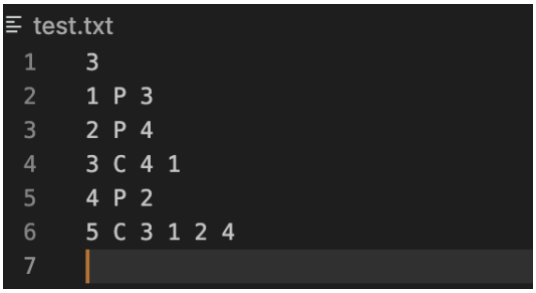
实验一、生产者和消费者问题

一、实验目的

掌握基本的同步互斥算法，理解生产者和消费者模型。

二、实验内容

创建测试用例程序：“test.txt”文件如下



第一行说明程序中设置几个缓冲区，其余每行分别描述了一个生产者或者消费者线程的信息。每一行的各字段间用 Tab 键隔开。不管是生产者还是消费者，都有一个对应的线程号，即每一行开始字段那个整数。第二个字段用字母 P 或者 C 区分是生产者还是消费者。第三个字段表示在进入相应线程后，在进行生产和消费动作前的休眠时间，以秒计时；这样做的目的是可以通过调整这一列参数，控制开始进行生产和消费动作的时间。如果是代表生产者，则该行只有三个字段。如果代表消费者，则该行后面还要若干字段，代表要求消费的产品所对应的生产者的线程号。所以务必确认这些对应线程号存在并且该线程代表一个生产者。

实验代码：

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <chrono>
#include <ctime>

// 全局变量
std::vector<int> buffer;
std::mutex mtx;
std::condition_variable cv;

```

```
16.     // 辅助函数：获取当前时间的字符串格式
17.     std::string getCurrentTime() {
18.         auto now = std::chrono::system_clock::now();
19.         std::time_t currentTime = std::chrono::system_clock::to_time_t(now);
20.         char buffer[100];
21.         strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", localtime(¤tTime))
;
22.         return std::string(buffer);
23.     }
24.
25.     // 生产者函数
26.     void producer(int id, int sleepTime) {
27.         std::this_thread::sleep_for(std::chrono::seconds(sleepTime));
28.         std::unique_lock<std::mutex> lock(mtx);
29.         int product = id;  // 生成产品，用线程号标识
30.
31.         // 尝试生产
32.         std::cout << "[" << getCurrentTime() << "] 生产者 " << id << " 尝试生产产
品 " << product << "\n";
33.         buffer.push_back(product);
34.         std::cout << "[" << getCurrentTime() << "] 生产者 " << id << " 生产了产
品 " << product << "\n";
35.
36.         // 显示当前缓冲区状态
37.         std::cout << "缓冲区: ";
38.         if (buffer.empty()) {
39.             std::cout << "空\n";
40.         } else {
41.             for (int item : buffer) std::cout << item << " ";
42.             std::cout << "\n";
43.         }
44.
45.         cv.notify_all();
46.     }
47.
48.     // 消费者函数
49.     void consumer(int id, int sleepTime, std::vector<int> requiredProducers) {
50.         std::this_thread::sleep_for(std::chrono::seconds(sleepTime));
51.         std::unique_lock<std::mutex> lock(mtx);
52.
53.         // 尝试消费
54.         std::cout << "[" << getCurrentTime() << "] 消费者 " << id << " 尝试消费产
品\n";
55.
```

```
        for (int prodId : requiredProducers) {
            cv.wait(lock, [&] { return !buffer.empty(); });

            // 找到并消费所需的产品
            auto it = std::find(buffer.begin(), buffer.end(), prodId);
            if (it != buffer.end()) {
                buffer.erase(it);
                std::cout << "[" << getCurrentTime() << "] 消费者 " << id << " 消费了生产者 " << prodId << " 的产品\n";
            } else {
                std::cout << "[" << getCurrentTime() << "] 消费者 " << id << " 未找到生产者 " << prodId << " 的产品\n";
            }

            // 显示当前缓冲区状态
            std::cout << "缓冲区: ";
            if (buffer.empty()) {
                std::cout << "空\n";
            } else {
                for (int item : buffer) std::cout << item << " ";
                std::cout << "\n";
            }
        }
    }

    // 主函数
    int main() {
        std::ifstream file("test.txt");
        int criticalSectionCount;
        file >> criticalSectionCount;

        std::vector<std::thread> threads;
        std::string line;
        std::getline(file, line); // 跳过第一行

        while (std::getline(file, line)) {
            std::istringstream ss(line);
            int thread_id, sleep_time;
            char role;
            ss >> thread_id >> role >> sleep_time;

            if (role == 'P') {
                threads.push_back(std::thread(producer, thread_id, sleep_time));
```

```
97.             } else if (role == 'C') {
98.                 std::vector<int> producer_ids;
99.                 int producer_id;
100.                while (ss >> producer_id) {
101.                    producer_ids.push_back(producer_id);
102.                }
103.                threads.push_back(std::thread(consumer, thread_id, sleep_time, p
roducer_ids));
104.            }
105.        }
106.
107.        // 等待所有线程执行完毕
108.        for (auto &t : threads) {
109.            t.join();
110.        }
111.
112.        return 0;
113.    }
```

测试用例 1 内容：

```
test.txt
1   3
2   1   P   3
3   2   P   4
4   3   C   4   1
5   4   P   2
6   5   C   3   1   4   2
```

- 第一行：3 表示在程序中设置了三个临界区（假设这指的是资源的并发访问限制）。
- 之后每行代表一个线程的配置：

1 P 3 表示 **生产者线程 1**，将在进入线程后休眠 3 秒再开始生产。

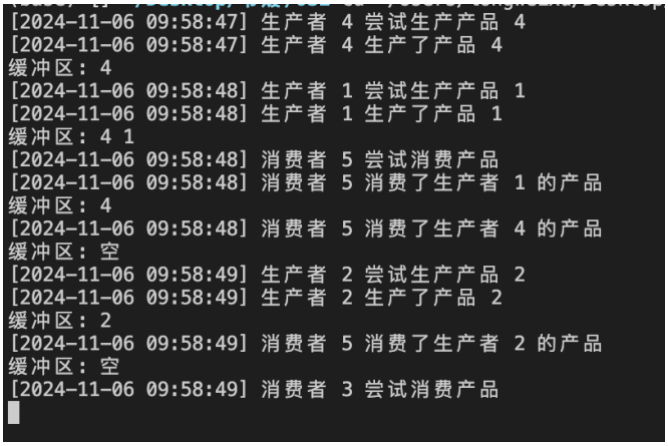
2 P 4 表示 **生产者线程 2**，将在进入线程后休眠 4 秒再开始生产。

3 C 4 1 表示 **消费者线程 3**，将在进入线程后休眠 4 秒再开始消费，并且它需要消费**生产者 1** 的产品。

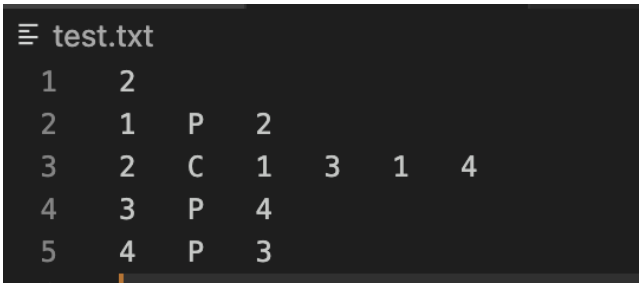
4 P 2 表示 **生产者线程 4**，将在进入线程后休眠 2 秒再开始生产。

5 C 3 1 4 2 表示 **消费者线程 5**，将在进入线程后休眠 3 秒再开始消费，并且它需要消费**生产者 1、4 和 2** 的产品。

运行结果如下：



测试用例 2 内容：



- 第一行 2 表示程序中设置了两个临界区（虽然此值在代码中未被直接使用）。
- 后续每行描述了一个线程的详细信息：

1 P 2 表示生产者线程 1，在进入线程后休眠 2 秒后开始生产。

2 C 1 3 1 4 表示消费者线程 2，在进入线程后休眠 1 秒后开始消费，并且消费来自生产者 3、1 和 4 的产品。

3 P 4 表示生产者线程 3，在进入线程后休眠 4 秒后开始生产。

4 P 3 表示生产者线程 4，在进入线程后休眠 3 秒后开始生产。

运行结果如下：