计算机系统体系结构 Project5

517030910326 王孝诚

2019.11.17

实验环境

Windows 10 下使用 VMWare Workstation 15 Player 创建和运行虚拟机，虚拟机环境是 Linux 发行版 Ubuntu16.04.6 LTS。

1 线程池

这部分要实现一个线程池，提供给用户几个 API：1）void pool_init(),2）int pool_submit(void (*somefunction)(void *p), void *p)，3）void pool_shutdown(void)。按照书上的描述，实现各自功能，并使用 semaphore 和 mutex 来实现同步和避免 race condition。

1.1 设计思路

1. **client.c.** 需要对原始代码进行一些修改，原本只调用了一次 pool_submit()，这里需要添加多个任务到线程池中来展示其功能。
2. **threadpool.c.** 这里按照用户调用的顺序说明其需要实现的功能。
   (a) 需要一个队列来保存提交的任务，这里使用数组实现。
   (b) 首先用户初始化线程池，需要初始化信号量（这里使用了命名的信号量）、互斥锁以及创建特定数量个线程。创建的线程传入 worker 函数，worker 在线程的生命周期中一直循环，每次循环等待表示有任务可执行的信号量，然后执行一次任务。这里取出任务的 dequeue 操作需要加锁，防止多个线程同时更改。
   (c) 接下来用户多次提交任务，在添加任务到队列的 enqueue 操作中，要对队列加锁，防止多个线程同时对其更改；若提交成功了（队列有位置可以添加）则调用 sem_post 以通知等待的线程有任务可取。然后 worker 会立即取出任务并执行。
   (d) 最后用户要关闭线程池，对每个线程调用 pthread_cancel。

1.2 核心代码解释

1. 声明的全局变量

```
// the work queue
task worktodo[QUEUE_SIZE];

// the worker bee
pthread_t bee[NUMBER_OF_THREADS];

int numTask = 0;
pthread_mutex_t mutex;
sem_t *sem;
```

2. 初始化线程池

```c
// initialize the thread pool
void pool_init(void)
{
    sem = sem_open("SEM", O_CREAT, 0666, 0);
    pthread_mutex_init(&mutex, NULL);
    for (int i=0; i<NUMBER_OF_THREADS; ++i){
        pthread_create(&bee[i],NULL,worker,NULL);
    }
}
```

3. 入队列操作 enqueue

```c
// insert a task into the queue
// returns 0 if successful or 1 otherwise,
int enqueue(task t)
{
    pthread_mutex_lock(&mutex);
    if(numTask >= QUEUE_SIZE){
        pthread_mutex_unlock(&mutex);
        return 1;
    }
    worktodo[numTask++]=t;
    pthread_mutex_unlock(&mutex);
    return 0;
}
```

4. 出队列操作 dequeue

```c
// remove a task from the queue
task *dequeue()
{
    pthread_mutex_lock(&mutex);
    if(numTask<=0){
        pthread_mutex_unlock(&mutex);
        return NULL;
    }
    task *next_work = &worktodo[--numTask];
    pthread_mutex_unlock(&mutex);
    return next_work;
}
```

5. 工作线程

```c
// the worker thread in the thread pool
void *worker(void *param)
{
    // execute the task
    while(TRUE){
        sem_wait(sem);
```

```
        task *newTask = dequeue();
        if (newTask == NULL) continue;
        execute(newTask->function, newTask->data);
    }
    pthread_exit(0);
}
```

6. 提交任务

```
int pool_submit(void (*somefunction)(void *p), void *p)
{
    task newTask;
    newTask.function = somefunction;
    newTask.data = p;
    int res = enqueue(newTask);
    if(!res) sem_post(sem);
    return res;
}
```

7. 关闭线程池

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

2 生产者-消费者问题

这里要使用两个信号量 empty 和 full 以及一个互斥锁 mutex 来解决生产者消费者问题。

2.1 设计思路

主函数的内容：首先获得命令行输入的 sleep time，生产者数量，消费者数量；然后初始化 buffer；然后分别创建若干个生产者线程和消费者线程，并让他们运行生产者/消费者的操作；然后主进程 sleep 输入的时间后终止所有线程。

对于生产者线程，将一直循环等待 buffer 有无空位，即 sem_wait(&empty); 若有空，则加锁，然后向 buffer 插入一条 item；随后解锁，并且对 full 信号量 sem_post(&full)。对于消费者线程，则相反，一直等待 buffer 有无 item，即 sem_wait(&full); 然后取出一条 item，随后解锁、sem_post(&empty);。

另外还要实现 buffer 的 insert 和 remove 操作，这里用了数组实现的循环队列。

2.2 核心代码解释

1. 一些变量的声明。

```c
buffer_item buffer[BUFFER_SIZE]; //manipulated as a circular queue
int rear = 0;
int front = 0;
pthread_mutex_t mutex;
sem_t empty;
sem_t full;
```

2. 首先是 buffer 的插入和删除。

```c
int insert_item(buffer_item item) {
    /* insert item into buffer
    return 0 if successful, otherwise
    return -1 indicating an error condition */
    if((rear+1)%BUFFER_SIZE==front){
        return -1;
    }
    rear = (rear+1)%BUFFER_SIZE;
    buffer[rear] = item;
    return 0;
}

int remove_item(buffer_item *item) {
    /* remove an object from buffer
    placing it in item
    return 0 if successful, otherwise
    return -1 indicating an error condition */
    if(front==rear){
        return -1;
    }
    front = (front+1)%BUFFER_SIZE;
    item = &buffer[front];
    return 0;
}
```

3. 然后是生产者的工作函数。消费者类似，这里就不展示了。

```c
void *producer(void *param) {
    buffer_item item;
    while (1) {
        /* sleep for a random period of time */
        sleep(rand()%3);
        /* generate a random number */
        item = rand();
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        if (insert_item(item))
            printf("insert error condition\n");
        else
            printf("producer produced %d\n",item);
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
```

```
    }
}
```

4. 最后是主函数。注意 empty 初始值应为 BUFFER_SIZE-1，因为循环队列需要区别空和满两种情况的 front 和 rear 关系，有一个空存储单元不能使用。

```c
int main(int argc, char *argv[]) {
/* 1. Get command line arguments argv[1],argv[2],argv[3] */
    int sleep_time, producer_threads, consumer_threads;
    if(argc != 4)
    {
        fprintf(stderr, "Input form:<sleep time> <producer threads number> <consumer threads number>\n");
        return -1;
    }
    sleep_time = atoi(argv[1]);
    producer_threads = atoi(argv[2]);
    consumer_threads = atoi(argv[3]);

/* 2.  Initialize  buffer */
    pthread_mutex_init(&mutex, NULL);
    sem_init(&full, 0, 0);
    sem_init(&empty, 0, BUFFER_SIZE-1);

/* 3. Create producer thread(s) */
    pthread_t producer_ids[producer_threads];
    for(int i=0;i<producer_threads;++i){
        pthread_create(&producer_ids[i], NULL, &producer, NULL);
    }

/* 4. Create consumer thread(s) */
    pthread_t consumer_ids[consumer_threads];
    for(int i=0;i<consumer_threads;++i){
        pthread_create(&consumer_ids[i], NULL, &consumer, NULL);
    }

/* 5. Sleep */
    sleep(sleep_time);

/* 6. Exit */
    return 0;

}
```

3 实验结果

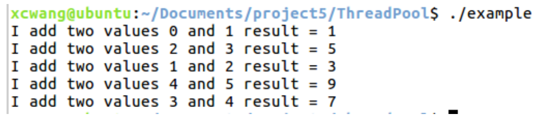

Figure 1: ThreadPool

Figure 2: Producer consumer