进程同步与互斥-生产者消费者问题

实验目的：

1. 理解操作系统中的同步与互斥问题。
2. 学习使用PV原语解决生产者消费者问题。
3. 掌握在不同操作系统上编写和调试多线程程序的基本技能。

实验内容：

生产者消费者问题是一个经典的多线程同步问题。在这个问题中，有一个生产者（Producer）和一个消费者（Consumer）共享一个有限大小的缓冲区（Buffer）。生产者的任务是生成数据并将其放入缓冲区；消费者的任务是从缓冲区中取出数据并使用它。当缓冲区满时，生产者需要等待；当缓冲区为空时，消费者需要等待。

实验伪代码：

```
// 初始化
semaphore mutex = 1; // 用于缓冲区的互斥访问
semaphore empty = N; // 缓冲区空余位置数量，初始为缓冲区大小N
semaphore full = 0;  // 缓冲区已占用位置数量

producer() {
    while (true) {
        produce_item();         // 生产一个新的项目
        P(empty);               // 等待缓冲区有空位
        P(mutex);               // 进入临界区
        insert_item();          // 将项目放入缓冲区
        V(mutex);               // 离开临界区
        V(full);                // 增加缓冲区已占用位置数量
    }
}

consumer() {
    while (true) {
        P(full);                // 等待缓冲区有数据
        P(mutex);               // 进入临界区
        remove_item();          // 从缓冲区取出项目
        V(mutex);               // 离开临界区
```

```
        V(empty);               // 增加缓冲区空余位置数量
        consume_item();         // 使用项目
    }
}
```

实验代码

```
#include <iostream>
#include <thread>
#include <semaphore>

const int N = 10; // 缓冲区大小
int buffer[N];    // 缓冲区
int in = 0;       // 下一个生产者将放置项目的位置
int out = 0;      // 下一个消费者将取出项目的位置

std::counting_semaphore<> empty(N); // 缓冲区空余位置数量
std::counting_semaphore<> full(0);  // 缓冲区已占用位置数量
std::binary_semaphore mutex(1);     // 用于缓冲区的互斥访问

void produce_item(int item) {
    // 生产项目的代码
    std::cout << "Produced: " << item << std::endl;
}

void consume_item(int item) {
    // 消费项目的代码
    std::cout << "Consumed: " << item << std::endl;
}

void producer() {
    for (int i = 0; i < 20; ++i) {
        produce_item(i);     // 生产一个新的项目

        // TODO: 等待缓冲区有空位

        // TODO: 进入临界区

        buffer[in] = i;      // 将项目放入缓冲区
        in = (in + 1) % N;   // 更新下一个生产者将放置项目的位置

        // TODO: 离开临界区

        // TODO: 增加缓冲区已占用位置数量
```

```
    }
}

void consumer() {
    for (int i = 0; i < 20; ++i) {
        // TODO: 等待缓冲区有数据

        // TODO: 进入临界区

        int item = buffer[out]; // 从缓冲区取出项目
        out = (out + 1) % N;    // 更新下一个消费者将取出项目的位置

        // TODO: 离开临界区

        // TODO: 增加缓冲区空余位置数量

        consume_item(item);  // 使用项目
    }
}

int main() {
    std::thread producerThread(producer);
    std::thread consumerThread(consumer);

    producerThread.join();
    consumerThread.join();

    return 0;
}
```

实验任务

将所给代码的 `Todo` 补充完毕，使得可以成功运行，完成生产者消费者问题，并完成实验报告。

运行环境

Windows

1. 安装编译器：确保安装了支持 C++20 的编译器，如最新版本的 Microsoft Visual Studio 或 MinGW-w64。推荐安装比较轻量版的编译器，如Codeblocks最新版、DevC++最新版，也可以使用其他的编译器如CLion或者VSCode。
2. 编译代码：

- 对于 Visual Studio或者其他IDE，您可以创建一个新的 C++ 项目，将代码复制到项目中，代码补全后，然后使用 IDE 编译和运行，确保勾选了C++20标准，因为信号量是C++20引入STL中的。
- 对于命令行方式，打开命令提示符或 PowerShell，导航到代码所在的目录，然后运行以下命令来编译代码：

```
g++ -std=c++20 -o producer_consumer producer_consumer.cpp
```

Linux

- 安装编译器：确保安装了支持 C++20 的编译器，如最新版本的 GCC 或 Clang。您可以使用包管理器来安装，例如，在 Ubuntu 上，您可以运行：

```
sudo apt update
sudo apt install build-essential
```

- 编译代码：在终端中，导航到代码所在的目录，然后运行以下命令来编译代码：

```
g++ -std=c++20 -o producer_consumer producer_consumer.cpp
```

- 运行程序：在终端中输入 `./producer_consumer` 来运行程序。

macOS

- 安装编译器：确保安装了支持 C++20 的编译器，如最新版本的 Clang。您可以通过安装 Xcode Command Line Tools 来获得编译器：

```
xcode-select --install
```

- 编译代码：在终端中，导航到代码所在的目录，然后运行以下命令来编译代码：

```
clang++ -std=c++20 -o producer_consumer producer_consumer.cpp
```

- 运行程序：在终端中输入 `./producer_consumer` 来运行程序。

输出样例

实验报告要求：

1. 完成缺失的代码部分，并确保程序能够正确运行。
2. 描述你的测试方法，并给出测试结果。例如，你可以通过打印日志来观察生产者和消费者的行为，以及缓冲区的状态变化。
3. 说明代码实现生产者消费者问题的思路和你的解决方案。
4. （选做）是否还有其他的方法可以来完成这个问题?tips：可以调研学习条件变量以及互斥锁。
5. 分析PV原语在解决生产者消费者问题中的作用和原理。并思考几个问题

- 在生产者消费者问题中，锁的作用是是什么?
- P原语和加锁的位置可以互换吗？为什么?

API说明：

1. `std::counting_semaphore`：
    - 这是 C++20 新引入的一个模板类，用于实现计数信号量。它用于同步对共享资源的访问，特别是在生产者-消费者问题中控制缓冲区的空闲和已满位置数量。
    - `std::counting_semaphore<> empty(N);` 创建一个初始计数为 `N` 的信号量 `empty`，表示缓冲区的空闲位置数量。

- `std::counting_semaphore<> full(0);` 创建一个初始计数为 `0` 的信号量 `full`，表示缓冲区中已占用的位置数量。
- `empty.acquire();` 和 `full.acquire();` 分别用于等待缓冲区有空位和有数据，会阻塞调用线程直到相应的计数大于零，然后减少计数。
- `empty.release();` 和 `full.release();` 分别用于增加缓冲区的空闲和已占用位置数量，会增加相应的计数。

2. `std::binary_semaphore`：
   - 这也是 C++20 新引入的一个类，是 `std::counting_semaphore` 的特殊情况，用于实现二元信号量，通常用作互斥锁。
   - `std::binary_semaphore mutex(1);` 创建一个初始计数为 `1` 的二元信号量 `mutex`，用于缓冲区的互斥访问。
   - `mutex.acquire();` 用于进入临界区，会阻塞调用线程直到 `mutex` 的计数大于零，然后将计数减为零。
   - `mutex.release();` 用于离开临界区，将 `mutex` 的计数增加为一。
3. `std::thread`：
   - 这是 C++11 引入的一个类，用于创建和管理线程。
   - `std::thread producerThread(producer);` 和 `std::thread consumerThread(consumer);` 分别创建了生产者和消费者线程，执行 `producer` 和 `consumer` 函数。
   - `producerThread.join();` 和 `consumerThread.join();` 分别用于等待生产者和消费者线程完成。
4. `std::cout`：
   - 这是 C++ 标准库中的一个输出流对象，用于向标准输出（通常是控制台）打印信息。
   - `std::cout << "Produced: " << item << std::endl;` 和 `std::cout << "Consumed: " << item << std::endl;` 分别用于打印生产和消费的项目。

注意：

- 请严格按照指定的文件命名和格式要求提交实验文件。
- 源代码文件命名为 `学号_lab编号.扩展名`（例如：`2019312676_lab1.cpp`），实验报告命名为 `学号_lab编号.pdf`（例如：`2019312676_lab1.pdf`）。