

操作系统大作业实验 1 报告

高艺轩 2022010639

1 实验目的

1. 通过对进程间通信同步/互斥问题的编程实现，加深理解信号量和 P、V 操作的原理；
2. 对 Windows 或 Linux 涉及的几种互斥、同步机制有更进一步的了解；
3. 熟悉 Windows 或 Linux 中定义的与互斥、同步有关的函数。

2 实验题目

本次实验选择的题目是“银行柜员服务问题”。

3 问题描述

银行有 n 个柜员负责为顾客服务，顾客进入银行先取一个号码，然后等着叫号。当某个柜员空闲下来，就叫下一个号。

编程实现该问题，用 P、V 操作实现柜员和顾客的同步。

4 思路与程序结构

这个问题的场景实际上就是一个典型的生产者-消费者问题。把顾客看作资源，柜员看作消费者，那么顾客在不停地到来就是生产者生产的过程，柜员服务即为消费者。因此所要竞争的主要内容即为“正在等待的顾客”，由柜员对顾客数执行 P，由到来的顾客对顾客数执行 V；另外，为了存储正在等待的顾客，用于在顾客线程和柜员线程中交换对应的顾客信息，需要维护一个队列以及对应的互斥锁，在访问前使用 P 原语锁定，访问结束后使用 V 原语释放。

为了便于观察现象和验证程序运行，本次实验中我将时间刻间隔设置为 200 ms。

具体来讲，在全局设置代表待服务的顾客数量的信号量：

```

35 // BEGIN_LST_COUNT_SEM
36 std::counting_semaphore<100> customer_num_sem{0};
37 // END_LST_COUNT_SEM

```

同时设置一个正在等待服务的客户队列，并用一个互斥锁保护它：

```

30 // BEGIN_LST_CUS_Q
31 std::mutex customer_q_mtx;
32 std::deque<Customer> customer_q;
33 // END_LST_CUS_Q

```

在设置好全局变量后，分别实现柜员和顾客。对于柜员，每到一个时间刻，如果正在服务的客户还没有服务完，那么就继续服务；如果处于空闲状态，则尝试执行P(customer_num_sem)获得一个新的客户。如果能获得客户，则进一步锁定客户队列，取出队头进行服务，再解锁客户队列。伪代码如下：

Algorithm 1: Server Procedure

Function Teller():

```

    while true do
        P(customers_num_sem);
        P(mutex);
        customer_number ← dequeue(customer_queue);
        V(mutex);
        serve(customer_number);

```

实际实现的代码：

```

46 // BEGIN_LST_TELLER
47 void teller_thread(
48     int id, std::chrono::time_point<std::chrono::steady_clock> start_tick) {
49     std::ostream(std::cout)
50         << "Teller id " << id << " started" << std::endl;
51     std::chrono::time_point curr_tick = start_tick;
52     unsigned int tick_count = 0;
53     bool is_serving = false;
54     std::chrono::time_point<std::chrono::steady_clock> start_serving_time;
55     struct Customer curr_customer;
56     while (bank_open) {
57         if (is_serving) {
58             if ((std::chrono::duration_cast<std::chrono::milliseconds>(
59                 (curr_tick - start_serving_time) / TIME_GRAN_MSEC))
60                 .count() ≤ curr_customer.service_time) {

```

```

61         // We are still serving, do nothing
62         std::ostream(std::cout) << "Teller id " << id << " is serving
           customer " << curr_customer.id << " at tick " << tick_count
           << std::endl;
63         goto next_tick;
64     } else {
65         // We have just finished serving.
66         std::ostream(std::cout) << "Teller id " << id << " finished
           serving customer " << curr_customer.id << " at tick " <<
           tick_count << std::endl;
67         is_serving = false;
68     }
69 }
70
71 // Try to acquire a new customer
72 if (customer_num_sem.try_acquire()) {
73     // Successfully acquired a customer
74     std::lock_guard<std::mutex> lock(customer_q_mtx);
75     if (!customer_q.empty()) {
76         curr_customer = customer_q.front();
77         customer_q.pop_front();
78
79         std::ostream(std::cout)
80             << "Teller id " << id << " is now serving customer "
81             << curr_customer.id << " for " << curr_customer.service_time
82             << " ticks at tick " << tick_count << std::endl;
83
84         is_serving = true;
85         start_serving_time = curr_tick;
86     }
87 } else {
88     std::ostream(std::cout)
89         << "Teller id " << id << " is idle at tick " << tick_count << std
           ::endl;
90 }
91
92 next_tick:
93     ++tick_count;
94     curr_tick += std::chrono::milliseconds(TIME_GRAN_MSEC);
95     std::this_thread::sleep_until(curr_tick);
96 }
97 std::ostream(std::cout)
98     << "Teller id " << id << " closing" << std::endl;

```

```

99 | }
100 | // END_LST_TELLER

```

对于客户线程，首先不断等待直到到达设定的进入时间，首先锁定客户队列，将自己加入，同时执行V(customer_num_sem)来向柜员线程提示有一个新的客户产生。伪代码如下：

Algorithm 2: Customer Procedure

```

Function Customer(info):
    sleep_until(arrival_time);
    P(mutex);
    enqueue(customer_queue, info);
    V(mutex);
    V(customers_num_sem);

```

实际实现的代码：

```

102 | // BEGIN_LST_CUSTOMER
103 | void customer_thread(
104 |     CustomerInfo info,
105 |     std::chrono::time_point<std::chrono::steady_clock> start_tick) {
106 |
107 |     Customer cus{.id = info.id, .service_time = info.service_time};
108 |
109 |     std::chrono::time_point curr_tick = start_tick;
110 |     unsigned int tick_count = 0;
111 |     std::chrono::time_point target_tick = start_tick + std::chrono::
112 |         milliseconds(TIME_GRAN_MSEC) * info.arrival_time;
113 |     while (curr_tick < target_tick) {
114 |         if (!bank_open) {
115 |             return;
116 |         }
117 |         ++tick_count;
118 |         curr_tick += std::chrono::milliseconds(TIME_GRAN_MSEC);
119 |         std::this_thread::sleep_until(curr_tick);
120 |     }
121 |
122 |     // Wake up and push self into queue, then V(customer_num_sem)
123 |     std::osyncstream(std::cout) << "Customer id " << info.id << " arrived at
124 |         tick " << tick_count << std::endl;
125 |     {
126 |         std::lock_guard<std::mutex> lock(customer_q_mtx);
127 |         customer_q.push_back(cus);

```

```

126     }
127     customer_num_sem.release();
128 }
129 // END_LST_CUSTOMER

```

在实现了客户与柜员时，main函数只需要读取配置文件、创建所有的柜员和客户进程即可：

```

131 // BEGIN_LST_MAIN
132 int main(int argc, char *argv[]) {
133     if (argc != 2) {
134         std::cout << "Invalid commandline arg, expected filename";
135         return 1;
136     }
137
138     std::signal(SIGINT, signal_handler);
139
140     // Read config from file
141     std::vector<CustomerInfo> customer_infos;
142
143     std::ifstream file(argv[1]);
144     if (!file.is_open()) {
145         std::cerr << "Cannot open file: " << argv[1] << std::endl;
146         return 1;
147     }
148
149     std::string line;
150     while (std::getline(file, line)) {
151         std::istringstream iss(line);
152         CustomerInfo cus;
153         if (iss >> cus.id >> cus.arrival_time >> cus.service_time) {
154             customer_infos.push_back(cus);
155         } else {
156             std::cerr << "Invalid line: " << line << std::endl;
157         }
158     }
159
160     file.close();
161
162     bank_open = true;
163     std::chrono::time_point start_tick = std::chrono::steady_clock::now();
164
165     // Set up tellers
166     std::vector<std::thread> tellers;

```

```

167
168     for (int i = 0; i < TELLER_NUM; ++i) {
169         try {
170             tellers.emplace_back(teller_thread, i, start_tick);
171         }
172         catch (const std::system_error &e) {
173             std::cerr << "Failed to create teller thread, i = " << i << ": "
174                 << e.what() << std::endl;
175         }
176     }
177
178     // Set up customers
179     std::vector<std::thread> customers;
180     for (auto cu : customer_infos) {
181         try {
182             customers.emplace_back(customer_thread, cu, start_tick);
183         } catch (const std::system_error &e) {
184             std::cerr << "Failed to create customer thread, id = " << cu.id << "
185                 << e.what() << std::endl;
186         }
187     }
188
189     // Wait for all children to shutdown (by ctrl-c handler)
190     for (auto &t: customers) {
191         if (t.joinable()) {
192             t.join();
193         }
194     }
195
196     for (auto &t : tellers) {
197         if (t.joinable()) {
198             t.join();
199         }
200     }
201
202     return 0;
203 }
204 // END_LST_MAIN

```

5 程序运行情况

使用指导书中给出的测试数据和自己编写的测试数据都进行了测试。使用指导书中的输入：

```
1 1 1 10
2 2 5 2
3 3 6 3
```

得到的输出为

```
1 Teller id 0 started
2 Teller id 1 started
3 Teller id 1 is idle at tick 0
4 Teller id 0 is idle at tick 0
5 Teller id 2 started
6 Teller id 2 is idle at tick 0
7 Teller id 0 is idle at tick 1
8 Teller id 2 is idle at tick 1
9 Teller id 1 is idle at tick 1
10 Customer id 1 arrived at tick 1
11 Teller id 0 is now serving customer 1 for 10 ticks at tick 2
12 Teller id 1 is idle at tick 2
13 Teller id 2 is idle at tick 2
14 Teller id 0 is serving customer 1 at tick 3
15 Teller id 1 is idle at tick 3
16 Teller id 2 is idle at tick 3
17 Teller id 0 is serving customer 1 at tick 4
18 Teller id 1 is idle at tick 4
19 Teller id 2 is idle at tick 4
20 Teller id 0 is serving customer 1 at tick 5
21 Customer id 2 arrived at tick 5
22 Teller id 2 is idle at tick 5
23 Teller id 1 is idle at tick 5
24 Teller id 0 is serving customer 1 at tick 6
25 Teller id 2 is now serving customer 2 for 2 ticks at tick 6
26 Customer id 3 arrived at tick 6
27 Teller id 1 is idle at tick 6
28 Teller id 0 is serving customer 1 at tick 7
29 Teller id 1 is now serving customer 3 for 3 ticks at tick 7
30 Teller id 2 is serving customer 2 at tick 7
31 Teller id 0 is serving customer 1 at tick 8
32 Teller id 1 is serving customer 3 at tick 8
33 Teller id 2 is serving customer 2 at tick 8
34 Teller id 0 is serving customer 1 at tick 9
```

```

35 Teller id 1 is serving customer 3 at tick 9
36 Teller id 2 finished serving customer 2 at tick 9
37 Teller id 2 is idle at tick 9
38 Teller id 1 is serving customer 3 at tick 10
39 Teller id 0 is serving customer 1 at tick 10
40 Teller id 2 is idle at tick 10
41 Teller id 0 is serving customer 1 at tick 11
42 Teller id 1 finished serving customer 3 at tick 11
43 Teller id 2 is idle at tick 11
44 Teller id 1 is idle at tick 11
45 Teller id 1 is idle at tick 12
46 Teller id 2 is idle at tick 12
47 Teller id 0 is serving customer 1 at tick 12
48 Teller id 0 finished serving customer 1 at tick 13
49 Teller id 0 is idle at tick 13
50 Teller id 2 is idle at tick 13
51 Teller id 1 is idle at tick 13
52 Teller id 0 is idle at tick 14
53 Teller id 1 is idle at tick 14
54 Teller id 2 is idle at tick 14
55 Teller id 0 is idle at tick 15
56 Teller id 2 is idle at tick 15
57 Teller id 1 is idle at tick 15
58 Teller id 0 is idle at tick 16
59 Teller id 1 is idle at tick 16
60 Teller id 2 is idle at tick 16
61 Teller id 2 is idle at tick 17
62 Teller id 0 is idle at tick 17
63 Teller id 1 is idle at tick 17
64 Teller id 0 is idle at tick 18
65 Teller id 1 is idle at tick 18
66 Teller id 2 is idle at tick 18
67 ^C
68 Caught SIGINT, shutting down...
69 Teller id 0 closing
70 Teller id 1 closing
71 Teller id 2 closing

```

再使用自己编写的测试数据：

```

1 1 1 10
2 2 2 10
3 3 3 10
4 4 4 10

```

```
5 | 5 5 5
6 | 6 6 7
7 | 8 7 10
```

得到的输出为

```
1 | Teller id 1 started
2 | Teller id 0 started
3 | Teller id 2 started
4 | Teller id 1 is idle at tick 0
5 | Teller id 2 is idle at tick 0
6 | Teller id 0 is idle at tick 0
7 | Teller id 0 is idle at tick 1
8 | Teller id 2 is idle at tick 1
9 | Teller id 1 is idle at tick 1
10 | Customer id 1 arrived at tick 1
11 | Teller id 0 is now serving customer 1 for 10 ticks at tick 2
12 | Teller id 1 is idle at tick 2
13 | Teller id 2 is idle at tick 2
14 | Customer id 2 arrived at tick 2
15 | Teller id 1 is now serving customer 2 for 10 ticks at tick 3
16 | Customer id 3 arrived at tick 3
17 | Teller id 0 is serving customer 1 at tick 3
18 | Teller id 2 is idle at tick 3
19 | Teller id 0 is serving customer 1 at tick 4
20 | Teller id 2 is now serving customer 3 for 10 ticks at tick 4
21 | Teller id 1 is serving customer 2 at tick 4
22 | Customer id 4 arrived at tick 4
23 | Teller id 0 is serving customer 1 at tick 5
24 | Teller id 2 is serving customer 3 at tick 5
25 | Teller id 1 is serving customer 2 at tick 5
26 | Customer id 5 arrived at tick 5
27 | Customer id 6 arrived at tick 6
28 | Teller id 1 is serving customer 2 at tick 6
29 | Teller id 0 is serving customer 1 at tick 6
30 | Teller id 2 is serving customer 3 at tick 6
31 | Teller id 1 is serving customer 2 at tick 7
32 | Teller id 0 is serving customer 1 at tick 7
33 | Customer id 8 arrived at tick 7
34 | Teller id 2 is serving customer 3 at tick 7
35 | Teller id 1 is serving customer 2 at tick 8
36 | Teller id 2 is serving customer 3 at tick 8
37 | Teller id 0 is serving customer 1 at tick 8
38 | Teller id 2 is serving customer 3 at tick 9
```

39 Teller id 1 is serving customer 2 at tick 9
40 Teller id 0 is serving customer 1 at tick 9
41 Teller id 0 is serving customer 1 at tick 10
42 Teller id 2 is serving customer 3 at tick 10
43 Teller id 1 is serving customer 2 at tick 10
44 Teller id 2 is serving customer 3 at tick 11
45 Teller id 1 is serving customer 2 at tick 11
46 Teller id 0 is serving customer 1 at tick 11
47 Teller id 1 is serving customer 2 at tick 12
48 Teller id 2 is serving customer 3 at tick 12
49 Teller id 0 is serving customer 1 at tick 12
50 Teller id 2 is serving customer 3 at tick 13
51 Teller id 1 is serving customer 2 at tick 13
52 Teller id 0 finished serving customer 1 at tick 13
53 Teller id 0 is now serving customer 4 for 10 ticks at tick 13
54 Teller id 2 is serving customer 3 at tick 14
55 Teller id 1 finished serving customer 2 at tick 14
56 Teller id 1 is now serving customer 5 for 5 ticks at tick 14
57 Teller id 0 is serving customer 4 at tick 14
58 Teller id 2 finished serving customer 3 at tick 15
59 Teller id 2 is now serving customer 6 for 7 ticks at tick 15
60 Teller id 1 is serving customer 5 at tick 15
61 Teller id 0 is serving customer 4 at tick 15
62 Teller id 2 is serving customer 6 at tick 16
63 Teller id 1 is serving customer 5 at tick 16
64 Teller id 0 is serving customer 4 at tick 16
65 Teller id 2 is serving customer 6 at tick 17
66 Teller id 0 is serving customer 4 at tick 17
67 Teller id 1 is serving customer 5 at tick 17
68 Teller id 2 is serving customer 6 at tick 18
69 Teller id 1 is serving customer 5 at tick 18
70 Teller id 0 is serving customer 4 at tick 18
71 Teller id 2 is serving customer 6 at tick 19
72 Teller id 0 is serving customer 4 at tick 19
73 Teller id 1 is serving customer 5 at tick 19
74 Teller id 0 is serving customer 4 at tick 20
75 Teller id 1 finished serving customer 5 at tick 20
76 Teller id 1 is now serving customer 8 for 10 ticks at tick 20
77 Teller id 2 is serving customer 6 at tick 20
78 Teller id 1 is serving customer 8 at tick 21
79 Teller id 0 is serving customer 4 at tick 21
80 Teller id 2 is serving customer 6 at tick 21
81 Teller id 2 is serving customer 6 at tick 22

82 Teller id 1 is serving customer 8 at tick 22
83 Teller id 0 is serving customer 4 at tick 22
84 Teller id 1 is serving customer 8 at tick 23
85 Teller id 2 finishied serving customer 6 at tick 23
86 Teller id 2 is idle at tick 23
87 Teller id 0 is serving customer 4 at tick 23
88 Teller id 1 is serving customer 8 at tick 24
89 Teller id 0 finishied serving customer 4 at tick 24
90 Teller id 2 is idle at tick 24
91 Teller id 0 is idle at tick 24
92 Teller id 1 is serving customer 8 at tick 25
93 Teller id 2 is idle at tick 25
94 Teller id 0 is idle at tick 25
95 Teller id 1 is serving customer 8 at tick 26
96 Teller id 2 is idle at tick 26
97 Teller id 0 is idle at tick 26
98 Teller id 1 is serving customer 8 at tick 27
99 Teller id 0 is idle at tick 27
100 Teller id 2 is idle at tick 27
101 Teller id 1 is serving customer 8 at tick 28
102 Teller id 2 is idle at tick 28
103 Teller id 0 is idle at tick 28
104 Teller id 1 is serving customer 8 at tick 29
105 Teller id 2 is idle at tick 29
106 Teller id 0 is idle at tick 29
107 Teller id 1 is serving customer 8 at tick 30
108 Teller id 0 is idle at tick 30
109 Teller id 2 is idle at tick 30
110 Teller id 1 finishied serving customer 8 at tick 31
111 Teller id 1 is idle at tick 31
112 Teller id 0 is idle at tick 31
113 Teller id 2 is idle at tick 31
114 Teller id 1 is idle at tick 32
115 Teller id 2 is idle at tick 32
116 Teller id 0 is idle at tick 32
117 Teller id 1 is idle at tick 33
118 Teller id 2 is idle at tick 33
119 Teller id 0 is idle at tick 33
120 Teller id 0 is idle at tick 34
121 Teller id 1 is idle at tick 34
122 Teller id 2 is idle at tick 34
123 Teller id 0 is idle at tick 35
124 Teller id 1 is idle at tick 35

```
125 | Teller id 2 is idle at tick 35
126 | ^C
127 | Caught SIGINT, shutting down...
128 | Teller id 1 closing
129 | Teller id 0 closing
130 | Teller id 2 closing
```

可以看到正确实现了客户排队与柜员依次服务。

6 思考题

1. 柜员人数和顾客人数对结果分别有什么影响?

答. 当顾客人数很少时, 几乎每时每刻都会有空闲的柜员, 因此所有的客户刚到来就能获得服务, 很少会有顾客等待, 服务的总时间基本上只取决于 (到达时间 + 服务时间) 最长的客户服务结束的时间; 当柜员人数很少时, 几乎时时刻刻所有的柜员都在工作, 因此总时间将会取决于 $\frac{\text{所有顾客所需要服务的总时间}}{\text{柜员的数量}}$ 。

2. 实现互斥的方法有哪些? 各自有什么特点? 效率如何?

答. (1) 将临界区设置禁止中断: 简单有效, 但是可靠性不高, 且不适用于多处理器;
(2) 锁变量: 出现忙等, 效率不高;
(3) 严格轮转法: 临界区外的进程也会阻塞其它进程, 效率不高;
(4) Peterson 算法: 可以正常工作, 但是也会出现忙等, 效率比较低
(5) 硬件实现: 需要硬件支持, 可以支持任意数目的进程, 但是也可能出现忙等。
(6) 信号量: 效率比较高, 但是信号量的控制不容易分析。

7 实验总结

在本次实验中, 我在实践中学习了线程间的同步和互斥, 从实践中考虑问题, 进一步让我理解了 P、V 原语工作的核心思路与应用, 也在编程过程中学习了一些 C++ 标准库的应用。

A 完整源代码

```
1 #include <atomic>
2 #include <chrono>
3 #include <csignal>
4 #include <deque>
5 #include <fstream>
6 #include <iostream>
7 #include <mutex>
8 #include <semaphore>
9 #include <sstream>
10 #include <syncstream>
11 #include <thread>
12 #include <vector>
13
14 #define TELLER_NUM 3
15 #define TIME_GRAN_MSEC 200
16
17 struct CustomerInfo {
18     unsigned int id;
19     unsigned int arrival_time;
20     unsigned int service_time;
21 };
22
23 struct Customer {
24     unsigned int id;
25     unsigned int service_time;
26 };
27
28 std::atomic<bool> bank_open{false};
29
30 // BEGIN_LST_CUS_Q
31 std::mutex customer_q_mtx;
32 std::deque<Customer> customer_q;
33 // END_LST_CUS_Q
34
35 // BEGIN_LST_COUNT_SEM
36 std::counting_semaphore<100> customer_num_sem{0};
37 // END_LST_COUNT_SEM
38
39
40 void signal_handler(int signum) {
41     // std::cout << "\nReceived signal " << signum << ", stopping ... \n";
```

```

42     write(STDOUT_FILENO, "\nCaught SIGINT, shutting down...\n", 33);
43     bank_open = false;
44 }
45
46 // BEGIN_LST_TELLER
47 void teller_thread(
48     int id, std::chrono::time_point<std::chrono::steady_clock> start_tick) {
49     std::ostream(std::cout)
50         << "Teller id " << id << " started" << std::endl;
51     std::chrono::time_point curr_tick = start_tick;
52     unsigned int tick_count = 0;
53     bool is_serving = false;
54     std::chrono::time_point<std::chrono::steady_clock> start_serving_time;
55     struct Customer curr_customer;
56     while (bank_open) {
57         if (is_serving) {
58             if ((std::chrono::duration_cast<std::chrono::milliseconds>(
59                 (curr_tick - start_serving_time) / TIME_GRAN_MSEC))
60                 .count() ≤ curr_customer.service_time) {
61                 // We are still serving, do nothing
62                 std::ostream(std::cout) << "Teller id " << id << " is serving
63                     customer " << curr_customer.id << " at tick " << tick_count
64                     << std::endl;
65                 goto next_tick;
66             } else {
67                 // We have just finished serving.
68                 std::ostream(std::cout) << "Teller id " << id << " finished
69                     serving customer " << curr_customer.id << " at tick " <<
70                     tick_count << std::endl;
71                 is_serving = false;
72             }
73         }
74
75         // Try to acquire a new customer
76         if (customer_num_sem.try_acquire()) {
77             // Successfully acquired a customer
78             std::lock_guard<std::mutex> lock(customer_q_mtx);
79             if (!customer_q.empty()) {
80                 curr_customer = customer_q.front();
81                 customer_q.pop_front();
82
83                 std::ostream(std::cout)
84                     << "Teller id " << id << " is now serving customer "

```

```

81         << curr_customer.id << " for " << curr_customer.service_time
82         << " ticks at tick " << tick_count << std::endl;
83
84         is_serving = true;
85         start_serving_time = curr_tick;
86     }
87 } else {
88     std::ostream(std::cout)
89     << "Teller id " << id << " is idle at tick " << tick_count << std
        ::endl;
90 }
91
92 next_tick:
93     ++tick_count;
94     curr_tick += std::chrono::milliseconds(TIME_GRAN_MSEC);
95     std::this_thread::sleep_until(curr_tick);
96 }
97 std::ostream(std::cout)
98     << "Teller id " << id << " closing" << std::endl;
99 }
100 // END_LST_TELLER
101
102 // BEGIN_LST_CUSTOMER
103 void customer_thread(
104     CustomerInfo info,
105     std::chrono::time_point<std::chrono::steady_clock> start_tick) {
106
107     Customer cus{.id = info.id, .service_time = info.service_time};
108
109     std::chrono::time_point curr_tick = start_tick;
110     unsigned int tick_count = 0;
111     std::chrono::time_point target_tick = start_tick + std::chrono::
        milliseconds(TIME_GRAN_MSEC) * info.arrival_time;
112     while (curr_tick < target_tick) {
113         if (!bank_open) {
114             return;
115         }
116         ++tick_count;
117         curr_tick += std::chrono::milliseconds(TIME_GRAN_MSEC);
118         std::this_thread::sleep_until(curr_tick);
119     }
120
121     // Wake up and push self into queue, then V(customer_num_sem)

```

```

122     std::ostream(std::cout) << "Customer id " << info.id << " arrived at
        tick " << tick_count << std::endl;
123     {
124         std::lock_guard<std::mutex> lock(customer_q_mtx);
125         customer_q.push_back(cus);
126     }
127     customer_num_sem.release();
128 }
129 // END_LST_CUSTOMER
130
131 // BEGIN_LST_MAIN
132 int main(int argc, char *argv[]) {
133     if (argc != 2) {
134         std::cout << "Invalid commandline arg, expected filename";
135         return 1;
136     }
137
138     std::signal(SIGINT, signal_handler);
139
140     // Read config from file
141     std::vector<CustomerInfo> customer_infos;
142
143     std::ifstream file(argv[1]);
144     if (!file.is_open()) {
145         std::cerr << "Cannot open file: " << argv[1] << std::endl;
146         return 1;
147     }
148
149     std::string line;
150     while (std::getline(file, line)) {
151         std::istringstream iss(line);
152         CustomerInfo cus;
153         if (iss >> cus.id >> cus.arrival_time >> cus.service_time) {
154             customer_infos.push_back(cus);
155         } else {
156             std::cerr << "Invalid line: " << line << std::endl;
157         }
158     }
159
160     file.close();
161
162     bank_open = true;
163     std::chrono::time_point start_tick = std::chrono::steady_clock::now();

```

```

164
165 // Set up tellers
166 std::vector<std::thread> tellers;
167
168 for (int i = 0; i < TELLER_NUM; ++i) {
169     try {
170         tellers.emplace_back(teller_thread, i, start_tick);
171     }
172     catch (const std::system_error &e) {
173         std::cerr << "Failed to create teller thread, i = " << i << ": "
174             << e.what() << std::endl;
175     }
176 }
177
178 // Set up customers
179 std::vector<std::thread> customers;
180 for (auto cu : customer_infos) {
181     try {
182         customers.emplace_back(customer_thread, cu, start_tick);
183     } catch (const std::system_error &e) {
184         std::cerr << "Failed to create customer thread, id = " << cu.id << "
185             << ": "
186             << e.what() << std::endl;
187     }
188 }
189 // Wait for all children to shutdown (by ctrl-c handler)
190 for (auto &t: customers) {
191     if (t.joinable()) {
192         t.join();
193     }
194 }
195
196 for (auto &t : tellers) {
197     if (t.joinable()) {
198         t.join();
199     }
200 }
201
202 return 0;
203 }
204 // END_LST_MAIN

```