线程调度大作业

改进的消费者和生产者问题

姓名：禹泽海 学号：2021302111

一、问题复述

问题 1.

假设 3 个消费者和 5 个生产者构成的应用系统。其中，生产者和消费者通过 一块共享内存进行通信，生产者向该共享内存写入数据、消费者从该共享内存取出并显示数据。请在 Windows 系统用多线程实现。

问题 2.

请分析并通过实验验证，如何设置生产者和消费者线程的优先级使得该系统性能最优或者较优。

问题 3.

纤程（Fiber）可以视作 Windows 系统的 “用户级线程”。正如我们所知，用户级线程的优势之一就在于允许我们灵活定义调度策略。请你在 Fiber 环境下实现包括彩票调度在内的不少于三种调度策略，并结合问题 1 的 Fiber 实现，对其性能进行分析。

二、思路及代码分析

问题一：

使用 C 语言和多线程在 Windows 系统上实现的生产者-消费者问题。使用了互斥锁（Critical Section）来保护共享缓冲区，并使用信号量来控制生产者和消费者的数量以及等待条件。代码使用 CreateThread 函数创建线程，并使用 CloseHandle 函数关闭线程句柄，最后使用 DeleteCriticalSection 函数删除临界区。

问题二：

设置线程的优先级可以对系统性能产生影响，但通常需要小心调整，以确保系统稳定性。通常，提高生产者线程的优先级可以在生产者-消费者问题中提高性能，因为这有助于更快地填充缓冲区。

在 Windows 系统中，线程的优先级通过 SetThreadPriority 函数来设置，可以设置以下优先级：

THREAD_PRIORITY_IDLE
THREAD_PRIORITY_LOWEST
THREAD_PRIORITY_BELOW_NORMAL
THREAD_PRIORITY_NORMAL
THREAD_PRIORITY_ABOVE_NORMAL
THREAD_PRIORITY_HIGHEST
THREAD_PRIORITY_TIME_CRITICAL

我们可以对生产者和消费者设置不同的优先级别分别实验并检验程序运行时间来判断性能。

问题三：

在 Windows 环境中，使用 Fibers（纤程）可以实现自定义的调度策略，其中包括彩票调度和其他多种策略。本问题中我们在 Fiber 环境下实现彩票调度、轮询调度和优先级调度。然后，我们可以对这三种调度策略的性能进行分析。

创建了 8 个 Fibers，每个 Fiber 代表一个线程。现在，我们可以实现不同的调度策略：

彩票调度：给每个 Fiber 分配一定数量的彩票，然后随机选择一个 Fiber 执行。

轮询调度：按照固定的顺序轮流执行每个 Fiber。

优先级调度：为每个 Fiber 分配不同的优先级，并按照优先级顺序执行。

对于性能分析，您可以考虑以下因素：

各种策略的公平性：不同策略可能会导致不同 Fibers 之间的公平性和响应时间。

各种策略的开销：不同策略可能会引入不同程度的开销，如彩票调度需要生成随机数，而优先级调度需要额外的排序。

不同策略下的系统吞吐量：通过测量每个 Fiber 的任务完成时间，可以比较不同策略下的系统吞吐量。

对于高负载情况的性能：测试不同策略在高负载时的性能表现。

性能测试通常需要使用性能分析工具或测量关键性能指标来确定最佳策略。根据具体应用的要求，不同策略可能会有不同的性能表现。在实际应用中，可以根据需求选择最适合的策略。

三、程序运行展示及分析

实验一：producer_consumer.c

```
Consumer 3 consumed: 53
Consumer 3 consumed: 11
Consumer 3 consumed: 41
Consumer 3 consumed: 64
Producer 1 produced: 90
Producer 1 produced: 42
Producer 1 produced: 88
Producer 1 produced: 6
Producer 1 produced: 40
Producer 1 produced: 42
Producer 1 produced: 64
Producer 5 produced: 62
Producer 4 produced: 57
Producer 3 produced: 38
Producer 3 timed out waiting for space in the buffer.
Producer 1 timed out waiting for space in the buffer.
Producer 4 timed out waiting for space in the buffer.
Producer 5 timed out waiting for space in the buffer.
running time is:1047ms
```

实验二：pc2.c

消费者优先级高于生产者：

```
DWORD WINAPI Producer(void* arg) {
    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_NORMAL);
    for (int i = 0; i < 100; i++) {
        // Produce an item
        int item = rand() % 100;

        // Wait if the buffer is full with a timeout
        if (WaitForSingleObject(producerSem, 1000) == WAIT_TIMEOUT) {
            printf("Producer %d timed out waiting for space in the buffer.\n", (int)arg);
            break;  // Exit the loop if timeout occurs
        }

        EnterCriticalSection(&bufferLock);

        // Add item to the buffer
        buffer[itemCount] = item;
        itemCount++;
        printf("Producer %d produced: %d\n", (int)arg, item);

        LeaveCriticalSection(&bufferLock);
        ReleaseSemaphore(consumerSem, 1, NULL);
    }

    return 0;
}

DWORD WINAPI Consumer(void* arg) {
    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);
```

```
Consumer 2 consumed: 37
Consumer 2 consumed: 57
Consumer 2 consumed: 62
Producer 1 produced: 59
Producer 1 produced: 23
Producer 1 produced: 41
Producer 3 produced: 44
Producer 5 produced: 41
Producer 4 produced: 62
Producer 1 timed out waiting for space in the buffer.
Producer 5 timed out waiting for space in the buffer.
Producer 4 timed out waiting for space in the buffer.
Producer 3 timed out waiting for space in the buffer.
running time is :1047
```

在这种情况下，生产者被设置为更高的优先级。这意味着生产者线程更频繁地获得 CPU 执行时间。结果显示生产者线程在一段时间内运行得更多。这对于需要高生产率的应用可能是有利的，但也可能导致消费者线程响应时间较长。

生产者优先级高于消费者

```
DWORD WINAPI Producer(void* arg) {
    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);
    for (int i = 0; i < 100; i++) {
        // Produce an item
        int item = rand() % 100;

        // Wait if the buffer is full with a timeout
        if (WaitForSingleObject(producerSem, 1000) == WAIT_TIMEOUT) {
            printf("Producer %d timed out waiting for space in the buffer.\n", (int)arg);
            break;  // Exit the loop if timeout occurs
        }

        EnterCriticalSection(&bufferLock);

        // Add item to the buffer
        buffer[itemCount] = item;
        itemCount++;
        printf("Producer %d produced: %d\n", (int)arg, item);

        LeaveCriticalSection(&bufferLock);
        ReleaseSemaphore(consumerSem, 1, NULL);
    }

    return 0;
}

DWORD WINAPI Consumer(void* arg) {
    SetThreadPriority(GetCurrentThread(), THREAD_PRIORITY_NORMAL);
    for (int i = 0; i < 100; i++) {
```

```
Consumer 3 consumed: 67
Consumer 3 consumed: 12
Producer 4 produced: 40
Producer 4 produced: 42
Producer 1 produced: 59
Producer 3 produced: 99
Producer 2 produced: 29
Producer 5 produced: 53
Producer 4 timed out waiting for space in the buffer.
Producer 5 timed out waiting for space in the buffer.
Producer 1 timed out waiting for space in the buffer.
Producer 3 timed out waiting for space in the buffer.
Producer 2 timed out waiting for space in the buffer.
running time is :1062
```

在这种情况下，消费者被设置为更高的优先级。这导致消费者线程更频繁地获得 CPU 执行时间。结果显示消费者线程在一段时间内运行得更多。这可以对需要快速响应和消费的应用有利。

总体而言，设置不同线程的优先级可以影响它们的执行顺序和相对权重。选择哪个线程优先运行可能会根据应用的具体需求而定。例如，如果某个线程的任务对应用的性能至关重要，那么提高其优先级可能是有益的。然而，需要小心管理线程的优先级，以避免导致某些线程被饿死（Starvation）或不公平分配 CPU 时间。

问题三：（1000 纤程运行）
彩票调度：代码文件：31.c

```
for (int i = 0; i < NUM_FIBERS; i++) {
    fibers[i] = CreateFiber(0, FiberFunc, (LPVOID)i);
    fiberInfo[i].fiber = fibers[i];
    fiberInfo[i].tickets = NUM_LOTTERY_TICKETS / NUM_FIBERS;
}

for(int i=0;i<1000;i++) {
    // 彩票延迟
    int winningTicket = rand() % NUM_LOTTERY_TICKETS;
    int winnerIndex = 0;

    for (int i = 0; i < NUM_FIBERS; i++) {
        winningTicket -= fiberInfo[i].tickets;
        if (winningTicket < 0) {
            winnerIndex = i;
            break;
        }
    }

    SwitchToFiber(fiberInfo[winnerIndex].fiber);
}

// Cleanup
for (int i = 0; i < NUM_FIBERS; i++) {
    DeleteFiber(fibers[i]);
}
```

```
Fiber 00000001 is running
Fiber 00000002 is running
Fiber 00000001 is running
Fiber 00000006 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000004 is running
Fiber 00000002 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000000 is running
Fiber 00000000 is running
Fiber 00000001 is running
running time is:62ms
```

彩票调度是一种随机调度策略，它为每个 Fiber 分配了彩票，并根据彩票的随机抽取来选择下一个执行的 Fiber。在这种情况下，因为彩票是随机分配的，所以某些 Fibers 可能被频繁地选择，而其他 Fibers 可能较少选择。这可以导致某些 Fibers 具有较低的响应时间，而其他 Fibers 具有较高的响应时间。

轮询调度：
代码文件：32.c

```c
for (int i = 0; i < NUM_FIBERS; i++) {
    fibers[i] = CreateFiber(0, FiberFunc, (LPVOID)i);
}

for(int i=0;i<1000;i++) {
    SwitchToFiber(fibers[currentIndex]);
    currentIndex = (currentIndex + 1) % NUM_FIBERS;
}

// Cleanup
for (int i = 0; i < NUM_FIBERS; i++) {
    DeleteFiber(fibers[i]);
}
```

```
Fiber 00000005 is running
Fiber 00000006 is running
Fiber 00000007 is running
Fiber 00000000 is running
Fiber 00000001 is running
Fiber 00000002 is running
Fiber 00000003 is running
Fiber 00000004 is running
Fiber 00000005 is running
Fiber 00000006 is running
Fiber 00000007 is running
running time is:78ms
```

轮询调度是一种固定顺序的调度策略，每个 Fiber 依次执行。在这种情况下，Fibers 按照固定顺序循环执行，没有考虑任务的优先级或不同 Fibers 的执行时间。这可能导致一些 Fibers 等待时间较长，而其他 Fibers 较快执行。

优先级调度：代码文件：33.c

```
for (int i = 0; i < NUM_FIBERS; i++) {
    fibers[i] = CreateFiber(0, FiberFunc, (LPVOID)i);
    fiberInfo[i].fiber = fibers[i];
    fiberInfo[i].priority = i % NUM_PRIORITIES;
}

for(int i=0;i<1000;i++) {
    int highestPriority = NUM_PRIORITIES - 1;
    int nextFiber = -1;

    for (int i = 0; i < NUM_FIBERS; i++) {
        if (fiberInfo[i].priority >= highestPriority) {
            highestPriority = fiberInfo[i].priority;
            nextFiber = i;
        }
    }

    if (nextFiber != -1) {
        SwitchToFiber(fiberInfo[nextFiber].fiber);
    }
}

// Cleanup
for (int i = 0; i < NUM_FIBERS; i++) {
    DeleteFiber(fibers[i]);
}
```

```
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
Fiber 00000005 is running
running time is:63ms
```

优先级调度是一种根据 Fiber 的优先级来选择下一个执行的策略。在这种情况下，Fibers 的执行顺序受其优先级的影响。具有较高优先级的 Fibers 将更频繁地执行。这可以导致优先级较高的任务更快完成，但可能导致低优先级任务响应时间较长。

综合分析：

彩票调度通常在随机性上具有较大的优势，但可能导致不公平性和较差的响应时间。这在生产者-消费者问题中可能会导致某些生产者或消费者的性能不稳定。

轮询调度是一种简单的调度策略，但可能导致某些 Fibers 等待时间较长，不太适用于需要灵活性和不同任务优先级的场景。

优先级调度在考虑任务优先级时可能具有优势，但需要谨慎管理任务的优先级以避免低优先级任务长时间等待。

在生产者-消费者问题中，选择适当的调度策略取决于具体的性能需求和公平性要求。如果要求高性能且不需要太大的公平性，彩票调度可能是一个不错的选择。如果需要更多的公平性，优先级调度可能更适合，但需要管理好任务的优先级。轮询调度在某些特定情况下可能有用，但通常不太适合要求高性能和公平性的场景。需要根据具体的应用需求来选择最适合的策略。

四、实验问题与解决方案

问题：
实验中执行生产者消费者模型时，规定固定执行数量后运行程序，程序在消费结束后并没有退出然后输出程序执行时间，而是卡在原地无响应。
解决方案：
在生产与消费函数中加入超时中断：

```
// Wait if the buffer is empty with a timeout
if (WaitForSingleObject(consumerSem, 1000) == WAIT_TIMEOUT) {
    printf("Consumer %d timed out waiting for items in the buffer.\n", (int)arg);
    break;  // Exit the loop if timeout occurs
}
```

即可在超时时既结束了固定数量进程运行，又输出了时间。

五、总结与反思

在本次实验中，我们探讨了多线程编程和 Fiber 编程的相关概念和应用。了解多线程和 Fiber 的基本原理对于并发编程非常重要，可以帮助解决各种问题。

在第一个问题中，我们实现了一个生产者-消费者问题的多线程解决方案，使用了互斥锁和信号量来控制线程的同步和协作。这展示了如何在多线程环境中管理共享资源和控制线程的数量。

在第二个问题中，我们实现了不同的 Fiber 调度策略，包括彩票调度、轮询调度和优先级调度。每种策略都有其优势和局限性，需要根据具体需求选择适当的策略。

性能分析对于确定最佳调度策略至关重要。不同策略可能会导致不同的性能表现，因此需要综合考虑吞吐量、响应时间和公平性等因素。

在实验过程中，我们学到了如何使用多线程和 Fiber 来实现并发编程，以及如何选择适当的同步和调度策略。这些技能对于解决实际应用中的并发问题非常有帮助。

总体而言，本次实验提供了一个深入了解多线程和 Fiber 编程的机会，并让我们掌握了处理并发问题的基本原则。随着更多的应用需要并发处理，这些知识和技能将变得越来越重要。