Operating System CH7

Susie Glitter

2025 年 7 月 7 日

注：本次实验使用了 VMware 中的 ubuntu-16.04.6-desktop

1 任务一：实现线程池

1.1 threadpool.c

在 threadpool 中，先使用初始化函数开辟数目合适的线程，它们从队列中不断取出任务（函数与参数），执行任务后继续等待新的任务。而用户可以通过提交任务到队列进行排队，等待任务被执行。

这里涉及多个线程对队列的存取操作，修改循环队列的首尾标识符与写入或拷贝任务时进入了关键区域，需要保证各个线程的操作互斥，因此引入互斥锁 mutex。另外队列存在容量的上限，为满时不可写入，以及为空时不可读取，这里用信号量 full 与 empty 进行标识

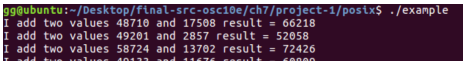
入队时，申请 full 信号量成功后使用 mutex 进行互斥操作，完成入队后释放 mutex 并增加 empty 信号量。

出队时，申请 empty 信号量成功后使用 mutex 进行互斥操作，完成入队后释放 mutex 并增加 full 信号量。

如此便可以保证各个线程对队列读写的互斥，避免 race condition

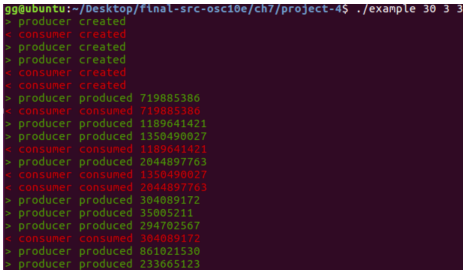
1.2 client.c

我们通过随机数生成一些加数，提交加法任务到线程池，观察结果如下，说明工作正常



2 任务二：解决生产者消费者问题

同任务一，生产者消费者问题主要考虑的也是队列的满、空、互斥问题，同样使用 full、empty、mutex 即可解决生产者消费者问题。以下是运行结果，可见每个产生的随机数都被正确且及时地消耗并打印了



3 任务三：线程池大小问题

若线程池的核心线程数设置过小，无法发挥多核处理器的能力，部分核会有闲置时间，效率将会降低

若线程池核心线程数设置过大，将会需要在一个核上进行频繁的上下文切换，以分时地实现并发，会有大量时间损耗在上下文切换上，且响应时间将会延长

在 CPU 密集型的程序中，最好设置核心线程数为 CPU 核心数的 1-2 倍，因为每个线程总是在几乎满负荷地使用对应核，控制在 1-2 倍有利于提升效率且降低响应时间

而在 I/O 密集型地程序中，核心线程数可以进一步提升，因为 I/O 中断将会使得每个线程频繁地中断，此时需要上下文切换为其他的线程，较多的线程数有利于避免所有线程均在等待中断而使得 CPU 闲置。据说有一个公式:

$$\text{核心线程数} = \text{CPU 核心数} * 1 + \text{平均等待时间}/\text{平均工作时间}$$