**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра «Информационные системы»**

Реферат №3

**по дисциплине «Системы реального времени»**

Тема: **Средства синхронизации**

|  |  |  |
| --- | --- | --- |
| Студентгр. 9373 |  | Блохина А.А. |
| Преподаватель |  | Сидельников В.В. |

Санкт-Петербург

2023

**содержание**

[1. Понятие критической секции, проблемы 3](#_Toc129602177)

[2. Семафоры и мьютексы, отличительные особенности 4](#_Toc129602178)

[3. Монитор и условная переменная 5](#_Toc129602179)

[4. Сигналы 6](#_Toc129602180)

[5. Механизм сообщений 6](#_Toc129602181)

[6. Примеры классических задач синхронизации 9](#_Toc129602182)

[**Список использованной литературы:** 10](#_Toc129602183)

# Понятие критической секции, проблемы

Критическая секция - это часть программного кода, которая обращается к общим ресурсам (например, разделяемой памяти, файлам и т.д.) и может быть выполнена только одним потоком в определенный момент времени. Другими словами, критическая секция - это участок кода, который необходимо защитить от параллельного доступа нескольких потоков, чтобы предотвратить состояние гонки или другие ошибки, которые могут привести к непредсказуемому поведению программы.

Проблемы, связанные с критическими секциями, включают в себя:

* Состояние гонки: если несколько потоков пытаются одновременно получить доступ к общим ресурсам внутри критической секции, то может возникнуть состояние гонки, когда результат выполнения программы становится непредсказуемым.
* Бесконечное ожидание: если один поток заблокировал критическую секцию и не освободил ее, то другие потоки, которые пытаются получить доступ к этой секции, могут застрять в бесконечном ожидании, что приводит к замедлению работы программы.
* Некорректное использование блокировок: если блокировки используются некорректно, то это может привести к дедлокам (когда два или более потока блокируют друг друга), отказу обслуживания и другим ошибкам.

Для предотвращения проблем, связанных с критическими секциями, необходимо использовать соответствующие механизмы синхронизации, такие как мьютексы, семафоры или условные переменные, которые позволяют управлять доступом к критическим секциям и избежать состояний гонки и других ошибок.

# Семафоры и мьютексы, отличительные особенности

Семафоры и мьютексы - это два распространенных механизма синхронизации, которые используются для управления доступом к критическим секциям в многопоточных программах. Однако они имеют некоторые отличительные особенности.

Функциональность: мьютексы позволяют только одному потоку входить в критическую секцию, в то время как семафоры могут разрешать доступ нескольким потокам, определяя максимальное количество потоков, которые могут одновременно получить доступ к общим ресурсам.

Значение: мьютексы имеют двоичное значение (заблокировано/разблокировано), тогда как семафоры имеют целочисленное значение, которое может изменяться в зависимости от того, сколько потоков запрашивает доступ к ресурсам.

Ожидание: потоки, которые запрашивают доступ к заблокированному мьютексу, блокируются до тех пор, пока мьютекс не будет разблокирован, в то время как потоки, которые запрашивают доступ к семафору, могут либо блокироваться, либо продолжать работу с уведомлением о том, что доступ к ресурсам ограничен.

Скорость: мьютексы быстрее семафоров, поскольку они требуют меньшего количества операций для блокировки и разблокировки.

Использование: мьютексы используются для управления доступом к критическим секциям в пределах одного процесса, тогда как семафоры могут использоваться для синхронизации процессов или потоков в разных процессах.

В целом, выбор между мьютексами и семафорами зависит от конкретных потребностей и особенностей приложения. Если нужен механизм синхронизации, который позволяет только одному потоку одновременно получать доступ к критической секции, то лучше использовать мьютексы. Если требуется управлять доступом нескольких потоков к общим ресурсам, то лучше выбрать семафоры.

# Монитор и условная переменная

Монитор - это абстрактный объект, который обеспечивает безопасное использование общих ресурсов в многопоточном программировании. Монитор может содержать переменные и процедуры, которые могут быть вызваны из потоков для доступа к общим ресурсам.

Условная переменная - это инструмент, который позволяет потокам ожидать определенных условий перед продолжением выполнения. Она связана с монитором и используется для реализации механизма ожидания/уведомления в многопоточных программах.

Рассмотрим более подробно, как работают мониторы и условные переменные:

Монитор: Монитор предоставляет защиту для общих ресурсов, которые должны использоваться потоками в многопоточном приложении. Он обычно содержит критическую секцию, которая позволяет только одному потоку входить в монитор одновременно. Когда поток покидает монитор, другой поток может войти в него и получить доступ к общим ресурсам. Мониторы также могут содержать процедуры, которые могут быть вызваны из потоков для доступа к общим ресурсам.

Условная переменная: Условная переменная - это инструмент, который используется в многопоточных программах для ожидания определенных условий перед продолжением выполнения. Вместо того, чтобы бесполезно выполнять цикл проверки условия в теле потока, он может быть заблокирован на условной переменной. Когда условие, которое ожидают потоки, выполняется, монитор может использовать условную переменную для уведомления заблокированных потоков об этом. При получении уведомления потоки продолжают выполнение.

# Сигналы

Сигналы в операционной системе используются для обработки событий, которые могут произойти в процессе выполнения задачи. Источником сигналов могут быть как сам процесс, так и другие процессы. Сигналы могут быть синхронными, то есть вызванными в результате работы процесса, или асинхронными, когда они поступают процессу от внешнего источника, например, от пользователя через терминал.

Синхронные сигналы могут возникнуть, например, при ошибке аппаратуры, такой как деление на нуль или нарушение защиты памяти. Асинхронные сигналы могут быть вызваны пользователем, например, когда он нажимает комбинацию клавиш для завершения процесса.

Программный код процесса может реагировать на поступивший сигнал стандартным действием, таким как завершение работы, либо выполнить специфические действия, определенные прикладным программистом. В распределенных системах синхронизация между процессами может быть реализована только посредством обмена сообщениями, так как блокирующие переменные, семафоры и сигналы, основанные на разделяемой памяти, неэффективны в таких системах.

# Механизм сообщений

Для того чтобы взаимодействующие процессы могли обмениваться информацией, операционная система должна обеспечивать соответствующие средства. Один из простых методов - использование общих областей памяти, к которым имеют доступ разные процессы для чтения/записи. Доступ к такой области должен быть защищен, например, с помощью семафоров. Главное преимущество общих областей памяти - возможность организовать прямой и мгновенный доступ. В языках высокого уровня используются глобальные переменные для общих областей памяти, доступные нескольким процессам. В случае возможных конфликтов доступа к общим областям, они должны быть защищены семафорами.

Другой метод, позволяющий одновременно осуществлять обмен данными и синхронизацию процессов, - это почтовые ящики. Почтовые ящики - это структуры данных для обмена сообщениями и синхронизации процессов. Они могут быть реализованы в виде логических файлов и поддерживают операции создания, открытия, записи/чтения сообщения, закрытия и удаления. Почтовые ящики хранятся в оперативной памяти или на диске и не имеют имен, только логические идентификаторы. Они могут быть созданы с помощью буфера или связанного списка. Операции записи и чтения сообщений осуществляются с помощью специальных операторов. Почтовые ящики являются классическими очередями FIFO, и операция чтения из пустого ящика может блокироваться до получения сообщения.

Каналы - средство обмена данными между процессами, разработанные для перенаправления ввода и вывода в UNIX. Они работают как файлы, которые могут быть перенаправлены в другой процесс с помощью символа "|". Каналы используются для связи между процессами в UNIX, OS/9 и Windows NT. Они подобны почтовым ящикам, но отличаются в организации потока данных - каналы работают с неструктурированными потоками символов, а почтовые ящики работают с сообщениями, где известны формат и длина. Операции над каналами включают чтение, запись, открытие, закрытие, удаление, а также установку флагов режима доступа и определение размера буфера. Каналы являются естественным средством взаимодействия между процессами в системах "клиент-сервер". В UNIX каналы могут зависеть от протокола TCP/IP, а в Windows NT они работают с любым транспортным протоколом. Однако, внешне простой механизм каналов может требовать больших накладных расходов при реализации, особенно в сетевых системах.

Удаленный вызов процедур (RPC) является основной процедурой обмена данными и синхронизации в среде "клиент-сервер". RPC позволяет вызывать удаленные процедуры так же, как локальные, при этом операционная система отвечает за маршрутизацию и доставку вызова к узлу, где находится эта подпрограмма. Это упрощает программирование и не зависит от того, где находится вызываемая процедура.

В системе реального времени важно, является ли RPC блокирующим или нет. Блокирующий RPC не возвращает управление вызывающему процессу, пока не закончит свою работу, в то время как неблокирующий RPC возвращает управление вызывающей процедуре по истечении некоторого времени (time out) независимо от того, завершила ли работу вызываемая процедура. В любом случае вызывающая программа получает код, идентифицирующий результат выполнения вызова - код возврата. Неблокирующие RPC имеют важное значение для гарантии живучести системы.

Методы синхронизации и обмена данными, хотя и разные, на самом деле направлены на достижение одной цели - организации параллельного программирования. Так, методы синхронизации могут быть использованы для организации обмена данными между процессами, а техника коммуникаций между процессами может служить для реализации функций синхронизации и взаимного исключения. Например, семафор может быть заменен почтовым ящиком, а операции signal и wait эквивалентны операциям put и get почтового ящика. Сообщения могут быть использованы в качестве "маркеров" для организации взаимного исключения и защиты ресурсов системы.

Если в системе используется только один подход, то связь между различными подходами имеет практическое значение только тогда, когда нужно строить все остальные на основе этого подхода. Современные операционные системы используют все упомянутые методы для поддержки многозадачного режима и операций в реальном времени. Выбор подхода для синхронизации и взаимодействия должен основываться на том, как лучше решить конкретную проблему, и эффективность конкретных решений должна оцениваться в имеющейся программной среде. В распределенных системах может существовать риск потери сообщения в сети, но если система контролирует правильность передачи и предоставляет средства для повторной передачи, прикладная программа не должна осуществлять дополнительные проверки. Однако, если контроль не предусмотрен, например, при использовании службы IP без транспортного протокола TCP, прикладная программа должна проверять результат передачи, что может быть сложным. Одним из возможных решений является помечать и нумеровать каждое сообщение для отслеживания порядка передачи.

# Примеры классических задач синхронизации

Несколько примеров классических задач синхронизации:

Проблема "Читатели-писатели": есть общий ресурс (например, база данных), который может быть одновременно изменен только одним писателем, но может быть прочитан множеством читателей. Задача состоит в том, чтобы синхронизировать доступ к этому ресурсу, чтобы избежать ситуации, когда писатель и одновременно несколько читателей работают с ресурсом одновременно.

Проблема "Обедающие философы": есть некоторое число философов, которые сидят за круглым столом и пытаются поесть. Между каждой парой тарелок лежит вилка, и философы могут есть только с помощью двух вилок. Задача состоит в том, чтобы разработать алгоритм, который позволит каждому философу взять две вилки и поесть, не позволяя возникновению ситуации, когда все вилки заняты.

Проблема "Производитель-потребитель": есть два процесса - производитель и потребитель, которые совместно работают с очередью данных. Производитель помещает данные в очередь, а потребитель их извлекает. Задача состоит в том, чтобы синхронизировать работу производителя и потребителя, чтобы избежать ситуации, когда потребитель пытается извлечь данные из пустой очереди или производитель пытается поместить данные в полную очередь.

Проблема "Файловый сервер": множество пользователей пытаются одновременно работать с файловым сервером. Задача состоит в том, чтобы синхронизировать доступ к файлам, чтобы избежать ситуации, когда два пользователей пытаются одновременно изменить один и тот же файл.

Эти проблемы являются классическими задачами синхронизации, которые возникают во многих операционных системах. Решение этих задач требует использования различных механизмов синхронизации, таких как мьютексы, семафоры, условные переменные и т.д.

**Список использованной литературы:**

1. Сергей Сорокин. Системы Реального Времени. // СТА. – 1997. - №2. – С 22-29.
2. IEEE Standard Glossary of Software Engineering Terminology. / New York : The Institute of Electrical and Electronics Engineers, 1990. – URL: <http://www.informatik.htw-dresden.de/~hauptman/SEI/IEEE_Standard_Glossary_of_Software_Engineering_Terminology%20.pdf>
3. Martin Timmerman, Bart Van Beneden, Lourent Uhres. RTOS Evaluation Kick Off! // Real-Time Magazine. – 1998. – N3 pp 6 – 10.
4. Сидельников В.В. Лекции по дисциплине «системы реального времени». СПб.: Изд-во СПбГЭТУ «ЛЭТИ»
5. Системное программное обеспечение управляющих систем реального времени: учебное пособие / М.В. Кавалеров. – Пермь: Изд-во ПНИПУ, 2013. – 190 с.
6. Гриценко Ю.Б. Системы реального времени: Учебное пособие. — Томск: Томский межвузовский центр дистанционного образования, 2009. — 256 с
7. Операционные системы реального времени: учебное пособие / Егор Горошко. – Харьков: Изд-во ИРЭ НАН Украины, 2003.