**操 作 系 统**

**实 验 报 告**

**实验名称：实验三 多线程程序实验**

**姓名： 王迎旭**

**学号： 16340226**

实验名称：**多线程程序实验**

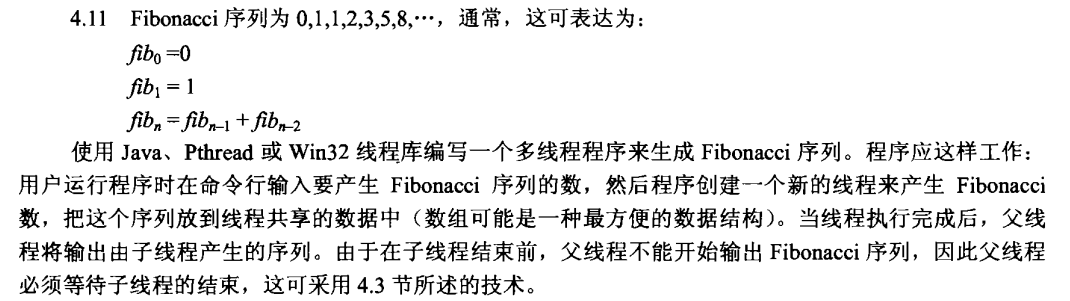
1. 实验目的：
2. 用线程生成Fibonacci数列
3. 完成多线程矩阵乘法程序设计
4. 实验要求：
5. 用pthread线程库，按照第四章习题4.11的要求生成并输出Fibonacci数列
6. 矩阵乘法

给定两个矩阵A和B，其中A是具有M行、K列的矩阵， B为K行、N列的矩阵， A和B的矩阵积为矩阵C， C为M行、N列。矩阵C中第i行、第j列的元素Cij就是矩阵A第i行每个元素和矩阵B第j列每个元素乘积的和，即



要求：每个Ci j的计算用一个独立的工作线程，因此它将会涉及生成M×N个工作线程。主线程(或称为父线程)将初始化矩阵A和B，并分配足够的内存给矩阵C，它将容纳矩阵A和B的积。这些矩阵将声明为全局数据，以使每个工作线程都能访问矩阵A、B和C。

1. 实验过程：
2. **用线程生成Fibonacci数列**



**1**.明确目标程序的需求：

① 使用线程完成斐波那契数列的设计

② 理清线程之间的关系

**2**.搜索题目所涉及相关资料：

**Ⅰ、线程问题相关函数介绍**

① pthread\_create()：创建线程。

② pthread\_join()：阻塞调用线程，直到threadid所指定的线程终止。  每个线程只能用pthread\_join()一次。若多次调用就会发生逻辑错误。

③ pthread\_exit()：终止调用线程。

④ pthread\_attr\_init ()：初始化线程属性为默认属性

⑤ pthread\_attr\_getscope ()：获得线程竞争范围

⑥ pthread\_attr\_setscope ()：设置线程竞争范围

**Ⅱ、pthread\_create()函数**

功能：

用于线程的创建

例：

int pthread\_create(pthread\_t \*restrict tid,  
 const pthread\_attr\_t \*restrict attr,  
 void \*(\*start\_rtn)(void),   
 void \*restrict arg)；

参数详解：

① 第一个参数为指向线程标识符的指针，用于输出线程标识符。

② 第二个参数用来设置线程属性。

③ 第三个参数是线程运行函数的起始地址。

④ 最后一个参数是运行函数的参数。

返回值：

线程创建成功返回0，否则返回错误编码

**注：**

**在编译时注意加上-lpthread参数，以调用静态链接库。因为pthread并非Linux系统的默认库。**

Ⅲ、pthread\_join()函数

功能:

这个函数是一个线程阻塞的函数，调用它的函数将一直等待到被等待的线程结束为止，当函数返回时，被等待线程的资源被收回

例：

int pthread\_join \_\_P (pthread\_t \_\_th,

void \*\*\_\_thread\_return);

参数详解：

① 第一个参数为被等待的线程标识符

　　 ② 第二个参数为一个用户定义的指针，它可以用来存储被等待 线程的返回值。

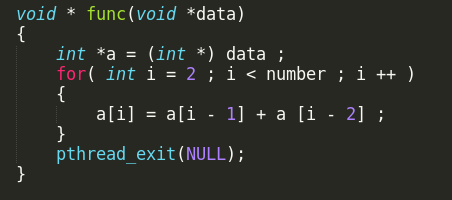
返回值：

成功调用返回0，否则返回错误编码

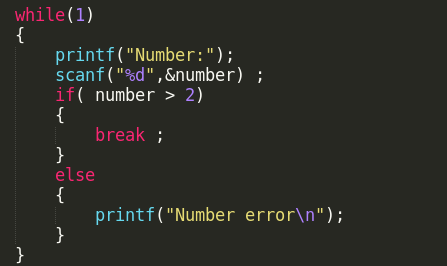
**3**.斐波拉契数列的程序设计与运行

**Ⅰ、针对斐波拉契数列数列理清设计思路**

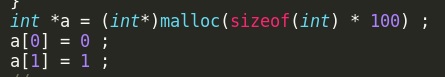
① 创建斐波那契数列求值函数



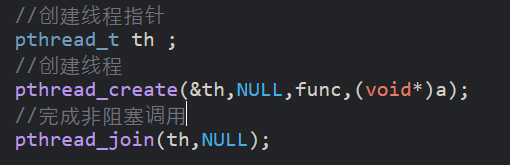
② 限制斐波那契数列项数必须大于2



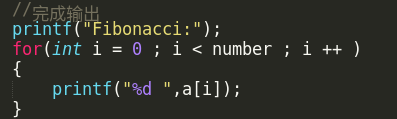
③ 为斐波那契数列动态分配空间



④ 创建线程并完成非阻塞调用

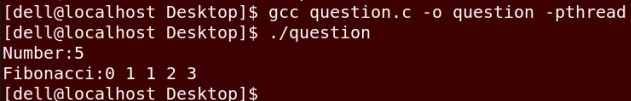


⑤ 完成输出

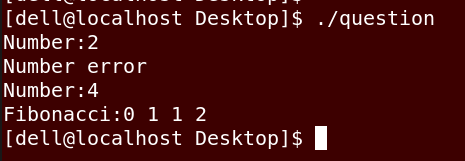


**Ⅱ、编译运行并输出结果**

① 正常编译运行



② 输入数字错误情况



分析：

1. 编译时加上-pthread指令，静态调用链接库，否则会出现编译错误的提示.
2. 创当输入的参数小于等于2的时候，程序提示错误，要求用户重新输入，直到输入的参数大于2.
3. **多线程矩阵乘法**

**1、明确目标程序的需求：**

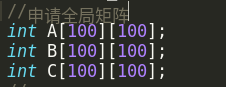
① 通过使用多线程，完成矩阵乘法的运算

② 主线程完成程序初始化功能，子线程完成乘法运算

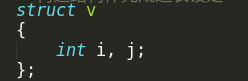
（**注：由于两个题之间均使用线程相关知识，（2）不再展示相关资料，直接完成程序的设计）**

1. **完成矩阵乘法的程序设计：**

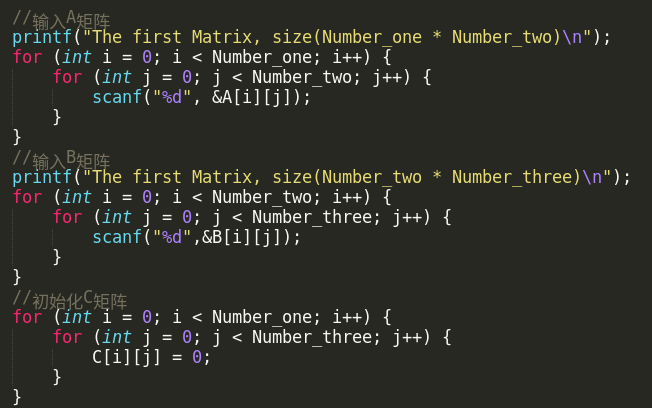
① 申请三个矩阵A,B,C最大规格为100\*100



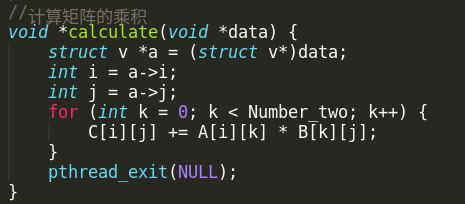
② 使用结构体记录某次需要计算C矩阵的参数位置



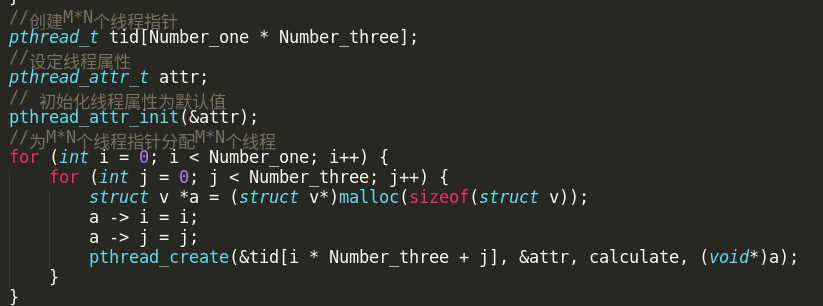
③ 完成A,B矩阵的数据输入，C的初始化



④ 设计矩阵乘法函数



⑤ 完成子线程的创建



**注：**

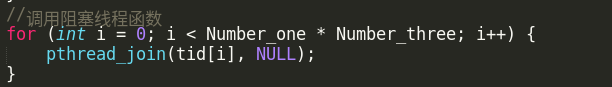
**[**

**这里由于需要使用子线程完成对乘法的运算，所以C矩阵（规格Number\_one \* Number\_three）需要Number\_one \* Number\_three个线程来完成相应的计算。**

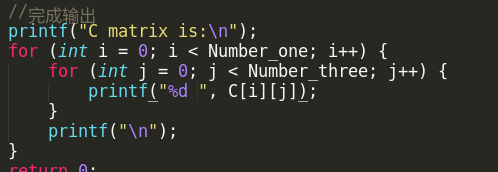
**使用结构体也方便我们找出需要计算的位置，以及需要调用的线程在数组中的位置。**

**]**

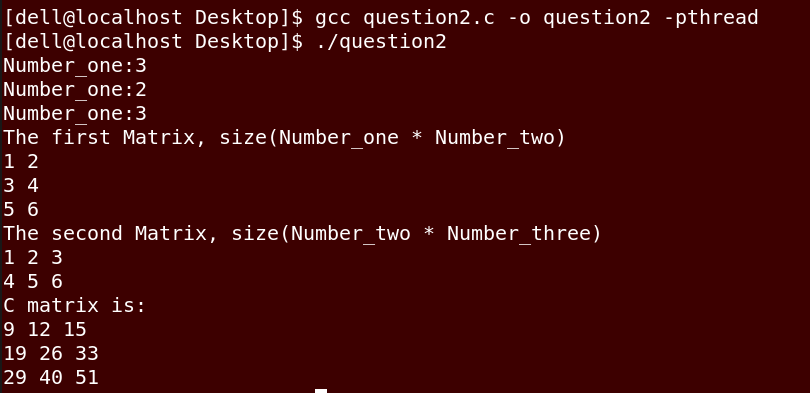
⑥ 非阻塞调用线程



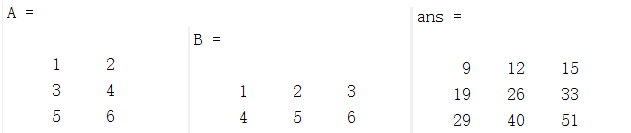
⑦ 输出计算结果



1. **编译与运行：**



**4.验证运算结果正确性：**



1. 实验心得：
2. 单线程序设计比较简单，只需要使用一次线程调用就可以解决问题，但是在完成任务之前，还是需要对线程函数进行学习，比如create线程不成功会出现在什么情况下以及如何判断线程的终止，同时也是对线程函数的参数进行了了解，随后完成了第一部分的任务。
3. 进行矩阵乘法程序设计，在单线程的程序设计基础上需要再多考虑几个问题，比如：

①、如何使用子线程指针调用多子线程进行计算

②、如何准确记录需要运算的位置

③、如何为线程指针分配对应的线程

等问题。为了处理多子线程指针的问题，我创建了一个一维数组用于对应C矩阵中的某个元素，对应格式为数组[i\*Number\_three + j];在设计这个程序的过程中，也遇到了段错误的问题，仔细研究之后，发现是访问目标矩阵越界；所以，在访问数组元素时候，一定要先判断自己的访问是否是非法的。

1. 同时我也看到了多线程程序的4个优点：响应度高、资源共享、经济、多处理器体系结构的利用。