

第三章编程作业实验报告

191220156 计算机科学与技术系 张桓

算法并行化问题

实现非多线程版本

按照黄宇老师的《算法设计与分析》编写即可。

简单多线程版本

归并排序使用了递归的思想，将左、右区间分别变为有序后，再分别将两个有序区间合并为一个有序区间，时间复杂度为 $O(n\log n)$ ，代码如下：

```
1 void MergeSort(int *a, int p, int r){
2     q = (p + r) / 2;
3     MergeSort(a, p, q);
4     MergeSort(a, q + 1, r);
5     Merge(a, p, q, r);
6 }
```

因为左右区间的处理过程不会使用同一临界资源，所以可以将左右区间的处理过程**并行化**。在并行化处理，在合并左右区间前需要保证左右区间已经达成有序，所以在合并前使用`pthread_join`即可。而且由于参数较多，将`merge_sort`需要的参数先设置成一个结构体`thread_data`，向`pthread_create`传参的时候传的是`void(*)thread_data`。

实验结果

分别对两种实现方式计时，得到如下结果：

```
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ time ./merge_sort1
the array's size is: 5
2 1 3 5 4
the raw list is: 2 1 3 5 4
the sorted list is: 1 2 3 4 5

real    0m6.842s
user    0m0.001s
sys     0m0.000s
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ time ./p_merge_sort
the array's size is: 5
2 1 3 5 4
the raw list is: 2 1 3 5 4
the sorted list is: 1 2 3 4 5

real    0m3.337s
user    0m0.003s
sys     0m0.000s
```

可以看到对同一组数据，多线程处理花费的时间比串行处理花费的时间要节省一倍左右。

信号量与PV操作实现同步问题

实现逻辑

用PV操作实现时，按照书上的示例，我们假设**读者有多个**，我的实现是最多有5个读者，而**写者只有一个**。使用的就是`lock`和`w_or_r`这两个信号量。`w_or_r`使用来保护对共享资源即文件`file`的访问，在`func_write`中对`share_data`的写操作之前，`sem_wait(w_or_r)`即信号量的P操作，操作完成之后`sem_post`。在读线程`func_read`中，用`reader_num`记录并发的读线程数量，对这个变量的操作用`lock`这个信号量来互斥（在并发的读线程之间互斥）。因为实现的是**读者优先**，第一个读线程发起读操作时（`reader_num == 1`）才会去`sem_wait(w_or_r)`，而且只有当所有的读线程都结束时（`reader_num == 0`），才会释放`w_or_r`，使得写线程可以修改。

对于文件`file`的读取和修改我的操作是在读线程中用 `int tmp = share_data.file;` 来模拟读。在写线程中不断将`file`修改成 `share_data.file = modify;`，其中 `modify = 233` 是一个预先定好的变量。

为了使得该程序能够停止下来，统计写线程的执行次数`write_counter`，当到达10000时就设置`stop`为1终止程序，同时也记录了读线程的执行次数`read_counter`，在程序结束时输出这两个数字。

实验结果

```
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ gcc 2PV.c -o 2PV -lpthread
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2PV
read thread's count is 50605
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2PV
read thread's count is 49509
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2PV
read thread's count is 53437
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2PV
read thread's count is 51185
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2PV
read thread's count is 51684
write thread's count is 10000
```

读者写者锁

实现逻辑

也是实现了5个读者和一个写者，每次读或者写时都需要为读写锁`rwlock`上锁才行，读进程上锁用 `pthread_rwlock_rdlock(&rwlock);`，写进程上锁用 `pthread_rwlock_wrlock(&rwlock);`，一次读或者写进程完成后释放锁 `pthread_rwlock_unlock(&rwlock);`，程序结束时销毁锁 `pthread_rwlock_unlock(&rwlock);`。

同样为统计各个进程执行的次数`write_counter`和`read_counter`，`write_counter`到达10000就结束程序，得到结果如下：

实验结果

```

njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ gcc 2mutex.c -o 2mutex -lpthread -w
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2mutex
read thread's count is 53346
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2mutex
read thread's count is 43888
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2mutex
read thread's count is 43171
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2mutex
read thread's count is 53046
write thread's count is 10000
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./2mutex
read thread's count is 45807
write thread's count is 10000

```

管程机制实现与管程应用问题

信号量与PV操作

实现逻辑

设置信号量`sem_t lock`; `lock` 用于对缓冲区的互斥操作。还有信号`not full`和`not empty`用来表示缓冲区非空和缓冲区非满的信号量。

测试时创造读写两个进程，设置缓冲区大小 $k = 10$ ，即 `CircleBuffer A(10)`;，写进程向其中写入 `0~14`，最后写入 `-1`，读进程读缓冲区，读到 `-1` 后结束程序，打印读写进程的运行情况。

实验结果

```

njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ gcc 3mutex.cpp -o 3mutex -lpthread -w -lstdc++
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./3PV
0 --->
--->0
1 --->
--->1
2 --->
--->2
3 --->
--->3
4 --->
--->4
5 --->
--->5
6 --->
--->6
7 --->
--->7
8 --->
--->8
9 --->
--->9
10 --->
--->10
11 --->
--->11
12 --->
--->12
13 --->
--->13
14 --->
--->14

```

互斥信号量和条件变量

实现逻辑

设置互斥体 `pthread_mutex_t lock;` 用于对缓冲区的互斥操作。条件变量 `pthread_cond_t notempty;` 和 `pthread_cond_t notfull` , 分别为用于指示缓冲区非空和未满的条件。

测试方法和上面一样, 打印读写进程的运行情况。

实验结果

```
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./3mutex
0 --->
--->0
1 --->
2 --->
3 --->
--->1
4 --->
--->2
5 --->
6 --->
--->3
7 --->
8 --->
--->4
9 --->
10 --->
--->5
11 --->
12 --->
--->6
13 --->
--->7
14 --->
--->8
--->9
--->10
--->11
--->12
--->13
--->14
```

死锁问题

实现逻辑

实现按照课件给出的PV操作不当导致死锁的例子:

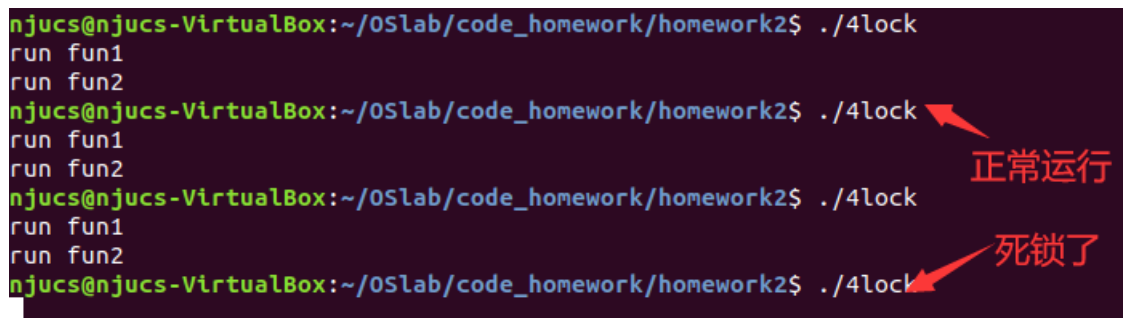
进程P	进程Q	产生死锁的 执行序列
①请求读卡机	①请求打印机	P①
②请求打印机	②请求读卡机	Q①
...	...	P②
③释放读卡机	③释放读卡机	Q②
④释放打印机	④释放打印机	

用信号量 `S1` 代表读卡机资源, `S2` 代表读卡机资源, 初始值都为1, 令 `fun1` 先 `sem_wait(&S1);` 再 `sem_wait(&S2);` , `fun2` 先 `sem_wait(&S2);` 再 `sem_wait(&S1);` 如果某个进程成功运行就会输出 `funi run` 的字样.

实验结果

由于死锁只能发生在特定指令序列下，即 $P_1 \ Q_1 \ P_2 \ Q_2$ ，其余情况下不会，多次尝试后出现死锁的情况，什么都不输出，也不退出程序：

```
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./4lock
run fun1
run fun2
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./4lock
run fun1
run fun2
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./4lock
run fun1
run fun2
njucs@njucs-VirtualBox:~/OSlab/code_homework/homework2$ ./4lock
```



正常运行

死锁了

参考文章

[多线程实现缓冲区](#)

[pthread_create传递参数问题](#)

[读者写者问题](#)

[多线程教程](#)

[多线程实现归并排序](#)