Project 5-1: Designing a Thread Pool

白骐硕 518030910102

任务概述

- 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 may be completed using either Pthreds and POSIX synchronization or Java.

具体实现

根据所下载的源码框架，在本实验中需要编写的内容有：

- threadpool.c
  - enqueue()
  - dequeue()
  - worker()
  - pool_submit()
  - pool_init()
  - pool_shutdown
- client.c
  - main()

threadpool.c

变量定义

为了实现threadpool.c中的几个函数，需要定义如下几个全局变量：

```
// the work queue
task worktodoQueue[QUEUE_SIZE+1]; //end is always empty
int queueBegin = 0, queueEnd = 0;

// the worker bee
pthread_t bees[NUMBER_OF_THREADS];

//mutex for enqueue and dequeue
pthread_mutex_t lock;
//semaphore
sem_t taskNum;
```

其中，共包含四个部分的变量：work queue，worker bee（线程数组），mutex，semaphore。

enqueue()

进队操作的逻辑如下：

　首先将mutex加锁，然后判断队列是否为满，如果满了，则释放mutex且返回1。如果不满，则将输入参数的 task进队，更新队列queueEnd（尾指针）的值，之后释放mutex且返回0。

队列的实现细节如下：

- 使用两个整型变量queueBegin，queueEnd来记录队列头尾的位置。queueEnd永远指向下一个空位置。
- queueBegin == queueEnd时队列为空
- (queueEnd+1)%QUEUE_SIZE == queueBegin 时队列为满

代码如下：

```
// insert a task into the queue
// returns 0 if successful or 1 otherwise,
int enqueue(task t)
{
    pthread_mutex_lock(&lock);
    if((queueEnd + 1)%QUEUE_SIZE == queueBegin){
        //full
        pthread_mutex_unlock(&lock);
        return 1;
    }else{
        worktodoQueue[queueEnd] = t;
        queueEnd = (queueEnd + 1)%QUEUE_SIZE;
        pthread_mutex_unlock(&lock);
        return 0;
    }
    return 0;
}
```

dequeue()

出队的执行逻辑为：

- 首先将mutex加锁
- 之后取出queueBegin指向的task，赋值给临时变量
- 更新queueBegin的值
- 释放锁，并返回刚才取到的task

在这里需要特殊说明的是，我们不需要在出队时进行判空的操作。因为在worker调用dequeue()之前会先等待信号量，保证了每次调用dequeue()时，队列不可能为空。

```
// remove a task from the queue
task dequeue()
{
    pthread_mutex_lock(&lock);
    task tmp = worktodoQueue[queueBegin];
    queueBegin = (queueBegin+1)%QUEUE_SIZE;
    pthread_mutex_unlock(&lock);
    return tmp;
}
```

worker()

worker为线程池线程的工作函数，其中包含了一个死循环的执行函数，只要线程池被创建且尚未被销毁时，所有的线程都一直执行该循环。在该循环体内，首先等待信号量（用于查看当前是否有可以执行的task），如果有则调用dequeue()取出一个任务task，调用execute()执行task，执行完毕后则进入下一次循环，寻找下一个task去执行。

```c
// the worker thread in the thread pool
void *worker(void *param)
{
    task tmp;
    while(TRUE){
        //execute the task
        sem_wait(&taskNum);
        tmp = dequeue();
        execute(tmp.function, tmp.data);
    }
    //pthread_exit(0);
}
```

pool_submit()

该函数中的执行逻辑为：

- 首先使用输入参数的somefunction和p，来构建一task结构体的实例
- 之后将该task进队
- 根据进队函数返回的结果判断是否成功进队：
    - 如果成功，则调用sem_post(&taskNum)增加信号量，且返回0
    - 如果不成功，则直接返回1

```c
/**
 * Submits work to the pool.
 */
int pool_submit(void (*somefunction)(void *p), void *p)
{
    int flag;
    task newTask;
    newTask.function = somefunction;
    newTask.data = p;
    flag = enqueue(newTask);
    if(!flag){
        //enqueue success
        sem_post(&taskNum);
    }
    return flag;
}
```

pool_init()

在线程池的初始化中需要做三件事：

- 初始化mutex
- 初始化semaphore
- 创建相应数量的thread

```
// initialize the thread pool
void pool_init(void)
{
    //mutex
    pthread_mutex_init(&lock,NULL);
    //semaphore
    sem_init(&taskNum,0,0);
    //thread create
    for(int i=0;i<NUMBER_OF_THREADS;++i){
        pthread_create(&bees[i],NULL,worker,NULL);
    }
}
```

pool_shutdown

在该函数中需要做的三件事恰好与pool_init()函数相对应：

- thread cancel
- destroy semaphore
- destroy mutex

```
// shutdown the thread pool
void pool_shutdown(void)
{
    //thread cancel
    for(int i = 0; i<NUMBER_OF_THREADS;++i){
        pthread_cancel(bees[i]);
        pthread_join(bees[i], NULL);
    }
    //semaphore
    sem_destroy(&taskNum);
    //mutex
    pthread_mutex_destroy(&lock);
}
```

client.c

为了更好的测试所编写的线程池的功能完备性，我重新编写的 client.c 的main函数

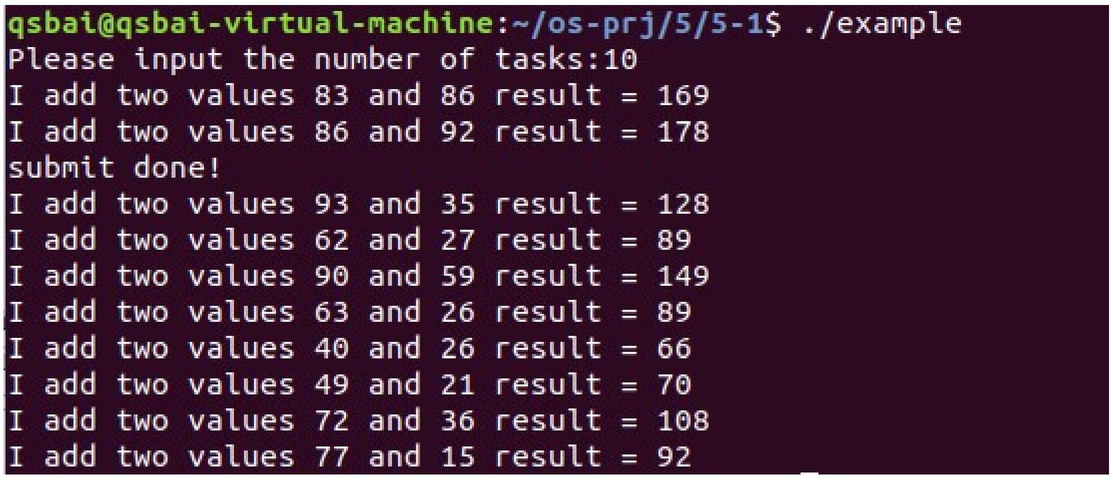
- 首先请求用户，输入任务个数（任务均为代码框架中的add()函数）
- 之后根据任务个数，用随机数随机生成相应个数的输入参数
- 初始化线程池
- 之后将所有的任务依次提交，如果提交不成功（pool_submit()返回1) 则重复提交至成功为止。
- 关闭线程池

```
int main(void)
{
    int n = 0;
    while(n <= 0){
        printf("Please input the number of tasks:");
        scanf("%d",&n);
    }
    // create some work to do
    struct data *work = malloc(n * sizeof(struct data));
    //initialize a and b
    for(int i = 0 ;i<n;++i){
```

```c
        work[i].a = rand()%100;
        work[i].b = rand()%100;
    }

    // initialize the thread pool
    pool_init();

    // submit the work to the queue
    for(int i = 0; i<n;++i){
        while(pool_submit(&add,&work[i]) != 0);
    }

    printf("submit done!\n");
    // may be helpful
    //sleep(3);

    pool_shutdown();
    free(work);

    return 0;
}
```

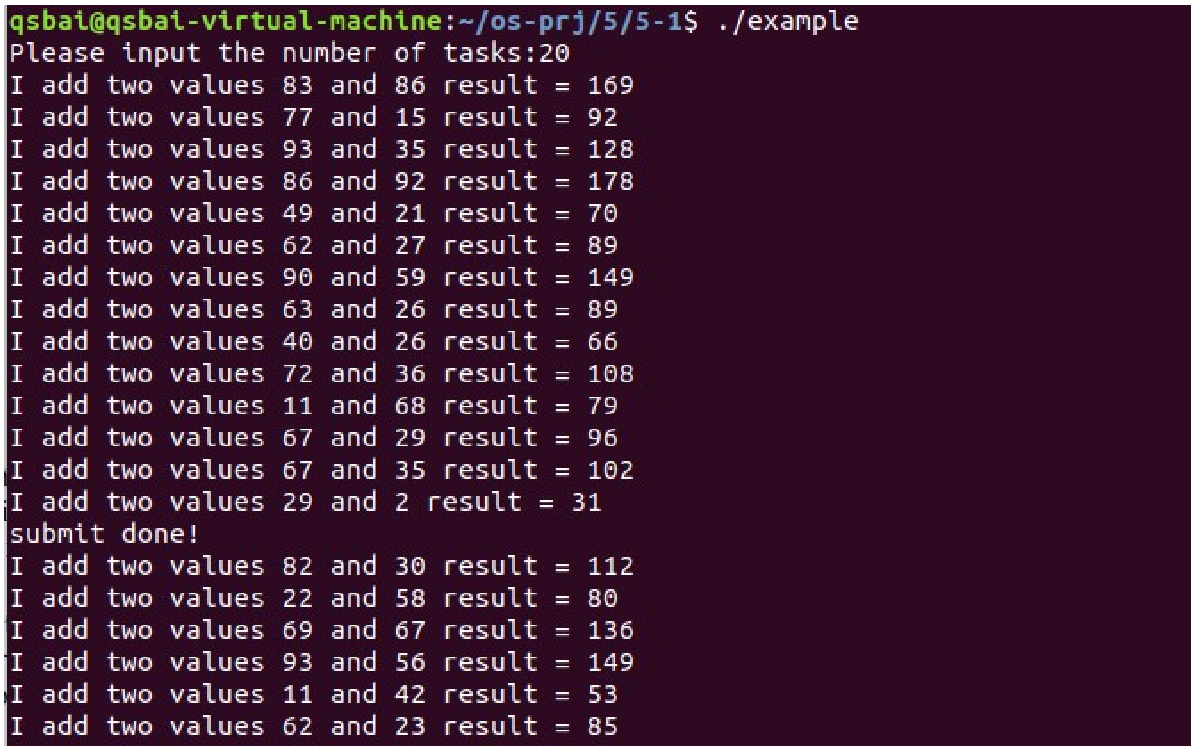
实验结果

输入任务个数为10个，结果如下：



```
qsbai@qsbai-virtual-machine:~/os-prj/5/5-1$ ./example
Please input the number of tasks:10
I add two values 83 and 86 result = 169
I add two values 86 and 92 result = 178
submit done!
I add two values 93 and 35 result = 128
I add two values 62 and 27 result = 89
I add two values 90 and 59 result = 149
I add two values 63 and 26 result = 89
I add two values 40 and 26 result = 66
I add two values 49 and 21 result = 70
I add two values 72 and 36 result = 108
I add two values 77 and 15 result = 92
```

输入任务个数为20个，结果如下：



```
qsbai@qsbai-virtual-machine:~/os-prj/5/5-1$ ./example
Please input the number of tasks:20
I add two values 83 and 86 result = 169
I add two values 77 and 15 result = 92
I add two values 93 and 35 result = 128
I add two values 86 and 92 result = 178
I add two values 49 and 21 result = 70
I add two values 62 and 27 result = 89
I add two values 90 and 59 result = 149
I add two values 63 and 26 result = 89
I add two values 40 and 26 result = 66
I add two values 72 and 36 result = 108
I add two values 11 and 68 result = 79
I add two values 67 and 29 result = 96
I add two values 67 and 35 result = 102
I add two values 29 and 2 result = 31
submit done!
I add two values 82 and 30 result = 112
I add two values 22 and 58 result = 80
I add two values 69 and 67 result = 136
I add two values 93 and 56 result = 149
I add two values 11 and 42 result = 53
I add two values 62 and 23 result = 85
```