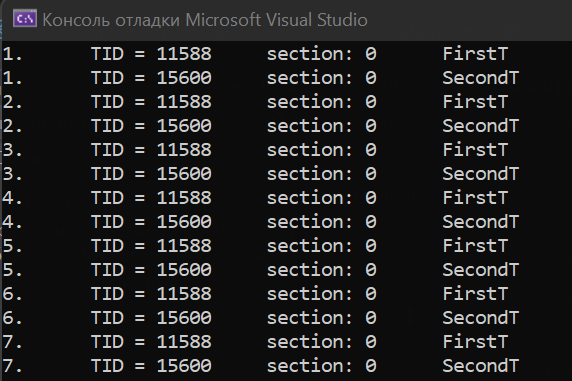
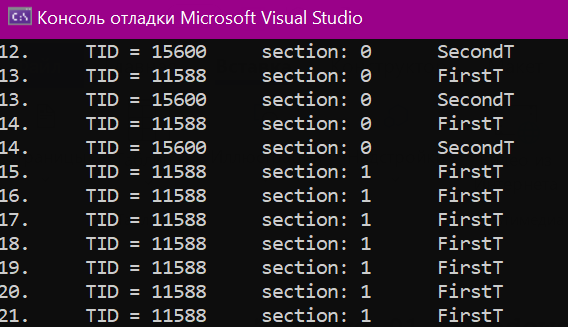
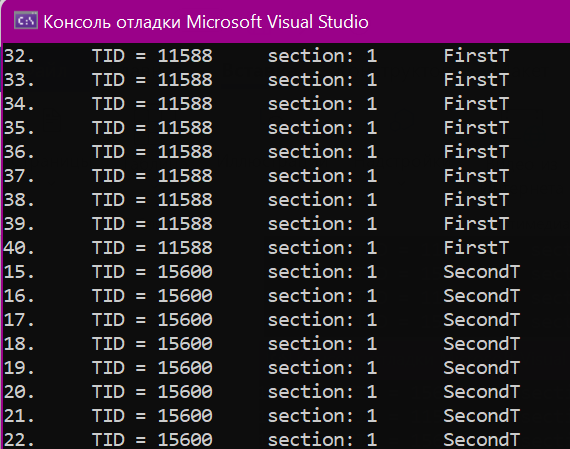
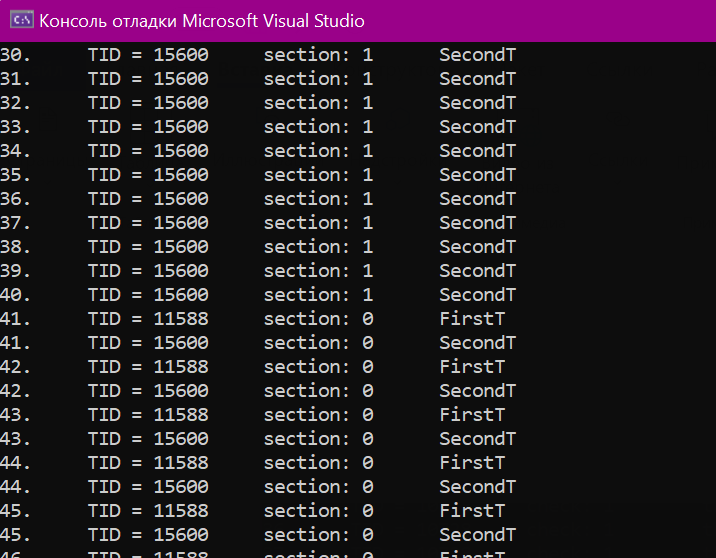
**Задание 01. Windows**

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



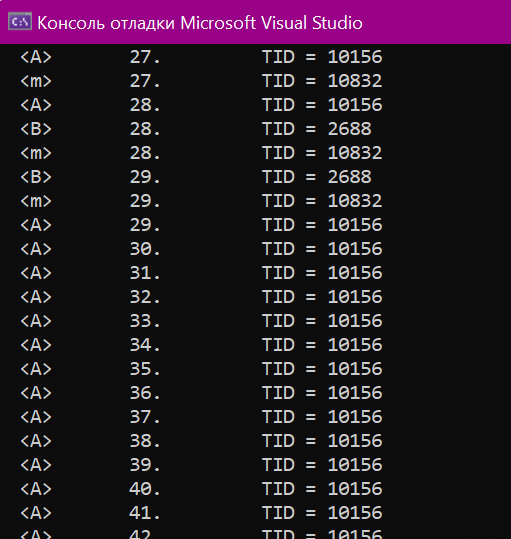


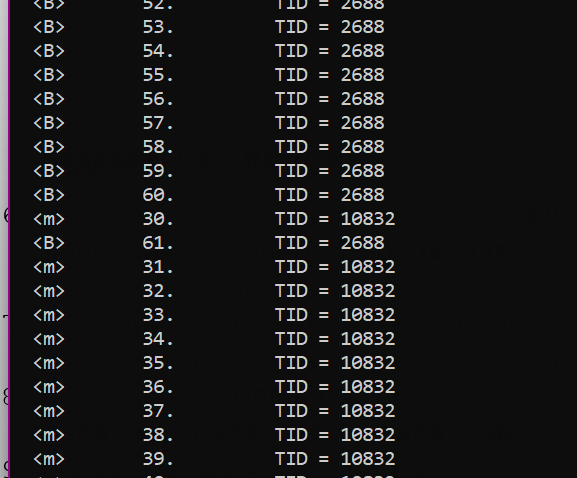


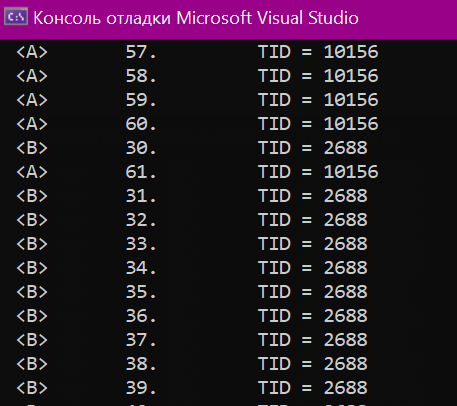


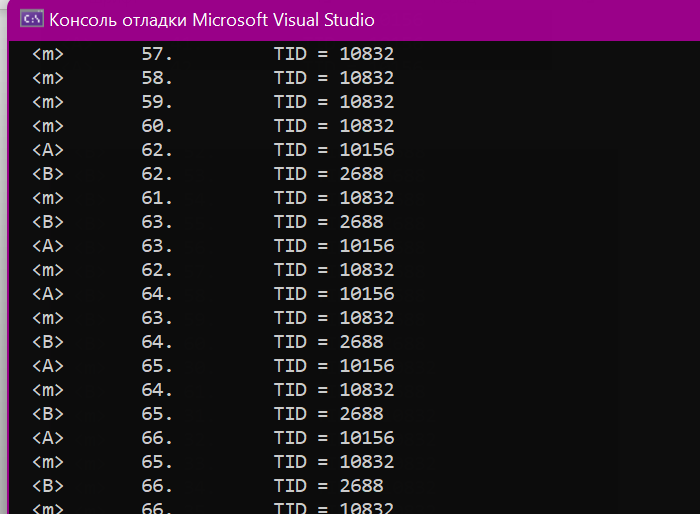
**Задание 02. Windows**

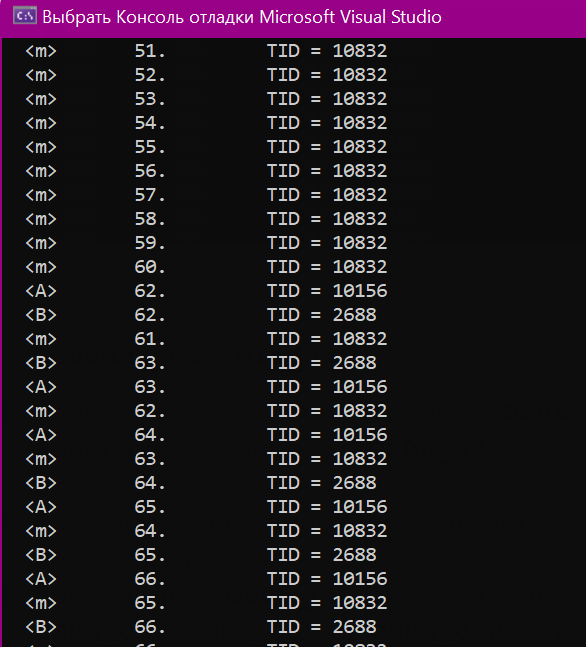
1. Разработайте приложение **OS06\_02**, запускающее два дочерних потока **A** и **B**.
2. Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_02** синхронизирует выполнение потоков **main**, **A** и **B** с помощью механизма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

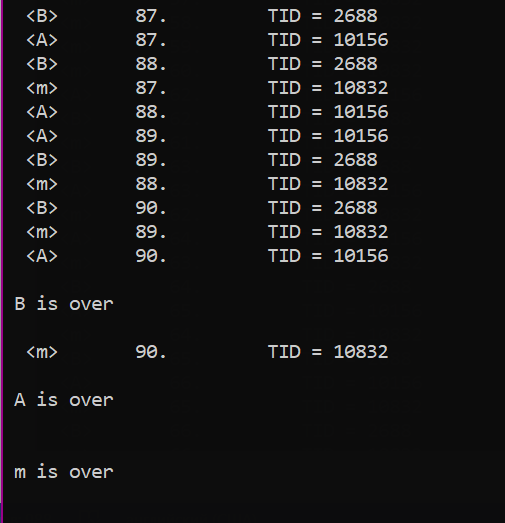






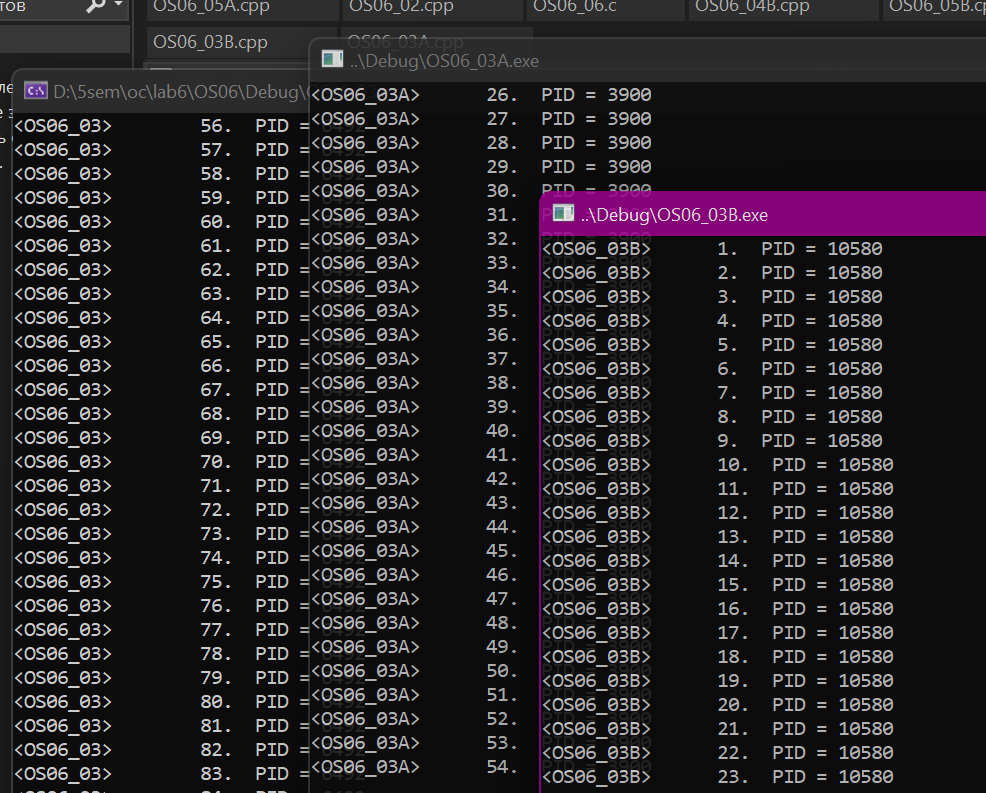


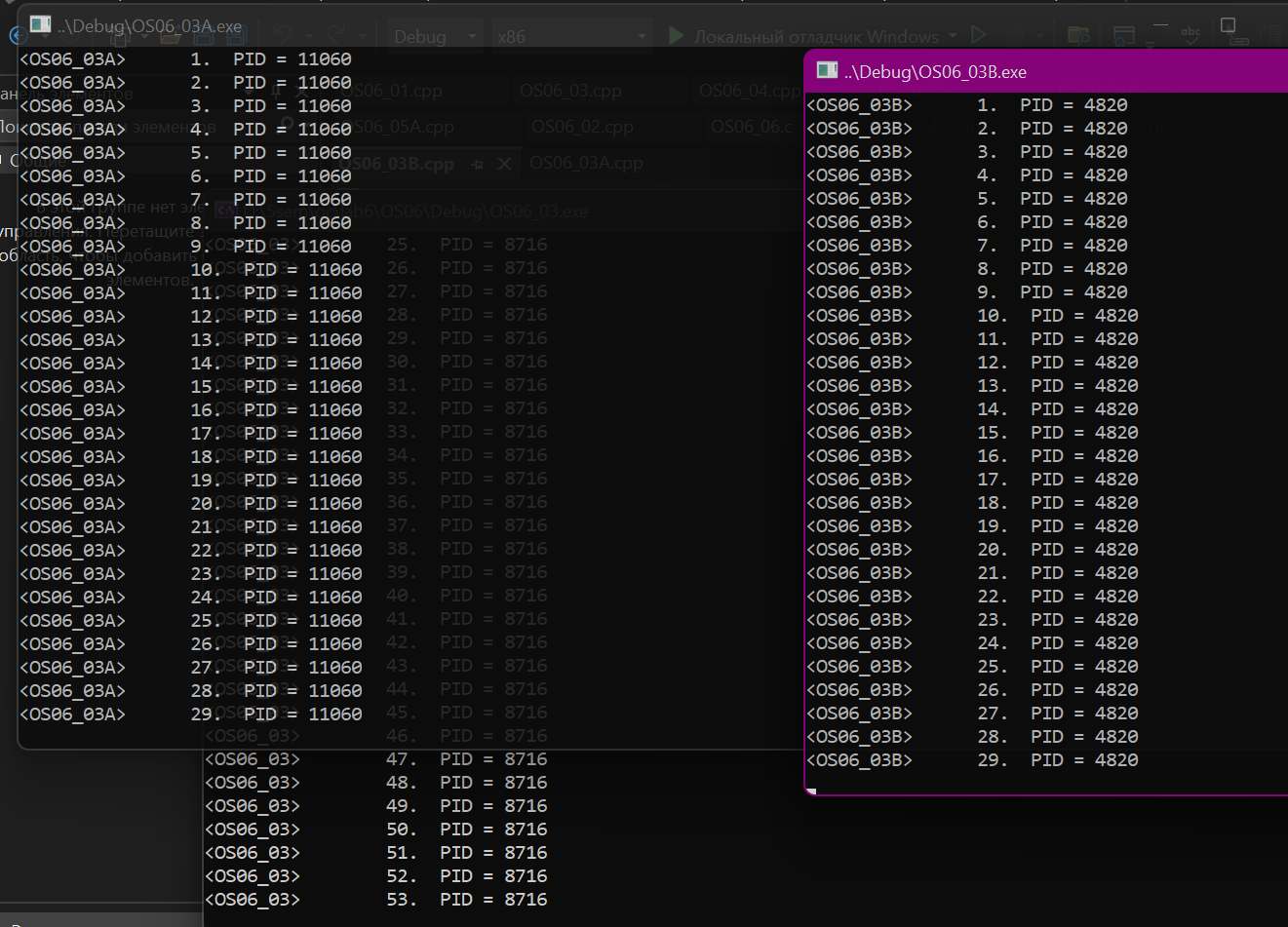




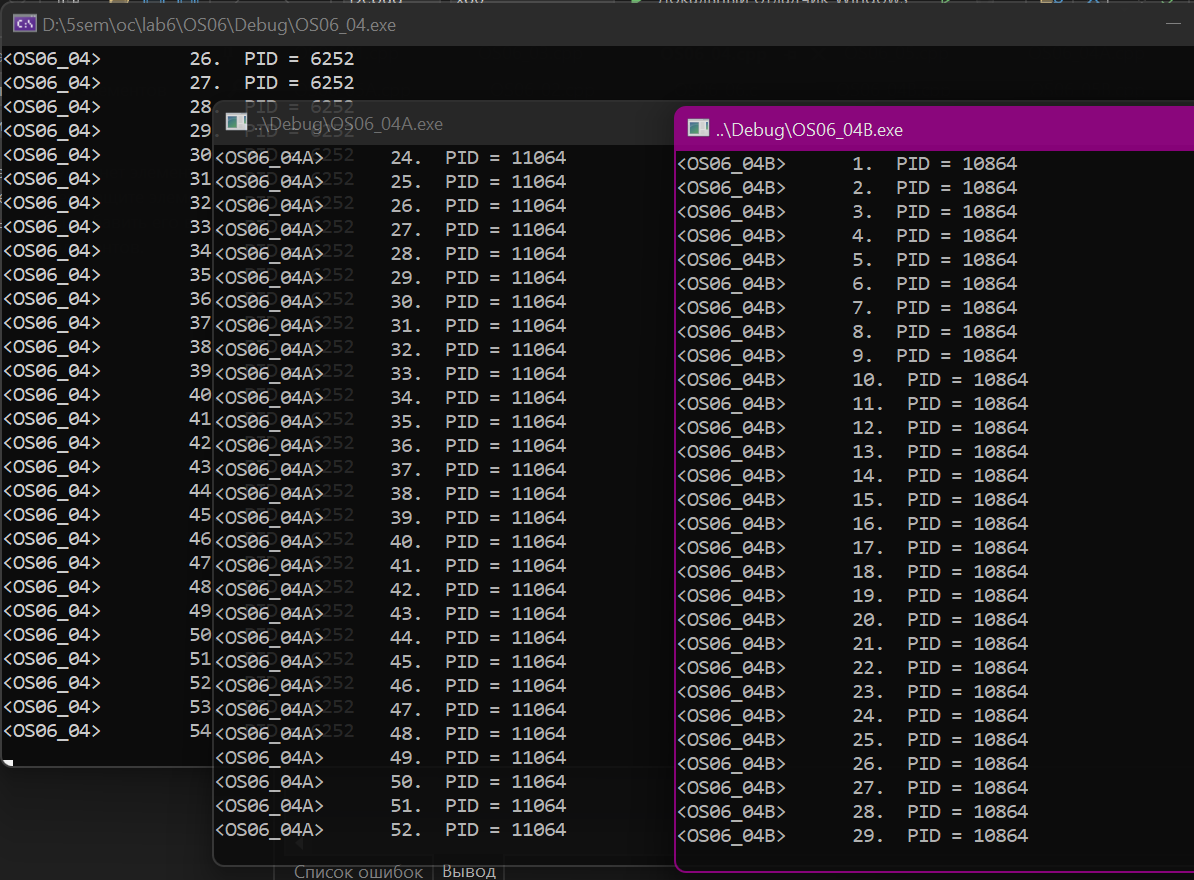
**Задание 03. Windows**

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.



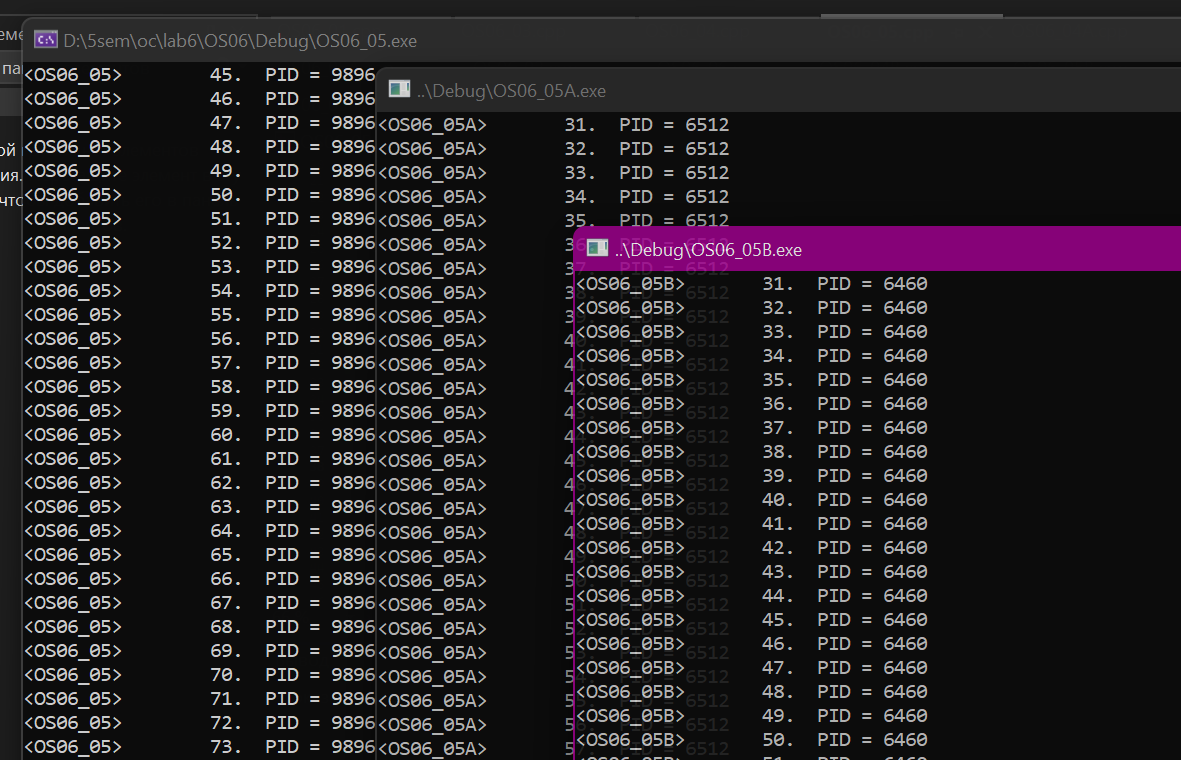


**Задание 04. Windows**

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. 

**Задание 05. Windows**

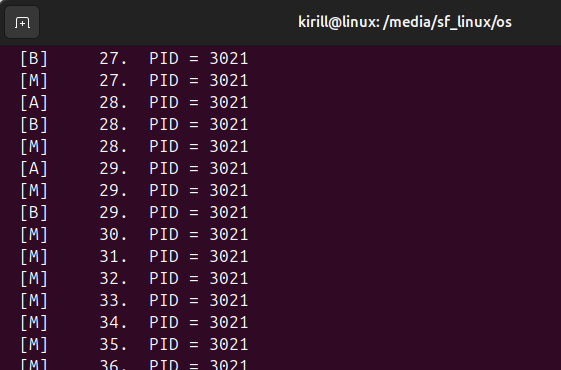
1. Разработайте приложение **OS06\_05**, запускающее два дочерних процесса **OS06\_05A** и **OS06\_05B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_05** синхронизирует выполнение 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.

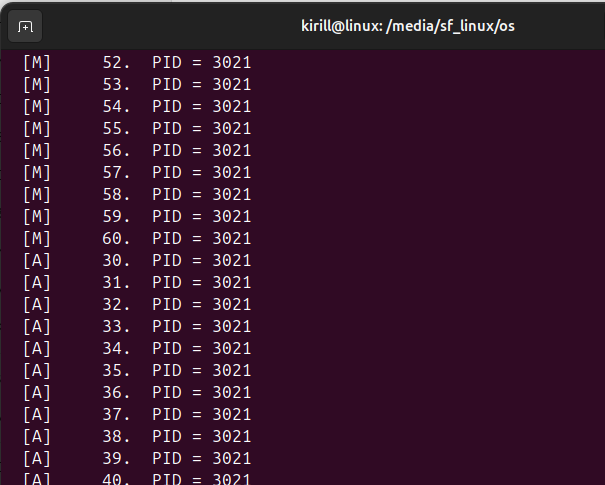


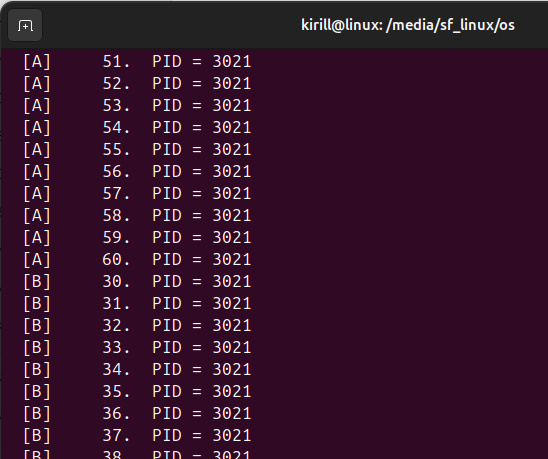
**Задание 06.Linux**

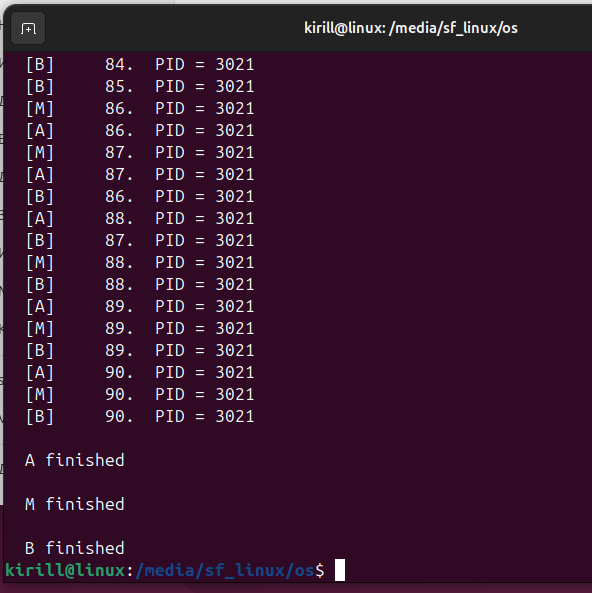
1. Разработайте приложение **OS06\_06**, запускающее два дочерних потока **A** и **B**.
2. Все потоки выполняют циклы в 90 итераций, выводящие имена потоков и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_06** синхронизирует выполнение потоков **main**, **A** и **B** с помощью механизма **mutex.**

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».
2. Объясните понятие «взаимная блокировка».
3. Перечислите механизмы авторизации OS.
4. Поясните в чем разница между механизмом **mutex** и **semaphore**.
5. Почему **mutex,** **semaphore, event** создают объект ядра OS, а **critical section** нет.
6. Дайте определение понятию «синхронизация потоков».

Понятие "синхронизация потоков" относится к координации выполнения нескольких потоков в многопоточной программе. Оно описывает методы и механизмы, используемые для упорядочивания доступа к общим ресурсам или обмена информацией между потоками. Синхронизация потоков позволяет избежать гонок данных (race conditions) и других проблем, возникающих при параллельном выполнении кода.

1. Объясните понятие «взаимная блокировка».

"Взаимная блокировка" (deadlock) возникает, когда два или более потока находятся в состоянии ожидания ресурсов, которые контролируют другие потоки. Каждый из этих потоков блокирует ресурс, который требуется другому потоку для продолжения выполнения, при этом сам ожидает ресурс, контролируемый другим потоком. В результате ни один из потоков не может продолжить свое выполнение, поскольку он ожидает освобождения ресурса, которое не произойдет. Взаимная блокировка является нежелательным состоянием и может привести к замедлению или даже зависанию программы.

1. Перечислите механизмы синхронизации OS.

Мьютексы: Используются для обеспечения взаимного исключения доступа к ресурсам

Семафоры: Позволяют ограничивать количество потоков, имеющих доступ к общему ресурсу

События: Используются для сигнализации между потоками о наступлении определенного события

Критические секции: Легковесный механизм синхронизации, используемый для защиты кода, который не должен выполняться одновременно несколькими потоками, но доступен только внутри одного процесса.

Барьер (barrier): Позволяет потокам ждать друг друга, пока все не достигнут определенного состояния.

<li><strong>Условные переменные (condition variable)</strong>: Используются для блокировки потоков, пока не произойдет определенное условие.

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

Разница между механизмом mutex и semaphore заключается в их функциональности и способе использования:

- Mutex (мьютекс) применяется для обеспечения взаимного исключения, то есть только один поток может захватить мьютекс и получить доступ к защищаемому ресурсу. Остальные потоки будут ожидать освобождения мьютекса. Mutex может быть разблокирован только тем потоком, который его захватил.

- Semaphore (семафор) используется для контроля доступа к определенному количеству ресурсов. Семафор содержит счетчик, который отслеживает количество доступных ресурсов. Потоки могут захватывать и освобождать семафоры. Если счетчик семафора равен нулю, поток будет ожидать освобождения ресурса.

Основная разница между mutex и semaphore заключается в их использовании. Mutex применяется для обеспечения взаимного исключения в отношении конкретного ресурса, в то время как semaphore используется для контроля доступа к определенному количеству ресурсов.

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

Mutex, semaphore и event создают объекты ядра операционной системы (OS) потому, что они предоставляют механизмы синхронизации, которые могут использоваться между разными процессами или потоками в рамках операционной системы. Эти объекты ядра предоставляют надежные и масштабируемые механизмы синхронизации, поддерживаемые операционной системой.

Mutex, semaphore и event могут быть использованы для синхронизации как внутри одного процесса, так и между разными процессами. Они могут быть именованными (named) или безымянными (unnamed). Именованные объекты могут быть открыты и использованы разными процессами, даже если они не являются родственными или не разделяют общую память. Безымянные объекты синхронизации обычно используются для синхронизации потоков внутри одного процесса.

С другой стороны, критическая секция (critical section) не создает объект ядра операционной системы. Критическая секция реализуется на уровне пользовательского кода и не требует взаимодействия с ядром операционной системы. Критическая секция представляет собой легковесный механизм синхронизации, который может использоваться только внутри одного процесса для синхронизации доступа к общим данным между потоками в этом процессе.

Поскольку критическая секция не требует взаимодействия с ядром операционной системы, она обычно работает быстрее и имеет меньшую накладную надежность по сравнению с объектами ядра, такими как mutex, semaphore и event. Однако, критическая секция имеет ограничения, например, она не может использоваться для синхронизации доступа между разными процессами.