OS project three 实验报告
——Matrix Multiplication

5140309201

黄晟

一、 实验要求

利用多线程功能完成矩阵乘法运算。

二、 实现过程

1、初始定义线程，假设矩阵行、列均不超过100；

```
pthread_t tid[100][100];
pthread_attr_t attr[100][100];
```

2、读取文件信息。

首先读取三个数字，即为第一个矩阵的行、第一个矩阵的列（第二个矩阵的行）、第二个矩阵的列。

其次再分别读取两个矩阵的信息，在此仅以第一个矩阵为例；

```
printf("\nthe first matrix is:\n");
for (i = 0; i < m; i++){
    for (j = 0; j < k; j++){
        fscanf(fileHandler,"%d",&a[i][j]);
        printf("%d\t",a[i][j]);
    }
    printf("\n");
}
printf("\n");
```

3、然后对于创建计算结果矩阵的每一个数字均创建一个进程进行计算；

```
for (i = 0; i < m; i++){
    for (j = 0; j < n; j++){
        struct v *data = (struct v*) malloc(sizeof(struct v));
        data -> i = i;
        data -> j = j;

        pthread_attr_init(&attr[i][j]);
        pthread_create(&tid[i][j], &attr[i][j], runner, data);

        //pthread_join(tid[i][j], NULL);

    }
}
```

4、编写计算函数；

```
void *runner(void *param)
{
    struct v *data = param;
    int n, sum = 0;

    for(n = 0; n < k; ++n){
        sum += a[data->i][n] * b[n][data->j];
    }

    c[data->i][data->j] = sum;

    pthread_exit(0);
}
```

5、结束进程，输出计算结果。

```
printf("\nthe result matrix is:\n");
for (i = 0; i < m; i++){
    for (j = 0; j < n; j++){
        printf("%d\t",c[i][j]);
    }
    printf("\n");
}
```

三、 实现效果

1、矩阵信息如图所示：

```
5 2 7
1 4
2 5
3 6
5 3
6 1
8 7 6 9 5 3 5
5 4 3 7 4 5 1
```

2、运行结果。

```
cyril@ubuntu:~/Desktop/project/3$ ./a.out

the first matrix has 5 rows, 2 cols
the second matrix has 2 rows, 7 cols
the first matrix is:
1       4
2       5
3       6
5       3
6       1


the second matrix is:
8       7       6       9       5       3       5
5       4       3       7       4       5       1


the result matrix is:
28      23      18      37      21      23      9
41      34      27      53      30      31      15
54      45      36      69      39      39      21
55      47      39      66      37      30      28
53      46      39      61      34      23      31
```

四、 心得与体会

1、 最初在创建进程的 for 循环中加入了 pthread_join 函数，使得整个函数的执行并没有变快多少。因为 pthread_join 是阻塞并等待线程退出，所以实际相当于单线程执行。因此把 pthread_join 单独放在一个 for 循环语句中，可以实现多线程运行。

```
for (i = 0; i < m; i++ ){
    for (j = 0; j < n; j++)
        pthread_join(tid[i][j], NULL);
}
```