

操作系统课程设计报告

设计题目：《多线程编程：生产者-消费者问题》

姓名：

学号：

专业班级：

指导老师：

完成日期： 2024年12月10日

**合肥工业大学 计算机与信息学院**

1. 课程设计任务

多线程编程：生产者-消费者问题：

a.设置两类线程，一类为生产者，一类为消费者；

b.建立缓冲区的数据结构；

c.使用菜单随机启动生产者或消费者；

d.在窗口上显示缓冲区；

e.随着线程每次操作缓冲区，更新窗口的显示；

本次课程设计的任务是实现一个基于 Windows 平台的生产者 - 消费者问题演示系统。通过该系统展示生产者线程生产数据并放入缓冲区，消费者线程从缓冲区取出数据进行消费的过程，同时要实现图形化界面，能够直观地呈现缓冲区状态、提供操作菜单以控制生产者和消费者的启动、暂停、清空缓冲区以及调整生产消费速度等功能，并通过线程同步机制确保缓冲区操作的正确性和程序的稳定性。

1. 课程设计目的和要求。
2. 目的：
   1. 深入理解操作系统中进程同步与互斥的概念和原理，特别是生产者 - 消费者问题的解决方法，通过实际编程实现来巩固理论知识。
   2. 掌握 Windows 图形界面编程技术，包括窗口创建、菜单设计、消息处理以及图形绘制等方面，能够开发出具有一定交互性的应用程序。
   3. 学会使用多线程编程来模拟并发执行的场景，理解线程间的协作与同步机制在实际应用中的重要性，并能够正确运用线程同步工具（如互斥锁、信号量）解决共享资源访问冲突问题。
3. 要求：
   1. 程序必须能够正确运行，实现生产者和消费者线程按照预期的逻辑进行数据生产和消费，并且在多线程并发环境下保证数据的一致性和正确性。
   2. 图形界面要简洁、直观、易用，能够清晰地展示缓冲区状态信息，菜单功能正常响应，例如点击 “启动生产者” 能成功创建生产者线程并开始生产数据，点击 “暂停” 能暂停生产和消费过程等。
   3. 代码结构要清晰，具有良好的可读性和可维护性，合理划分功能模块，适当添加注释以便他人理解代码意图。
4. 开发环境。
5. 操作系统：Windows
6. 集成开发环境（IDE）：可以使用 Dev-C++ 等支持 Windows 编程和 C 语言开发的 IDE。在 Dev-C++ 中，能够方便地创建 Windows 应用程序项目，配置相关编译链接选项，并且其编辑器提供了代码自动补全、语法检查等功能有助于提高开发效率。
7. 编译工具：使用 MinGW（GNU Compiler Collection for Windows）作为编译器，它能够将 C 语言代码编译成可在 Windows 平台上运行的可执行文件，并且支持 Windows API 的调用。
8. 相关原理及算法
9. **生产者 - 消费者原理：**

生产者和消费者共享一个缓冲区，生产者负责生产数据并放入缓冲区，消费者则从缓冲区取出数据进行消费。在这个过程中，需要解决缓冲区的同步与互斥问题，以避免数据冲突和错误。例如，当缓冲区已满时，生产者需要等待消费者取出数据后才能继续生产（通过信号量 full 和 empty 控制）；当缓冲区为空时，消费者需要等待生产者生产数据后才能消费。同时，在访问缓冲区的过程中，无论是生产者写入数据还是消费者读取数据，都需要使用互斥锁 mutex 来保证同一时间只有一个线程能操作缓冲区，防止数据不一致。

1. **线程同步算法：**

a. 使用互斥锁（mutex）实现对共享资源（缓冲区）的互斥访问。在生产者线程和消费者线程中，当需要操作缓冲区时，首先通过 WaitForSingleObject 函数等待获取互斥锁，操作完成后通过 ReleaseMutex 函数释放互斥锁，这样确保了同一时刻只有一个线程能进入临界区（操作缓冲区的代码段）。

b.利用信号量（empty 和 full）来控制生产者和消费者的执行顺序和节奏。empty 信号量初始化为缓冲区的大小，表示缓冲区中空位的数量；full 信号量初始化为 0，表示缓冲区中初始没有数据可供消费。生产者在生产数据前先等待 empty 信号量，表示有空位才能生产，生产完成后释放 full 信号量，表示缓冲区中有了新数据可供消费；消费者则先等待 full 信号量，有数据时才能消费，消费完成后释放 empty 信号量，表示缓冲区腾出了空位。

1. 系统结构和主要的算法设计思路描述
2. **系统结构**：
   1. **界面层**：负责创建 Windows 图形界面，包括窗口、菜单等元素。通过处理窗口消息（如 WM\_COMMAND 处理菜单点击事件，WM\_PAINT 处理窗口绘制事件）与用户进行交互，并将用户操作反馈给逻辑层。例如，当用户点击菜单中的 “启动生产者” 选项时，界面层会调用逻辑层的函数来创建生产者线程。
   2. **逻辑层**：包含生产者线程函数 ProducerThread、消费者线程函数 ConsumerThread 以及相关的同步控制逻辑。这一层主要实现生产者 - 消费者的核心算法，包括数据生产、消费以及缓冲区的同步操作。同时，还维护着一些全局变量（如 buffer 缓冲区结构体、isRunning 程序运行状态、isPaused 暂停状态等）来记录系统的状态信息，并通过这些变量与界面层进行信息交互，例如根据 isPaused 的状态来决定生产者和消费者线程的执行或暂停。
   3. **数据层**：主要是缓冲区结构体 Buffer，用于存储生产者生产的数据，并供消费者消费。它是生产者和消费者线程之间共享的数据结构，通过逻辑层的同步机制来保证数据的正确读写。
3. **主要算法设计思路：**
   1. **生产者线程算法**：
      * 首先增加生产者线程计数 producerCount。
      * 在主循环中，根据程序运行状态 isRunning 和暂停状态 isPaused 判断是否继续执行生产操作。
      * 若允许生产，先等待 empty 信号量获取缓冲区空位，再获取 mutex 互斥锁以独占缓冲区访问权。
      * 生产一个随机数据并放入缓冲区 buffer 的指定位置（由 buffer.in 指示），更新缓冲区相关指针和计数（buffer.in 和 buffer.count），同时增加总生产数据计数 totalProduced。
      * 触发窗口重绘 InvalidateRect 更新界面显示，然后释放 mutex 互斥锁和 full 信号量，通知消费者有新数据。
      * 随机等待一段时间后，清除当前活动缓冲区标记 activeBuffer，再次触发窗口重绘，循环继续。
   2. **消费者线程算法**：
      * 类似生产者线程，先增加消费者线程计数 consumerCount。
      * 循环中依据 isRunning 和 isPaused 判断执行条件。
      * 若可消费，先等待 full 信号量获取数据，再获取 mutex 互斥锁。
      * 从缓冲区 buffer 取出数据（由 buffer.out 指示），标记该位置为已消费（赋值 - 1），更新缓冲区指针和计数（buffer.out 和 buffer.count），并增加总消费数据计数 totalConsumed。
      * 触发窗口重绘更新显示，释放 mutex 互斥锁和 empty 信号量，通知生产者有空位。
      * 随机等待后，清除 activeBuffer 标记，再次重绘，循环继续。
   3. **窗口过程算法**：
      * 针对不同的窗口消息进行处理。对于 WM\_COMMAND 消息，根据具体的菜单命令标识符执行相应操作，如创建线程、切换暂停状态、清空缓冲区、调整速度等，并更新菜单状态显示。
      * 收到 WM\_PAINT 消息时，绘制窗口界面，包括显示标题、缓冲区状态（绘制矩形框并显示数据或空状态）以及生产消费指针位置信息。
      * 当 WM\_DESTROY 消息到来时，设置 isRunning 为 FALSE 停止线程，发送退出消息结束程序，其他消息则调用默认窗口过程处理。
4. 程序实现-主要数据结构

在本程序中，核心的数据结构是自定义的 Buffer 结构体，其定义如下：

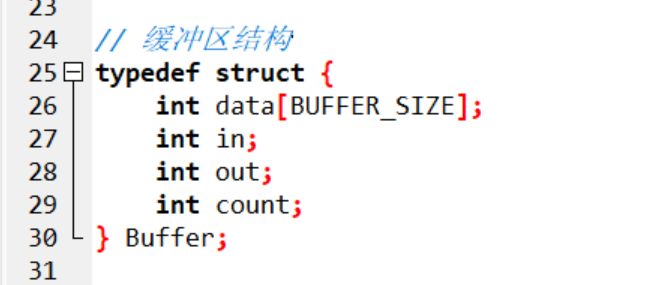


图1-缓冲区结构

* data 数组：它是一个长度由 BUFFER\_SIZE（在代码中定义为 5）决定的整型数组，用于实际存储生产者生产的数据以及消费者从中获取的数据。数组中的每个元素代表缓冲区的一个存储单元，生产者将生成的数据放入这些单元，消费者从这些单元取出数据进行消费。例如，data[0] 表示缓冲区的第一个存储位置，以此类推。
* in 成员变量：这是一个整型变量，用于指示生产者下次将数据放入缓冲区的位置索引。它的取值范围是 0 到 BUFFER\_SIZE - 1，通过取余运算（buffer.in = (buffer.in + 1) % BUFFER\_SIZE;）实现循环利用缓冲区空间，当达到缓冲区末尾时，会自动回到起始位置继续存放数据，就像一个环形缓冲区的 “写指针”。
* out 成员变量：同样是整型变量，用于标记消费者下次从缓冲区取出数据的位置索引，其取值范围和循环方式与 in 类似，起到环形缓冲区 “读指针” 的作用，确保按照先进先出的顺序从缓冲区获取数据。
* count 成员变量：记录当前缓冲区中已经存放的数据个数，通过生产者放入数据时增加、消费者取出数据时减少该变量的值，方便判断缓冲区是否已满（count == BUFFER\_SIZE）或者为空（count == 0），进而协调生产者和消费者线程的操作。

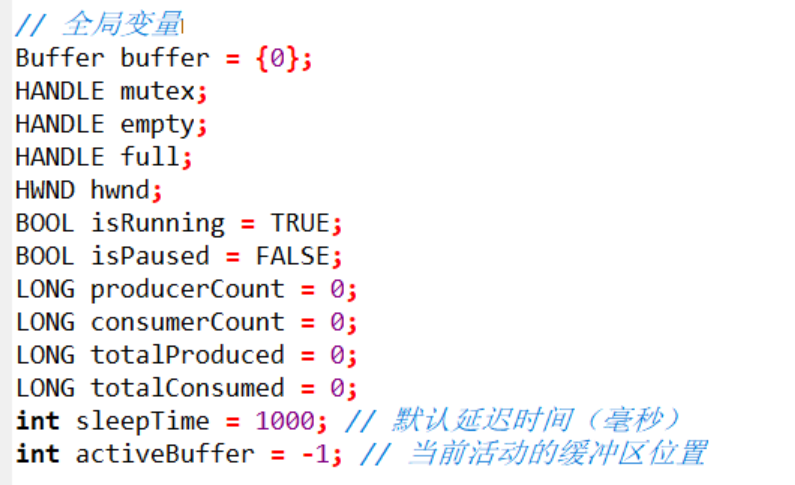


图2-全局变量定义

除了 Buffer 结构体，程序中还使用了一些全局变量来记录系统的状态以及辅助多线程操作，例如 mutex（互斥锁句柄）、empty 和 full（信号量句柄）用于线程同步，hwnd（窗口句柄）用于操作窗口相关功能，isRunning 和 isPaused 分别控制程序整体的运行和暂停状态，producerCount、consumerCount、totalProduced、totalConsumed 用于统计线程数量以及生产和消费的数据总量等，sleepTime 控制线程操作后的等待时间以调节生产消费速度，activeBuffer 标记当前正在操作的缓冲区位置方便界面更新。

1. 程序实现-细节描述

以下是关键代码部分的详细解释

1. **WinMain 函数（主函数）**：

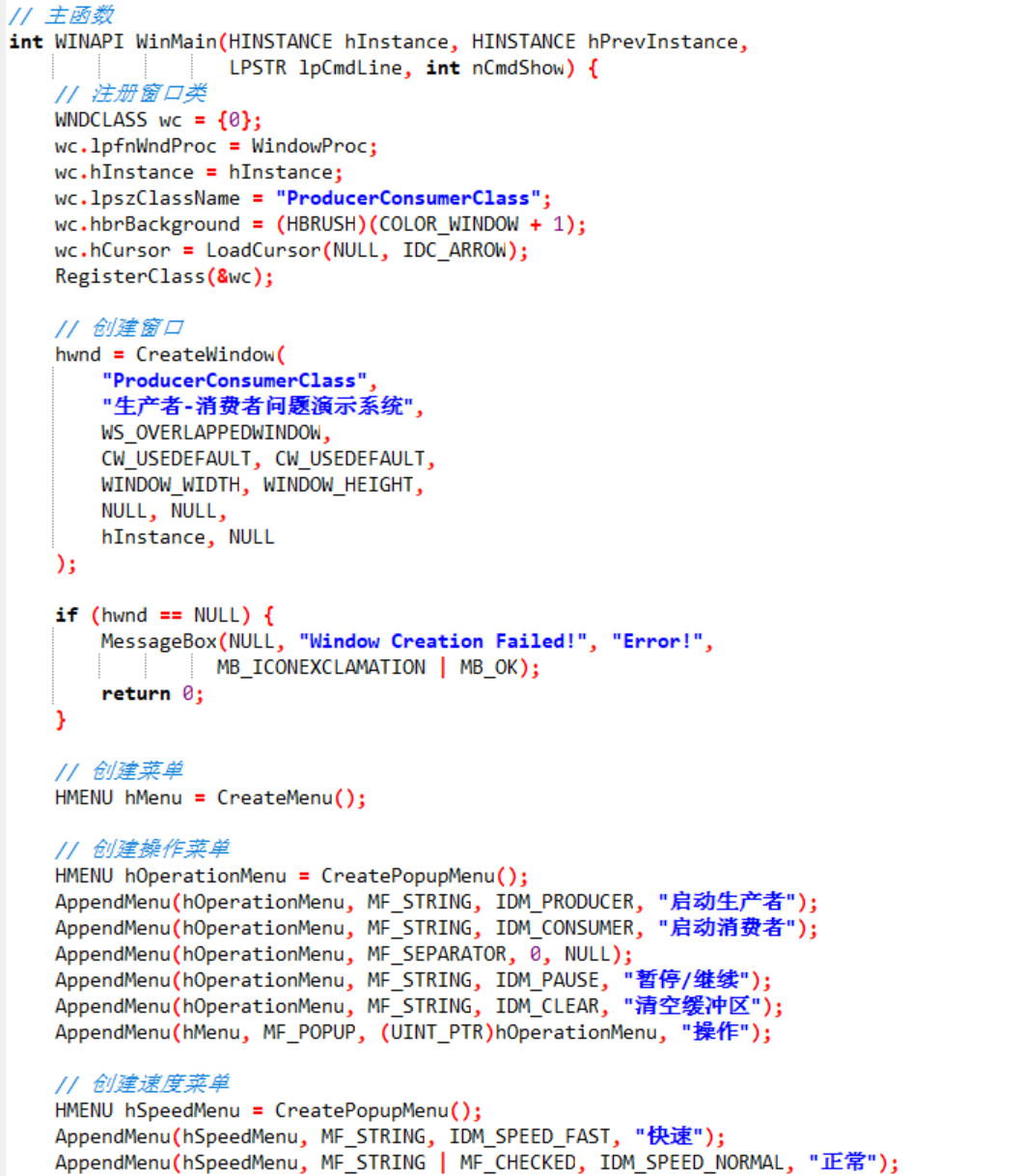


图3- **WinMain 函数（部分）**

* + 首先注册窗口类，设置窗口过程函数为 WindowProc，以及窗口的各种属性（如实例句柄、类名、背景颜色、光标样式等），然后创建窗口。
  + 创建菜单，包括操作菜单（启动生产者、消费者、暂停、清空缓冲区）和速度菜单（快速、正常、慢速），并将菜单设置到窗口上。
  + 初始化同步对象（mutex 互斥锁、empty 和 full 信号量），以及缓冲区（将缓冲区数据全部初始化为 - 1，设置 buffer.in、buffer.out 和 buffer.count 初始值）。
  + 初始化随机数生成器，以便生产者线程能生成不同的随机数据。
  + 显示窗口并进入消息循环，不断获取窗口消息，通过 TranslateMessage 和 DispatchMessage 函数将消息分发给相应的窗口过程函数进行处理，直到收到退出消息。最后清理资源，关闭同步对象句柄。

1. **ProducerThread 函数**：

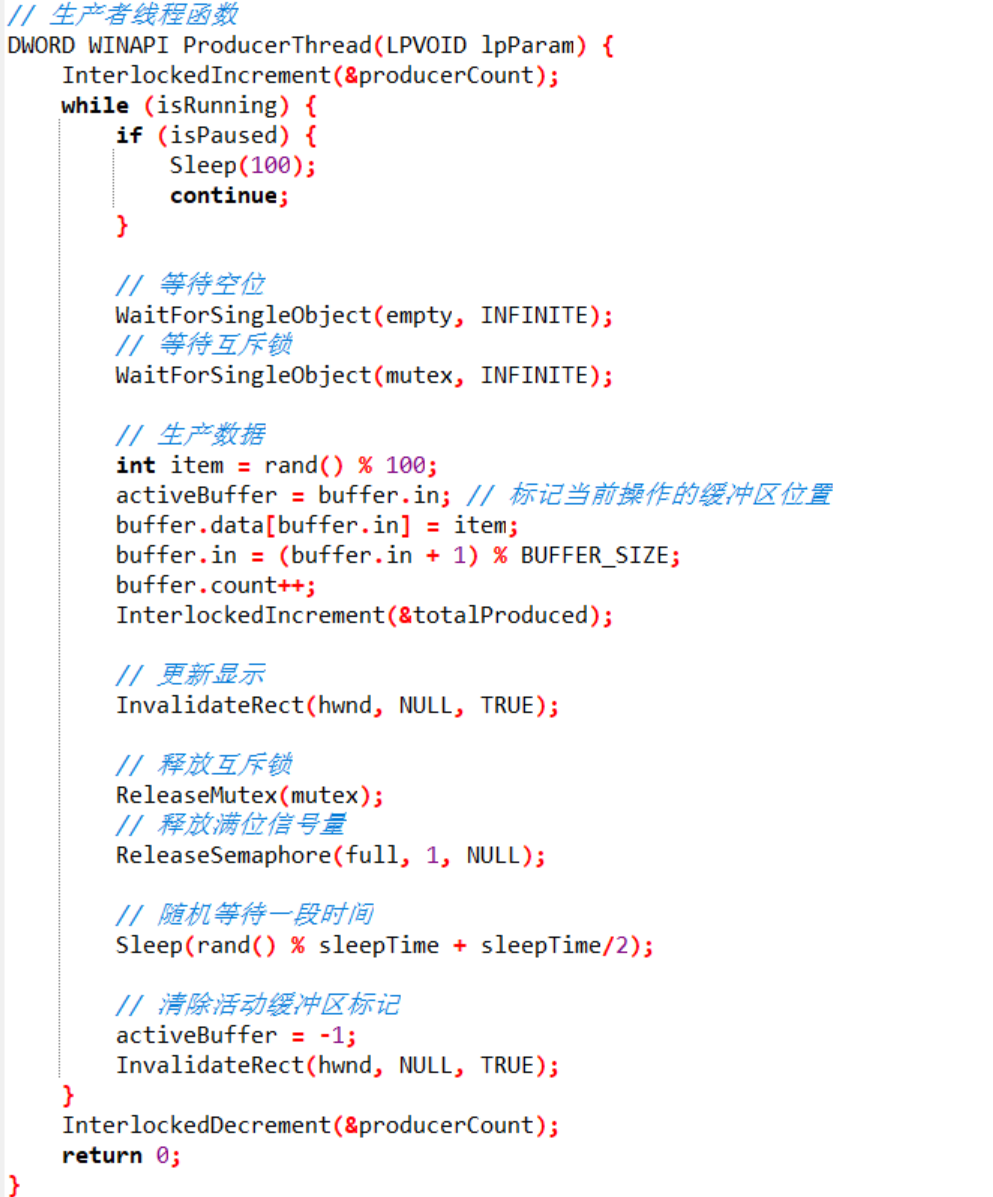


图4 - **ProducerThread 函数**

* + 如前面算法设计思路所述，在函数内部通过循环实现生产者的核心逻辑，包括同步等待、数据生产、缓冲区操作、界面更新以及线程计数维护等操作，确保在多线程环境下正确地与其他线程协作并生产数据放入缓冲区。

1. **ConsumerThread 函数**：

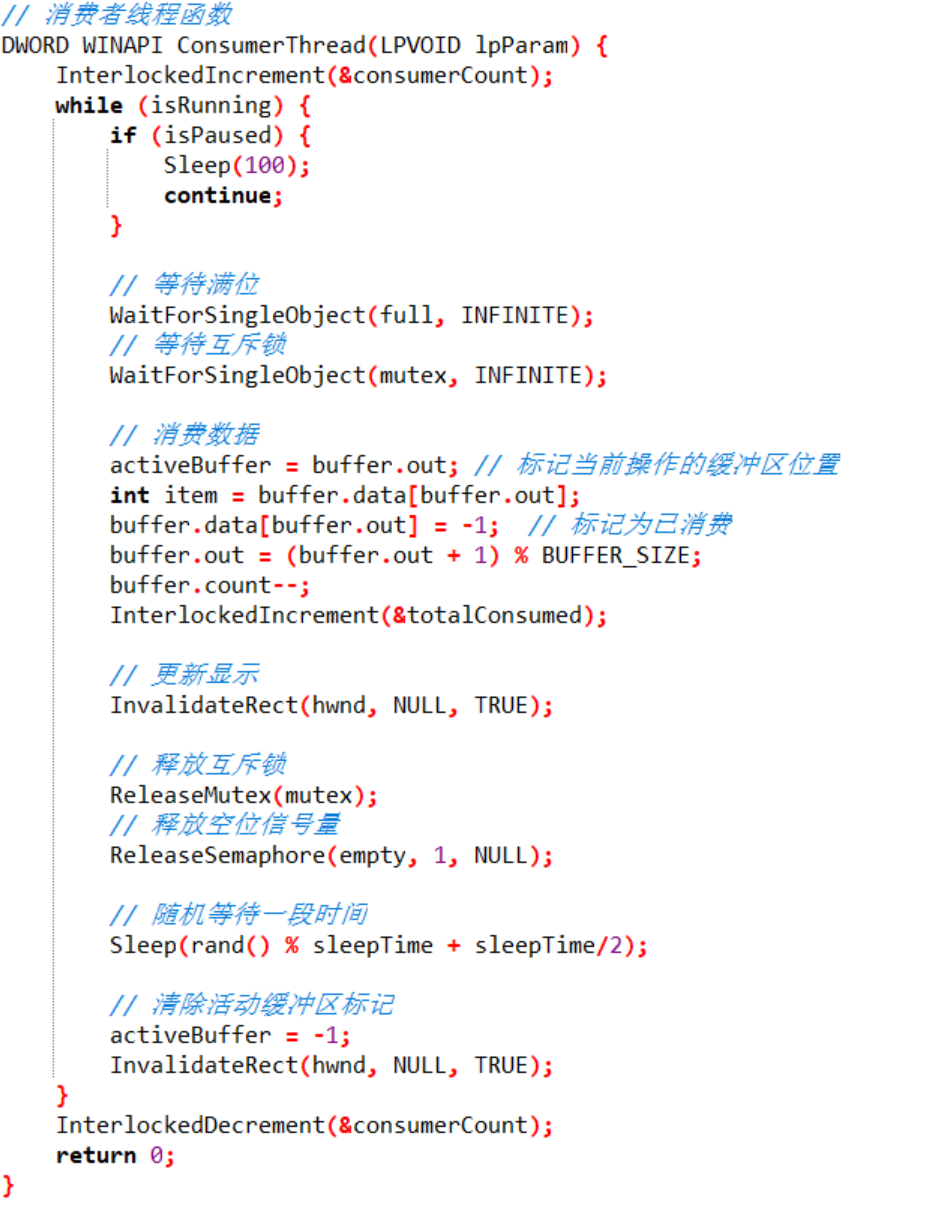


图5 – **ConsumerThread函数**

* + 类似 ProducerThread 函数，通过循环实现消费者的核心逻辑，从缓冲区获取数据进行消费，同时处理好同步、界面更新以及线程计数等相关操作，保证消费过程的正确性和与其他线程的协调。

1. **WindowProc 函数**：



图6- **WindowProc 函数（部分）**

* + 根据不同的窗口消息类型进行处理，在 WM\_COMMAND 消息中处理各种菜单操作，在 WM\_PAINT 消息中绘制窗口界面，在 WM\_DESTROY 消息中处理窗口关闭相关逻辑，对于其他消息则调用默认窗口过程处理，以此实现与用户的交互以及窗口的显示和控制。

1. 程序运行的主要界面和结果截图



图7-程序运行初始界面

初始界面：运行程序后，会弹出一个标题为 “生产者 - 消费者问题演示系统” 的窗口，窗口中有菜单栏，包括 “操作” 和 “速度” 两个菜单选项。

缓冲区显示：在窗口的中心区域会显示缓冲区的状态，以 5 个矩形框表示缓冲区的 5 个单元，每个矩形框内如果有数据则显示数据值（例如 12、34 等随机生成的数字），如果为空则显示 “空” 字。在矩形框下方会显示 “生产位置、 消费位置、当前数量”，分别指示当前生产者生产数据放入缓冲区的位置索引、消费者从缓冲区取出数据的位置索引以及缓冲区中现有数据的数量。



图8-程序运行菜单操作栏



图9-情况缓冲区提示

操作菜单效果：点击 “操作” 菜单中的 “启动生产者”，会启动生产者线程开始生产数据，此时可以看到缓冲区中逐渐有数据填充；点击 “启动消费者” 则启动消费者线程开始消费数据，缓冲区中的数据会被逐渐取出；点击 “暂停”，生产和消费过程会暂停，再次点击则继续；点击 “清空缓冲区” 会弹出确认对话框，点击 “是” 后缓冲区会被清空，所有矩形框显示 “空”，生产消费指针位置重置为 0，数量为 0。

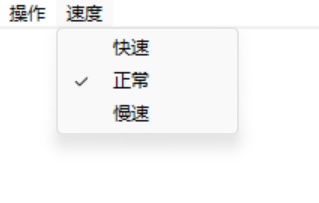


图9-程序运行菜单速度栏

速度菜单效果：点击 “速度” 菜单中的 “快速”“正常”“慢速” 选项，可以调整生产者和消费者线程每次操作后的等待时间，从而改变生产消费的速度。例如选择 “快速” 后，数据的生产和消费速度会明显加快，选择 “慢速” 则速度变慢，相应地可以看到缓冲区数据变化的频率也随之改变。

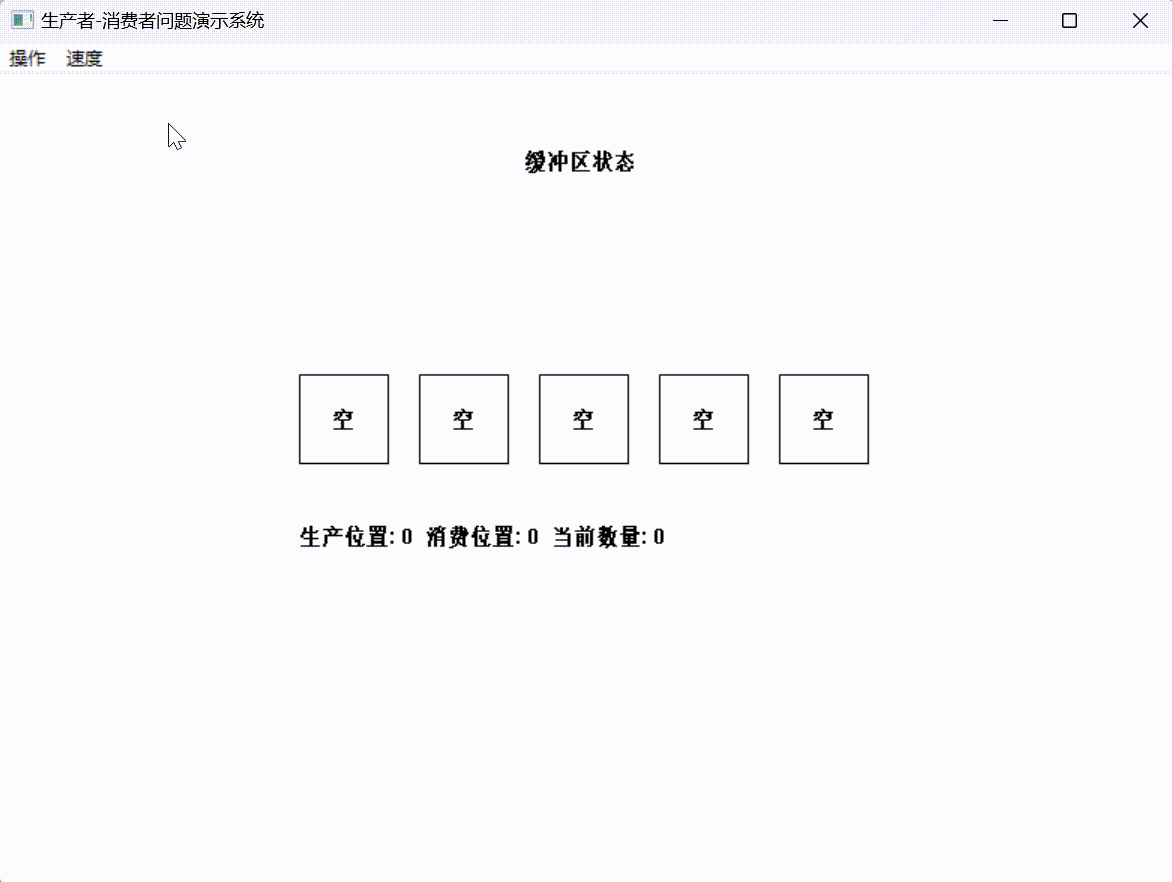
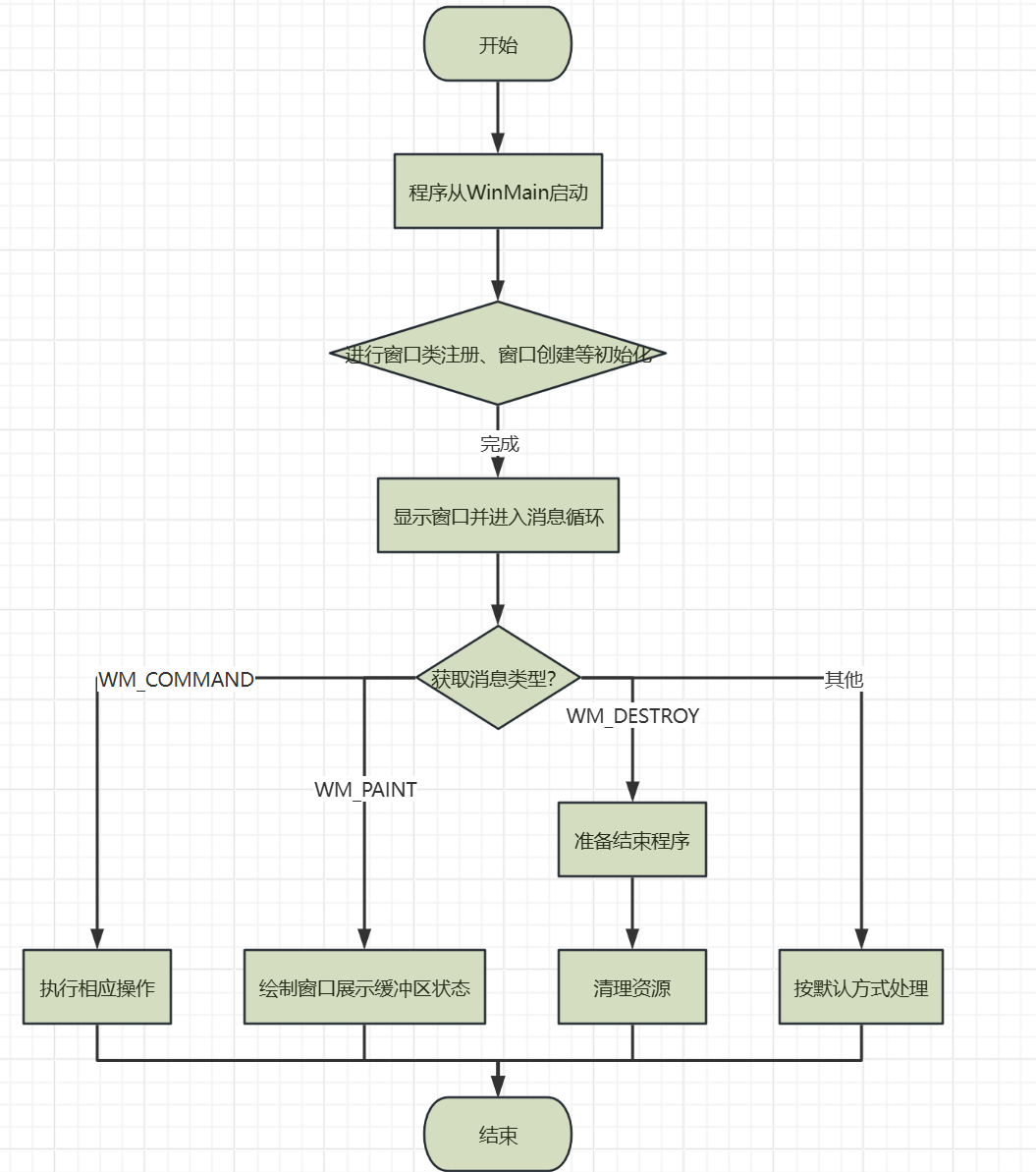


图10-整体程序演示gif图



程序流程图

1. 总结和感想体会

通过本次课程设计，我深入理解了操作系统中的多线程编程、进程同步与互斥机制，特别是生产者-消费者问题的解决方法。在实现过程中，我不仅巩固了理论知识，还掌握了Windows平台下的图形界面编程技术，包括窗口创建、菜单设计、消息处理以及图形绘制等方面的技能。

**总结：**

1. **多线程同步机制的理解与应用**：通过使用互斥锁（mutex）和信号量（empty、full），我成功实现了生产者和消费者线程之间的同步与互斥。互斥锁确保了同一时刻只有一个线程能够访问缓冲区，而信号量则控制了生产者和消费者的执行顺序，避免了缓冲区溢出或空读的问题。
2. **图形界面的设计与实现**：本次设计的一个重要任务是实现图形化界面，展示缓冲区的状态和操作结果。通过Windows API，我成功创建了一个简洁、直观的窗口界面，能够实时显示缓冲区的数据变化、生产者和消费者的操作位置等信息。菜单的设计也使得用户可以方便地控制程序的运行状态，如启动、暂停、清空缓冲区以及调整生产消费速度。
3. **代码结构与可维护性**：在编写代码时，我注重模块化设计，将界面层、逻辑层和数据层分离，使得代码结构清晰，易于理解和维护。同时，我也为关键代码添加了详细的注释，方便他人阅读和修改。
4. **调试与优化**：在多线程编程中，调试是一个复杂的过程。由于线程的并发执行，常常会出现难以复现的错误。通过仔细分析代码逻辑和使用调试工具，我逐步解决了线程同步中的问题，确保了程序的稳定性和正确性。

**感想与体会：**

1. **理论与实践的结合**：通过本次课程设计，我深刻体会到理论知识与实际编程的结合是多么重要。书本上的概念和算法在实际应用中可能会遇到各种问题，只有通过动手实践，才能真正理解并掌握这些知识。
2. **多线程编程的复杂性**：多线程编程虽然强大，但也非常复杂。线程之间的同步与互斥问题需要非常小心地处理，稍有不慎就会导致数据不一致或死锁等问题。通过本次设计，我学会了如何正确使用同步工具来避免这些问题。
3. **图形界面编程的挑战**：图形界面编程与传统的命令行编程有很大不同，需要考虑用户交互、界面布局、消息处理等多方面的问题。虽然挑战较大，但通过不断学习和实践，我逐渐掌握了Windows图形界面编程的基本技巧。
4. **团队合作与沟通**：在本次设计中，我与同学们进行了多次讨论和交流，互相帮助解决遇到的问题。这让我意识到，团队合作和沟通在项目开发中的重要性。通过分享经验和知识，我们可以更快地解决问题，提高开发效率。

**未来展望：**

本次课程设计让我对操作系统和多线程编程有了更深的理解，也让我意识到自己在编程和系统设计方面还有很多需要学习的地方。未来，我计划进一步学习操作系统的高级主题，如内存管理、文件系统、进程调度等，同时也会继续提升自己的编程能力，尤其是在多线程和并发编程方面。此外，我还希望学习更多的图形界面编程技术，掌握跨平台的GUI开发框架，以便能够开发出更加复杂和实用的应用程序。

总的来说，本次课程设计不仅让我掌握了生产者-消费者问题的解决方法，还让我在编程实践、问题解决和团队合作等方面得到了全面的锻炼。这些经验和技能将为我未来的学习和工作打下坚实的基础。

1. 附录

源码