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».
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.

22. Синхронизация потоков (или процессов) - это процесс управления одновременным выполнением нескольких потоков (или процессов) с целью обеспечения правильного и безопасного выполнения их задач. Синхронизация потоков включает в себя координацию доступа к разделяемым ресурсам, предотвращение гонок данных и управление порядком выполнения потоков.

23. Взаимная блокировка (deadlock) - это ситуация, при которой два или более потока (или процесса) оказываются заблокированными, ожидая освобождения ресурсов, которые удерживают другие потоки в этом же наборе. В результате ни один из потоков не может продолжить выполнение, и программа зависает.

24. Механизмы авторизации (или аутентификации) операционных систем включают в себя:

a. Пароль (что-то, что знает пользователь).

b. Биометрические данные (например, скан отпечатка пальца).

c. Смарт-карты или USB-токены (что-то, что пользователь имеет).

d. Двухфакторная аутентификация (комбинация чего-то, что пользователь знает и чего-то, что пользователь имеет).

e. Публичные и приватные ключи (используется в криптографических системах).

25. Разница между механизмом mutex и semaphore заключается в их применении:

- Mutex (сокращение от "mutual exclusion") - это механизм синхронизации, который предназначен для обеспечения монопольного доступа к разделяемым ресурсам. Только один поток (или процесс) может владеть mutex в определенный момент времени, что предотвращает гонки данных и конфликты доступа к ресурсам.

- Semaphore - это механизм синхронизации, который предоставляет счетчик, который может быть использован для ограничения количества потоков (или процессов), имеющих доступ к разделяемым ресурсам. Семафор может иметь начальное значение и уменьшаться и увеличиваться потоками. Это позволяет контролировать параллельное выполнение определенного количества задач.

26. Mutex, semaphore и event создают объекты ядра операционной системы (например, в Windows это могут быть объекты средства синхронизации), потому что они предоставляют более сложные и мощные механизмы синхронизации, которые могут применяться в разных частях программы и между процессами. Они предоставляют более широкий спектр функциональности и могут использоваться для различных целей.

Critical section (критическая секция) - это более простой механизм синхронизации, который используется для ограничения доступа к разделяемым ресурсам внутри одного процесса или потока. Critical section обычно реализуется с помощью библиотечных функций и не требует создания объектов ядра операционной системы. Critical section обеспечивает синхронизацию только внутри одного процесса и не предоставляет такую гибкость и мощность, как mutex, semaphore и event.