Лабораторная работа 07

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.

**Задание 02. Windows**

1. Разработайте приложение **OS07\_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 **OS07\_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 **OS07\_03**, запускающее два дочерних процесса **OS07\_03A** и **OS07\_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 **OS07\_03** синхронизирует выполнение процессов **OS07\_03**, **OS07\_03A** и **OS07\_03B** с помощью механизма **mutex.**
4. Синхронизация должна обеспечивать поочередное выполнение итераций цикла с 30 по 60.

**Задание 04. Windows**

1. Разработайте приложение **OS07\_04**, запускающее два дочерних процесса **OS07\_04A** и **OS07\_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 **OS07\_04** синхронизирует выполнение процессов **OS07\_04**, **OS07\_04A** и **OS07\_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 **OS07\_05**, запускающее два дочерних процесса **OS07\_05A** и **OS07\_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 **OS07\_05** синхронизирует выполнение процессов **OS07\_05**, **OS07\_05A** и **OS07\_05B** с помощью механизма **event**.
4. Синхронизация должна обеспечивать выполнение приложения в два этапа:
5. выполнение итераций с 1 по 15 процесса **OS07\_05**;
6. одновременное выполнение всех трех процессов: **OS07\_05** – продолжает выполнение итераций; процессы **OS07\_05A** и **OS07\_05B** выполняются начиная с первой итерации.

**Задание 06.Linux**

1. Разработайте приложение **OS07\_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 **OS07\_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.

**Задание 07.** Ответьте на следующие вопросы

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

Синхронизация потоков - это процесс координирования выполнения множества потоков в многозадачной среде, чтобы избежать конфликтов и обеспечить правильное взаимодействие между ними.

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

Взаимная блокировка - это ситуация, когда два или более потока или процесса блокируют друг друга, ожидая ресурсы или события, и не могут продолжить выполнение. Это приводит к зависанию программы.

1. **Перечислите механизмы синхронизации OS.**

Mutex (блокировки): Обеспечивают эксклюзивный доступ к ресурсу одним потоком или процессом за раз.

Semaphore (семафоры): Используются для управления доступом нескольких потоков к ресурсу, ограничивая их количество.

Event (события): Позволяют потокам ожидать определенных событий или сигналов, которые могут произойти в других потоках или процессах.

Critical Section: Это программная конструкция, предназначенная для синхронизации доступа к общим данным внутри одного процесса или потока.

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

Mutex беспечивает эксклюзивный доступ к ресурсу одним потоком или процессом за раз.

Semaphore используется для управления доступом нескольких потоков к ресурсу, ограничивая их количество.

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 представляют собой более общие и универсальные механизмы синхронизации, которые могут использоваться между разными процессами и даже в разных приложениях. Эти объекты имеют глобальную видимость и могут использоваться для синхронизации между разными процессами, даже если они работают параллельно и в разных частях операционной системы.

Однако, когда речь идет о "critical section" (критической секции), это - концепция, связанная непосредственно с программным кодом внутри одного процесса или потока. Она предполагает, что определенный участок кода внутри процесса должен быть выполнен только одним потоком в конкретный момент времени. В этом случае, синхронизация происходит на уровне самого приложения, и нет необходимости создавать объект ядра операционной системы. Критические секции используются для предотвращения конфликтов доступа к общим данным внутри процесса.