OS编程作业2

姓名	学号	邮箱	院系
张洋彬	191220169	1016466918@qq.com	计算机科学与技术系

一、算法并行化问题

1、不考虑线程的话，算法的核心代码如下

```
void merge(int left, int right){
    int mid = (left + right) >> 1;
    int size1 = mid - left + 1;
    int size2 = right - mid;
    int t1[size1];
    int t2[size2];

    memcpy(t1, a+left, sizeof(int) * (mid-left+1));
    memcpy(t2, a+mid+1, sizeof(int) * (right-mid));

    int i = 0, j = 0;
    int k = left;
    while (i < size1 && j < size2) {
        if (t1[i] <= t2[j]) {
            a[k] = t1[i];
            i++;
        }
        else {
            a[k] = t2[j];
            j++;
        }
        k++;
    }


    while (i < size1) {
        a[k] = t1[i];
        k++;
        i++;
    }

    while (j < size2) {
        a[k] = t2[j];
        j++;
        k++;
    }
}
```

```
38
39
40  void* merge_sort(void* arg){
41      int *argu = (int*)arg;
42      int left = argu[0];
43      int right = argu[1];
44
45      int mid = (left + right) >> 1;
46      int arg1[2];
47      int arg2[2];
48
49
50      arg1[0] = left;
51      arg1[1] = mid;
52
53      arg2[0] = mid + 1;
54      arg2[1] = right;
55
56      if (left >= right) {
57          return NULL;
58      }
59      merge_sort(arg1);
60      merge_sort(arg2);
61
62      merge(left, right);
63      return NULL;
64  }
65
66
```

上述算法经测试，结果正确。

2、加入线程机制后，merge_sort的实现如下：

```
1   pthread_t t2;
2      pthread_t t1;
3
4      if (numofThread == maxThreadNumber) {
5          flag = 1;
6      }
7
8      if (numofThread < maxThreadNumber) {
9          numofThread += 1;
10         pthread_create(&t1, NULL, merge_sort, (void*)arg1);
11         pthread_join(t1, NULL);
12         pthread_exit(NULL);
13         numofThread -= 1;
14     }
15     else {
```

```
        merge_sort(arg1);
    }

    if (numofThread < maxThreadNumber) {
        numofThread += 1;
        pthread_create(&t2, NULL, merge_sort, arg2);
        pthread_join(t2, NULL);
        pthread_exit(NULL);
        numofThread -= 1;
    }
    else {
        merge_sort(arg2);
    }

    merge(left, right);
```

将数据设置为100个，线程的数量设置0，1，10，20，100，结果如下图所示

```
njucs@njucs-VirtualBox:~/pa2$ ./merge
The running time is 335 microsecond
njucs@njucs-VirtualBox:~/pa2$ gcc merge.c -o merge -pthread
njucs@njucs-VirtualBox:~/pa2$ ./merge
The number of thread that I use : 1
The running time is 225 microsecond
njucs@njucs-VirtualBox:~/pa2$ gcc merge.c -o merge -pthread
njucs@njucs-VirtualBox:~/pa2$ ./merge
The number of thread that I use : 10
The running time is 1480 microsecond
njucs@njucs-VirtualBox:~/pa2$ gcc merge.c -o merge -pthread
njucs@njucs-VirtualBox:~/pa2$ ./merge
The running time is 2292 microsecond
njucs@njucs-VirtualBox:~/pa2$ gcc merge.c -o merge -pthread
njucs@njucs-VirtualBox:~/pa2$ ./merge
The running time is 1809 microsecond
```

1000个数据，执行的次数为二叉树非叶子结点的个数，同时运行的线程个数应小于20，所以线程数量超过20时，将达不到最大线程数，线程的数量会导致开销增大，所以单线程的时间消耗较少，多线程消耗较多。

(Ps:由于每次运行结果都不一样，所以直接随机取值)

二、信号量与PV操作实现同步问题

实现的方式是写者优先，测试用例如下

```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 7 3
```

```
Create the 1 thread: Reader
Create the 2 thread: Writer
Create the 3 thread: Reader
Create the 4 thread: Reader
Create the 5 thread: Writer
Thread 1: waiting to read
Thread 1: start reading
Thread 2: waiting to write
Thread 3: waiting to read
Thread 4: waiting to read
Thread 5: waiting to write
Thread 1: end reading
Thread 2: start writing
Thread 2: end writing
Thread 5: start writing
Thread 5: end writing
Thread 3: start reading
Thread 4: start reading
Thread 3: end reading
Thread 4: end reading
```

由上图分析可得，虽然thread3和4先于5进入等待队列中，但是实现的方式是写者优先，5会在thread2写完后再去写，写操作是可以同步的，所以3、4一起运行读过程。

三、管程机制实现与管程应用问题

使用C++语言和pthread库实现一个有界环形缓冲区类(CircleBuffer), 该类提供一个带参(int K)构造方法，用于指定缓冲区的大小, 提供一个get和一个put方法分别用于从缓冲区取出一个元素和向缓冲区存入一个元素。另外需要编写一个多线程测试用例，测试该有界环形缓冲区类是否能够正确同步。具体实现要求：

1. 只能使用pthread库提供的一般信号量(semaphore)，参考课堂上介绍的使用信号量与PV操作实现Hoare类型管程，来实现CircleBuffer;
2. 使用pthread库提供的互斥信号量(mutex)和条件变量(condition)，实现CircleBuffer。

   没有去测试，代码如下：

```
#include <iostream>
#include <ctype.h>
#include <unistd.h>
#include <stdio.h>
#include <queue>
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <time.h>
#include <boost.h>
class mesa_monitor : boost::noncopyable {
public:
    typedef boost::unique_lock<mesa_monitor> lock_type;
    friend class lock_type;
    mesa_monitor() : m_notify(0) {}
public:
    void lock() const {
        m_mutex.lock();
        m_notify = 0;  // 进入管程时要把m_notify归0
    }
    void unlock() const {
        notify_impl(m_notify);
        m_mutex.unlock();
    }
    bool try_lock() const {
        bool ret = m_mutex.try_lock();
        if (ret) {
            m_notify = 0;
        }
        return ret;
    }
    void wait() const {
```

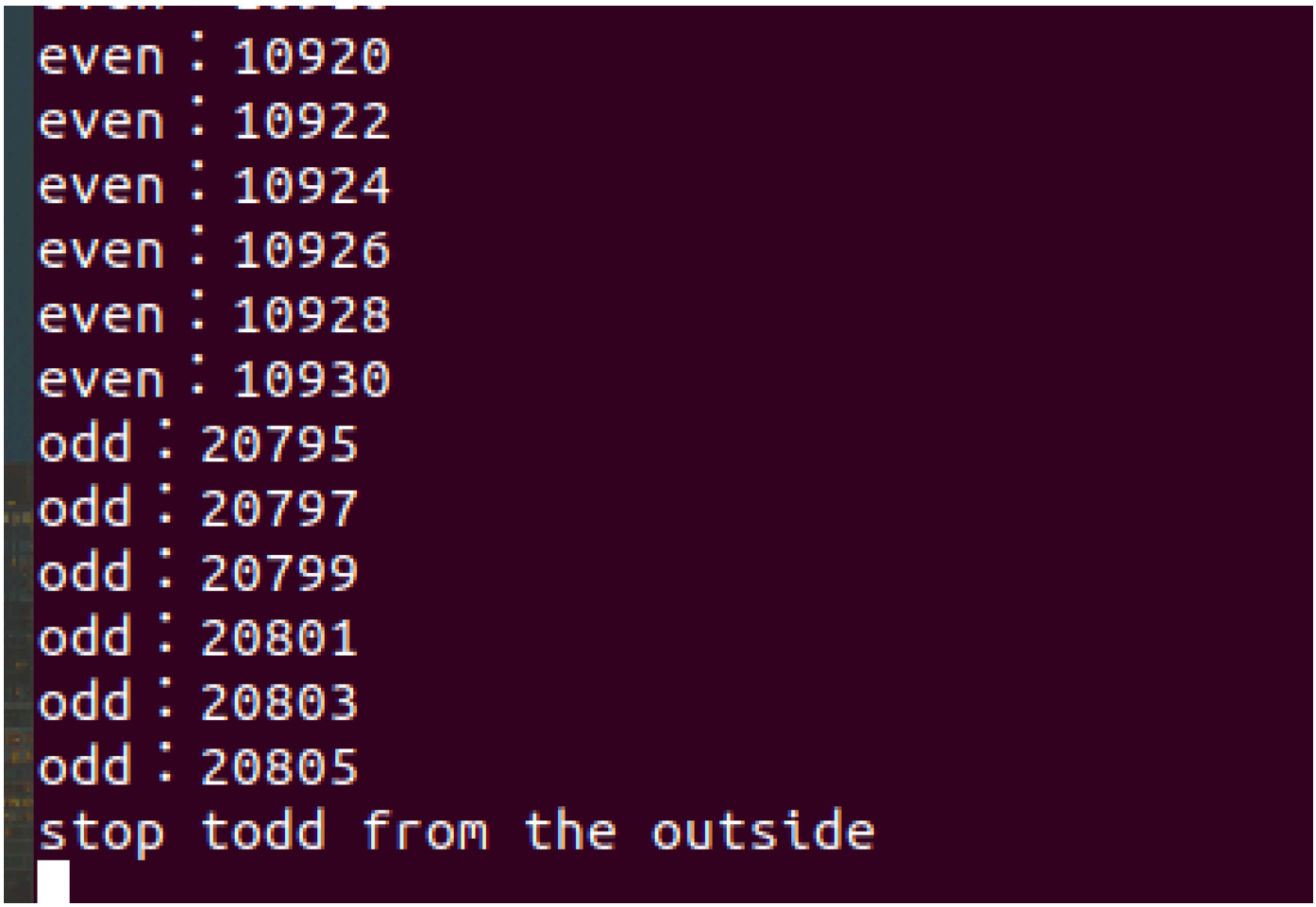
```
34          notify_impl(m_notify);
35          m_cond.wait(m_mutex);
36          m_notify = 0;
37      }
38      void notify_one() {
39          if (m_notify != -1) {
40              ++m_notify;
41          }
42      }
43
44      void notify_all() {
45          m_notify = -1;
46      }
47
48  private:
49      void notify_impl(int nnotify) const {
50          if (nnotify != 0) {
51              if (nnotify = -1) {
52                  m_cond.notify_all();
53                  return;
54              } else {
55                  while (nnotify > 0) {
56                      m_cond.notify_one();
57                      --nnotify;
58                  }
59              }
60          }
61      }
62  private:
63      mutable boost::condition_variable_any m_cond;
64      mutable boost::mutex m_mutex;
65      mutable int m_notify;
66  };
67
68  template <typename T>
69  class threadsafe_queue : mesa_monitor {
70      std::queue<T> m_data;
71  public:
72      threadsafe_queue() {}
73      void pop(T& val) {
74          mesa_monitor::lock_type lk(*this);
75          while (m_data.empty()) {
76              wait();
77          }
78          val = m_data.front();
79          m_data.pop();
80      }
81      void push(const T& val) {
82          mesa_monitor::lock_type lk(*this);
```

```
        m_data.push(val);
        notify_one();
    }
};

```

四、死锁问题

两个进程分别是打印奇数和偶数，当A线程进入锁定状态是，主线程突然异常将A线程停止，这时将导致B线程也无法继续执行，处于死锁状态，如下图所示：



贴一段进程创建的代码实现

```
pthread_t todd,teven;
        pthread_mutex_init(&m,0);
        pthread_create(&todd,0,runodd,0);
        pthread_create(&teven,0,runeven,0);
        sleep(5);
        printf("stop todd from the outside\n");
        pthread_cancel(todd);
        pthread_join(todd,(void**)0);
        pthread_join(teven,(void**)0);
        pthread_mutex_destroy(&m);
```