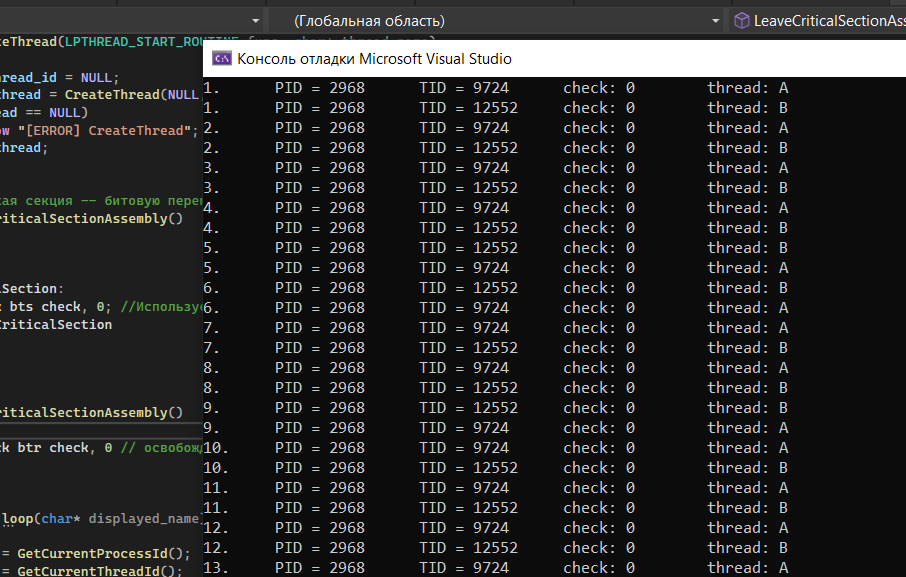
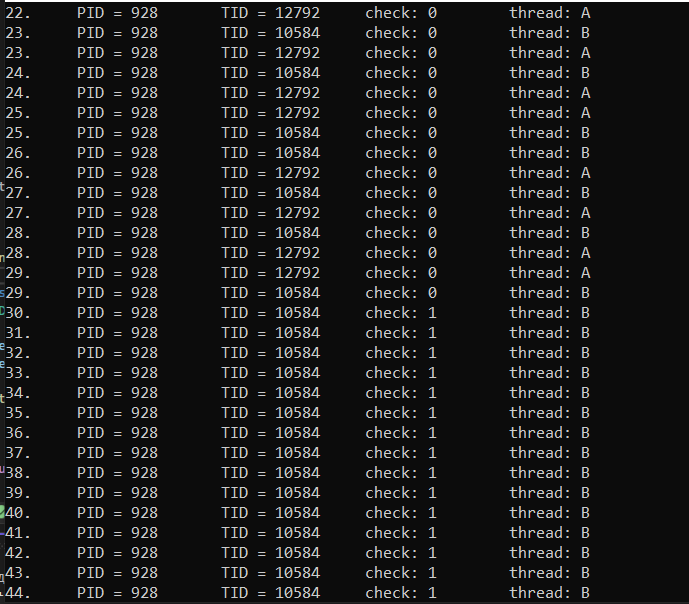
**Задание 01. Windows**

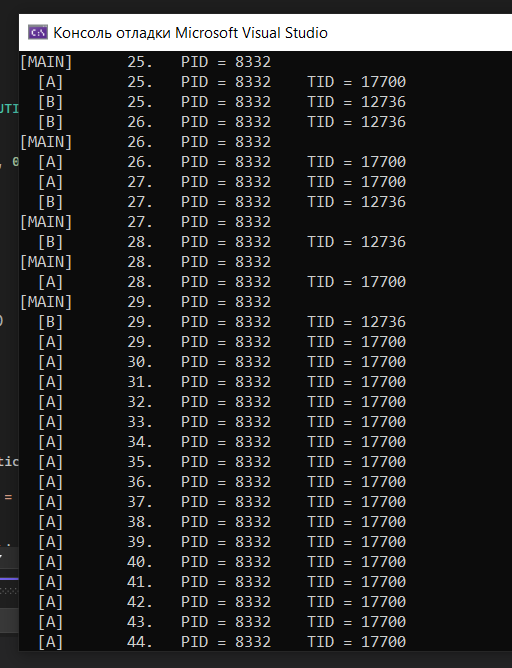
1. Напишите ассемблерный код с применением команд BTS или BTR, демонстрирующий реализацию механизма синхронизации двух потоков одного процесса и поясните его работу.





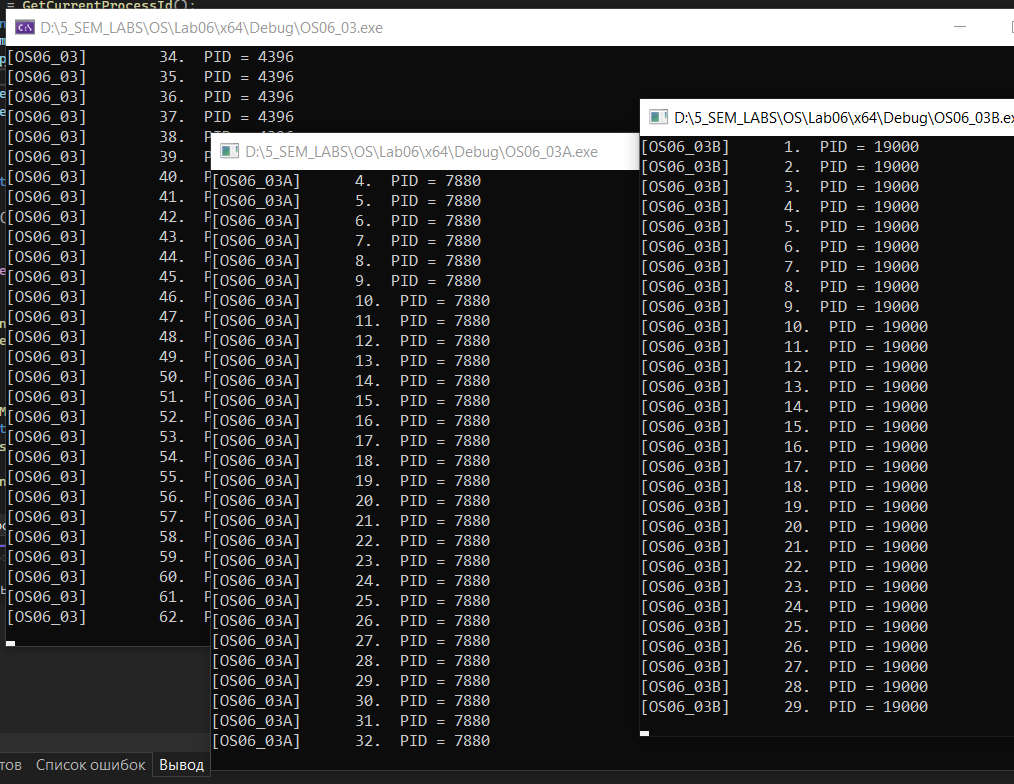
**Задание 02. Windows**

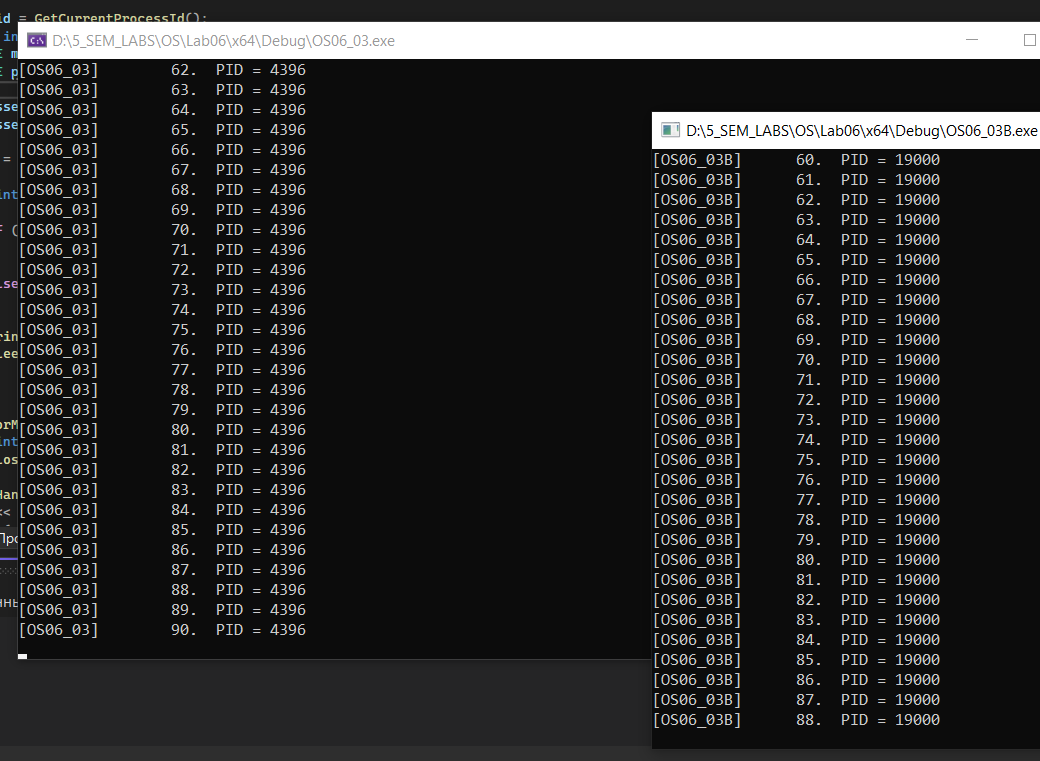
1. Разработайте приложение **OS06\_02**, запускающее два дочерних потока **A** и **B**.
2. Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_02** синхронизирует выполнение потоков **main**, **A** и **B** с помощью механизма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



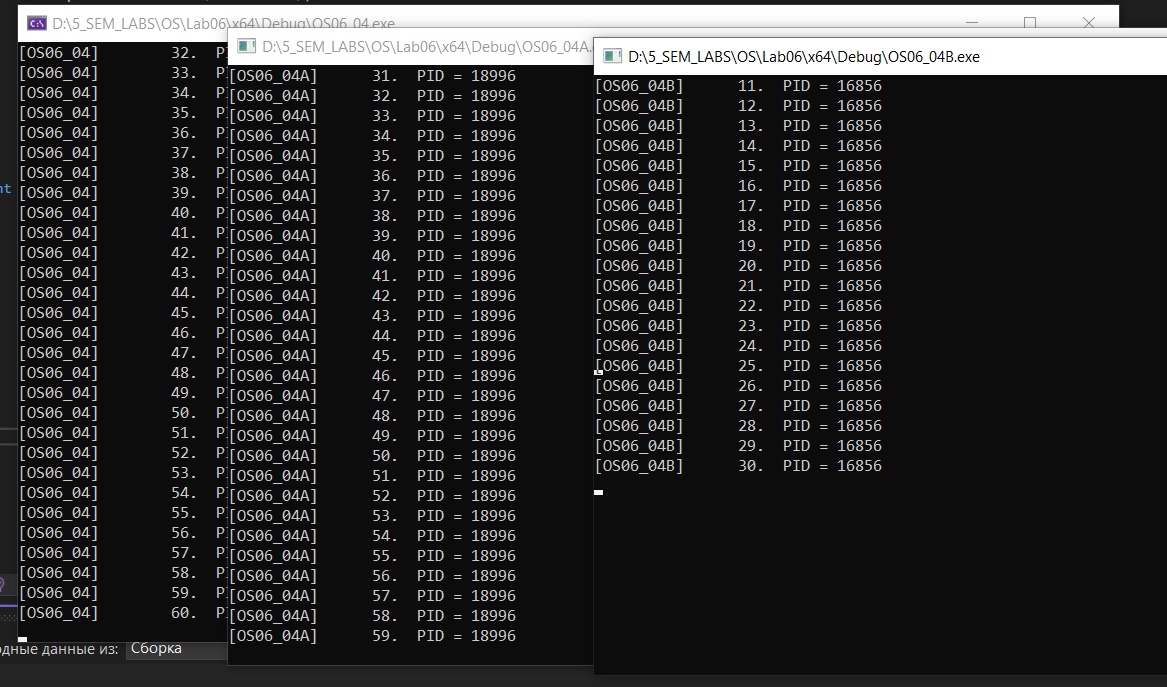
**Задание 03. Windows**

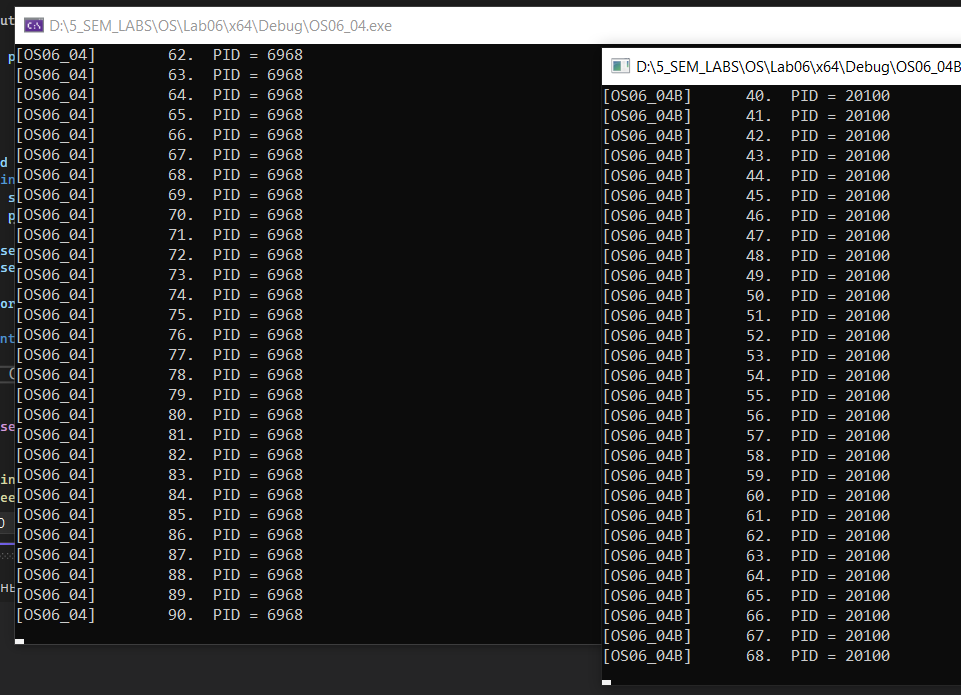
1. Разработайте приложение **OS06\_03**, запускающее два дочерних процесса **OS06\_03A** и **OS06\_03B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_03** синхронизирует выполнение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





**Задание 04. Windows**

1. Разработайте приложение **OS06\_04**, запускающее два дочерних процесса **OS06\_04A** и **OS06\_04B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_04** синхронизирует выполнение процессов **OS06\_04**, **OS06\_04A** и **OS06\_04B** с помощью механизма **semaphore.**
4. 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.
5. 



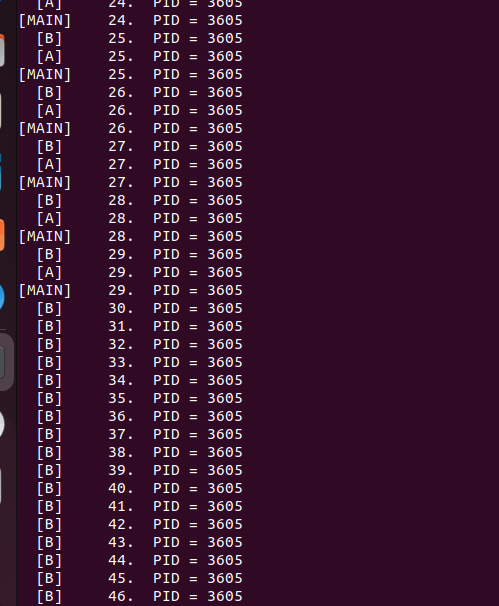
**Задание 05. Windows**

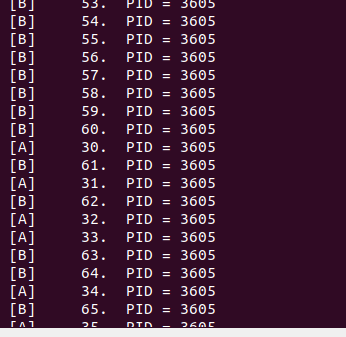
1. Разработайте приложение **OS06\_05**, запускающее два дочерних процесса **OS06\_05A** и **OS06\_05B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_05** синхронизирует выполнение процессов **OS06\_05**, **OS06\_05A** и **OS06\_05B** с помощью механизма **event**.
4. Синхронизация должна обеспечивать выполнение приложения в два этапа:
5. выполнение итераций с 1 по 15 процесса **OS06\_05**;
6. одновременное выполнение всех трех процессов: **OS06\_05** – продолжает выполнение итераций; процессы **OS06\_05A** и **OS06\_05B** выполняются начиная с первой итерации.

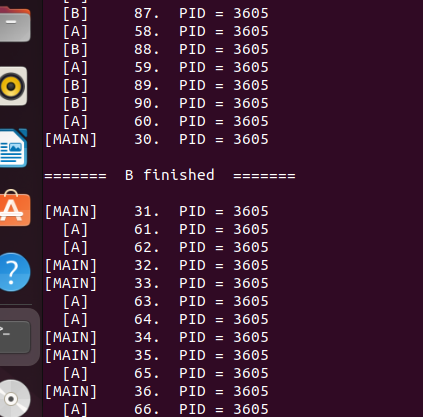


**Задание 06.Linux**

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







**22. Что такое «синхронизация потоков»?**

**Синхронизация потоков** — это процесс управления последовательностью выполнения потоков в многопоточном приложении. Она необходима для обеспечения корректного доступа к общим ресурсам и предотвращения состояний гонки, когда несколько потоков одновременно пытаются изменить одни и те же данные. Синхронизация помогает избежать ошибок, связанных с некорректным порядком выполнения операций, и обеспечивает согласованность данных.

**23. Что такое «взаимная блокировка»?**

**Взаимная блокировка** (deadlock) — это состояние в многопоточном или многопроцессном окружении, когда два или более потоков (или процессов) блокируют друг друга, ожидая освобождения ресурсов, которые они захватили. Например, поток A удерживает ресурс 1 и ждет ресурс 2, в то время как поток B удерживает ресурс 2 и ждет ресурс 1. В результате ни один из потоков не может продолжить выполнение, возникшая ситуация называется взаимной блокировкой.

**1. Мьютексы (Mutex)**

* **Описание**: Предназначен для взаимного исключения. Позволяет только одному потоку владеть мьютексом в любой момент времени.
* **Использование**: Когда поток захватывает мьютекс, другие потоки, пытающиеся его захватить, блокируются до его освобождения.

**2. Семафоры (Semaphore)**

* **Описание**: Позволяет контролировать доступ к ресурсу с помощью счетчика. Семафор может разрешить доступ нескольким потокам одновременно.
* **Использование**: Если счетчик больше нуля, поток может получить доступ к ресурсу. Если он равен нулю, поток блокируется до освобождения ресурса.

**3. События (Event)**

* **Описание**: События используются для сигнализации между потоками. Один поток может установить (сигнализировать) событие, в то время как другой поток может ожидать его.
* **Использование**: Полезно для синхронизации выполнения потоков, когда один поток должен ждать завершения работы другого.

**4. Критические секции (Critical Section)**

* **Описание**: Легковесный механизм синхронизации, используемый для защиты доступа к общим ресурсам в пределах одного процесса.
* **Использование**: Позволяет избежать гонок данных, но не может использоваться между процессами.

**5. Мониторы (Monitor)**

* **Описание**: Высокоуровневый механизм синхронизации, который объединяет данные и операции, которые могут управлять доступом к ним.
* **Использование**: Автоматически управляет доступом, предоставляя методы для блокировки и разблокировки.

**6. Барьеры (Barrier)**

* **Описание**: Синхронизация нескольких потоков или процессов, которая заставляет их ждать друг друга в определенной точке выполнения.
* **Использование**: Все потоки должны достигнуть барьера, прежде чем они смогут продолжить выполнение.

**Разница между механизмом mutex и semaphore**

* **Mutex (взаимное исключение)**:
  + Предназначен для управления доступом к ресурсу, позволяя только одному потоку владеть мьютексом в любой момент времени.
  + Если поток пытается захватить мьютекс, который уже занят, он блокируется до тех пор, пока мьютекс не будет освобожден.
* **Semaphore (семафор)**:
  + Позволяет контролировать доступ к ресурсу, но может разрешить доступ нескольким потокам одновременно. Семафор имеет счетчик, который указывает, сколько потоков могут одновременно получить доступ к ресурсу.
  + Если счетчик больше нуля, поток может увеличить счетчик и получить доступ к ресурсу. Если счетчик равен нулю, поток будет заблокирован до тех пор, пока другой поток не освободит ресурс.

 **Объекты ядра** (такие как mutex, semaphore и event) управляются операционной системой и имеют свои дескрипторы, что позволяет использовать их между процессами. Они могут быть доступны из разных процессов и обеспечивают синхронизацию на уровне ядра.

 **Critical Section** (критическая секция) — это механизм синхронизации, который используется только в пределах одного процесса. Она более легковесна и быстрее, чем объекты ядра, так как не требует переключения контекста между пользовательским и системным режимами. Это делает критическую секцию менее универсальной, но более эффективной для синхронизации потоков внутри одного процесса.