515030910605

李洋

2017年11月11日

Project3

Matrix Multiplication

1. Introduction

In this experiment, we need to write a code file to calculate matrix multiplication by multithread.

These are the things I have done in this code file:

a) Create child threads to calculate result matrix C[M][N]

b) Pass the parameters to the calculate function by struct.

c) Execute the pthread_join() to wait for finish of all threads.

I will introduce my method in the next part.

2. Steps

2.1. Workflow

a) Set 2 matrices A[M][K] and B[K][N].

b) Create M * N threads to calculate result matrix C[M][N].

c) Pass the parameters to the calculate function by struct.

d) Execute the pthread_join() to wait for finish of all threads.

e) Display the result

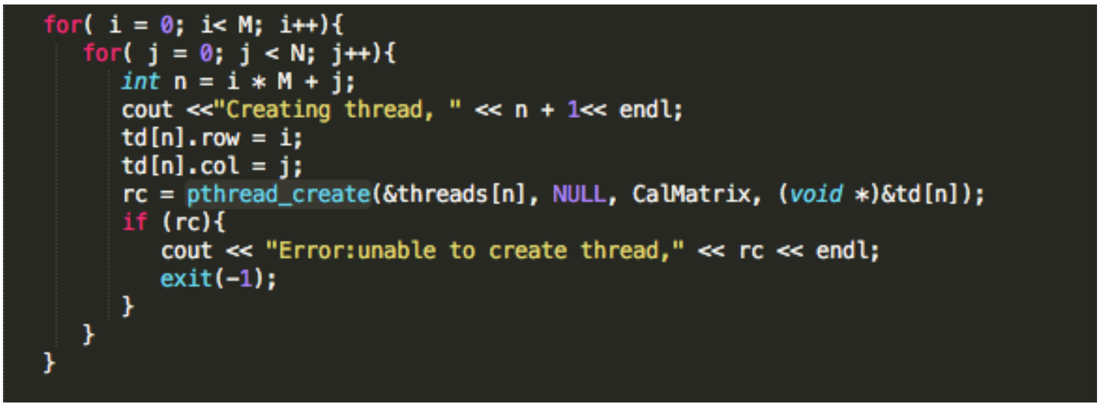
2.2. Create threads

We create threads by pthread_create() function.

The function has 4 parameters. They are the id of thread, the parameter of thread, the function which is called and the parameters of function. The function will return 0 if it creates a thread successfully.

We create M * N threads and calculate each element in C[M][N] by one thread. So we create a array of id of threads.

The following is the process of creating threads:



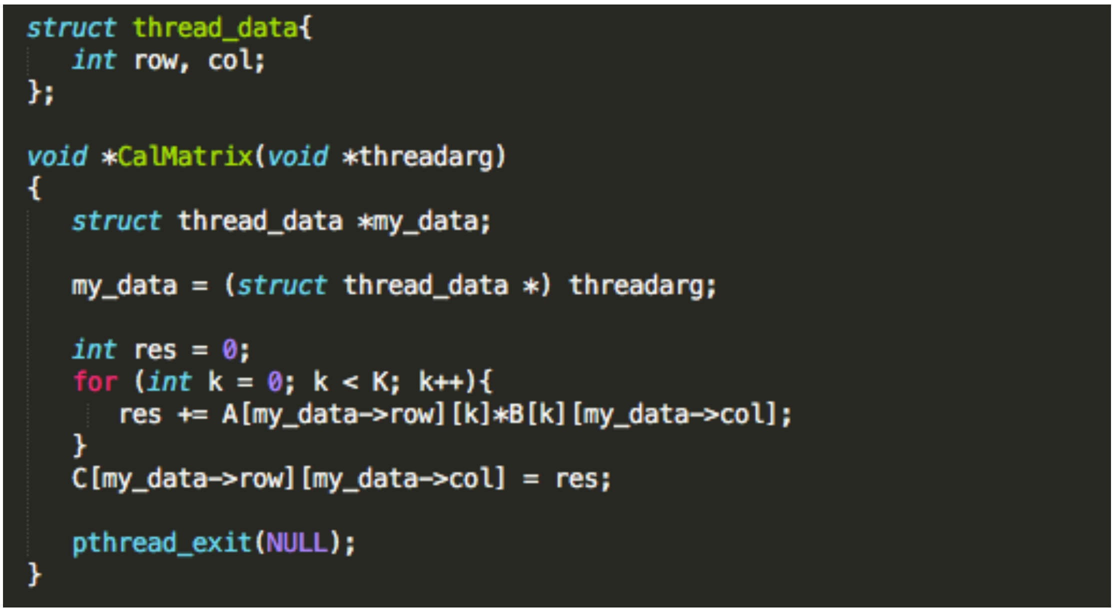
```
for( i = 0; i< M; i++){
    for( j = 0; j < N; j++){
        int n = i * M + j;
        cout <<"Creating thread, " << n + 1<< endl;
        td[n].row = i;
        td[n].col = j;
        rc = pthread_create(&threads[n], NULL, CalMatrix, (void *)&td[n]);
        if (rc){
            cout << "Error:unable to create thread," << rc << endl;
            exit(-1);
        }
    }
}
```

2.3. Pass the parameters to function

This operation are executed by the forth parameter of pthread_create() function. And we also need *struct* to help us pass more parameters. We use a thread_data() to pass the parameters of row and column. And we pass the pointer of struct at the forth parameter.

We also define a function to calculate a value at the specific position of C[M][N].

In this situation, pthread_exit() end thread when the thread finish its work and the thread doesn't need to exist anymore.



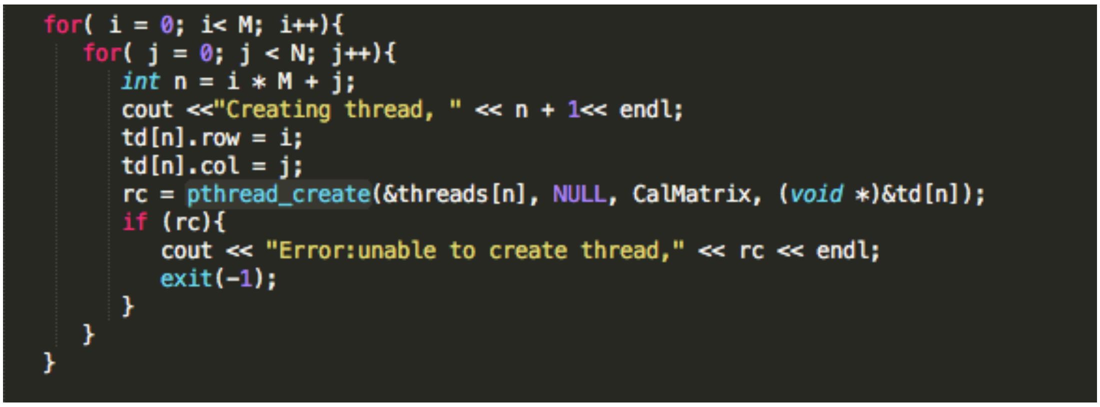
```
struct thread_data{
    int row, col;
};

void *CalMatrix(void *threadarg)
{
    struct thread_data *my_data;

    my_data = (struct thread_data *) threadarg;

    int res = 0;
    for (int k = 0; k < K; k++){
        res += A[my_data->row][k]*B[k][my_data->col];
    }
    C[my_data->row][my_data->col] = res;

    pthread_exit(NULL);
}
```

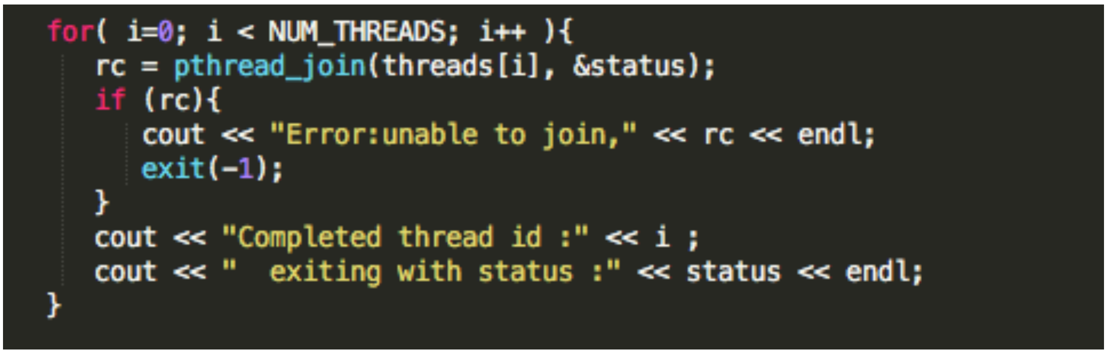


```
for( i = 0; i< M; i++){
    for( j = 0; j < N; j++){
        int n = i * M + j;
        cout <<"Creating thread, " << n + 1<< endl;
        td[n].row = i;
        td[n].col = j;
        rc = pthread_create(&threads[n], NULL, CalMatrix, (void *)&td[n]);
        if (rc){
            cout << "Error:unable to create thread," << rc << endl;
            exit(-1);
        }
    }
}
```

2.4. Execute the pthread_join() to wait.

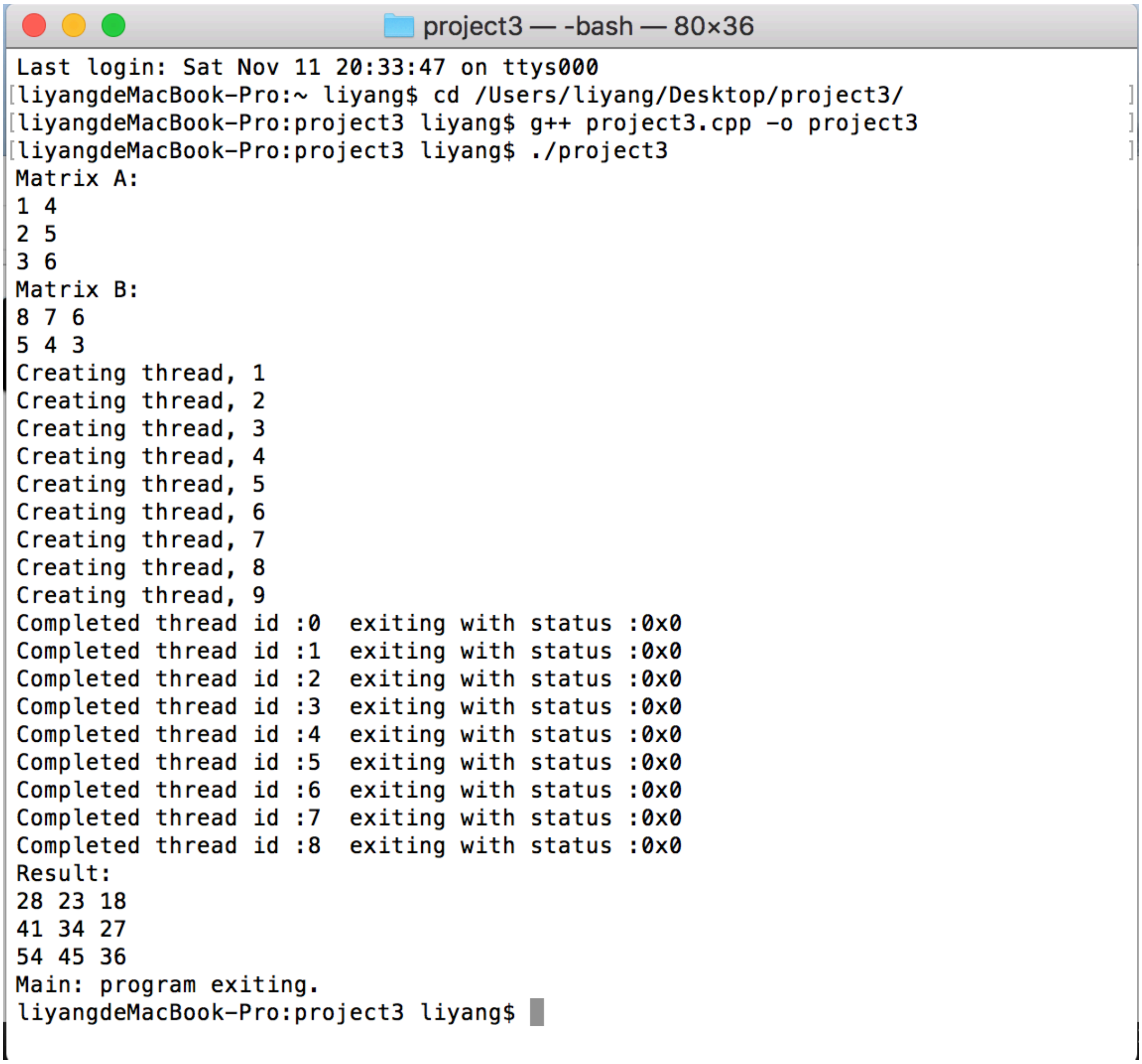
This operation are executed in main() function.

We use pthread_join() to make sure all the threads finish their work before main() end. We use a circle to check wether every thread finish.



```
for( i=0; i < NUM_THREADS; i++ ){
    rc = pthread_join(threads[i], &status);
    if (rc){
       cout << "Error:unable to join," << rc << endl;
       exit(-1);
    }
    cout << "Completed thread id :" << i ;
    cout << "  exiting with status :" << status << endl;
}
```

3. Test result show



```
Last login: Sat Nov 11 20:33:47 on ttys000
liyangdeMacBook-Pro:~ liyang$ cd /Users/liyang/Desktop/project3/
liyangdeMacBook-Pro:project3 liyang$ g++ project3.cpp -o project3
liyangdeMacBook-Pro:project3 liyang$ ./project3
Matrix A:
1 4
2 5
3 6
Matrix B:
8 7 6
5 4 3
Creating thread, 1
Creating thread, 2
Creating thread, 3
Creating thread, 4
Creating thread, 5
Creating thread, 6
Creating thread, 7
Creating thread, 8
Creating thread, 9
Completed thread id :0  exiting with status :0x0
Completed thread id :1  exiting with status :0x0
Completed thread id :2  exiting with status :0x0
Completed thread id :3  exiting with status :0x0
Completed thread id :4  exiting with status :0x0
Completed thread id :5  exiting with status :0x0
Completed thread id :6  exiting with status :0x0
Completed thread id :7  exiting with status :0x0
Completed thread id :8  exiting with status :0x0
Result:
28 23 18
41 34 27
54 45 36
Main: program exiting.
liyangdeMacBook-Pro:project3 liyang$ 
```

We calculate A*B (A = { {1,4}, {2,5}, {3,6} } B = { {8,7,6}, {5,4,3} }). The result is C = {{28, 23, 18},{41, 34, 27},{54, 45, 36}}.

4. Conclusion

In this experiment, I understand how to create a thread and how to pass the parameters to it. And I also know how to use pthread_join() to wait for finish of all threads.

This experiment let me understand the method of thread execution deeply. It really improve my skill of multithread programming.

Appendix: code

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

using namespace std;

#define M   3
#define K   2
#define N   3
#define NUM_THREADS     M * N

int A[M][K] = { {1,4}, {2,5}, {3,6} };
int B[K][N] = { {8,7,6}, {5,4,3} };
int C[M][N];

struct thread_data{
  int row, col;
};

void *CalMatrix(void *threadarg)
{
  struct thread_data *my_data;

  my_data = (struct thread_data *) threadarg;

  int res = 0;
  for (int k = 0; k < K; k++){
    res += A[my_data->row][k]*B[k][my_data->col];
  }
  C[my_data->row][my_data->col] = res;

  pthread_exit(NULL);
}

int main ()
{
  pthread_t threads[NUM_THREADS];
  struct thread_data td[NUM_THREADS];
  int rc;
  int i, j;
  void *status;

  cout<<"Matrix A:"<<endl;
  for( i = 0; i< M; i++){
    for( j = 0; j < K; j++){
      cout<<A[i][j]<<' ';
    }
    cout<<endl;
  }

  cout<<"Matrix B:"<<endl;
  for( i = 0; i< K; i++){
    for( j = 0; j < N; j++){
      cout<<B[i][j]<<' ';
    }
    cout<<endl;
  }

  for( i = 0; i< M; i++){
    for( j = 0; j < N; j++){
      int n = i * M + j;
      cout <<"Creating thread, " << n + 1<< endl;
      td[n].row = i;
      td[n].col = j;
      rc = pthread_create(&threads[n], NULL, CalMatrix, (void *)&td[n]);
      if (rc){
        cout << "Error:unable to create thread," << rc << endl;
        exit(-1);
      }
    }
  }

  for( i=0; i < NUM_THREADS; i++ ){
    rc = pthread_join(threads[i], &status);
    if (rc){
      cout << "Error:unable to join," << rc << endl;
      exit(-1);
    }
    cout << "Completed thread id :" << i ;
    cout << "  exiting with status :" << status << endl;
  }

  cout<<"Result:"<<endl;
  for( i = 0; i< M; i++){
    for( j = 0; j < N; j++){
      cout<<C[i][j]<<' ';
    }
    cout<<endl;
  }

  cout << "Main: program exiting." << endl;
  pthread_exit(NULL);
}
```