БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет НиДО

Специальность ИиТП

Контрольная работа № 2

по дисциплине «Системное программирование»

Выполнил студент: Дегтярев А.А.

группа 393551

Зачетная книжка № 902021-26

Минск 2015

**Взаимодействие процессов и потоков – основные задачи, проблемы, решения.**

Процесс по сути - загруженный в память исполняемый код, связанные данные и ресурсы. В многозадачной среде, коими являются большинство современных ОС, одновременно может выполнятся множество процессов. Многопроцессорные системы сейчас широко распространены, однако принцип квантования времени для процессов все еще актуален. Суть его заключается в том, что процессор выполняет работу для конкретной задачи какой-то определенный промежуток времени. Планировщик задач(scheduler) контролирует переключние задач. Основной тип многозадачности применяемый в современных ОС – вытесняющая многозадачность – инициатива смены задачи за планировщиком. Для ос типа UNIX, Win32 существует 2 основных объекта составляющие задачу: процесс и поток.   
 Процесс – динамичиеский объект ассоциированный с выполняющейся программой, имеет свое изолированное виртуальное адресное пространство, пространство описателей объектов. Каждый процесс как минимум имеет 1 главный поток.  
 Поток – динамиеский объект ассоциированный с последовательностью команд, выполняющийся независимо и асинхронно.   
 Процесс – единица планирования ресурсов, Поток – процессорного времени.

Также существует элемент JobObject имеющий аналог в Unix в виде групп процессов. Позволяет объединять несколько связных задач.

Элемент Fiber – нить – часть потока. Объект исполнения который может быть назначен для исполнения программой а не планировщиком . Нити сами определяют какая следующая нить будет исполнена. Они также имеют свой отдельный стек. В отличии от потоков, система не занет о существовании нитей, весь контроль лежит на плечах разработчика. Типичные схемы для управления нитями: Master/Slave – когда каждая нить передает управление следующей за ней. Peer-to-peer – нить передает управление следующей на основе какого-то алгоритма из пула нитей.

Для ОС является затратным переключение между процессами, потоки позволяют избежать затрат на переключение при этом сохраняя возможность одновременного исполнения задач.

Не смотря на различные среды, процессы могут взаимодействовать друг с другом через «(pipes)» и использование общих ресурсов

Для управления несколькими паралельными взаимодействующими задачами ( ожидание ввода, передачи данных по сети, долгие вычисления) использование различных процессов затратно и сложно, в отличие от использования потоков

Основные проблемы связанные с использование потоков:

- Потоки используют общее хранилище данных, и другие ресурсы внутри одного процесса. Поток может изменить данные используемые другим потоком что может привести к тупиковым ситуациям вроде deadlock и race conditions

- В некоторых случаях параллелизм может сильно замедлить выполнение задачи

- Многие функции стандартной библиотеки C полагаются на глобальные состояния, которые могут быть одновреенно изменены разными потоками, что может вызвать ошибки в работе. Это также следует учитывать в разработке собственных функций.

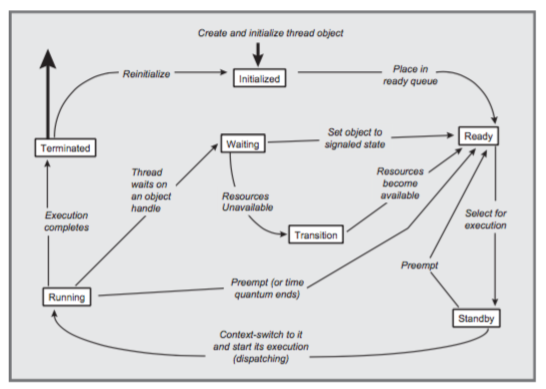
Потоки в дополнине к общим ресурсам процесса имеют свое уникальное хранилище данных. В случае ОС Windows каждый поток получает свой стэк вызова функций. Вызывающий процесс может передать аргумент(чаще всего указатель) во время создания потока, этот аргумент будет находится в стеке потока. Каждый поток может выделить локальное хранилище TLS, которое гарантированно не будет изменено другими потоками.

Существует несколько типичных моделей дизайна многопоточных программ:  
 Boss/Worker модель, когда 1 поток (часто главный поток процесса), назначает задачи worker потоку, который собственно выполняет работу и возвращает результат основному потоку

Work crew модель (разделяй и властвуй) – в которой рабочие потоки кооперируются для решения одной задачи выполняя лишь часть работы. В такой модели рабочие потоки могут сами разделять работу между собой без участия главного потока(boss thread), пример: паралельный Merge-Sort. Преимуществом является то, тчо нет необходимости блокировки потоков, использования сложных схем синхронизации. Производительность легко масштабируется с ростом числа процессоров.   
 Client/Server модель - \*\*\*\*\*\*  
 Pipeline модель – когда задача перемещается от потока к потоку \*\*\*

Использование данных моделий сильно облегчает процесс разработки, т.к они соответствуют решению многих типичных задач на паралельность. Ошибки легче обнаружить при сравнение на соответствие реализации с какой-то из данных моделей.

Для упроавления приоритетами исполнения потоков, каждому потоку можно выставить приоритет исполнения. Не стоит использовать приоритеты для явной синхронизации потоков.

**Основные состояния потоков и преходы между ними:** 

**Потоко-безопасный код**1) Переменные, локальные для потока не должны быть в глобальной области видимости, а находитьсся на стеке или в структуре данных или TLS доступных только этому потоку.  
2)Если функция вызывается из разных потоков, и сохраняет состояние, данные должны быть либо привязаны к потоку, либо записаны в TLS.

3)Потоки не должны изменять окружение процесса, ибо это повлияет на другие потоки. Исключение составляет лишь основной поток.

4)Переменные общие для нескольких потоков должны быть статическими или глобальными и должны быть защищены механизмами интерлока или объектами синхронизации

**Объекты синхронизации потоков**

Поток может ожидать окончание другого процесса или потока с помощью WaitForSingleObject / WaitForMultipleObjects

File Lock специфичны для синхронизации доступа к файлам

**Critical Section -** секция гарантирующая одновременное использование лишь одним потоком. Это обычный LOCK механизм. Critical section объекты не имеют описателей, и не могут быть доступны из других процессов. EnterCriticalSection блокирует поток, если другой в это время выполняет критическую секцию. Critical Section рекурсивна, поэтому поток должен выйти из секции LeaveCriticalSection столько же раз, сколько вошел в нее.  **Mutex** (mutual exclusion) – более сложный именованный объект, хранят описатели и могут быть использованы для синхронизации потоков различных процессов(могут синзронизировать допуск к общей памяти) . Они допускают time-out значения, и получают сигнал о уничтожении потока. Аналогично CS mutex должен быть освобожден столько же раз сколько был захвачен.   
  
При неправильном использовании Mutex и CS могут привести к ошибкам вроде deadlock. Когда 2 потока остаются заблокированными ожидая друг от друга освобождения ресурсов.

С точки зрения производительности CS намного быстрее мютексов. ПОэтому если нет необходимосте в гибком функционале мютекса лучше использовать CS.

HeapLock/HeapUnlcok – позволяют синхронизировать доступ к куче.  **Semaphore –** счетчик, когда значение > 0 – семафор находится в сигналируемом состоянии; счетчик показывает число ожидающих потоков, когда поток освобожается, счетчик уменьшает свое значение. Считается что мютекс это часттный случай семафора с макисмальным значением счетчика в 1, но в отличие от мютекса, у семафора нет владельца. Любой поток может добавлять значение в счетчик.

**Event –** может отправлять сигнал другм потокам чтобы установить какое-то условие, например что сообщение заблокировано. Особенностью является то, что event может сразу снять ожидание с нескольких потоков одновременно. Manual-reset event – оповещает множество потоков, auto-reset event – один.

**Другие методы синхронизации потоков**

**SRW (slim reader/writer)**

**Condition variables**

**Waitable timers**

**I/O completion ports**

**Для обмена данными между процессами**

В win32 api существует набор средств, называющийся IPC Inter process communications.

Memory mapped files – отображаемы в память файлы могут быть использованы одновременно несколькими процессами, на основе этого работают динамические библиотеки DLL

Pipes ( транспортеры) – если породить второй процесс так чтобы он наследовал описатели можно использовать описатели транспортера для передачи данных.

Именованные транспортеры позволя/т создавать глобальное имя для транспортеров вида [\\.\pipe\name](file:///\\.\pipe\name) его использование возможно из других процессов.  
  
Mailslots() аналогичен Named Pipe, но является однонаправленным транспортом с именем [\\.\mailslot\path\slot](file:///\\.\mailslot\path\slot) ящик существует пока существуетт родительский процесс.

Socket – универсальное решение практически всех задач на процессорное взаимподействие. Обеспечивает взаимное обнаружение и индентификацию, синхронизацию передачу данных в наличии и отсутствия соединения.

Clipboard – Любое приложение может поместить данные в буфер обмена и получить из него даннеы