的修改。在我的设计中, 线程池和等待队列均为数组, 最大数分别为 3 和 10。

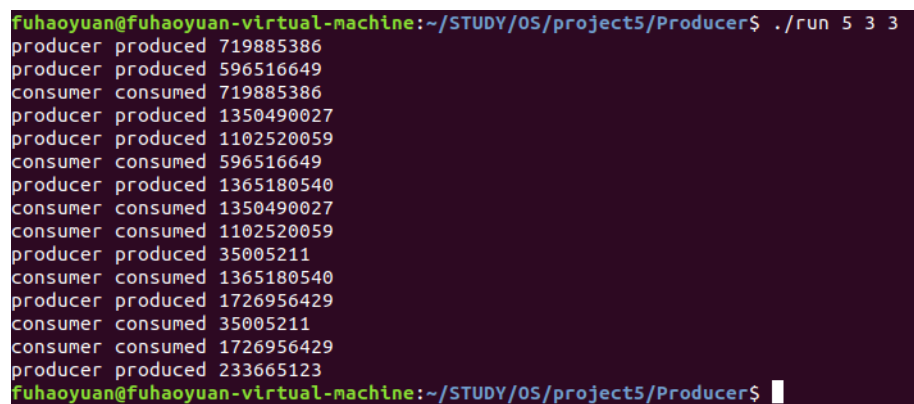
首先是 `enqueue()` 和 `dequeue()` 函数。最先进入队列的数据下标最小, 其他的实现较为简单, 但需注意队列内容的修改需要加互斥锁 `mutex`。接下来是 `worker()` 函数, 它由线程调用, 通过 `execute()` 函数, 执行 `task` 中的函数。在 `worker()` 函数返回之前, 需要改变本线程的工作状态 `working` 并将信号量 `sem` 减一。在 `pool_submit()` 函数的实现中, 应先通过参数获取 `work` 函数和数据 `data`, 压入队列, 调用可用线程执行 `work()` 函数, 最后返回一个值表明是否成功添加。线程池的初始化和回收相对简单, 只需将线程、信号量、互斥锁初始化或回收。

2 Producer-Consumer Problem

2.1 程序设计思想

本 project 需要我们模拟生产者-消费者问题，线程之间共享一个数组 buffer，每次只能有一个生产者/消费者对数组进行修改。根据书上给出的解决方案，我们采用一个互斥锁 mutex 和两个信号量 full、empty。以消费者进程为例，首先它需要检查数组中是否有元素可以进行消费，即 full，然后使用互斥锁 mutex 加锁，保证单次只有一个线程对数组进行修改，结束后对 empty 增加，表示数组中被消费了一个元素。生产者进程完全类似，只有将 empty 和 full 调换即可。

2.2 运行结果截图



```
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project5/Producer$ ./run 5 3 3
producer produced 719885386
producer produced 596516649
consumer consumed 719885386
producer produced 1350490027
producer produced 1102520059
consumer consumed 596516649
producer produced 1365180540
consumer consumed 1350490027
consumer consumed 1102520059
producer produced 35005211
consumer consumed 1365180540
producer produced 1726956429
consumer consumed 35005211
consumer consumed 1726956429
producer produced 233665123
fuhaoyuan@fuhaoyuan-virtual-machine:~/STUDY/OS/project5/Producer$
```

图 4: Producer-Consumer Problem

2.3 核心数据结构及代码解释

首先先对 buffer.c 文件进行简单的解释。这个文件中定义了 buffer_init()、insert_item() 和 remove_item() 函数。其中 buffer_init() 函数初始化公用数组 buffer 以及互斥锁和信号量，insert_item() 和 remove_item() 函数的实现很类似，只是前者用于加入元素，后者用于减少元素，而且需注意信号量的判断逻辑。

接下来对 problem.c 文件进行说明。从命令行中得到三个值：程序运行时间、生产者线程数量和消费者线程数量。这里我们将生产者和消费者线程

分别定义为两个线程链表 LinkThread，从命令行中读入参数后完成链表的初始化，并让各线程开始执行消费/生产的任务，即 consumer() 和 producer() 函数。在这两个函数中，各线程先等待随机时间，再开始向 buffer 中随机生产或从 buffer 中消费元素。

需要注意的是，由于线程需要在我们给定的程序运行时间后结束，因此在 consumer() 和 producer() 函数中，每次循环过后要使用 pthread_testcancel() 函数。在主函数中，sleep() 后，调用 pthread_cancel() 函数，代替 pthread_join() 函数，进行线程的强行结束。