Operating System: lock **Fall 2024**

Homework 5 — September 25

Lecturer: Jiang Dejun *Completed by: Zhang Jiawei*

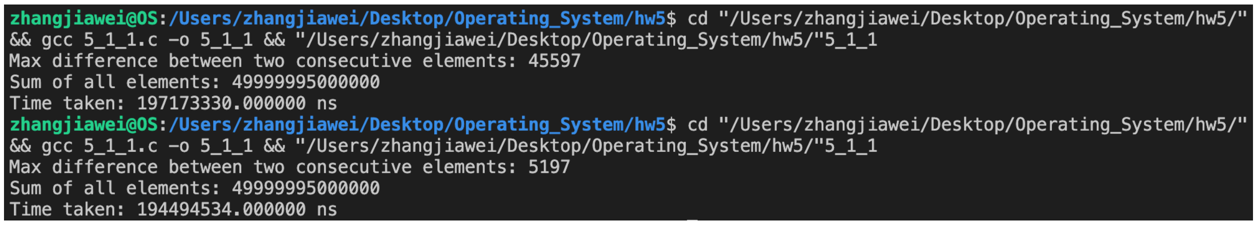
5.1

(1) 使用 Peterson 算法实现互斥访问,代码如下:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>
#define MAX 10000000
#define max(a, b) ((a) > (b) ? (a) : (b))

bool flag[2] = {false, false};
int turn = 0;
int idx = 0;
int idx_even = 0;
int idx_odd = 0;
int data[MAX] = {0};

void *even(void *arg) {
   while (idx < MAX){
      flag[0] = true;
      turn = 1;
      while (flag[1] && turn == 1);
      // begin critical section
      int count = 0;
      int i = idx_even;
      while (count < 200 && i < MAX){
         data[idx] = i;
         idx++;
         i+=2;
         count++;
      }
      idx_even += 400;
      // end critical section
      flag[0] = false;
   }
   return NULL;
}

void *odd(void *arg) {
   while (idx < MAX){
      flag[1] = true;
      turn = 0;
      while (flag[0] && turn == 0);
      // begin critical section
      int count = 0;
      int i = idx_odd;
      while (count < 200 && i < MAX){
         data[idx] = i+1;
         idx++;
         i+=2;
         count++;
      }
      idx_odd += 400;
      // end critical section
      flag[1] = false;
   }
   return NULL;
}

int main() {
   pthread_t t1, t2;
   struct timespec start, end;
   clock_gettime(CLOCK_MONOTONIC, &start);
   pthread_create(&t1, NULL, even, NULL);
   pthread_create(&t2, NULL, odd, NULL);
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   clock_gettime(CLOCK_MONOTONIC, &end);

   int max_diff = 0;
   long sum = 0;
   for (int i = 0; i+1 < MAX; i++){
      max_diff = max(max_diff, abs(data[i] - data[i+1]));
   }
   for (int i = 0; i < MAX; i++)
      sum += data[i];

```

```c
    printf("Max difference between two consecutive elements: %d\n", max_diff);
    printf("Sum of all elements: %ld\n", sum);
    printf("Time taken: %lf ns\n", (end.tv_sec - start.tv_sec) * 1e9 + (end.tv_nsec - start.tv_nsec));
    return 0;
}
```

临界区已经在代码中标注，运行结果如下：



```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_1_1.c -o 5_1_1 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_1_1
Max difference between two consecutive elements: 45597
Sum of all elements: 49999995000000
Time taken: 197173330.000000 ns
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_1_1.c -o 5_1_1 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_1_1
Max difference between two consecutive elements: 5197
Sum of all elements: 49999995000000
Time taken: 194494534.000000 ns
```

图 **5.1.** Peterson 算法运行结果

可以看出，Peterson 算法算出了最大相邻元素差值和所有元素的和，同时计算了程序运行时间。由元素和得知，程序运行正确，但是最大相邻元素差值具有随机性，这是因为 CPU 调度的不确定性导致的。

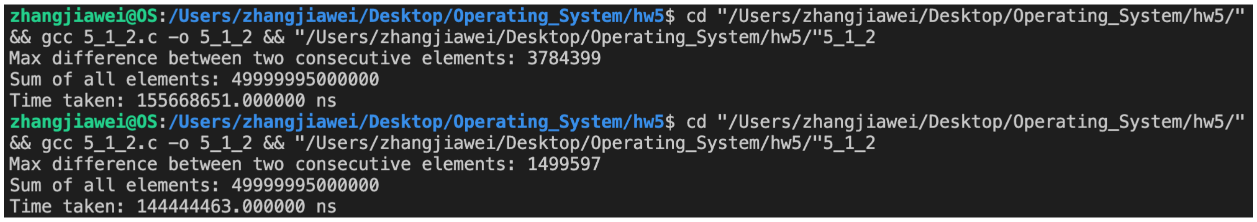
(2) 使用 `pthread_mutex_lock/unlock()` 实现互斥访问，代码如下：

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>
#define MAX 10000000
#define max(a, b) ((a) > (b) ? (a) : (b))

int idx = 0;
int idx_even = 0;
int idx_odd = 0;
int data[MAX] = {0};
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *even(void *arg) {
    while (idx < MAX){
        pthread_mutex_lock(&mutex);
        // begin critical section
        int count = 0;
```

```
            int i = idx_even;
            while (count < 200 && i < MAX){
               data[idx] = i;
               idx++;
               i+=2;
               count++;
            }
            idx_even += 400;
            // end critical section
            pthread_mutex_unlock(&mutex);
         }
         return NULL;
      }

      void *odd(void *arg) {
         while (idx < MAX){
            pthread_mutex_lock(&mutex);
            // begin critical section
            int count = 0;
            int i = idx_odd;
            while (count < 200 && i < MAX){
               data[idx] = i+1;
               idx++;
               i+=2;
               count++;
            }
            idx_odd += 400;
            // end critical section
            pthread_mutex_unlock(&mutex);
         }
         return NULL;
      }

      int main() {
         pthread_t t1, t2;
         struct timespec start, end;
         pthread_mutex_init(&mutex, NULL);
         clock_gettime(CLOCK_MONOTONIC, &start);
         pthread_create(&t1, NULL, even, NULL);
         pthread_create(&t2, NULL, odd, NULL);
         pthread_join(t1, NULL);
         pthread_join(t2, NULL);
         clock_gettime(CLOCK_MONOTONIC, &end);
```

```c
        pthread_mutex_destroy(&mutex);

        int max_diff = 0;
        long sum = 0;
        for (int i = 0; i+1 < MAX; i++){
           max_diff = max(max_diff, abs(data[i] - data[i+1]));
        }
        for (int i = 0; i < MAX; i++)
           sum += data[i];

        printf("Max difference between two consecutive elements: %d\n",
            max_diff);
        printf("Sum of all elements: %ld\n", sum);
        printf("Time taken: %lf ns\n", (end.tv_sec - start.tv_sec) * 1e9 +
            (end.tv_nsec - start.tv_nsec));
        return 0;
    }
```

临界区已经在代码中标注，运行结果如下：



```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_1_2.c -o 5_1_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_1_2
Max difference between two consecutive elements: 3784399
Sum of all elements: 49999995000000
Time taken: 155668651.000000 ns
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_1_2.c -o 5_1_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_1_2
Max difference between two consecutive elements: 1499597
Sum of all elements: 49999995000000
Time taken: 144444463.000000 ns
```

图 5.2. pthread_mutex_lock/unlock() 运行结果

与上一题的结果类似，最大相邻元素差值仍因为 CPU 调度的不确定性而具有随机性，但是元素和的结果表明程序运行正确，运行时间比上一题稍短。

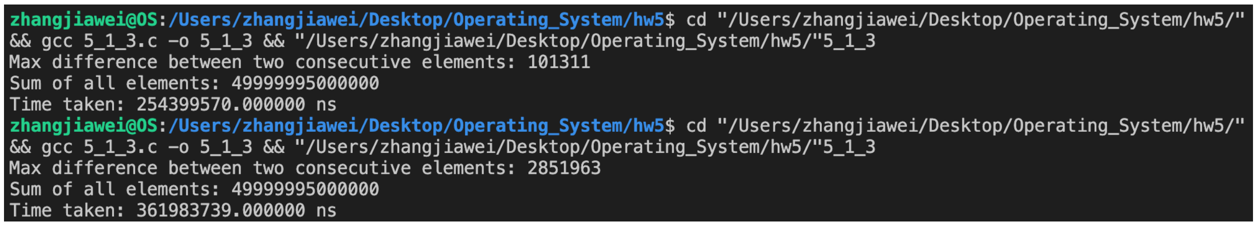
(3) 使用 atomic_add_fetch 实现互斥访问，代码如下：

```c
    #include <stdio.h>
    #include <stdlib.h>
    #include <stdbool.h>
    #include <pthread.h>
    #include <time.h>
    #include <unistd.h>
    #include <stdatomic.h>

    #define MAX 10000000
    #define max(a, b) ((a) > (b) ? (a) : (b))
```

```c

atomic_long idx = 0;
atomic_long idx_even = 0;
atomic_long idx_odd = 0;
long data[MAX] = {0};

void *even(void *arg) {
   while (atomic_load(&idx) < MAX) {
      int count = 0;
      long i = atomic_load(&idx_even);
      while (count < 200 && i < MAX) {
         data[atomic_fetch_add(&idx, 1)] = i;
         i += 2;
         count++;
      }
      atomic_fetch_add(&idx_even, 400);
   }
   return NULL;
}

void *odd(void *arg) {
   while (atomic_load(&idx) < MAX) {
      int count = 0;
      long i = atomic_load(&idx_odd);
      while (count < 200 && i < MAX) {
         data[atomic_fetch_add(&idx, 1)] = i + 1;
         i += 2;
         count++;
      }
      atomic_fetch_add(&idx_odd, 400);
   }
   return NULL;
}

int main() {
   pthread_t t1, t2;
   struct timespec start, end;
   clock_gettime(CLOCK_MONOTONIC, &start);
   pthread_create(&t1, NULL, even, NULL);
   pthread_create(&t2, NULL, odd, NULL);
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   clock_gettime(CLOCK_MONOTONIC, &end);
```

```c

        long max_diff = 0;
        long sum = 0;
        for (int i = 0; i + 1 < MAX; i++) {
            max_diff = max(max_diff, labs(data[i] - data[i + 1]));
        }
        for (int i = 0; i < MAX; i++)
            sum += data[i];

        printf("Max difference between two consecutive elements: %ld\n",
             max_diff);
        printf("Sum of all elements: %ld\n", sum);
        printf("Time taken: %lf ns\n", (end.tv_sec - start.tv_sec) * 1e9 +
             (end.tv_nsec - start.tv_nsec));
        return 0;
    }
```

临界区已经在代码中标注，运行结果如下：



```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_1_3.c -o 5_1_3 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_1_3
Max difference between two consecutive elements: 101311
Sum of all elements: 49999995000000
Time taken: 254399570.000000 ns
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_1_3.c -o 5_1_3 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_1_3
Max difference between two consecutive elements: 2851963
Sum of all elements: 49999995000000
Time taken: 361983739.000000 ns
```

图 **5.3.** atomic_add_fetch 运行结果

本题使用了 atomic_add_fetch 函数实现互斥访问，但是最大相邻元素差值具有随机性，这是因为 CPU 调度的不确定性导致的。除此之外，在 Linux 系统下，如果不使用 long 类型，而是使用 int 类型，可能会导致溢出，原因未知。

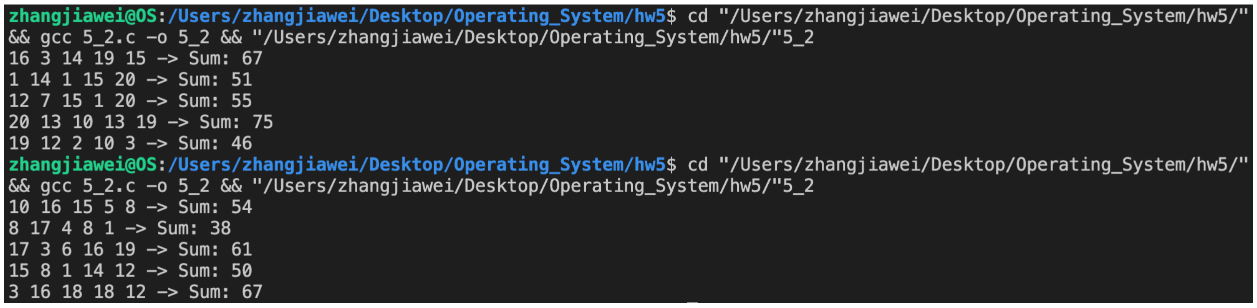
5.2

可以实现，代码如下：

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#define LEN 5
#define CYCLE 5

int data[LEN] = {0};
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

```c

void *randomize(void *arg) {
   struct timespec ts;
   clock_gettime(CLOCK_REALTIME, &ts);
   srand(ts.tv_nsec);

   pthread_mutex_lock(&mutex);
   for (int i = 0; i < LEN; i++)
      data[i] = rand() % 20 + 1;
   pthread_mutex_unlock(&mutex);
   return NULL;
}

void *printArray(void *arg) {
   int sum = 0;

   pthread_mutex_lock(&mutex);
   for (int i = 0; i < LEN; i++){
      printf("%d ", data[i]);
      sum += data[i];
   }
   pthread_mutex_unlock(&mutex);
   printf("-> Sum: %d\n", sum);
   return NULL;
}

int main(){
   pthread_t t1, t2;
   for (int i = 0; i < CYCLE; i++){
      pthread_create(&t1, NULL, randomize, NULL);
      pthread_join(t1, NULL);
      pthread_create(&t2, NULL, printArray, NULL);
      pthread_join(t2, NULL);
   }
   return 0;
}
```

运行结果如下:



```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_2.c -o 5_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_2
16 3 14 19 15 -> Sum: 67
1 14 1 15 20 -> Sum: 51
12 7 15 1 20 -> Sum: 55
20 13 10 13 19 -> Sum: 75
19 12 2 10 3 -> Sum: 46
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw5$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw5/"
&& gcc 5_2.c -o 5_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw5/"5_2
10 16 15 5 8 -> Sum: 54
8 17 4 8 1 -> Sum: 38
17 3 6 16 19 -> Sum: 61
15 8 1 14 12 -> Sum: 50
3 16 18 18 12 -> Sum: 67
```

图 **5.4.** 随机生成数组并打印

可以看出，程序成功使用互斥锁实现了随机生成数组并打印的功能，每次生成的数组元素和都不相同。需要注意的是，随机数种子使用了当前时间的纳秒部分，以保证每次生成的随机数不同，若不使用纳秒部分，可能会导致每次生成的随机数相同。