OS-Project5

常烁晨-521021910369

Task1：线程池

1、任务要求：使用Pthread API，补充 threadpool.c 和client.c，要求实现在 client.c 中定义的 add 任务的多线程运行。

2、任务实现：

（1）初始化：

首先定义线程池相关变量。本实验使用数组来模拟线程池以及任务队列。线程池中的线程用于执行任务队列中保存的任务，此处定义一个数组，用于标识线程池中的线程是否正在被占用。

此外定义信号量以及互斥锁，用于确保多线程并发时任务队列非负/输入输出过程互斥。

```
task worktodo;
task task_queue[QUEUE_SIZE];
int length = 0;
pthread_t threads_pool[NUMBER_OF_THREADS];
pthread_mutex_t mutex;
sem_t semphore;
int threads_flag[NUMBER_OF_THREADS];
```

（2）enqueue（）和dequeue（）函数实现：

首先检查当前队列是否已经占满，若否，则执行对应功能（如将任务添加到队尾/从头部删除）。注意通过互斥锁保证出入队的互斥。

```
        task_queue[length] = t;
        length++;
// or
        for(int i = 0; i < length - 1; i++)
            task_queue[i] = task_queue[i + 1];
```

（3）执行代码：

已有给定的 execute（）函数，只需在 worker（）函数中打印相关信息（如执行内容、执行线程），并在 exe（）执行前后操作互斥锁/信号量。

最后实现 submit（）函数，用于选择一个空闲的线程来执行任务队列中的下一个任务。init（）和shutdown（）函数只需要初始化/释放空间、销毁信号量。

3、执行结果：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch7/project-1/posix$ ./example
input numbers of works!
7
input two numbers for the NO.1 work:
1 1
input two numbers for the NO.2 work:
2 2
input two numbers for the NO.3 work:
3 3
input two numbers for the NO.4 work:
4 4
input two numbers for the NO.5 work:
5 5
input two numbers for the NO.6 work:
6 6
input two numbers for the NO.7 work:
7 7
The NO.0 thread is running...
I add two values 1 and 1, result = 2
The NO.0 thread is done

The NO.0 thread is running...
I add two values 2 and 2, result = 4
The NO.0 thread is done

The NO.0 thread is running...
I add two values 3 and 3, result = 6
The NO.0 thread is done

The NO.0 thread is running...
I add two values 4 and 4, result = 8
The NO.0 thread is done

The NO.0 thread is running...
I add two values 5 and 5, result = 10
The NO.0 thread is done

The NO.0 thread is running...
I add two values 6 and 6, result = 12
The NO.0 thread is done

The NO.0 thread is running...
I add two values 7 and 7, result = 14
The NO.0 thread is done
```

Task2：生产者-消费者问题

1、任务要求：与 Task1 类似，使用线程库模拟该问题执行结果。

2、任务实现：

（1）初始化：定义缓冲区、以及一个标识缓冲区占用情况的数组。

同时定义工具函数 insert_item（）和 remove_item（）用于操作缓冲区。

定义信号量 full、empty，互斥锁 mutex。

（2）定义生产者函数/消费者函数。生产者开始前调用sem_wait(&empty)，结束后调用 sem_wait(&full)，消费者相反。

（3）编写 main（）函数。首先通过命令行读入睡眠时间、生产者个数、消费者个数，之后初始化信号量及互斥锁。接下来使用线程库创建线程并执行生产者/消费者过程，同时打印相关信息到终端中。

经过一段时间睡眠后终止线程，最后销毁互斥锁信号量，结束函数。

代码内容比较简单，与前面的project类似，因此报告中限于篇幅，不再赘述。

3、执行结果：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch7/project-2$ gcc ex2.c -o ex2
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch7/project-2$ ./ex2 4 6 5
Start
Sleeping begin
The 1350490027 is produced
The 1350490027 is consumed
The 2044897763 is produced
The 1365180540 is produced
The 304089172 is produced
The 2044897763 is consumed
The 521595368 is produced
The 521595368 is consumed
The 1365180540 is consumed
The 861021530 is produced
The 861021530 is consumed
The 2145174067 is produced
The 2145174067 is consumed
The 1801979802 is produced
The 635723058 is produced
The 1801979802 is consumed
The 1059961393 is produced
The 1059961393 is consumed
The 1656478042 is produced
The 1653377373 is produced
The 1914544919 is produced
The 1656478042 is consumed
The 1734575198 is produced
The 1734575198 is consumed
The 2038664370 is produced
The 2038664370 is consumed
The 412776091 is produced
The 412776091 is consumed
The 749241873 is produced
The 749241873 is consumed
The 635723058 is consumed
The 135497281 is produced
The 2084420925 is produced
Sleeping end
Finish
```

Bonus：

核心数过大：（1）由于核心线程会消耗CPU、内存等资源，因此过大的核心数会导致系统开销过大。（2）线程过多，导致上下文切换开销增大。

核心数过小：（1）多线程规模小，线程数目少，没有足够的核心数用于完成任务，导致大量任务等待。（2）不能充分利用多核处理器的性能。

合理设置核心数：（1）根据任务密集程度，选择适当的核心数。（2）通过监控任务执行/提交的相关指标，改变核心数，寻找较好的核心数完成任务。（3）考虑设备性能、系统资源等，在确保系统正常运行时尽可能高效执行。