操作系统(D)

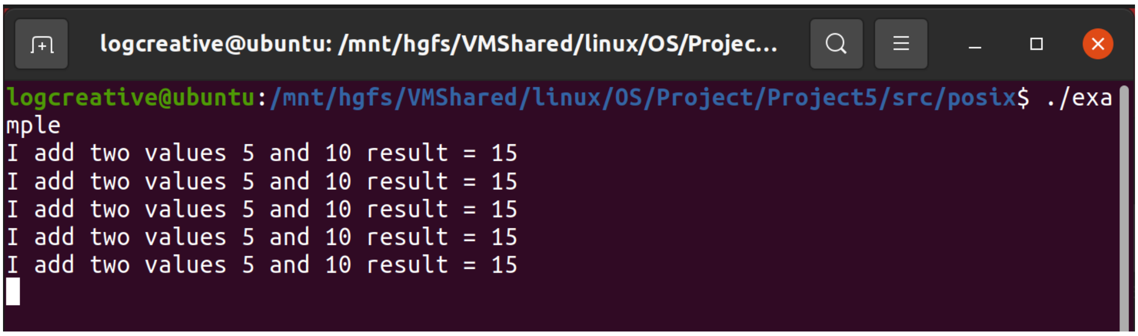
项目 5

李子龙 518070910095

2021 年 3 月 27 日

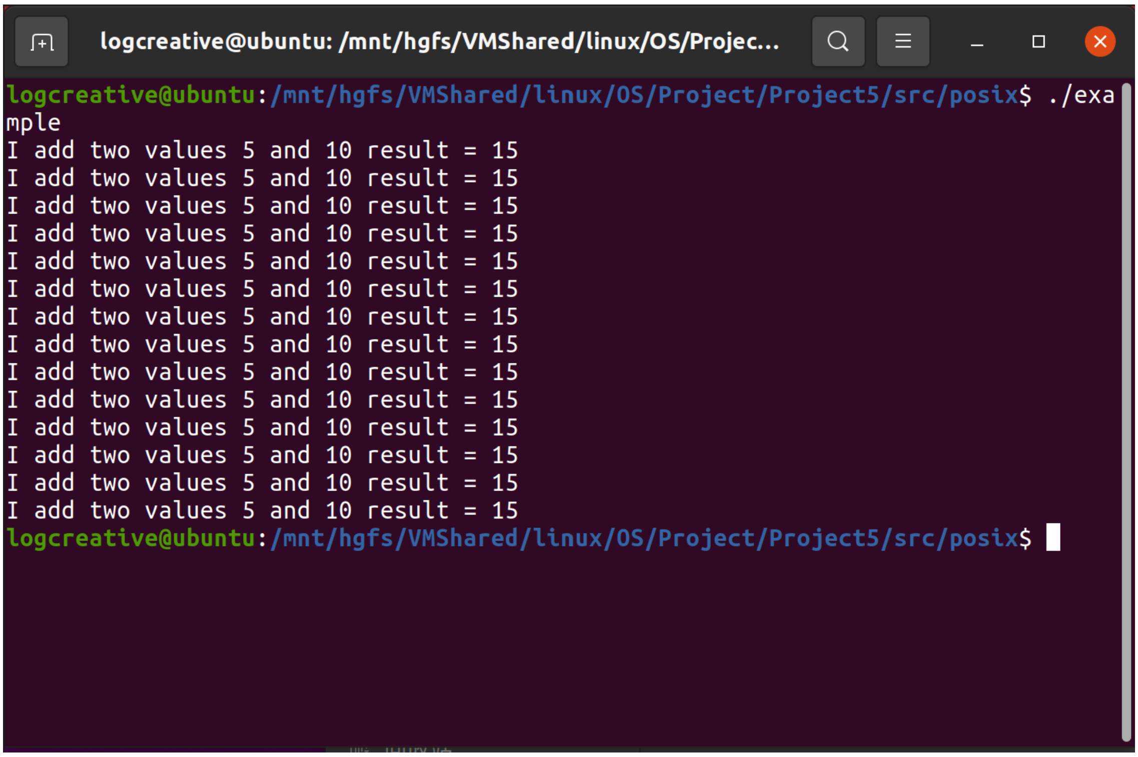
一 设计线程池

开始时刻，输入5个线程，线程池处理这5个线程。



```
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/OS/Project/Project5/src/posix$ ./exa
mple
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
```

后来，输入10个线程，但是线程池上限为9个线程，所以又只处理了9个线程。



```
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/OS/Project/Project5/src/posix$ ./exa
mple
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
I add two values 5 and 10 result = 15
logcreative@ubuntu:/mnt/hgfs/VMShared/linux/OS/Project/Project5/src/posix$ 
```

1. pool_init() 中初始化一个互斥锁和一个信号量。首先全局定义了两者后，再初始化 NUMBER_OF_THREADS = 3 个 worker 线程。

```c
// mutex
pthread_mutex_t queue_mutex;

// semaphore
sem_t sem_submit;

// initialize the thread pool
void pool_init(void)
{
    // mutual-exclusion locks
    pthread_mutex_init(&queue_mutex, NULL);

    // semaphores
    sem_init(&sem_submit, 0, 0);

    for(int i = 0; i < NUMBER_OF_THREADS; ++i)
        pthread_create(&bee[i],NULL,worker,NULL);
}
```

2. pool_submit() 需要使用队列存储任务。这里采用了循环队列。注意循环队列的容量是数据总量 - 1，也就是最多有 9 个任务可以在线程池中。

```c
// the work queue
task workqueue[QUEUE_SIZE];

int front = 0, rear = 0;

// insert a task into the queue
// returns 0 if successful or 1 otherwise,
int enqueue(task t)
{

    pthread_mutex_lock(&queue_mutex);

    int res = 0;
    if((rear + 1) % QUEUE_SIZE == front) res = 1;
    else {
        rear = (rear + 1) % QUEUE_SIZE;
        workqueue[rear] = t;
    }

    pthread_mutex_unlock(&queue_mutex);

    return res;
}

// remove a task from the queue
task dequeue()
{

    pthread_mutex_lock(&queue_mutex);

    front = (front + 1) % QUEUE_SIZE;
    task taskfront = workqueue[front];

    pthread_mutex_unlock(&queue_mutex);

    return taskfront;
}
```

3. worker() 进程，根据线程池的定义，一旦有可用进程就会从队列中弹出一个进程执行，并将需要服务的请求传递给它。一旦线程完成了服务，它会返回到池中再等待操作。如果池内没有可用线程，那么会等待，直到有空线程为止。这里使用一个信号量管理临界区入口。

```
// the worker thread in the thread pool
void *worker(void *param)
{
    while(TRUE){
        sem_wait(&sem_submit);
        // execute the task
        task worktodo = dequeue();
        execute(worktodo.function, worktodo.data);
    }

    pthread_exit(0);
}
```

4. 为了防止对队列的同时操作，设置了相关互斥锁，在第 2. 点可见使用了 queue_mutex 进行管理。

5. pool_shutdown() 会首先对每一个线程进行线程撤销，最后进行线程合并。信号量是一个线程撤销点。

```
// shutdown the thread pool
void pool_shutdown(void)
{
    for(int i = 0; i < NUMBER_OF_THREADS; ++i)
        pthread_cancel(bee[i]);
    for(int i = 0; i < NUMBER_OF_THREADS; ++i)
        pthread_join(bee[i],NULL);
}
```

二　生产者–消费者问题