Project 2 Report

花振东　jackeyhua@sjtu.edu.cn

2024 年 4 月 15 日

目录

1 Burger buddies problem

1.1 Description

We need to design a C solution for the burger buddies process synchronization problem.

1.2 Semaphore and Mutex

I use five semaphores and one mutex to solve the problem.

- `sem_customer = 0`, it is used to show whether there is a customer waiting for the cashier.
- `sem_customer_private = 1`. Because every customer has to know the cashier's information, so we need to use this to fetch that info. It is set to 1 so it acts like a mutex.
- `sem_cashier = 0`, it reflects the number of cashiers available.
- `sem_rack = 0`, it shows the number of burgers in the rack.
- `sem_vacant = NUMRACKS`, it shows the number of vacant place in the rack, namely how many burgers can be made currently.
- `mutex = 1`, it is used by cashier thread and provides a critical section to modify the cashier's information, which will be fetched by corresponding customer thread.

Mutex is no different from semaphore. They are just semaphore with value 1. But the name has better readability.

1.3 Cashier struct

When a customer is served by a cashier, they actually bond together. The customer should know the cashier's id. And there should be private semaphore between them. Namely `order, served, takeaway`. The first one indicates whether the customer has ordered. The second one indicates whether the cashier has fetched the burger and gave it to the customer. The last one indicates whether the customer has taken the burger away. They are key to the synchronization between customer and cashier. The struct is designed as follows:

```
1 typedef struct {
2     uint8_t id;          // 服务当前客户的服务员id
```

```
3     sem_t* order;        // 客户是否点单
4     sem_t* takeaway;     // 客户是否将汉堡拿走，不拿走服务员不能继续服务
5     sem_t* served;       // 服务员是否拿到了汉堡
6 } cashier_arg_t;
```

1.4 Design

Cook, cashier and customer are abstracted as three functions. We create `NUMCOOKS` cook threads, `NUMCASHIERS` cashier threads and `NUMCUSTOMERS` customer threads, running corresponding functions.

Cooks run in infinite loop. They wait for a vacancy in the rack, then make a burger and put it in the rack. I use `sleep(rand() % SLEEP)` to simulate the time of making a burger.

Cashier also runs in infinite loop. They wait for a customer to come. When a customer matchs a cashier, the cashier need to get the mutex, and update his id and private semaphore to global variable `cashier_cur`. After that, he can signal that he is ready to serve. Then he need to wait for the customer to order, using private semaphore `order`. After the customer orders, he need to wait for a burger. After the burger is ready, he can give it to the customer and wait for the customer to take it away. After the customer takes it away, he can serve another customer or sleep based on the number of customers.

Customer does not run in infinite loop. They will exit after they are served. They first need to get the `sem_customer_private` semaphore, and signal the cashier that they are waiting and wait for a cashier. Then they copy the value of the global variable `cashier_cur` to their private variable. After that, they can release the `sem_customer_private` semaphore. Then they can order, wait for the burger and take it away. After that, they can exit. It is very important that they first get the `sem_customer_private` semaphore, and then `sem_post(&sem_customer)`. Because `sem_post` does not have concurrency control, so there maybe several customer threads signal concurrently. A later `sem_post` thread may first get `sem_customer_private`, leading to the wrong id of the cashier.

Note that there is no order for cashiers to fetch a burger. Namely, if there are 2 cashiers waiting for a burger, any of them can fetch the burger if ready. There is no order between them. It mimics the reality. The cashier will not fetch the burger until the cook shouts out that "A burger is ready." (lazy fetch). So whether the cashier can get this burger or not depends on his speed to fetch the burger.

Additionally, there is no order for the customers which is waiting to be served.

Namely, if there are 2 customers waiting for a cashier, any of them can be served by the next available cashier. There is no order between them. It is reasonable, because there is no concept of queue here. The customer waiting is like a pool, and they compete for the next available cashier.

I write detailed comments in the code, please refer to the code for implementation details.

1.5 Verification

I test my program on several cases, and i will list and analyze respectively.

1.5.1 Case 1

```
./BBC 2 4 2 3

Cooks [2], Cashiers [4], Customers [2], Rack[3]
Begin run.

Customer [1] comes.
Customer [1] is being served by cashier [1].
Customer [2] comes.
Cook [2] cooks a burger and puts it into the rack.
Cook [2] cooks a burger and puts it into the rack.
Cook [1] cooks a burger and puts it into the rack.
Cashier [1] accepts an order.
Customer [2] is being served by cashier [2].
Cashier [2] accepts an order.
Cashier [2] takes a burger from the rack to the customer.
Customer [2] receives the burger from cashier [2] and leaves.
Cashier [1] takes a burger from the rack to the customer.
Customer [1] receives the burger from cashier [1] and leaves.

All customers have left, DONE.
```

In this case, there are more cashiers than customers and rack size greater than the number of customers.

We can see the correct order of a customer and a cashier. First, customer comes, and then a cashier serves him. After that, the customer orders a food. The cashier fetches the burger and gives it to the customer. Finally, the customer takes the burger away and

leaves.

We can also see the interleaving of different customer-cashier pair. Because there are no contraints between different customer-cashier pairs. We can also see that the cook can make burgers concurrently, not being affected by the cashier or customer other than the rack size.

So it is correct.

1.5.2 Case 2

```
./BBC 4 2 10 20

Cooks [4], Cashiers [2], Customers [10], Rack[20]
Begin run.

Cook [2] cooks a burger and puts it into the rack.
Customer [1] comes.
Customer [1] is being served by cashier [1].
Cashier [1] accepts an order.
Customer [3] comes.
Customer [7] comes.
Customer [6] comes.
Customer [5] comes.
Customer [4] comes.
Customer [2] comes.
Customer [10] comes.
Customer [9] comes.
Customer [8] comes.
Customer [3] is being served by cashier [2].
Cook [1] cooks a burger and puts it into the rack.
Cashier [1] takes a burger from the rack to the customer.
Customer [1] receives the burger from cashier [1] and leaves.
Cook [4] cooks a burger and puts it into the rack.
Cashier [2] accepts an order.
Customer [7] is being served by cashier [1].
Cook [3] cooks a burger and puts it into the rack.
Cook [2] cooks a burger and puts it into the rack.
Cook [3] cooks a burger and puts it into the rack.
Cook [4] cooks a burger and puts it into the rack.
Cashier [2] takes a burger from the rack to the customer.
```

```
31  Cook [4] cooks a burger and puts it into the rack.
32  Customer [3] receives the burger from cashier [2] and leaves.
33  Cashier [1] accepts an order.
34  Customer [6] is being served by cashier [2].
35  Cook [4] cooks a burger and puts it into the rack.
36  Cook [1] cooks a burger and puts it into the rack.
37  Cook [4] cooks a burger and puts it into the rack.
38  Cashier [1] takes a burger from the rack to the customer.
39  Customer [7] receives the burger from cashier [1] and leaves.
40  Cook [1] cooks a burger and puts it into the rack.
41  Cook [3] cooks a burger and puts it into the rack.
42  Customer [5] is being served by cashier [1].
43  Cashier [2] accepts an order.
44  Cashier [2] takes a burger from the rack to the customer.
45  Customer [6] receives the burger from cashier [2] and leaves.
46  Cook [2] cooks a burger and puts it into the rack.
47  Cook [1] cooks a burger and puts it into the rack.
48  Cashier [1] accepts an order.
49  Cook [4] cooks a burger and puts it into the rack.
50  Customer [4] is being served by cashier [2].
51  Cook [4] cooks a burger and puts it into the rack.
52  Cook [2] cooks a burger and puts it into the rack.
53  Cook [3] cooks a burger and puts it into the rack.
54  Cook [4] cooks a burger and puts it into the rack.
55  Cashier [1] takes a burger from the rack to the customer.
56  Customer [5] receives the burger from cashier [1] and leaves.
57  Cook [1] cooks a burger and puts it into the rack.
58  Cook [2] cooks a burger and puts it into the rack.
59  Customer [2] is being served by cashier [1].
60  Cashier [2] accepts an order.
61  Cook [3] cooks a burger and puts it into the rack.
62  Cook [1] cooks a burger and puts it into the rack.
63  Cashier [2] takes a burger from the rack to the customer.
64  Customer [4] receives the burger from cashier [2] and leaves.
65  Cook [4] cooks a burger and puts it into the rack.
66  Cook [2] cooks a burger and puts it into the rack.
67  Cashier [1] accepts an order.
68  Customer [10] is being served by cashier [2].
69  Cashier [1] takes a burger from the rack to the customer.
```

```
70     Customer [2] receives the burger from cashier [1] and leaves.
71     Customer [9] is being served by cashier [1].
72     Cashier [2] accepts an order.
73     Cashier [1] accepts an order.
74     Cook [3] cooks a burger and puts it into the rack.
75     Cashier [2] takes a burger from the rack to the customer.
76     Customer [10] receives the burger from cashier [2] and leaves.
77     Customer [8] is being served by cashier [2].
78     Cashier [1] takes a burger from the rack to the customer.
79     Customer [9] receives the burger from cashier [1] and leaves.
80     Cook [1] cooks a burger and puts it into the rack.
81     Cashier [2] accepts an order.
82     Cashier [2] takes a burger from the rack to the customer.
83     Customer [8] receives the burger from cashier [2] and leaves.
84
85     All customers have left, DONE.
86     Cook [2] cooks a burger and puts it into the rack.
```

In this case, there are more customers than cashiers and rack size greater than the number of customers.

The analysis is similar to the previous case. We can see customer waiting for the cashier. It is obvious that the output is correct.

1.5.3 Case 3

```
1  ./BBC 4 3 4 1
2
3  Cooks [4], Cashiers [3], Customers [4], Rack[1]
4  Begin run.
5
6  Customer [1] comes.
7  Customer [1] is being served by cashier [1].
8  Customer [2] comes.
9  Customer [4] comes.
10 Customer [3] comes.
11 Cashier [1] accepts an order.
12 Customer [2] is being served by cashier [2].
13 Cook [1] cooks a burger and puts it into the rack.
14 Cashier [2] accepts an order.
15 Customer [4] is being served by cashier [3].
```

```
Cashier [1] takes a burger from the rack to the customer.
Customer [1] receives the burger from cashier [1] and leaves.
Cook [2] cooks a burger and puts it into the rack.
Cashier [2] takes a burger from the rack to the customer.
Customer [2] receives the burger from cashier [2] and leaves.
Cook [3] cooks a burger and puts it into the rack.
Cashier [3] accepts an order.
Customer [3] is being served by cashier [1].
Cashier [1] accepts an order.
Cashier [3] takes a burger from the rack to the customer.
Customer [4] receives the burger from cashier [3] and leaves.
Cook [4] cooks a burger and puts it into the rack.
Cashier [1] takes a burger from the rack to the customer.
Customer [3] receives the burger from cashier [1] and leaves.
Cook [1] cooks a burger and puts it into the rack.

All customers have left, DONE.
```

In this case, there are more cooks than the rack size. We can obviously see that the cook is blocked by the rack size. And cashier is waiting for the burger. So the output is correct.

1.5.4 Case 4

```
    ./BBC 0 2 3 10

    Invalid input.

    ./BBC 105 1 200 299

    Too many cooks/cashiers/customers/racks. Risks of threads number overflow.
```

In this case, we test for edge cases. We do not allow non-postive number of cooks, cashiers, customers and racks. We also prohibit the number of threads to be too large, which may lead to thread number overflow. The threshold is set to 100 in `MAXSIZE`.

2 Santa Claus problem

2.1 Description

We need to design a C solution for the Santa Claus synchronization problem.

2.2 Semaphore, Mutex, and Condition Variable

There are quite a lot of semaphores, mutexes and condition variables in this problem. I will expand on them respectively and explain their usage.

- `santaSem = 0`. It is used to wake up Santa Claus. When there are 3 elves or 9 reindeers, they will `sem_post` this semaphore to wake up Santa Claus.
- `reindeerSem = 0`. It is used for currently waiting reindeers. They use this semaphore to wait to be hitched. When santa claus is ready, he will `sem_post` this semaphore for 9 times to wake up the reindeers.
- `elfSem = 0`. It is used for currently waiting elves. They use this semaphore to wait for help. When santa claus is ready, he will `sem_post` this semaphore for 3 times to wake up the elves.
- `elfallow = 3`. When there are 3 elves waiting, any other coming elves should be blocked.
- `reindeerEnd = 0`. It is used to show whether the reindeers have finished their task. It will be `sem_post` by the santa claus after he has come back from the ride. Only when santa claus manually `sem_wait` this semaphore can the reindeers go back.
- `mutex = 1`. It is the mutex for visiting the global variables.
- `reindeerCount = 0`. It is used to count how many reindeers are currently waiting. When it reaches 9, the santa claus will be waken up.
- `elfCount = 0`. It is used to count how many elves are currently waiting. When it reaches 3, the santa claus will be waken up.
- `sleighCount = 0`. It is used to count how many times the santa claus is waken up by reindeers.
- `helpCount = 0`. It is used to count how many times the santa claus is waken up by elves.

- `hitchedCount = 0`. It is used to count how many reindeers have been hitched. Only when it reaches 9 can the santa claus go to the ride.
- `elvesBuf[3]`. It is used to store the id of the elves who are currently being helped by santa claus.
- `flag_r`. It is used to block reindeer from waiting right after a ride is finished. Take this as an example. When santa claus finished a ride, all 9 reindeers will be released and go back. No reindeers should wait for the next ride before all the reindeers have gone back. But due to the concurrency of thread, there may be some reindeers go back again to wait for the santa claus when some reindeers are still waiting to go back to their inhabitants.
- `flag_e`. It is quite silimar to `flag_r`. When the santa claus have finished helping 3 elves, no elves are allowed to be helped until the 3 elves being helped have left. It is actually redundant, since we use `elfallow` to block the elves. But is is a good practice to use this flag to avoid some potential bugs.
- `flag_exit`. It is used to tell the elves and reindeers who are still blocked by semaphores to exit when end requirements are satisfied.

2.3 Design

There are 3 types of threads in this problem, namely santa claus, reindeer and elf. They are all represented as a function.

We first look at reindeer function. Reindeer runs in infinite loop. At the beginning of the loop, when the end requirements are satisfied, they will exit. When `flag_r` is 0, it means that santa claus has finished the ride, and the 9 reindeers are being released. So the current reindeer should not come to wait (it should iterate in a while loop) until all previous reindeers have been released. After that, they will rest for random time and then come to wait to be hitched. Then it will get the mutex, and add the `reindeerCount`. If the `reindeerCount` reaches 9, it will wake up the santa claus. Then it will wait for santa claus to signal `reindeerSem` to hitch them. I add a judgment after this semaphore. If the `flag_exit` is 1, then the reindeer should exit. That is why we need this flag. (Because when end requirements are satisfied, some reindeers may be blocked by `reindeerSem`). Then the reindeer will get hitched and add the `hitchedCount`. If the `hitchedCount` reaches 9, the santa claus will go for a ride. After that, it will wait for the santa claus to end the ride. At the end, it will be released and decrement the `reindeerCount` by 1.

Then we take a look at elf function. The starting part is same as the reindeer. They will exit when the end requirements are satisfied. They should iterate until the `flag_e` is

0, which means that the previous 3 elves being helped have left. Then they will rest for random time and come to wait for help. It will first wait for `elfallow`, which will be 0 if there have been 3 elves waiting. After that i add another judgment for `flag_exit` in case of the same situation as reindeer. Then it will get the mutex, increment the `elfCount` by 1 and record its id in the `elvesBuf`. If the `elfCount` reaches 3, it will wake up the santa claus. Then it will wait for the santa claus to signal `elfSem`, which is the sign of the end of help. After that, it will get the mutex and decrement the `elfCount` by 1.

Finally, let's dive into the santa claus function. At the beginning of the function, it is still the judgment for the end requirements. If satisfied, the santa claus will set the `flag_exit` to 1 and `sem_post {elfSem, elfallow, reindeerSem}`, thus reindeers and elves will exit decently. Then he will wait for the reindeers or elves to wake him up. If waken up by reindeers, it will first set the `flag_r` to 0 (the purpose has been stated above). Then it will post the `reindeerSem` for 9 times to prepare a sleigh and wait for the reindeers to be hitched. When all the reindeers are hitched, he will go out for a ride, sending gifts to children all around the world. After the ride, he will post the `reindeerEnd` to let the reindeers go back. After all the reindeers have gone back, this round has finished. He will then set the `flag_r` to 1, and wait for the next round. If waken up by elves, likewise, he will set the `flag_e` to 0, and then start to help the elves. I use random sleep to simulate this process. After that, he will `sem_post` the `elfSem` for 3 times to end the help. He will set the `flag_e` to 1 until all three elves have left, namely `elfCount == 0`. At last, he will `sem_post` the `elfallow` to let other elves come to wait.

2.4 Verification

I set the requirements of the end to 3 rounds of reindeers and 3 rounds of elves (because 30 rounds will be too long). The results are as follows:

```
Reindeer 6 is waiting for the santa claus.
Elf 0 is waiting for the santa claus.
Elf 4 is waiting for the santa claus.
Elf 8 is waiting for the santa claus.
Santa Claus is waken up.
Santa Claus: helping elves.
Sleigh count : 0, Help elves count : 1.
Elf 0 is being helped.
Elf 4 is being helped.
Elf 8 is being helped.
Reindeer 2 is waiting for the santa claus.
Santa Claus finishes helping the elves.
```

```
13 Elf 7 is waiting for the santa claus.
14 Reindeer 7 is waiting for the santa claus.
15 Reindeer 3 is waiting for the santa claus.
16 Reindeer 1 is waiting for the santa claus.
17 Reindeer 4 is waiting for the santa claus.
18 Elf 2 is waiting for the santa claus.
19 Elf 3 is waiting for the santa claus.
20 Santa Claus is waken up.
21 Santa Claus: helping elves.
22 Sleigh count : 0, Help elves count : 2.
23 Elf 7 is being helped.
24 Elf 2 is being helped.
25 Elf 3 is being helped.
26 Reindeer 5 is waiting for the santa claus.
27 Reindeer 0 is waiting for the santa claus.
28 Reindeer 8 is waiting for the santa claus.
29 Santa Claus finishes helping the elves.
30 Elf 6 is waiting for the santa claus.
31 Santa Claus is waken up.
32 Santa Claus: preparing sleigh.
33 Sleigh count : 1, Help elves count : 2.
34 Reindeer 2 getting hitched.
35 Reindeer 6 getting hitched.
36 Reindeer 1 getting hitched.
37 Reindeer 4 getting hitched.
38 Reindeer 0 getting hitched.
39 Reindeer 8 getting hitched.
40 Elf 9 is waiting for the santa claus.
41 Elf 8 is waiting for the santa claus.
42 Reindeer 3 getting hitched.
43 Reindeer 7 getting hitched.
44 Reindeer 5 getting hitched.
45 Santa Claus together with 9 reindeers are going to send gifts to children
       around the world.
46 Santa Claus finishes sending gift to children.
47 Santa Claus is waken up.
48 Santa Claus: helping elves.
49 Sleigh count : 1, Help elves count : 3.
50 Elf 6 is being helped.
```

```
Elf 9 is being helped.
Elf 8 is being helped.
Reindeer 4 is waiting for the santa claus.
Santa Claus finishes helping the elves.
Elf 1 is waiting for the santa claus.
Elf 0 is waiting for the santa claus.
Elf 5 is waiting for the santa claus.
Santa Claus is waken up.
Santa Claus: helping elves.
Sleigh count : 1, Help elves count : 4.
Elf 1 is being helped.
Elf 0 is being helped.
Elf 5 is being helped.
Reindeer 1 is waiting for the santa claus.
Reindeer 2 is waiting for the santa claus.
Reindeer 6 is waiting for the santa claus.
Reindeer 5 is waiting for the santa claus.
Reindeer 7 is waiting for the santa claus.
Reindeer 0 is waiting for the santa claus.
Reindeer 8 is waiting for the santa claus.
Reindeer 3 is waiting for the santa claus.
Santa Claus finishes helping the elves.
Elf 4 is waiting for the santa claus.
Elf 2 is waiting for the santa claus.
Elf 7 is waiting for the santa claus.
Santa Claus is waken up.
Santa Claus: preparing sleigh.
Sleigh count : 2, Help elves count : 4.
Reindeer 4 getting hitched.
Reindeer 2 getting hitched.
Reindeer 6 getting hitched.
Reindeer 1 getting hitched.
Reindeer 5 getting hitched.
Reindeer 7 getting hitched.
Reindeer 0 getting hitched.
Reindeer 8 getting hitched.
Reindeer 3 getting hitched.
Santa Claus together with 9 reindeers are going to send gifts to children
    around the world.
```

```
89 Santa Claus finishes sending gift to children.
90 Santa Claus is waken up.
91 Santa Claus: helping elves.
92 Sleigh count : 2, Help elves count : 5.
93 Elf 4 is being helped.
94 Elf 2 is being helped.
95 Elf 7 is being helped.
96 Reindeer 5 is waiting for the santa claus.
97 Reindeer 3 is waiting for the santa claus.
98 Reindeer 4 is waiting for the santa claus.
99 Reindeer 2 is waiting for the santa claus.
100 Santa Claus finishes helping the elves.
101 Reindeer 0 is waiting for the santa claus.
102 Elf 3 is waiting for the santa claus.
103 Elf 8 is waiting for the santa claus.
104 Elf 9 is waiting for the santa claus.
105 Santa Claus is waken up.
106 Santa Claus: helping elves.
107 Sleigh count : 2, Help elves count : 6.
108 Elf 3 is being helped.
109 Elf 8 is being helped.
110 Elf 9 is being helped.
111 Santa Claus finishes helping the elves.
112 Elf 0 is waiting for the santa claus.
113 Elf 5 is waiting for the santa claus.
114 Elf 7 is waiting for the santa claus.
115 Santa Claus is waken up.
116 Santa Claus: helping elves.
117 Sleigh count : 2, Help elves count : 7.
118 Elf 0 is being helped.
119 Elf 5 is being helped.
120 Elf 7 is being helped.
121 Reindeer 1 is waiting for the santa claus.
122 Reindeer 7 is waiting for the santa claus.
123 Reindeer 8 is waiting for the santa claus.
124 Reindeer 6 is waiting for the santa claus.
125 Santa Claus finishes helping the elves.
126 Elf 6 is waiting for the santa claus.
127 Santa Claus is waken up.
```

```
Santa Claus: preparing sleigh.
Sleigh count : 3, Help elves count : 7.
Reindeer 5 getting hitched.
Reindeer 1 getting hitched.
Reindeer 3 getting hitched.
Reindeer 2 getting hitched.
Reindeer 4 getting hitched.
Reindeer 7 getting hitched.
Reindeer 6 getting hitched.
Reindeer 8 getting hitched.
Elf 4 is waiting for the santa claus.
Elf 1 is waiting for the santa claus.
Elf 7 DONE.
Elf 0 DONE.
Elf 8 DONE.
Reindeer 0 getting hitched.
Santa Claus together with 9 reindeers are going to send gifts to children
    around the world.
Santa Claus finishes sending gift to children.
Reindeer 5 DONE.
Reindeer 1 DONE.
Reindeer 3 DONE.
Reindeer 2 DONE.
Reindeer 4 DONE.
Reindeer 7 DONE.
Reindeer 6 DONE.
Reindeer 8 DONE.
Reindeer 0 DONE.
Santa Claus DONE.
Elf 6 DONE.
Elf 3 DONE.
Elf 1 DONE.
Elf 4 DONE.
Elf 2 DONE.
Elf 9 DONE.
Elf 5 DONE.
```

We can easily check the correctness of the output. I have printed sufficient information to indication the current status of the threads. We can also see the interleave of the

threads. For example, line1 and line2, line11 and line13, line40 and line41, and so on. I set different output different color, and it will be included in the `typescript.md` file.