Thread Pool：

为了验证线程池的功能，修改client.c的main函数，将多个线程设置并用pthread\_submit()函数加入到线程池中。

实现poolthread.c

Enqueue()函数的功能是将任务加入到等待队列中，需判断队列是否已满，若已满则报错，并且返回1；未满则将任务加到队列尾端即可，并返回0。需注意的是，为了防止队列读写的冲突，需用互斥锁避免竞争关系。本实验中以数组作为队列。

Dequeue()函数的功能是将任务从队列中取出，也需要判断队列是否为空，若为空则报错；若不为空，则从队列头部取出任务返回，由于本实验中任务队列是由数组实现，故取出头部任务后，所有的任务需向前移动一位。同样，需用互斥锁避免竞争。

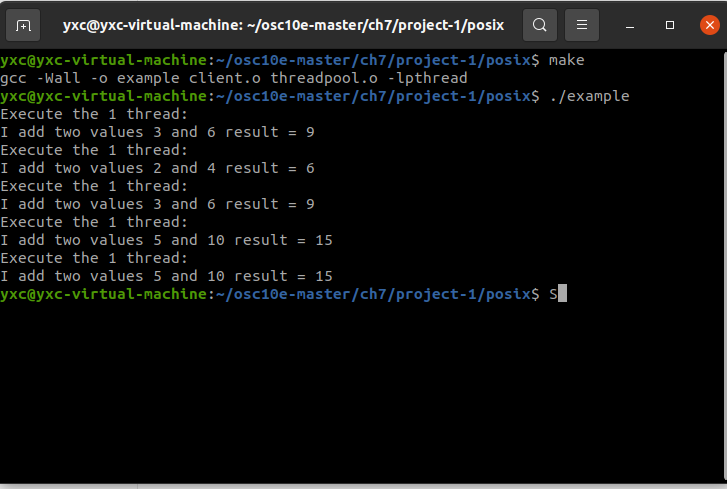
Worker()函数的功能即为线程的执行，需用dequeue()函数将线程从队列中取出，然后调用execute()函数运行指定函数，运行完成后，将工作队列中对应的id位置置零（表示可用），并增加信号量，终止线程。

Pool\_submit()函数的功能，即是将给定的任务添加到等待队列中，这用enqueue()函数实现即可；还将重复检索可以工作的任务（检索前需等待信号量），即对应id的工作队列位置为0，将其置1后，创建线程调用worker执行线程。

Pool\_init()函数的功能，即初始化线程池，那么将等待队列和工作队列清空置零，并调用sem\_init()函数和pthread\_mutex\_init()函数，分别将信号量和互斥锁初始化即可。

Pool\_shutdown()函数的功能即取消所用的工作线程，调用pthread\_join()函数等待每个线程终止，然后调用sem\_destroy()函数和pthread\_mutex\_destroy()函数销毁信号量和互斥锁。、

演示的结果如图所示。



Producer-consumer：

在buffer.h文件中定义任务队列，并用一个waiting数组表示队列中对应的位置是否为空，waiting[i]=1表示i位置为空，可插入，不可移除；waiting[i]=0表示i位置为非空，不可插入，可移除；

因此，在insert\_item()函数和remove\_item()函数中（在buffer.h中定义），在执行插入和移除操作前，需检查对应位置的waiting[i]后进行操作，并改变waiting[i]的值。

在main.c文件中，首先定义producer()函数和remove()函数作为生产者和消费者，重复着睡眠一段随机时间（有rand()函数生成）后，执行生产和消费的动作。

生产动作即随机生成一个数，而后调用insert\_item()函数将其插入到任务队列中，并将其打印到工作台。

消费动作即调用remove\_item()函数，从任务队列中取出一个数，并将其打印到工作台。

由于生产动作和消费动作出于计算缓冲区中，因此根据书中7.1.1节中所述，用两个信号量：full和empty 来计算缓冲区中空槽和满槽的数量，以及用一个互斥锁mutex来确保读写不发生竞争。

有一点意外的是，根据书本指导，本应将错误情况（即已满时插入或已空时移除）以fprintf()函数打印，但是在编译时该处一直不通过，在查阅资料并按照资料指导编写后仍然无法通过，故只能暂且用printf()函数输出错误了。

具体代码见main.c文件。

在main函数中，即按照书本指导进行6个步骤。

1. 从控制台读取main所需要的参数，即main函数睡眠时间sleep\_time、将创建的生产者线程数num\_producer、将创建的消费者线程数num\_consumer。
2. 初始化buffer，即调用pthread\_mutex\_init()函数初始化互斥锁，调用sem\_init()函数初始化full和empty,以及初始化任务列表（将所有位置置空）
3. 创建num\_producer个生产者线程。
4. 创建num\_consumer个消费者线程。
5. 令main函数睡眠sleep\_time的时间。
6. 用pthread\_cancel()函数取消所有生产者和消费者线程。用sem\_destroy()和pthread\_mutex\_destroy()函数销毁两个信号量和互斥锁，结束整个函数的运行。

运行结果如下