Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Немкович А.В.

ФИТ 3 курс 1 группа

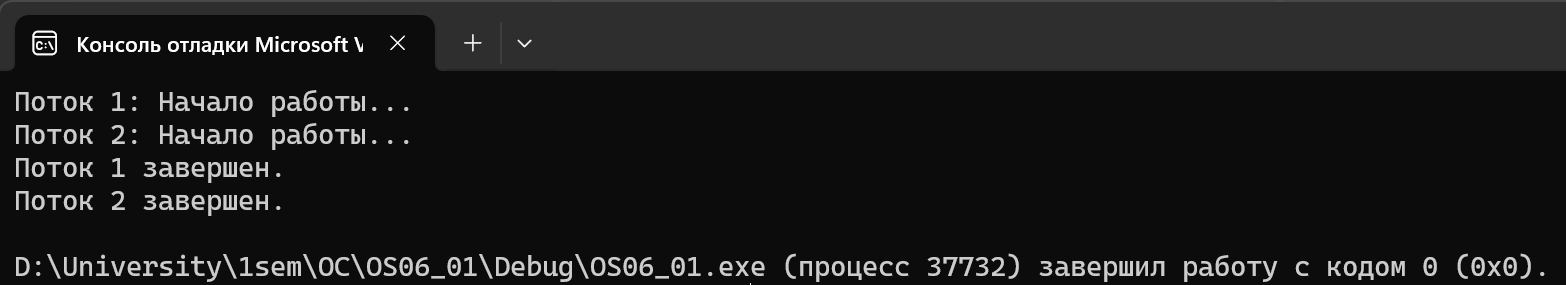
Преподаватель: Бернацкий П.В

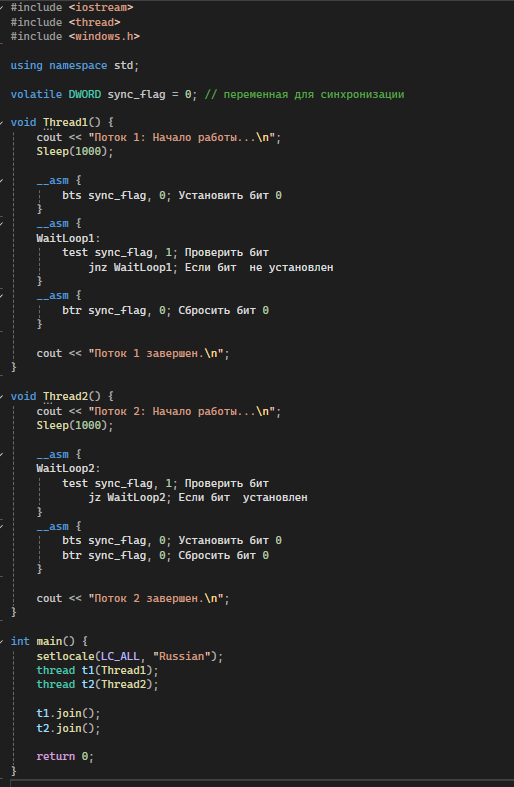
Минск 2024

**Лабораторная работа №6**

**Задание 01**

Напишите ассемблерный код с применением команд BTS или BTR, демонстрирующий реализацию механизма синхронизации двух потоков одного процесса и поясните его работу.





BTS – Bit Test and Set, BTR - Bit Test and Reset. Эти команды используется для изменения значения битов в операнде.

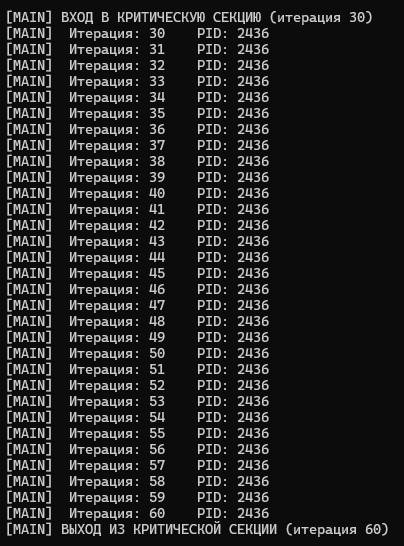
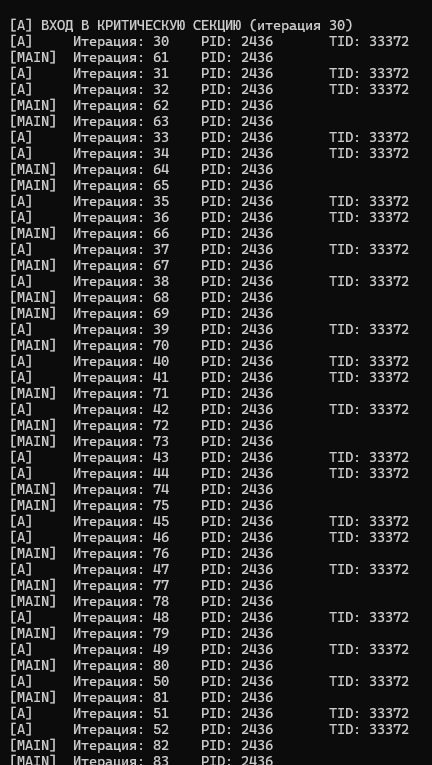
Команда BTS (Bit Set) устанавливает указанный бит в операнде

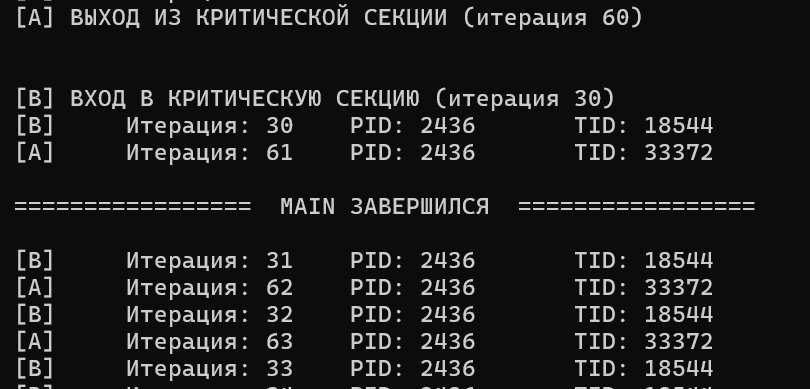
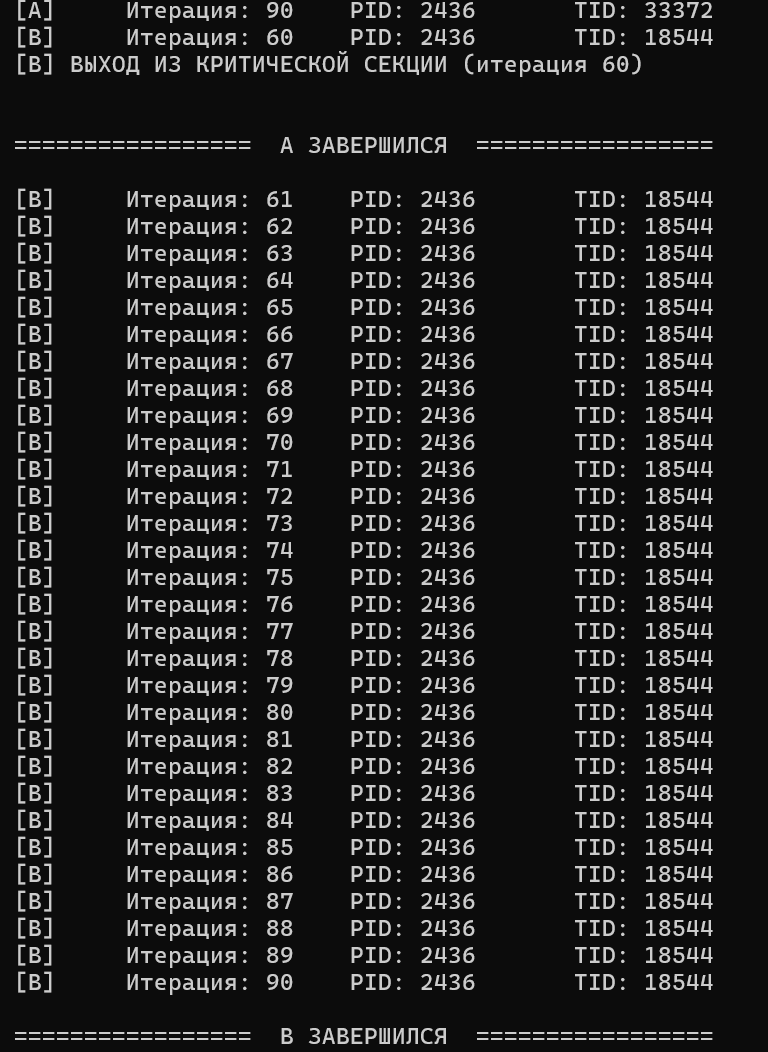
Команда BTR (Bit Test and Reset) сбрасывает указанный бит в операнде (в sync\_flag) в значение 0. Команда проверяет бит в операнде, выполняя побитовую операцию "И" с другим операндом. Если результат операции равен нулю, значит бит в проверяемом месте равен 0. Если результат ненулевой — значит, бит установлен (равен 1).

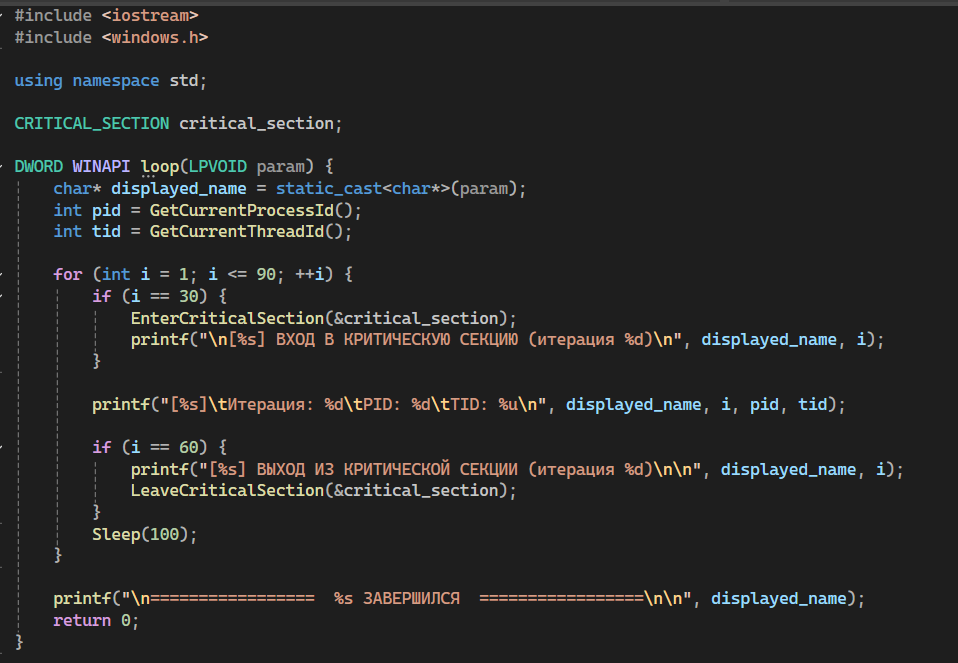
Команда JC (Jump if Carry) выполняет переход по условию, если в предыдущей операции флага переноса (carry flag) был установлен. В данном случае флаг переноса устанавливается, когда выполняется операция BTS, а это значит, что бит был успешно установлен.

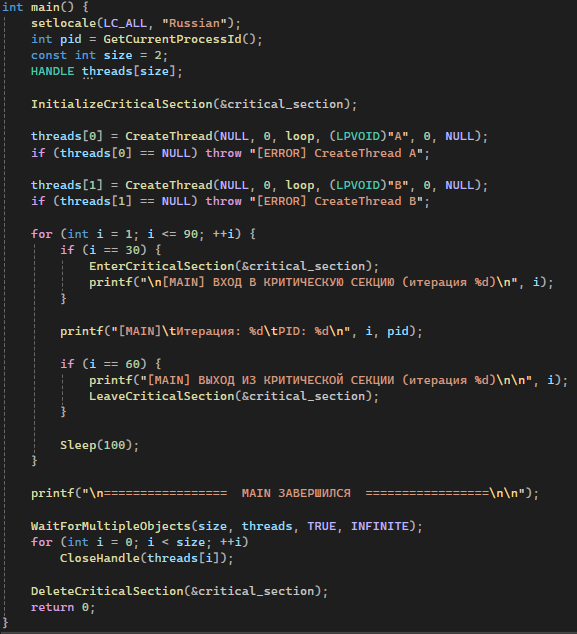
**Задание 02**

Разработайте приложение OS06\_02, запускающее два дочерних потока A и B. Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек. Приложение OS06\_02 синхронизирует выполнение потоков main, A и B с помощью механизма critical section. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 в каждом потоке.





CRITICAL\_SECTION — это объект синхронизации в Windows, который позволяет управлять доступом к разделяемым ресурсам из разных потоков.

1. Перед использованием CRITICAL\_SECTION нужно инициализировать её вызовом функции InitializeCriticalSection.

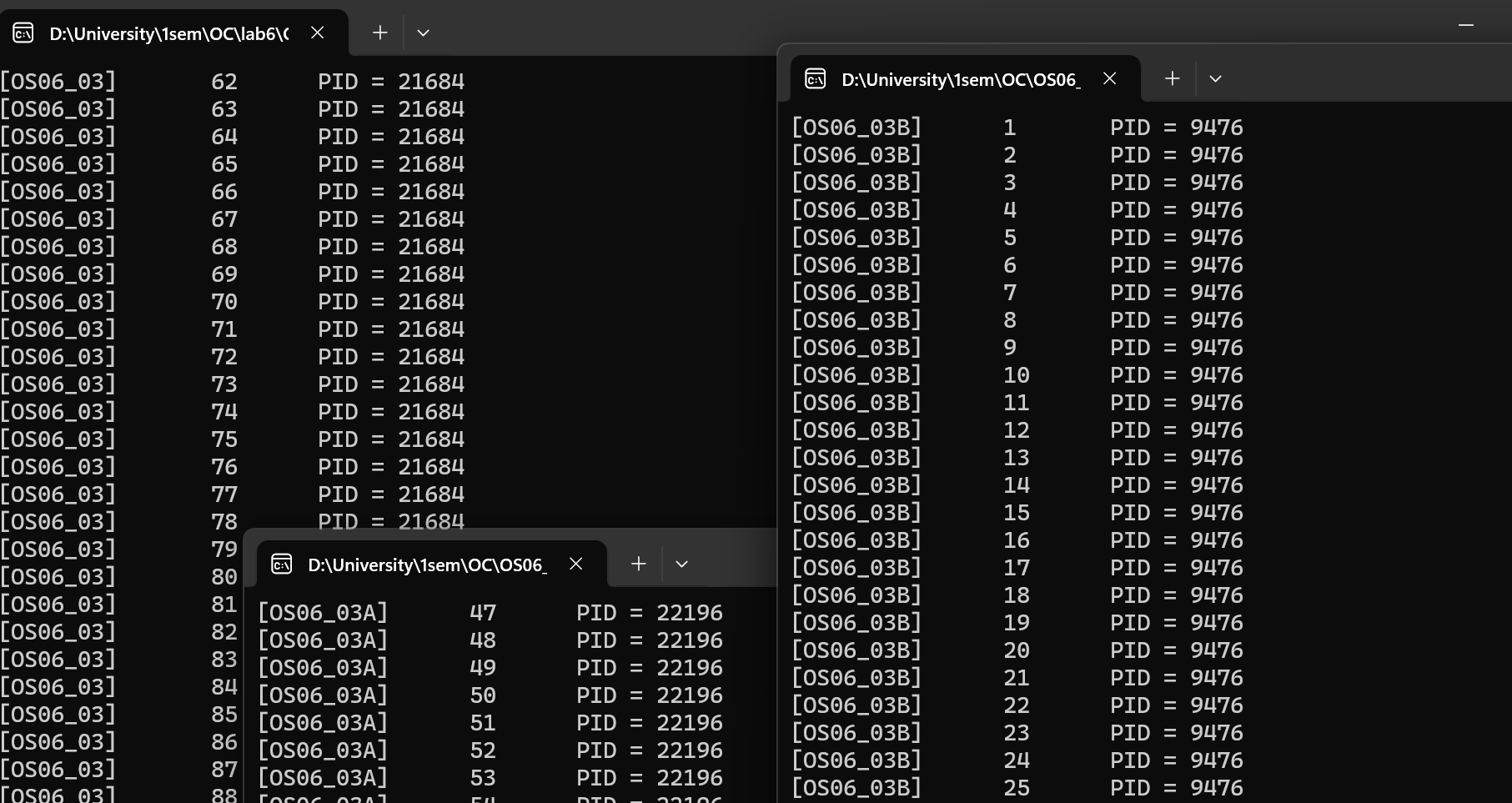
2. Когда поток начинает выполнение кода, защищённого критической секцией, он вызывает EnterCriticalSection. Это действие блокирует доступ другим потокам, заставляя их ждать, пока текущий поток не выйдет из критической секции.

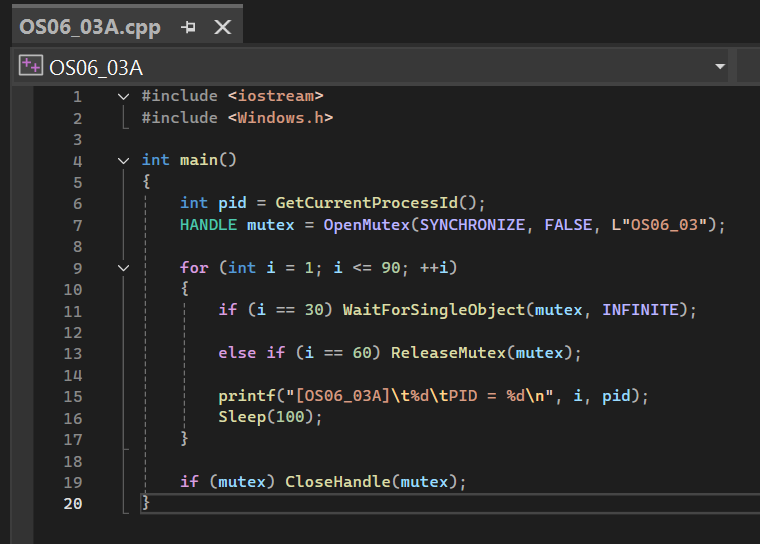
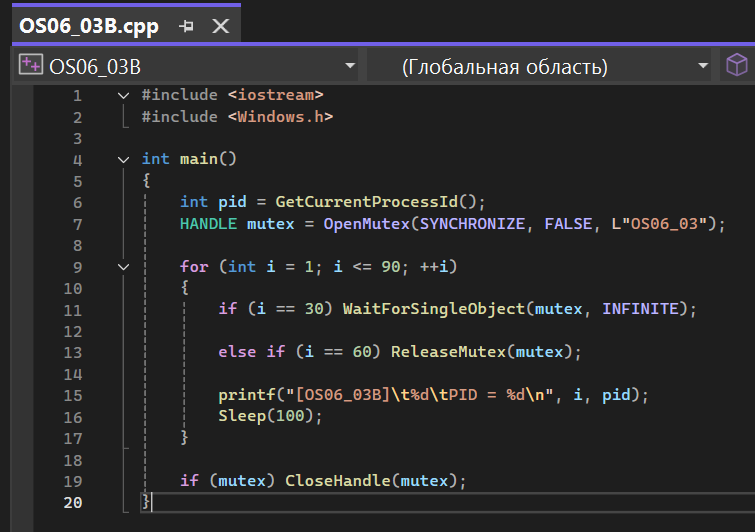
3. Когда поток завершает выполнение защищенного кода, он вызывает LeaveCriticalSection, чтобы разблокировать доступ для других потоков.

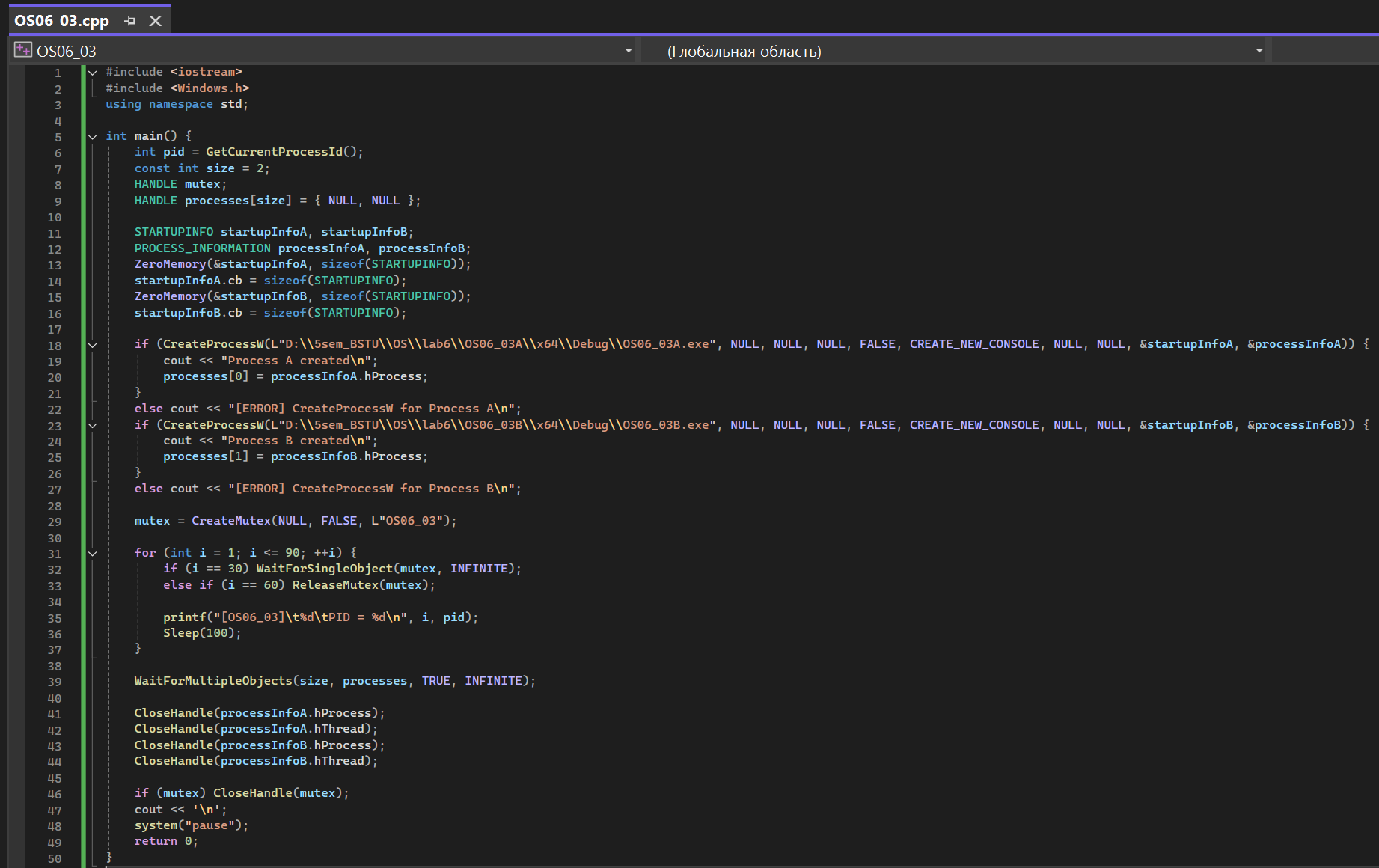
4. В конце работы CRITICAL\_SECTION освобождается вызовом DeleteCriticalSection. Это позволяет очистить ресурсы, выделенные для объекта.

**Задание 03**

Разработайте приложение OS06\_03, запускающее два дочерних процесса OS06\_03A и OS06\_03B с выводом в отдельные консоли.Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек. Приложение OS06\_03 синхронизирует выполнение процессов OS06\_03, OS06\_03A и OS06\_03B с помощью механизма mutex. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60.



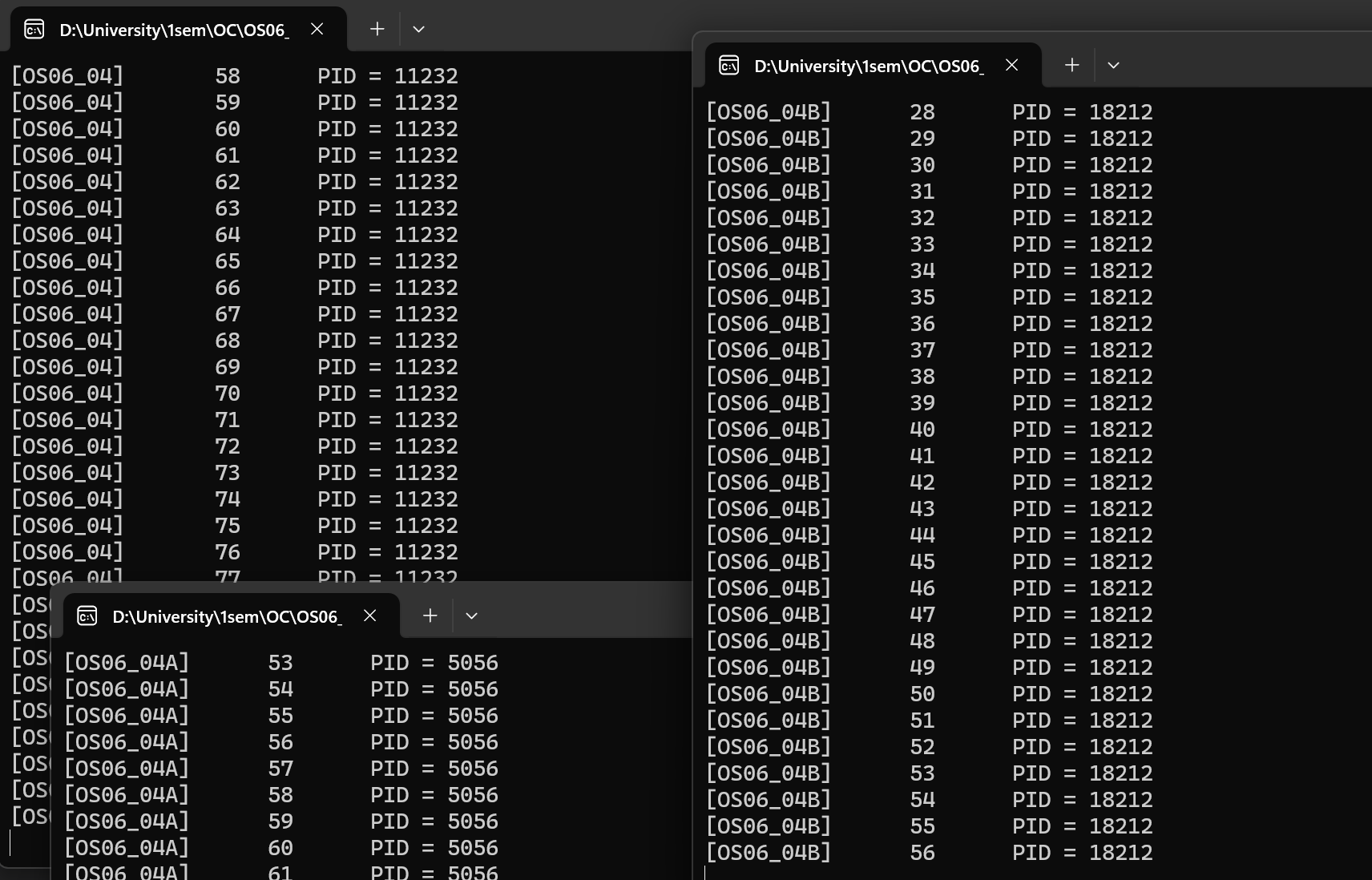


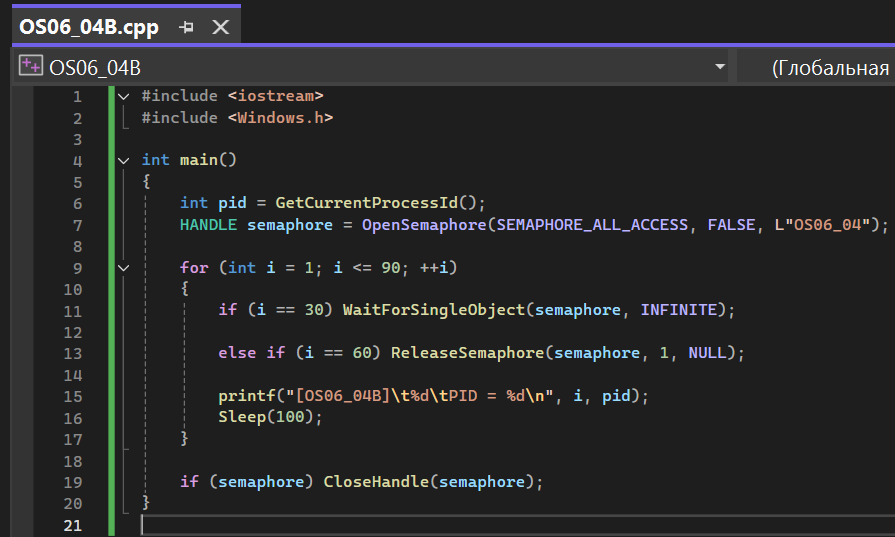
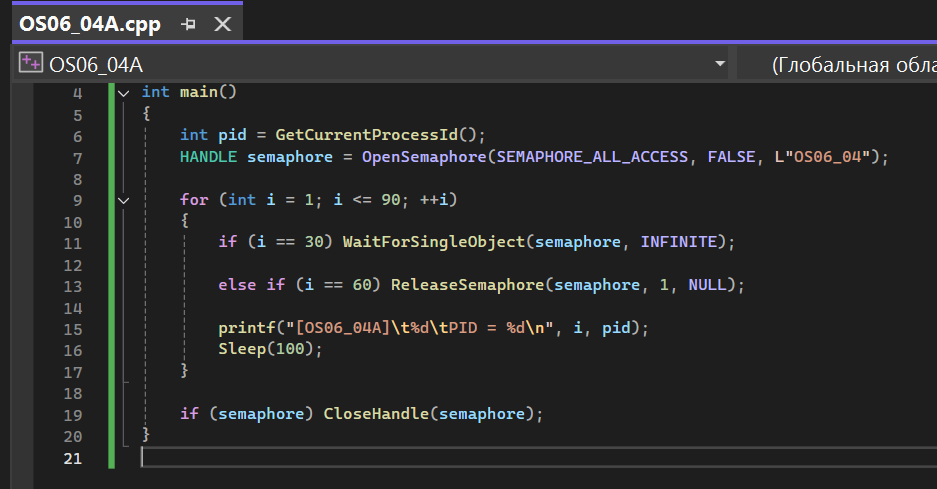
Мьютекс — это объект синхронизации, который используется для управления доступом к общим ресурсам. Он гарантирует, что в любой момент времени к защищённому ресурсу может обратиться только один поток или процесс. Мьютекс находится в сигнальном состоянии, если он не принадлежит ни одному потоку. В противном случае мьютекс находится в несигнальном состоянии. Одновременно мьютекс может принадлежать только одному потоку.

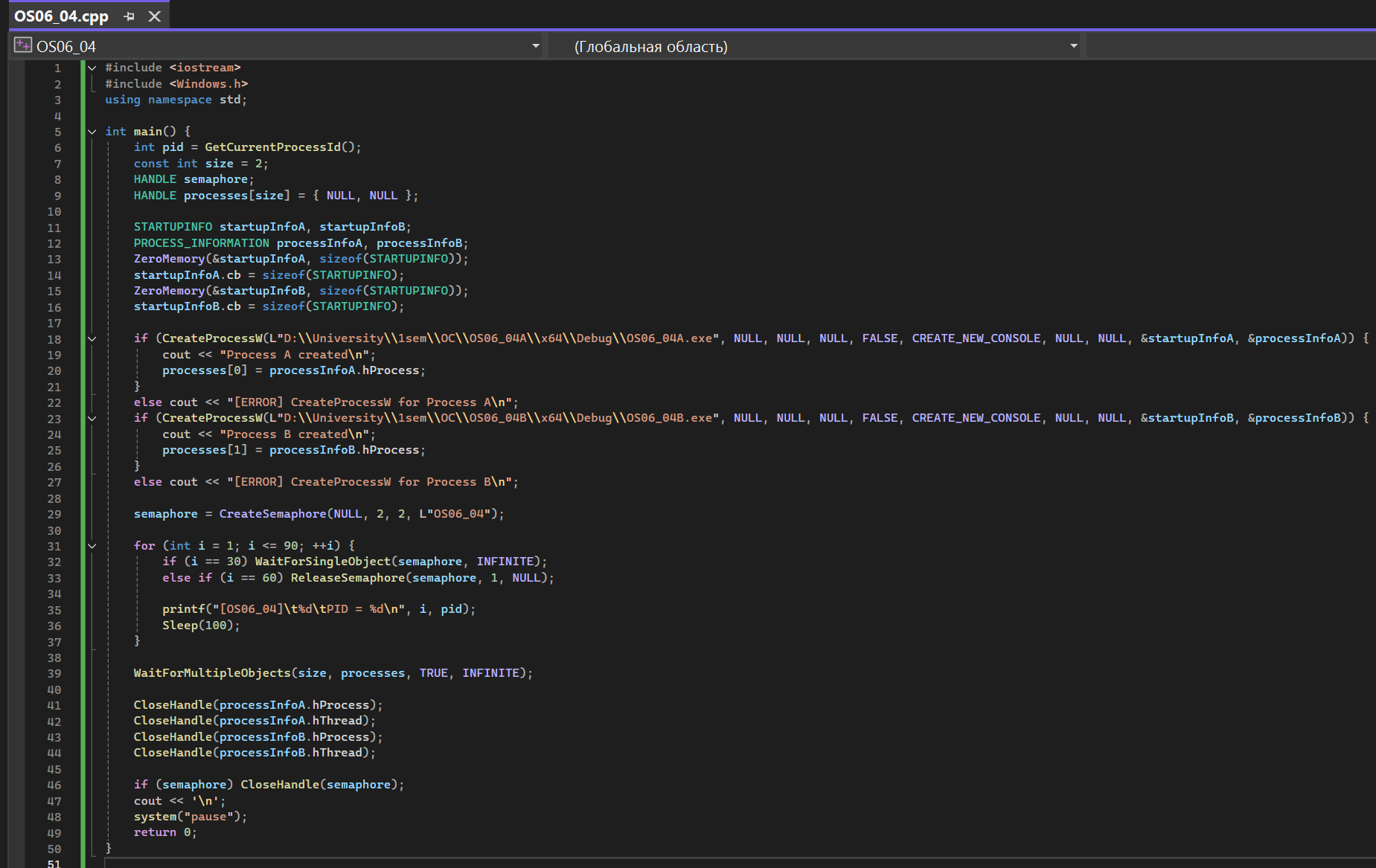
1. Мьютекс создаётся с помощью функции CreateMutex. Мьютекс может быть именованным или безымянным: Безымянный мьютекс используется только в пределах одного процесса. Именованный мьютекс доступен для всех процессов в операционной системе, которые знают его имя.
2. Когда поток или процесс хочет получить доступ к защищённому ресурсу, он вызывает функцию ожидания мьютекса WaitForSingleObject. Если мьютекс свободен, поток захватывает его, и теперь мьютекс считается занятым. Другие потоки или процессы, которые попытаются захватить этот мьютекс, будут заблокированы до тех пор, пока он не освободится. Если мьютекс занят, поток или процесс попадает в состояние ожидания, пока мьютекс не будет освобождён.
3. Когда поток или процесс завершает работу с защищённым ресурсом, он освобождает мьютекс с помощью функции ReleaseMutex. Это делает мьютекс доступным для захвата другим потоком или процессом.
4. После завершения работы мьютекс должен быть закрыт с помощью функции CloseHandle. Это освобождает ресурсы, занятые мьютексом.

**Задание 04**

Разработайте приложение OS06\_04, запускающее два дочерних процесса OS06\_04A и OS06\_04B с выводом в отдельные консоли. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек. Приложение OS06\_04 синхронизирует выполнение процессов OS06\_04, OS06\_04A и OS06\_04B с помощью механизма semaphore. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 одного (любого) процесса и двух других процессов. Другими словами, итерации с 30 по 60 должны одновременно выполняться только в двух из трех процессов.







1. При создании семафору назначается максимальное количество разрешений, которые он может выдавать. Если ресурс может использоваться максимум 3 потоками, то начальное значение счетчика будет равно 3.
2. Когда поток хочет получить доступ к ресурсу, он запрашивает разрешение у семафора. Если счетчик семафора больше 0 (т.е., есть свободные разрешения), семафор "выдает разрешение" (уменьшает счетчик на 1) и позволяет потоку продолжить выполнение. Если разрешений нет (счетчик равен 0), поток блокируется и ждет, пока семафор снова освободит разрешение.
3. После завершения работы с ресурсом поток освобождает разрешение, увеличивая значение счетчика на 1. Если в очереди ожидания есть заблокированные потоки, один из них "разблокируется" и получит освобожденное разрешение.

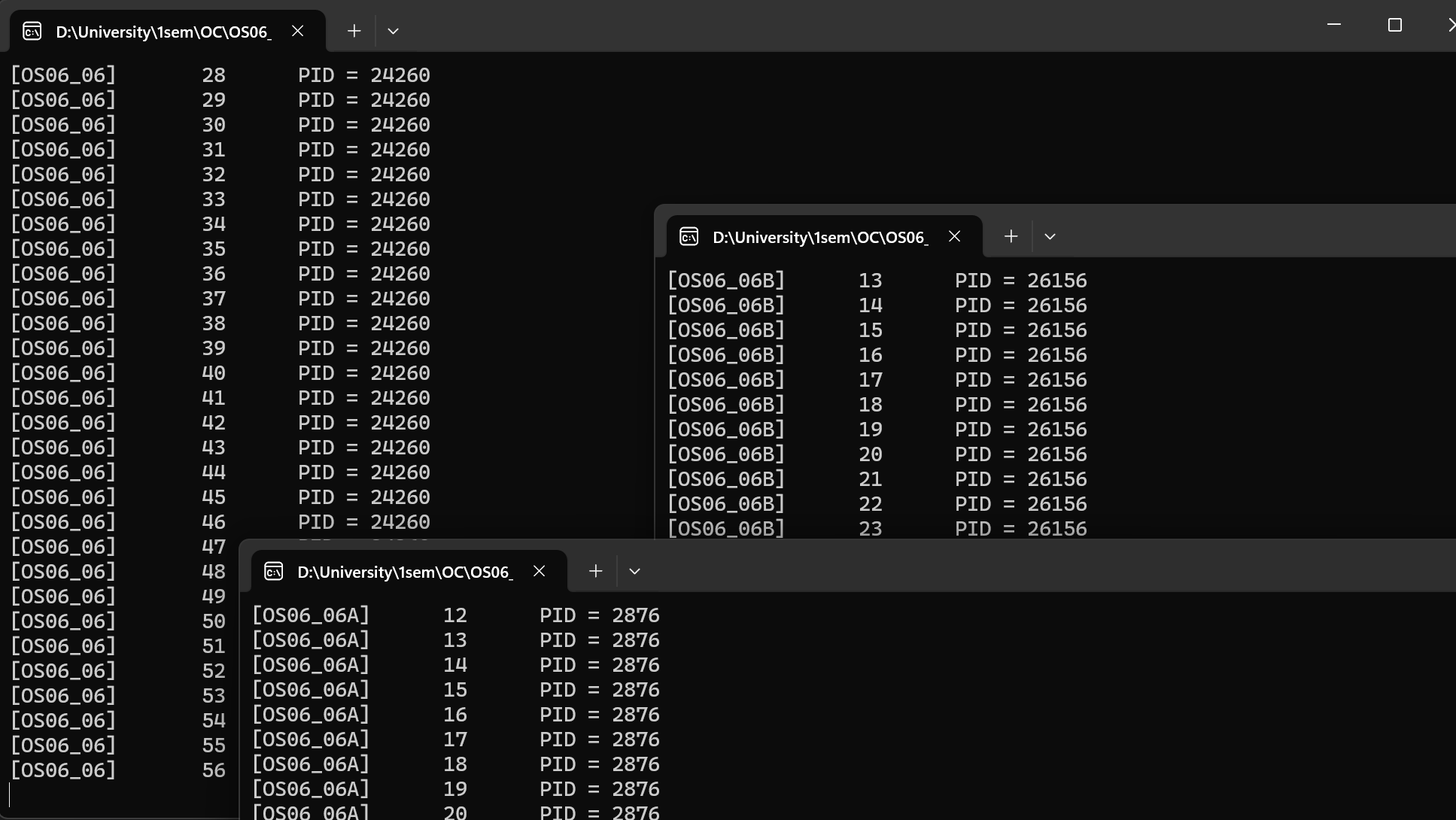
**Задание 05**

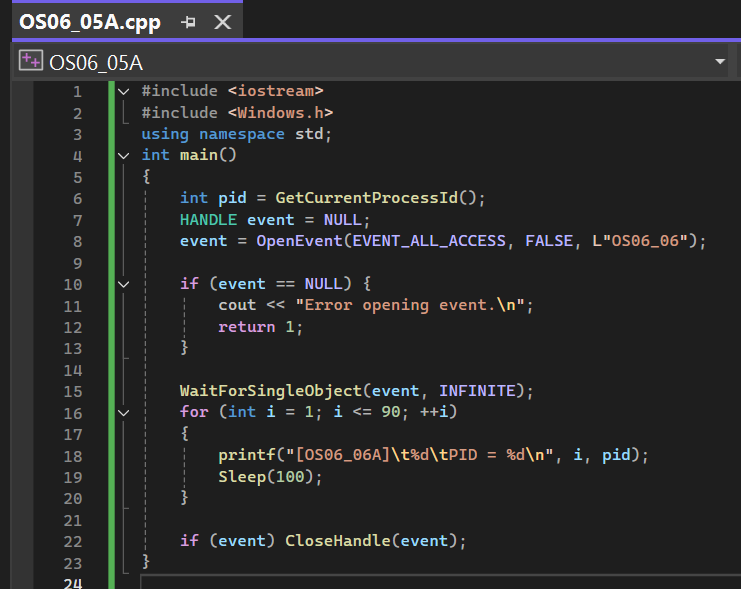
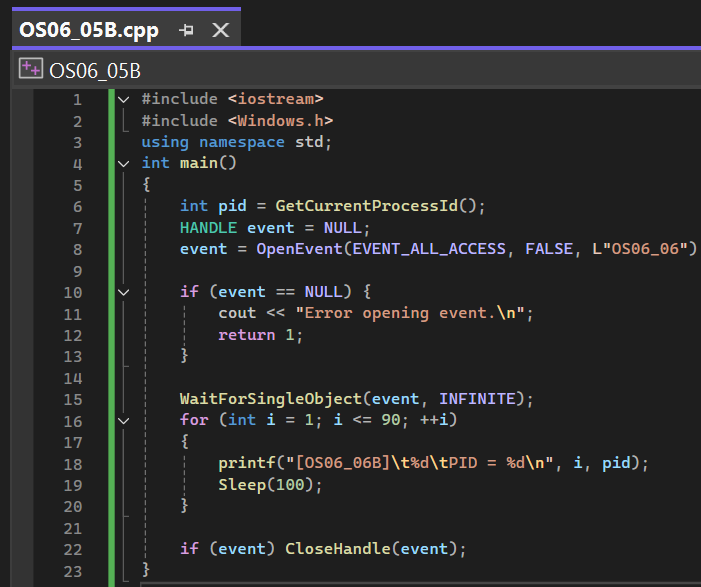
Разработайте приложение OS06\_05, запускающее два дочерних процесса OS06\_05A и OS06\_05B с выводом в отдельные консоли. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек. Приложение OS06\_05 синхронизирует выполнение процессов OS06\_05, OS06\_05A и OS06\_05B с помощью механизма event.

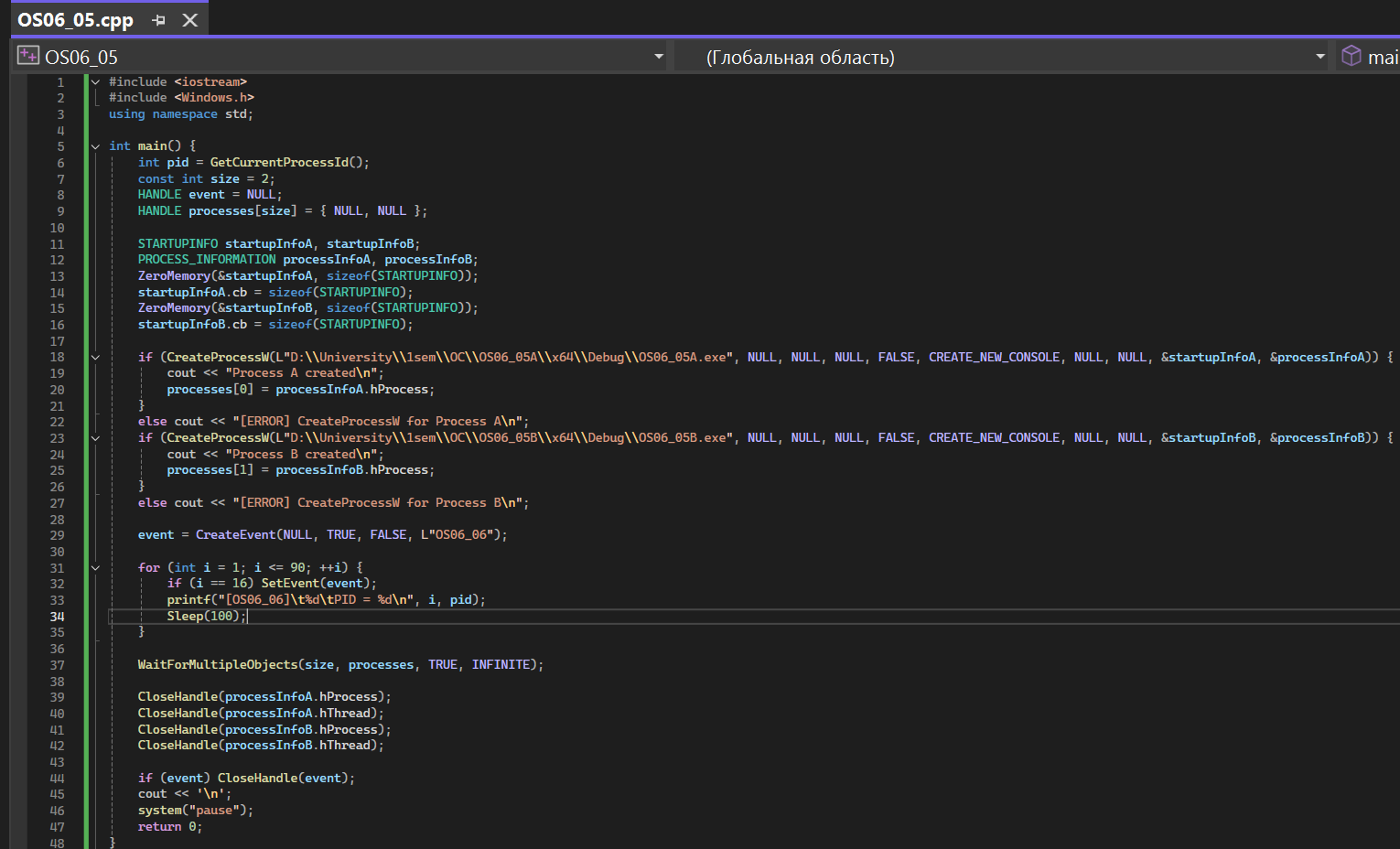
Синхронизация должна обеспечивать выполнение приложения в два этапа:

1) выполнение итераций с 1 по 15 процесса OS06\_05;

2) одновременное выполнение всех трех процессов: OS06\_05 – продолжает выполнение итераций; процессы OS06\_05A и OS06\_05B выполняются начиная с первой итерации.





**События** (events) являются механизмом синхронизации, позволяющим координировать действия между несколькими потоками или процессами. Событие в данном контексте представляет собой объект ядра, который может находиться в двух состояниях: **установленное** (signaled) и **неустановленное** (nonsignaled). События используются для того, чтобы процессы или потоки ждали до наступления определённого момента или действия.

В основном процессе вызывается функция CreateEvent для создания события. При создании события второй параметр указывает на автоматический сброс события (то есть после того, как событие будет установлено, оно автоматически сбросится в состояние nonsignaled, как только первый процесс, ожидающий его, продолжит выполнение).

После запуска, каждый дочерний процесс пытается получить доступ к событию, вызвав функцию OpenEvent.

После успешного подключения к событию, каждый дочерний процесс использует WaitForSingleObject, чтобы ждать, пока событие станет установленным. Пока событие находится в состоянии nonsignaled (неустановленном), дочерние процессы продолжают находиться в состоянии ожидания и не переходят к выполнению основного кода.

SetEvent переводит состояние события из nonsignaled в signaled. Это означает, что теперь все процессы, которые ждали установку этого события (дочерние процессы), могут продолжить выполнение своего кода.

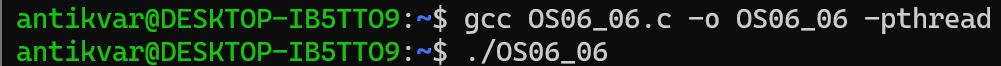
По завершении основного кода основной процесс закрывает дескриптор события с помощью CloseHandle.

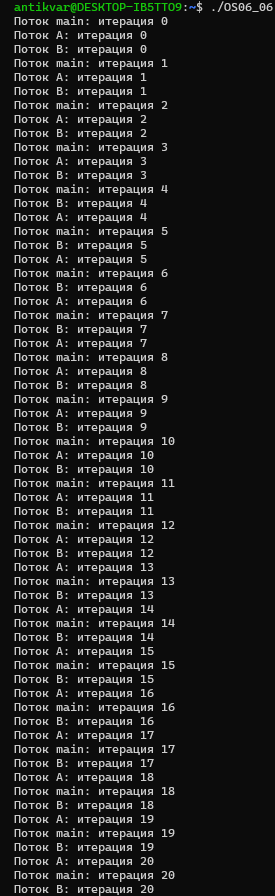
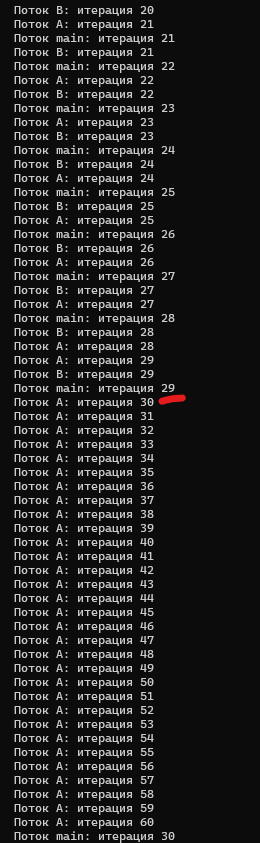
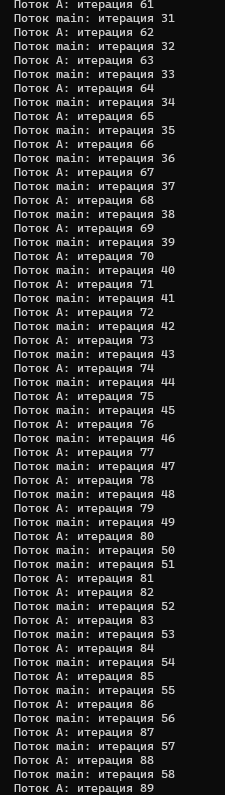
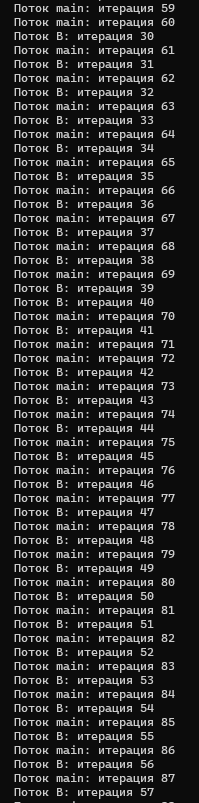
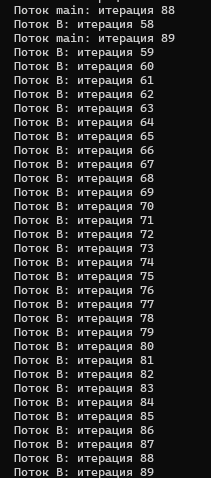
**Задание 06 -LINUX**

Разработайте приложение OS06\_06, запускающее два дочерних потока A и B. Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек. Приложение OS06\_06 синхронизирует выполнение потоков main, A и B с помощью механизма mutex. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60 в каждом потоке.







**Задание 07. Ответьте на следующие вопросы**

**1. Дайте определение понятию «синхронизация потоков».**

Синхронизация потоков — это процесс координации выполнения нескольких потоков в многозадачной среде для обеспечения их правильного взаимодействия и предотвращения конфликтов при доступе к общим ресурсам, таким как память, файлы или устройства.

**2. Объясните понятие «взаимная блокировка».**

**Взаимная блокировка (или deadlock)** — это состояние в многопоточной программе, когда два или более потоков (или процессов) находятся в состоянии ожидания ресурсов, заблокированных друг другом, что приводит к бесконечному ожиданию и невозможности завершить работу.

**3. Перечислите механизмы авторизации OS.**

Механизмы синхронизации: критические секции, мьютексы, семафоры, события, барьеры, атомарные операции

Атомарные операции — это операции, которые выполняются как единое целое, недоступное для других потоков. Это означает, что они не могут быть прерваны или нарушены.

Барьеры — это механизмы, которые позволяют потокам синхронизироваться на определенной точке выполнения. Все потоки должны достичь барьера, прежде чем любой из них сможет продолжить выполнение.

**4. Поясните в чем разница между механизмом mutex и semaphore.**

Мьютекс используется для эксклюзивного доступа к ресурсу. Это объект синхронизации, который разрешает доступ только одному потоку в конкретный момент времени. Мьютекс может быть освобожден только тем потоком, который его захватил. Это предотвращает ситуации, когда один поток пытается освободить мьютекс, захваченный другим потоком.

Семафор используется для управления доступом к ограниченному ресурсу. Он имеет внутренний счетчик, который определяет, сколько потоков могут одновременно получить доступ к ресурсу. Семафор позволяет большему количеству потоков одновременно захватывать ресурс, в отличие от мьютекса, который разрешает доступ только одному потоку.

Семафоры могут быть двух типов:

Бинарный семафор (двоичный семафор): похож на мьютекс (считает 0 или 1), только в этом случае может быть освобожден любым потоком, а не только тем, который его захватил.

Подсчитывающий семафор: использует счетчик для управления количеством доступных ресурсов (например, ограниченное количество потоков, которые могут одновременно использовать ресурс).

**5. Почему mutex, semaphore, event создают объект ядра OS, а critical section нет.**

Mutex, Semaphore, Event синхронизации являются объектами ядра операционной системы. Это означает, что они поддерживаются и управляются ядром ОС, которое отвечает за их создание, управление состоянием и синхронизацию. Такие объекты могут использовать механизмы ОС для организации блокировки, сигнализации и ожидания между потоками. Они также могут работать между различными процессами, так как поддерживаются в контексте всего операционного окружения.

Critical Section — это более легкий механизм синхронизации, который управляется в контексте конкретного процесса. Это значит, что он существует только внутри одного процесса и не требует взаимодействия с ядром ОС.