作业 5 多线程问题

解雲暄 3190105871

索引

1 问题描述

问题要求模拟一个加油站，有 1 个排队序列和 2 个加油枪；如果加油枪空闲，加油工会召唤队列中队头汽车过来加油。如果排队长度超过 5，则到达的汽车会自行离开。

该问题的核心难点在于，车的到来以及两个加油枪都会对同一个队列做操作，我们需要采取一些同步方式维持这个队列的一致性。

2 代码设计

我们建立 3 个子线程，2 个用于模拟加油枪，1 个用于模拟汽车的到来。为了防止队列同时被多个进程操控，我们使用一个信号量 `queueSemaphore` 进行同步。为了便于展示，我们讨论省略了输出的各线程的 handler 的操作：

```
1 // station thread handler
```

```
2  DWORD WINAPI station(LPVOID param) {
3      int stationIndex = *((int *) param);
4      int oiledCar = -1;
5      srand(time(NULL));
6      // run permantly
7      while (true) {
8          WaitForSingleObject(queueSemaphore, INFINITE);
9          // get oiled car number
10         if (!cars.empty()) {
11             oiledCar = cars.front();
12             cars.pop();
13         }
14         ReleaseSemaphore(queueSemaphore, 1, NULL);
15
16         // oiling
17         if (oiledCar != -1)
18             goToOil(stationIndex, oiledCar, rand() % 2000 + 5000);
19
20         oiledCar = -1;
21     }
22 }
```

这是模拟加油枪的代码。在第 8 行到 14 行，我们用 queueSemaphore 进行了保护，因为其中我们要从队列中取出队头（如果有）。在第 18 行，我们对选出的车进行加油，消耗的时间是 5~7 秒中的随机数。

```
1  DWORD WINAPI arrival(LPVOID param) {
2      srand(time(NULL));
3      int nextArrival;
4      int carIndex = 0;
5      // run permanently
6      while (true) {
7          // wait a random duration for the next arrival
8          nextArrival = rand() % 3000;
9          Sleep(nextArrival);
10
11         // next arrived car index
12         ++carIndex;
13
14         WaitForSingleObject(queueSemaphore, INFINITE);
15         // if queue size <= 5, the arriving car will begin to queue
16         if (cars.size() <= 5) {
17             cars.push(carIndex);
18         }
```

```
19          // else, the car will leave
20          ReleaseSemaphore(queueSemaphore, 1, NULL);
21      }
22  }
```

这是模拟汽车到来的代码，第 8~9 行可以看到，每 0~3 秒就会到来一辆车。在 14~20 行，我们也用了 queueSemaphore 进行保护，因为我们会对队列进行操作。

实际上，上述线程还会输出一些记录信息；在此略去。在第 3 节结果测试中可以看到输出结果，在第 4 节的讨论中可以看到设计过程中出现的问题以及一些具体的优化思路。

主函数的代码如下：

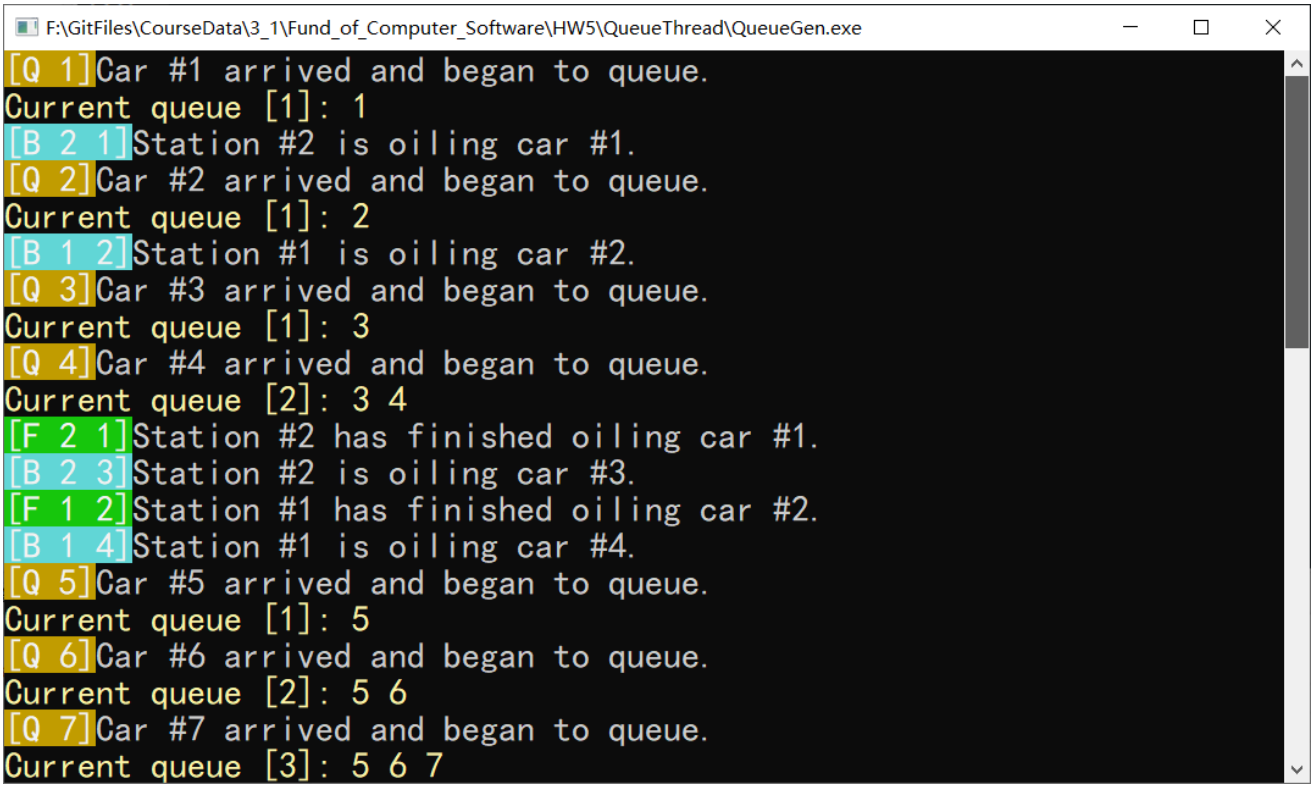
```
1   int main() {
2       queueSemaphore  = CreateSemaphore(NULL, 1, 1, NULL);
3       stdoutSemaphore = CreateSemaphore(NULL, 1, 1, NULL);
4
5       carArrival = CreateThread(NULL, 0, arrival, NULL, 0, NULL);
6       int stationNum1 = 1;
7       station1 = CreateThread(NULL, 0, station, &stationNum1, 0, NULL);
8       int stationNum2 = 2;
9       station2 = CreateThread(NULL, 0, station, &stationNum2, 0, NULL);
10
11      // for running threads
12      while(true);
13
14      return 0;
15  }
```

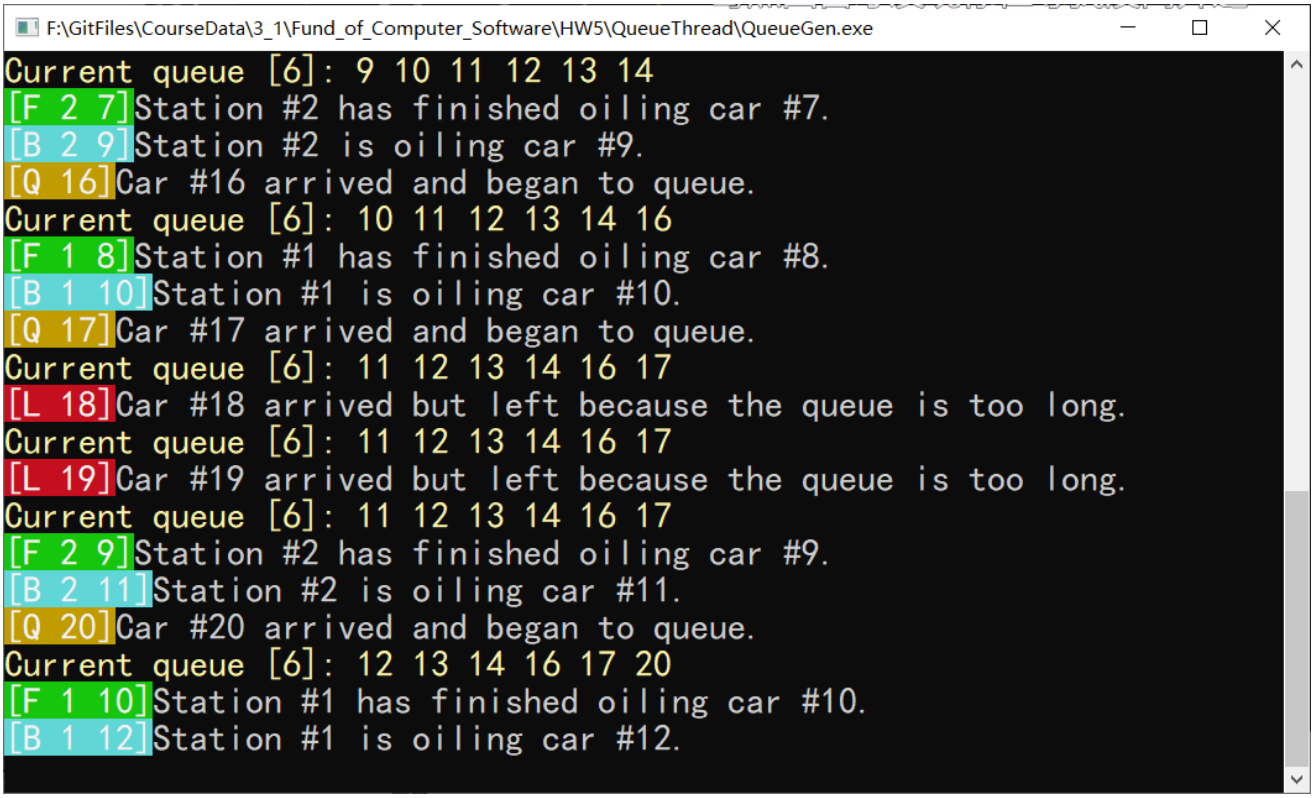
实际上，我们也可以让 carArrival 放在主线程进行操作，建立两个子线程模拟 station 即可；因为此时的主线程实际上是在 busy-waiting 的。

更多部分的代码可以查看源文件；重点部分已经用注释标明。

3 结果测试

- 为了方便观看结果，我们给每一个时间进行一个打印，并在开头用不同颜色的**摘要**进行描述：
    - 我们使用 **黄色底色** 表示车到来且开始排队，如 `[Q 1]`。
    - 使用 **红色底色** 表示车到来但因为队伍长度已经超过 5 所以离开（如图 2），如 `[L 18]`。
    - 在上述两种情况后，我们都会用 **黄色字体** 打印当前的队列长度和队列中车的序号。
    - 使用 **蓝色底色** 表示某个站开始给某辆车加油，如 `[B 2 1]`。
    - 使用 **绿色底色** 表示某个站已经完成了给当前车的加油，如 `[F 2 1]`。

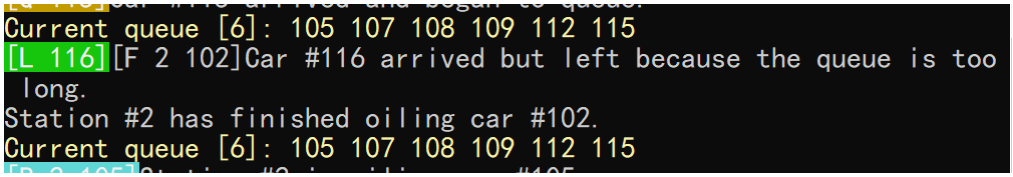




可以看到，由于车到来的时间间隔和加油的用时都是一定范围内随机的，因此输出也产生了各种形式。但是经过充分的检验，我们设计的程序可以保证数据的一致性，同时也能够良好地展示各个事件的发生以及当前系统的状态。

4 讨论

我们初期的程序设计中，实际上是将输出和队列操作分开做的；只有涉及数据同步性的队列操作才用信号量进行了保护。但是实际上我们会发现这样的情况：



即，虽然程序做出的动作是正确的，但是由于有可能出现两个线程同时输出的情况，因此输出的结果是有失同步性的。因此我们在后面的程序设计中将输出也放入了信号量部分。

这样的决定会使得 CPU 利用率有所降低，因为我们实际上会让 CPU 等待 I/O 的时间。为了优化，我们引入了第二个信号量。这样也是合理的，因为我们将队列和 stdout 这两个资源分别用一个信号量去控制，可以在保证同步性的前提下减少空闲时间。