北京郵電大學

Beijing University of Posts and Telecommunications

实 验 报 告

课 程 名 称	操作系统
题 目	多线程编程
姓 名	沈原灏
学 号	2021211108
班 级	2021211306
指 导 教 师	李文生

2023 年 10 月

一、概览

1.1 实验内容

1.1.1 实验一

Write a multithreaded program that calculates various statistical values for a list of numbers. This program will accept a series of numbers from the keyboard and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value.

For example, suppose your program accepted the integers

90 81 78 95 79 72 85

The program will report:

The average value is 82

The minimum value is 72

The maximum value is 95

The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited.

1.1.2 实验二

编写多线程程序，实现矩阵相乘，具体要求见教材 P149 或附件。

1.2 实验环境

- Ubuntu 22.04.01 LTS
- gcc 11.4.0
- Visual Studio Code

二、实验一

2.1 相关 API

`void *calc_avg(void *num_array)`	接收一个数组，返回平均值
`void *calc_max(void *num_array)`	接收一个数组，返回最大值
`void *calc_min(void *num_array)`	接收一个数组，返回最小值
`int pthread_create (pthread_t *__restrict __newthread, const pthread_attr_t *__restrict __attr, void *(*__start_routine) (void *), void *__restrict __arg)`	在当前进程创建一个新线程，新线程通过唤起 start_routine 开始执行，arg 参数被传递到 start_routine 的参数，attr 指向的结构体的内容用来决定新线程创建时的一些参数，如果为 NULL，则采用默认参数。在该函数返回之前，新线程的 ID 将被存储到 thread 中。当成功时，该函数返回 0。
`int pthread_join(pthread_t thread, FAR pthread_addr_t *pexit_value)`	该函数等待 thread 指定的线程终止，如果这个线程已经终止，则该函数立即返回。如果 retval 不是 NULL，则该函数复制目标线程给 pthread_exit() 的参数到 retval 所指向的地址。如果成功，该函数返回 0。
`int pthread_mutex_lock(FAR pthread_mutex_t *mutex)`	用于实现互斥锁的加锁操作。参数是一个指向互斥锁对象的指针，如果互斥锁已经被其他线程锁定，那么调用线程将被阻塞，直到互斥锁可用。
`int pthread_mutex_unlock(FAR pthread_mutex_t *mutex)`	这个函数的作用是解锁指定的互斥锁，允许其他线程访问被保护的共享资源。

2.2 设计概述

- 将输入的数字序列存储在 Number 中：

```
for (int i = 0; i < num_cnt; ++i)
     numbers[i] = atoi(argv[i + 1]);
```

- 创建了三个线程，并分别将这些线程绑定到三个计算函数上：

```
pthread_t worker[NUM_THREADS];

// create 3 threads
pthread_create(&worker[0], NULL, calc_avg, numbers);
pthread_create(&worker[1], NULL, calc_max, numbers);
pthread_create(&worker[2], NULL, calc_min, numbers);
```

- 线程函数如下，分别求平均值、求最大值和求最小值：

  这里我用到了线程互斥锁，因为 avg/max/min 都是全局变量，所以程序应该只允许一个线程同时锁定资源，以避免其他线程同时访问这个资源，防止竞争条件和数据损坏。

```
void *calc_avg(void *num_array)
{
   int sum = 0;
   int *numbers = (int *)num_array;
   for (int i = 0; i < num_cnt; ++i)
      sum += numbers[i];
   pthread_mutex_lock(&mutex);
   avg = sum / num_cnt;
   pthread_mutex_unlock(&mutex);
   pthread_exit(0);
}

void *calc_max(void *num_array)
{
   int *numbers = (int *)num_array;
   max = INT_MIN;
   for (int i = 0; i < num_cnt; ++i)
      if (max < numbers[i])
      {
         pthread_mutex_lock(&mutex);
         max = numbers[i];
         pthread_mutex_unlock(&mutex);
      }
   pthread_exit(0);
}

void *calc_min(void *num_array)
{
```

```
  int *numbers = (int *)num_array;
  min = INT_MAX;
  for (int i = 0; i < num_cnt; ++i)
    if (min > numbers[i])
    {
      pthread_mutex_lock(&mutex);
      min = numbers[i];
      pthread_mutex_unlock(&mutex);
    }
  pthread_exit(0);
}
```

- 等待线程执行完成后，输出结果：

```
// wait for all threads done
for (int i = 0; i < NUM_THREADS; ++i)
pthread_join(worker[i], NULL);

// print
printf("The average value is %d\n", avg);
printf("The maximum value is %d\n", min);
printf("The minimum value is %d\n", max);
```

2.3 运行结果及分析

2.3.1 运行结果

```
arycra07@arycra07-virtual-machine:~/Desktop/os_lab/lab2/task1$ ./task1 90 81 78 95 79 72 85
The average value is 82
The maximum value is 72
The minimum value is 95
arycra07@arycra07-virtual-machine:~/Desktop/os_lab/lab2/task1$
```

2.3.2 分析

该程序正确地输出了该数字序列的平均值、最大值和最小值。

三、实验二

3.1 相关 API

`int pthread_create (pthread_t *__restrict __newthread, const pthread_attr_t *__restrict __attr, void *(*__start_routine) (void *), void *__restrict __arg)`	在当前进程创建一个新线程，新线程通过唤起 start_routine 开始执行，arg 参数被传递到 start_routine 的参数，attr 指向的结构体的内容用来决定新线程创建时的一些参数，如果为 NULL，则采用默认参数。在该函数返回之前，新线程的 ID 将被存储到 thread 中。当成功时，该函数返回 0。
`int pthread_join(pthread_t thread, FAR pthread_addr_t *pexit_value)`	该函数等待 thread 指定的线程终止，如果这个线程已经终止，则该函数立即返回。如果 retval 不是 NULL，则该函数复制目标线程给 pthread_exit() 的参数到 retval 所指向的地址。如果成功，该函数返回 0。
`int pthread_mutex_lock(FAR pthread_mutex_t *mutex)`	用于实现互斥锁的加锁操作。参数是一个指向互斥锁对象的指针，如果互斥锁已经被其他线程锁定，那么调用线程将被阻塞，直到互斥锁可用。
`int pthread_mutex_unlock(FAR pthread_mutex_t *mutex)`	这个函数的作用是解锁指定的互斥锁，允许其他线程访问被保护的共享资源。

3.2 设计概述

出于实用性与灵活性的考虑，我没有采用书本中建议的方式将矩阵以静态或文件的方式存储，而是读入两个矩阵的大小与数值，这样做更加实用灵活。我做了用户提示与输入检查：

```
#define MAXN 10

int A[MAXN][MAXN];
int B[MAXN][MAXN];
int C[MAXN][MAXN];
int A_ROW, A_COL, B_ROW, B_COL;

……
```

```
printf("Enter the number of rows and columns in the matrix A:\n");
  scanf("%d %d", &A_ROW, &A_COL);
  printf("Enter values of A:\n");
  for (int i = 0; i < A_ROW; i++)
  {
    for (int j = 0; j < A_COL; j++)
    {
      scanf("%d", &A[i][j]);
    }
  }

  printf("Enter the number of rows and columns in the matrix B:\n");
  scanf("%d %d", &B_ROW, &B_COL);
  printf("Enter values of B:\n");
  for (int i = 0; i < B_ROW; i++)
  {
    for (int j = 0; j < B_COL; j++)
    {
      scanf("%d", &B[i][j]);
    }
  }

  if (A_COL != B_ROW || A_ROW > MAXN || A_COL > MAXN || B_ROW > MAXN
|| B_COL > MAXN)
  {
    fprintf(stderr, "Usage: task2 invalid input!\n");
    return -1;
  }
```

接着，我为结果矩阵的每一个元素的计算都创建了一个线程，并绑定计算函数。

```
struct Node
{
  int row;
  int col;
};

……

pthread_t worker[MAXN][MAXN];
struct Node mat_node[MAXN][MAXN];
```

```
……

for (int i = 0; i < A_ROW; ++i)
  {
    for (int j = 0; j < B_COL; ++j)
    {
      mat_node[i][j].row = i;
      mat_node[i][j].col = j;

      int ret = pthread_create(&worker[i][j], NULL, Calc, (void
*)&mat_node[i][j]);
      if (ret)
      {
        fprintf(stderr, "Usage: task2 create thread fail!\n");
        return -1;
      }
    }
  }
```

计算函数如下，这里依然用到了线程互斥锁，因为 C 矩阵数组是全局变量，在修改值的时候需要互斥访问：

```
void *Calc(void *node)
{
  int res = 0;
  int r = ((struct Node *)node)->row;
  int c = ((struct Node *)node)->col;

  for (int i = 0; i < A_COL; ++i)
  {
    res += A[r][i] * B[i][c];
  }
  pthread_mutex_lock(&mutex);
  C[r][c] = res;
  pthread_mutex_unlock(&mutex);
}
```

最后，等待线程执行完毕返回结果即可。

```
printf("Here is the result:\n");
for (int i = 0; i < A_ROW; ++i)
{
  for (int j = 0; j < B_COL; ++j)
  {
```

```
      pthread_join(worker[i][j], NULL);
   }
}

for (int i = 0; i < A_ROW; ++i)
{
   for (int j = 0; j < B_COL; ++j)
   {
      printf("%-5d", C[i][j]);
   }
   printf("\n");
}
```

3.3 运行结果及分析

3.3.1 运行结果

```
arycra07@arycra07-virtual-machine:~/Desktop/os_lab/lab2/task2$ ./task2
Enter the number of rows and columns in the matrix A:
3 2
Enter values of A:
1 4
2 5
3 6
Enter the number of rows and columns in the matrix B:
2 3
Enter values of B:
8 7 6
5 4 3
Here is the result:
28   23   18
41   34   27
54   45   36
```

```
arycra07@arycra07-virtual-machine:~/Desktop/os_lab/lab2/task2$ ./task2
Enter the number of rows and columns in the matrix A:
1 1
Enter values of A:
1
Enter the number of rows and columns in the matrix B:
2 1
Enter values of B:
1 1
Usage: task2 invalid input!
arycra07@arycra07-virtual-machine:~/Desktop/os_lab/lab2/task2$
```

3.3.2 分析

在输入合法的矩阵 A 和 B 之后，程序可以输出正确的结果，反之则对非法输入进行报错。

四、总结

本次实验中，我使用 POSIX 的相关 API 编写了多线程相关的程序，我通过查阅 pthread 库相关文档以及书本上关于线程部分的知识，巩固了自己对于多线程的理解。本次实验提高了我的 Linux-C 编程能力与操作系统掌握水平，使我收益颇丰。