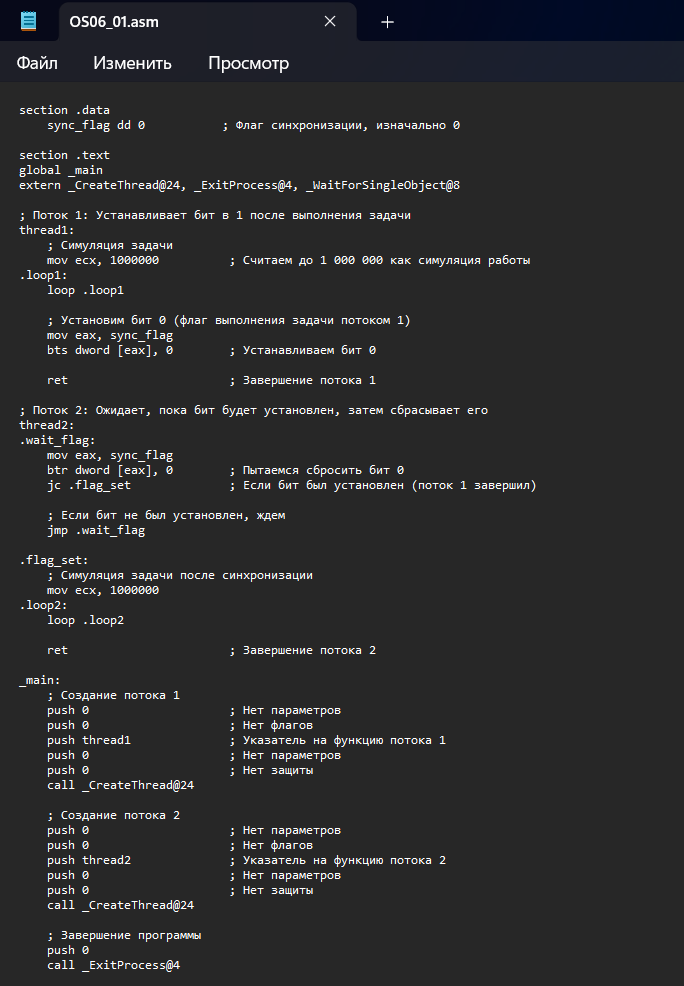
Лабораторная работа 06

Синхронизация

OC, ПОИТ-3

**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.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Команда BTS позволяет получить значение определенного бита первого операнда, а затем устанавливает значение этого бита равным единице. Номер проверяемого бита задается вторым операндом. Перед установкой бита, значение этого бита сохраняется в флаг CF (флаг переноса).

Команда BTR позволяет получить значение определенного бита первого операнда, а затем сбрасывает этот бит в нулевое значение. Номер проверяемого бита задается вторым операндом.

JC выполняет переход к метке, если флаг переноса установлен.

**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.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

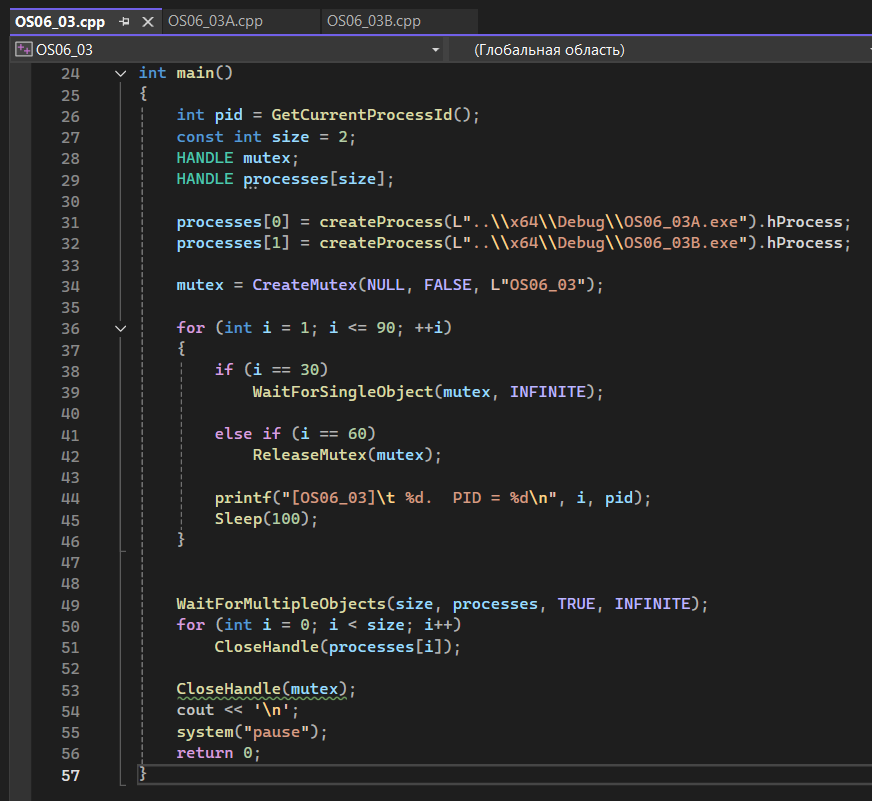
Автоматически созданное описание Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

**Задание 03. Windows**

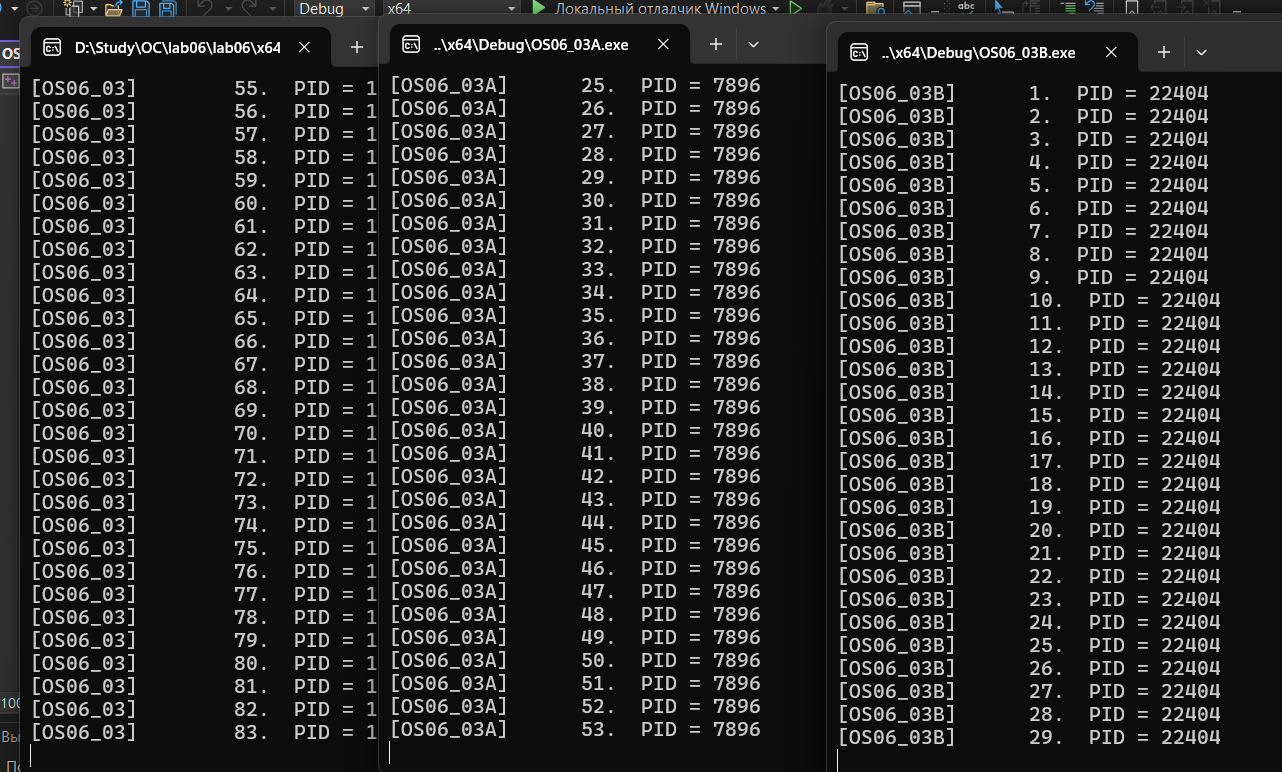
1. Разработайте приложение **OS06\_03**, запускающее два дочерних процесса **OS06\_03A** и **OS06\_03B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_03** синхронизирует выполнение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



Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

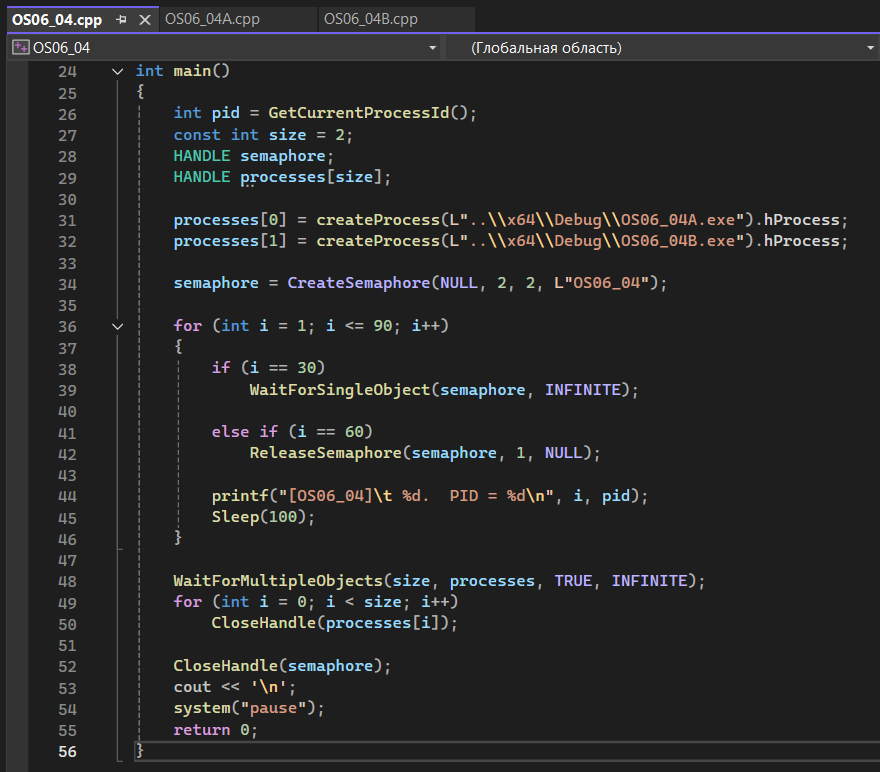
Автоматически созданное описание Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание



**Задание 04. Windows**

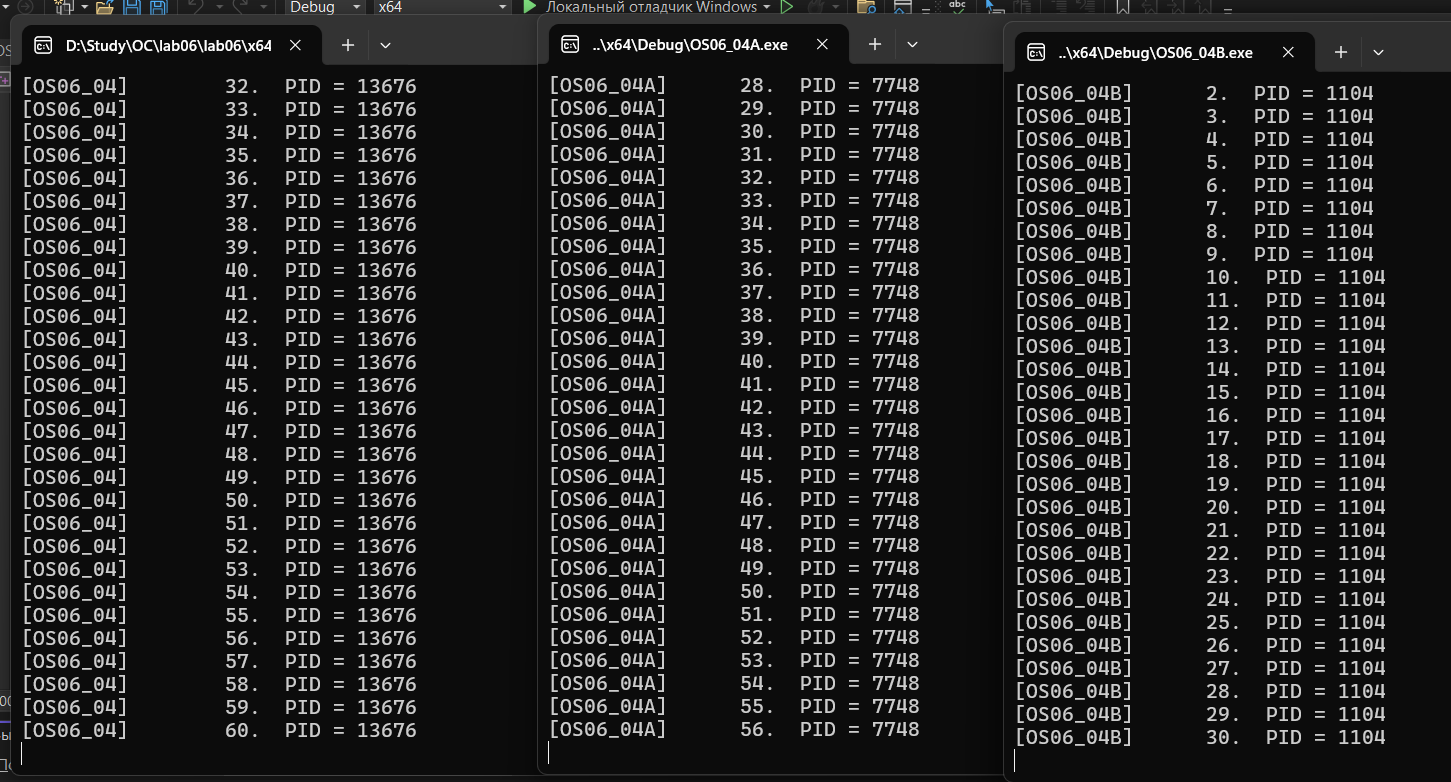
1. Разработайте приложение **OS06\_04**, запускающее два дочерних процесса **OS06\_04A** и **OS06\_04B** свыводом в отдельные консоли.
2. Все процессы выполняют циклы в 90 итераций, выводящие имена процессов и номера итерации с задержкой в 0.1 сек.
3. Приложение **OS06\_04** синхронизирует выполнение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



Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание



**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.

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

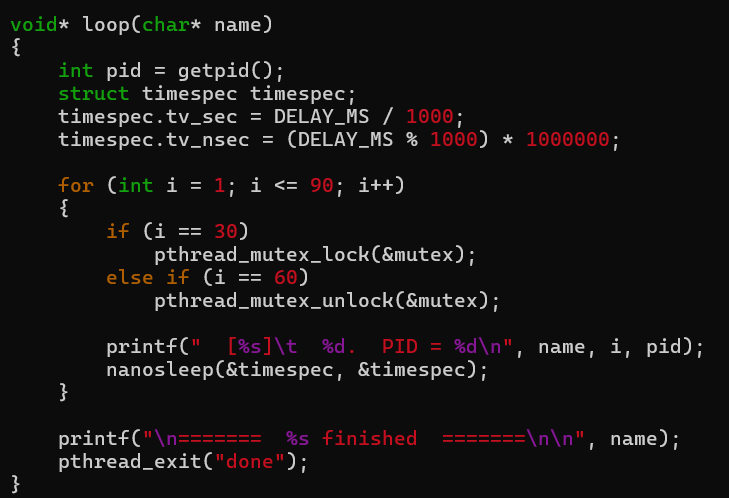
**Задание 06.Linux**

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.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

 Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Задание 07.** Ответьте на следующие вопросы

1. Дайте определение понятию «синхронизация потоков».

Синхронизация потоков — это процесс координации выполнения нескольких потоков, чтобы предотвратить одновременный доступ к общим ресурсам, избежать состояния гонки и обеспечить правильность выполнения программы. В многопоточном программировании синхронизация гарантирует, что потоки взаимодействуют последовательно и корректно.

1. Объясните понятие «взаимная блокировка».

Взаимная блокировка (или "deadlock") — это состояние, при котором два или более потоков (или процессов) ожидают освобождения ресурсов, занятых друг другом, и не могут продолжить работу. Взаимная блокировка происходит, когда каждый поток держит один ресурс и ожидает освобождения другого, что приводит к бесконечному ожиданию.

1. Перечислите механизмы авторизации OS.

Основные механизмы авторизации в операционных системах включают:

* **Списки контроля доступа (ACL)** — определяют права доступа для каждого пользователя к объектам системы.
* **Ролевая модель (RBAC)** — права доступа назначаются ролям, а роли присваиваются пользователям.
* **Мандатное управление доступом (MAC)** — права назначаются исходя из уровня конфиденциальности и потребностей безопасности.
* **Идентификация и аутентификация пользователей** — проверка подлинности пользователя перед тем, как дать ему доступ к ресурсам.
* **Модели доверенных доменов** — для управления доступом между разными системами или доменами.

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

* **Mutex (взаимоисключение)** — это объект синхронизации, который используется для блокировки доступа к ресурсу одним потоком на время его использования. Он позволяет только одному потоку владеть блокировкой в данный момент времени.
* **Семафор** — это механизм, который позволяет управлять доступом к ресурсу для нескольких потоков. Семафор имеет счётчик, который контролирует количество потоков, которые могут одновременно использовать ресурс. Если счётчик равен нулю, то доступ блокируется, пока один из потоков не освободит ресурс.

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

* **Мьютексы, семафоры и события** создают объекты ядра ОС, так как они позволяют синхронизировать потоки не только внутри одного процесса, но и между разными процессами. Эти объекты управляются ядром, обеспечивая безопасный межпроцессорный доступ к ресурсам.
* **Критическая секция** не создает объект ядра ОС, так как используется для синхронизации потоков только внутри одного процесса. Она работает быстрее, так как не требует перехода в режим ядра и позволяет управлять доступом к ресурсам в пределах одного процесса.

1. Основные механизмы синхронизации:

* **Critical section (Критическая секция)** — участок кода, к которому разрешается доступ только одному потоку в пределах одного процесса. Быстрая и эффективная для работы внутри процесса, так как не требует обращения к ядру ОС.
* **Mutex (Мьютекс)** — объект, который предоставляет взаимное исключение, позволяя только одному потоку в системе захватить его в определенный момент времени. Используется как внутри одного процесса, так и между процессами. Требует обращения к ядру ОС.
* **Semaphore (Семафор)** — объект синхронизации со счетчиком, который контролирует доступ к ресурсу для нескольких потоков. Подходит для ситуаций, когда ресурс может использоваться одновременно несколькими потоками (ограниченное количество раз). Может быть использован как внутри процесса, так и между процессами.
* **Event (Событие)** — объект, который используется для уведомления потоков о наступлении определенного события. Один поток может "сигнализировать" событие, а другие, ожидающие его, продолжат выполнение. Применяется для синхронизации между потоками и процессами, когда важно оповещать о выполнении задачи или достижении состояния.