实验二 多线程程序实验

陈志扬 15331046

一、实验目的

1. 进一步理解线程
2. 学习使用pthread线程库

二、实验运行环境

虚拟机VMware下的Ubuntu16.04系统

三、实验内容

1. 用线程生成Fibonacci数列
2. 多线程矩阵乘法

四、实验原理

线程使用说明——主要系统调用：
pthread_create():创建线程
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。
pthread_exit():终止调用线程
pthread_attr_init():初始化线程属性为默认属性
pthread_attr_getscope():获得线程竞争范围
pthread_attr_setscope():设置线程竞争范围

使用pthread的程序编译命令：
若程序文件是main.c
传统命令为：gcc main.c -o main -lpthread
现在命令为：gcc main.c -o main -pthread
差别：后一个会选用线程安全的库实现
若程序文件是main.cc
传统命令为：g++ main.cc -o main -lpthread
现在命令为：g++ main.cc -o main -pthread
差别：后一个会选用线程安全的库实现

五、实验过程

1. 用线程生成Fibonacci数列
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。

代码如下：

```
#include <iostream>
#include <pthread.h>

using namespace std;

int n;// the size of fibonacci array

void *fibonacci(void *data) {
    int *a = (int*)data;
    // calculate the fibonacci array
    for (int i = 2; i < n; i++) {
        a[i] = a[i - 1] + a[i - 2];
    }
    pthread_exit(NULL);
```

```
}

int main() {
    cout << "Please enter the number n(n>2):" << endl;
    cin >> n;
    while (n <= 2) {
        cout << "The number should be larger than 2." << endl;
        cout << "Please enter the number n(n>2):" << endl;
        cin >> n;
    }
    int a[1000];
    // initial a[0] and a[1]
    a[0] = 0;
    a[1] = 1;
    pthread_t th;
    // create a thread to calculate
    pthread_create(&th, NULL, fibonacci, (void*)a);
    // a thread to be joined upon
    pthread_join(th, NULL);

    cout << "Fibonacci:" << endl;
    // output the result
    for (int i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    return 0;
}
```

编译运行结果如下：
由于我限制了n的大小，只有当n大于2才输出Fibonacci数列。

```
linjiafengyang@ubuntu:~/Desktop$ g++ -g fibonacci.cpp -o fibonacci -pthread
linjiafengyang@ubuntu:~/Desktop$ ./fibonacci
Please enter the number n(n>2):
2
The number should be larger than 2.
Please enter the number n(n>2):
3
Fibonacci:
0 1 1
```

输入9：

```
Please enter the number n(n>2):
9
Fibonacci:
0 1 1 2 3 5 8 13 21
```

输入20：

```
Please enter the number n(n>2):
20
Fibonacci:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

根据上述实验结果可得出：程序正确，证明已实现用线程生成Fibonacci数列。

---

2. 多线程矩阵乘法

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列的矩阵。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

$$C_{i,j} = \sum_{n=1}^{K} A_{i,n} \times B_{n,j}$$

要求：每个Cij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。

代码如下：

```
#include <iostream>
#include <pthread.h>
#include <stdlib.h>

using namespace std;

int M, K, N;
// the size of matrix
```

```
int A[100][100];
int B[100][100];
int C[100][100];
// structure for passing data to threads
struct v
{
    int i, j;
};
// calculate the matrix product in C[row][col]
void *calculate(void *data) {
    struct v *a = (struct v*)data;
    int i = a->i;
    int j = a->j;
    for (int k = 0; k < K; k++) {
        C[i][j] += A[i][k] * B[k][j];
    }
    pthread_exit(NULL);
}

int main() {
    cout << "Please enter three numbers(M/K/N) that are less than 100:" << endl;
    cin >> M >> K >> N;
    cout << "Please enter the first matrix(M*K):" << endl;
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < K; j++) {
            cin >> A[i][j];
        }
    }
    cout << "Please enter the second matrix(K*N):" << endl;
    for (int i = 0; i < K; i++) {
        for (int j = 0; j < N; j++) {
            cin >> B[i][j];
        }
    }
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
        }
    }
    pthread_t tid[M * N];
    pthread_attr_t attr;
    // get the default attributes
    pthread_attr_init(&attr);
    // we have to create M*N pthreads
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            struct v *a = (struct v*)malloc(sizeof(struct v));
            a->i = i;
            a->j = j;
            pthread_create(&tid[i * N + j], &attr, calculate, (void*)a);
        }
    }
    // join upon each thread
    for (int i = 0; i < M * N; i++) {
        pthread_join(tid[i], NULL);
    }
    // output the result
    cout << "The result(M*N) is:" << endl;
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            cout << C[i][j] << " ";
            if (j == N - 1) cout << endl;
        }
    }
    return 0;
}
```

编译运行结果如下：

```
linjiafengyang@ubuntu:~/Desktop$ g++ -g matrix.cpp -o matrix -pthread
linjiafengyang@ubuntu:~/Desktop$ ./matrix
Please enter three numbers(M/K/N) that are less than 100:
3 2 3
Please enter the first matrix(M*K):
1 2
3 4
5 6
Please enter the second matrix(K*N):
1 2 3
4 5 6
The result(M*N) is:
9 12 15
19 26 33
29 40 51
```

与MATLAB软件得出的结果作对比可得出实验结果正确：

```
>> A=[1,2;3,4;5,6]

A =

     1     2
     3     4
     5     6

>> B=[1,2,3;4,5,6]

B =

     1     2     3
     4     5     6

>> A*B

ans =

     9    12    15
    19    26    33
    29    40    51

>>
```

再测试一组较大的数据：
运行结果如下图

```
Please enter three numbers(M/K/N) that are less than 100:
4 4 4
Please enter the first matrix(M*K):
12 14 16 18
20 22 24 26
28 30 32 34
36 38 40 42
Please enter the second matrix(K*N):
55 57 59 61
63 65 67 69
71 73 75 77
79 81 83 85
The result(M*N) is:
4100 4220 4340 4460
6244 6428 6612 6796
8388 8636 8884 9132
10532 10844 11156 11468
```

与MATLAB软件得出的结果对比可得实验结果正确，证明用多线程实现矩阵乘法算法准确。

```
A =

    12    14    16    18
    20    22    24    26
    28    30    32    34
    36    38    40    42

>> B=[55,57,59,61;63,65,67,69;71,73,75,77;79,81,83,85]

B =

    55    57    59    61
    63    65    67    69
    71    73    75    77
    79    81    83    85

>> A*B

ans =

        4100        4220        4340        4460
        6244        6428        6612        6796
        8388        8636        8884        9132
       10532       10844       11156       11468

fx >>
```

五、实验总结

总的来说，这次实验相对比较简单，根据课本的知识和老师的提示，可以很快就实现这两个实验内容。但是，我们不能仅仅满足于实验结果的正确，最重要的是我们要去理解线程的意义和作用，学习pthread线程库主要的几个系统调用：pthread_create(), pthread_join(), pthread_exit(), pthread_attr_init(), pthread_attr_getscope(), pthread_attr_setscope()。
通过这次实验，我进一步加深了对线程的理解，一个应用程序通常是作为一个具有多个控制线程的独立进程实现的，深刻体会到多线程的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。