

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:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <pthread.h>
5  #include <time.h>
6  #include <unistd.h>
7  #define MAX 10000000
8  #define max(a, b) ((a) > (b) ? (a) : (b))
9
10 bool flag[2] = {false, false};
11 int turn = 0;
12 int idx = 0;
13 int idx_even = 0;
14 int idx_odd = 0;
15 int data[MAX] = {0};
16
17 void *even(void *arg) {
18     while (idx < MAX){
19         flag[0] = true;
20         turn = 1;
21         while (flag[1] && turn == 1);
22         // begin critical section
23         int count = 0;
24         int i = idx_even;
25         while (count < 200 && i < MAX){
26             data[idx] = i;
27             idx++;
28             i+=2;
29             count++;
30         }
31         idx_even += 400;
32         // end critical section
33         flag[0] = false;
34     }
```

```
35     return NULL;
36 }
37
38 void *odd(void *arg) {
39     while (idx < MAX){
40         flag[1] = true;
41         turn = 0;
42         while (flag[0] && turn == 0);
43         // begin critical section
44         int count = 0;
45         int i = idx_odd;
46         while (count < 200 && i < MAX){
47             data[idx] = i+1;
48             idx++;
49             i+=2;
50             count++;
51         }
52         idx_odd += 400;
53         // end critical section
54         flag[1] = false;
55     }
56     return NULL;
57 }
58
59 int main() {
60     pthread_t t1, t2;
61     struct timespec start, end;
62     clock_gettime(CLOCK_MONOTONIC, &start);
63     pthread_create(&t1, NULL, even, NULL);
64     pthread_create(&t2, NULL, odd, NULL);
65     pthread_join(t1, NULL);
66     pthread_join(t2, NULL);
67     clock_gettime(CLOCK_MONOTONIC, &end);
68
69     int max_diff = 0;
70     long sum = 0;
71     for (int i = 0; i+1 < MAX; i++){
72         max_diff = max(max_diff, abs(data[i] - data[i+1]));
73     }
74     for (int i = 0; i < MAX; i++)
75         sum += data[i];
76 }
```

```

77     printf("Max difference between two consecutive elements: %d\n",
            max_diff);
78     printf("Sum of all elements: %ld\n", sum);
79     printf("Time taken: %lf ns\n", (end.tv_sec - start.tv_sec) * 1e9 +
            (end.tv_nsec - start.tv_nsec));
80     return 0;
81 }

```

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:

```

1     #include <stdio.h>
2     #include <stdlib.h>
3     #include <stdbool.h>
4     #include <pthread.h>
5     #include <time.h>
6     #include <unistd.h>
7     #define MAX 100000000
8     #define max(a, b) ((a) > (b) ? (a) : (b))
9
10    int idx = 0;
11    int idx_even = 0;
12    int idx_odd = 0;
13    int data[MAX] = {0};
14    pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
15
16    void *even(void *arg) {
17        while (idx < MAX){
18            pthread_mutex_lock(&mutex);
19            // begin critical section
20            int count = 0;

```

```
21         int i = idx_even;
22         while (count < 200 && i < MAX){
23             data[idx] = i;
24             idx++;
25             i+=2;
26             count++;
27         }
28         idx_even += 400;
29         // end critical section
30         pthread_mutex_unlock(&mutex);
31     }
32     return NULL;
33 }
34
35 void *odd(void *arg) {
36     while (idx < MAX){
37         pthread_mutex_lock(&mutex);
38         // begin critical section
39         int count = 0;
40         int i = idx_odd;
41         while (count < 200 && i < MAX){
42             data[idx] = i+1;
43             idx++;
44             i+=2;
45             count++;
46         }
47         idx_odd += 400;
48         // end critical section
49         pthread_mutex_unlock(&mutex);
50     }
51     return NULL;
52 }
53
54 int main() {
55     pthread_t t1, t2;
56     struct timespec start, end;
57     pthread_mutex_init(&mutex, NULL);
58     clock_gettime(CLOCK_MONOTONIC, &start);
59     pthread_create(&t1, NULL, even, NULL);
60     pthread_create(&t2, NULL, odd, NULL);
61     pthread_join(t1, NULL);
62     pthread_join(t2, NULL);
63     clock_gettime(CLOCK_MONOTONIC, &end);
```

```

64     pthread_mutex_destroy(&mutex);
65
66     int max_diff = 0;
67     long sum = 0;
68     for (int i = 0; i+1 < MAX; i++){
69         max_diff = max(max_diff, abs(data[i] - data[i+1]));
70     }
71     for (int i = 0; i < MAX; i++)
72         sum += data[i];
73
74     printf("Max difference between two consecutive elements: %d\n",
75           max_diff);
76     printf("Sum of all elements: %ld\n", sum);
77     printf("Time taken: %lf ns\n", (end.tv_sec - start.tv_sec) * 1e9 +
78           (end.tv_nsec - start.tv_nsec));
79     return 0;
80 }

```

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:

```

1     #include <stdio.h>
2     #include <stdlib.h>
3     #include <stdbool.h>
4     #include <pthread.h>
5     #include <time.h>
6     #include <unistd.h>
7     #include <stdatomic.h>
8
9     #define MAX 10000000
10    #define max(a, b) ((a) > (b) ? (a) : (b))

```

```
11
12     atomic_long idx = 0;
13     atomic_long idx_even = 0;
14     atomic_long idx_odd = 0;
15     long data[MAX] = {0};
16
17     void *even(void *arg) {
18         while (atomic_load(&idx) < MAX) {
19             int count = 0;
20             long i = atomic_load(&idx_even);
21             while (count < 200 && i < MAX) {
22                 data[atomic_fetch_add(&idx, 1)] = i;
23                 i += 2;
24                 count++;
25             }
26             atomic_fetch_add(&idx_even, 400);
27         }
28         return NULL;
29     }
30
31     void *odd(void *arg) {
32         while (atomic_load(&idx) < MAX) {
33             int count = 0;
34             long i = atomic_load(&idx_odd);
35             while (count < 200 && i < MAX) {
36                 data[atomic_fetch_add(&idx, 1)] = i + 1;
37                 i += 2;
38                 count++;
39             }
40             atomic_fetch_add(&idx_odd, 400);
41         }
42         return NULL;
43     }
44
45     int main() {
46         pthread_t t1, t2;
47         struct timespec start, end;
48         clock_gettime(CLOCK_MONOTONIC, &start);
49         pthread_create(&t1, NULL, even, NULL);
50         pthread_create(&t2, NULL, odd, NULL);
51         pthread_join(t1, NULL);
52         pthread_join(t2, NULL);
53         clock_gettime(CLOCK_MONOTONIC, &end);
```

```

54
55     long max_diff = 0;
56     long sum = 0;
57     for (int i = 0; i + 1 < MAX; i++) {
58         max_diff = max(max_diff, labs(data[i] - data[i + 1]));
59     }
60     for (int i = 0; i < MAX; i++)
61         sum += data[i];
62
63     printf("Max difference between two consecutive elements: %ld\n",
64           max_diff);
64     printf("Sum of all elements: %ld\n", sum);
65     printf("Time taken: %lf ns\n", (end.tv_sec - start.tv_sec) * 1e9 +
66           (end.tv_nsec - start.tv_nsec));
66     return 0;
67 }

```

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:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <time.h>
5  #define LEN 5
6  #define CYCLE 5
7
8  int data[LEN] = {0};
9  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

```

```
10
11 void *randomize(void *arg) {
12     struct timespec ts;
13     clock_gettime(CLOCK_REALTIME, &ts);
14     srand(ts.tv_nsec);
15
16     pthread_mutex_lock(&mutex);
17     for (int i = 0; i < LEN; i++)
18         data[i] = rand() % 20 + 1;
19     pthread_mutex_unlock(&mutex);
20     return NULL;
21 }
22
23 void *printArray(void *arg) {
24     int sum = 0;
25
26     pthread_mutex_lock(&mutex);
27     for (int i = 0; i < LEN; i++){
28         printf("%d ", data[i]);
29         sum += data[i];
30     }
31     pthread_mutex_unlock(&mutex);
32     printf("-> Sum: %d\n", sum);
33     return NULL;
34 }
35
36 int main(){
37     pthread_t t1, t2;
38     for (int i = 0; i < CYCLE; i++){
39         pthread_create(&t1, NULL, randomize, NULL);
40         pthread_join(t1, NULL);
41         pthread_create(&t2, NULL, printArray, NULL);
42         pthread_join(t2, NULL);
43     }
44     return 0;
45 }
```

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。