ВикипедиЯ

Многопоточность

Материал из Википедии — свободной энциклопедии

Многопото́чность — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких *потоков*, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

Такие *потоки* называют также *потоками выполнения* (от англ. *thread of execution*); иногда называют «нитями» (буквальный перевод англ. *thread*) или неформально «тредами».

Содержание

Описание

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

К достоинствам многопоточной реализации той или иной системы перед многозадачной можно отнести следующее:

- Упрощение программы в некоторых случаях за счёт использования общего адресного пространства.
- Меньшие относительно процесса временны́е затраты на создание потока.

К достоинствам многопоточной реализации той или иной системы перед однопоточной можно отнести следующее:

- Упрощение программы в некоторых случаях, за счёт вынесения механизмов чередования выполнения различных слабо взаимосвязанных подзадач, требующих одновременного выполнения, в отдельную

подсистему многопоточности.
- Повышение производительности процесса за счёт распараллеливания процессорных вычислений и операций ввода-вывода.

В случае, если потоки выполнения требуют относительно сложного взаимодействия друг с другом, возможно проявление проблем многозадачности, таких как взаимные блокировки.

Аппаратная реализация

На обычном процессоре управление потоками осуществляется операционной системой. Поток исполняется до тех пор, пока не произойдёт аппаратное прерывание, системный вызов или пока не истечёт отведённое для него операционной системой время. После этого процессор переключается на код операционной системы, который сохраняет состояние потока (его контекст) или переключается на состояние другого потока, которому тоже выделяется время на исполнение. При такой многопоточности достаточно большое количество тактов процессора тратится на код операционной системы, переключающий контексты. Если поддержку потоков реализовать аппаратно, то процессор сам сможет переключаться между потоками, а в идеальном случае - выполнять несколько потоков одновременно за каждый такт. Для операционной системы и пользователя один такой физический процессор будет виден как несколько логических процессоров.

Различают две формы многопоточности, которые могут быть реализованы в процессорах аппаратно:

- Временная многопоточность (англ. Temporal multithreading)
- Одновременная многопоточность (англ. Simultaneous multithreading)

Типы реализации потоков

- Поток в пространстве пользователя. Каждый процесс имеет таблицу потоков, аналогичную таблице процессов ядра. К недостаткам можно отнести:

1. Отсутствие прерывания по таймеру внутри одного процесса
2. При использовании блокирующего системного запроса для процесса все его потоки блокируются.
3. Сложность реализации

- Поток в пространстве ядра. Наряду с таблицей процессов в пространстве ядра имеется таблица потоков.
- «Волокна» (англ. *fibers*). Несколько потоков режима пользователя, исполняющихся в одном потоке режима ядра. Поток пространства ядра потребляет заметные ресурсы, в первую очередь физическую память и диапазон адресов режима ядра для стека режима ядра. Поэтому было введено понятие «волокна» — облегчённого потока, выполняемого исключительно в режиме пользователя. У каждого потока может быть несколько «волокон».

Взаимодействие потоков

В многопоточной среде часто возникают задачи, требующие приостановки и возобновления работы одних потоков в зависимости от работы других. В частности это задачи, связанные с предотвращенем конфликтов доступа при использовании одних и тех же данных или устройств из параллельно исполняемых потоков. Для решения таких задач используются специальные объекты для взаимодействия потоков, такие как взаимоисключения (мьютексы), семафоры, критические секции, события и т.п. Многие из этих объектов являются объектами ядра и могут применяться не только между потоками одного процесса, но и для взаимодействия между потоками разных процессов.

- Взаимоисключения (mutex, мьютекс) — это объект синхронизации, который устанавливается в особое сигнальное состояние, когда не занят каким-либо потоком. Только один поток владеет этим объектом в любой момент времени, отсюда и название таких объектов (от английского **mut**ually **ex**clusive access — взаимно исключающий доступ) — одновременный доступ к общему ресурсу исключается. После всех необходимых действий мьютекс освобождается, предоставляя другим потокам доступ к общему ресурсу. Объект может поддерживать рекурсивный захват второй раз тем же потоком, увеличивая счётчик, не блокируя поток, и требуя потом многократного освобождения. Такова, например, критическая секция в

Win32. Тем не менее, есть и такие реализации, которые не поддерживают такое и приводят к взаимной блокировке потока при попытке рекурсивного захвата. Например, это FAST_MUTEX в ядре Windows.

- Критические секции обеспечивают синхронизацию подобно мьютексам, за исключением того, что объекты, представляющие критические секции, доступны лишь в пределах одного процесса. События, мьютексы и семафоры также можно использовать в потоках однопроцессного приложении, однако реализации критических секций в некоторых ОС (например, Windows NT) обеспечивают более быстрый и более эффективный[1][2] механизм взаимно-исключающей синхронизации — операции «получить» и «освободить» на критической секции оптимизированы для случая единственного потока (отсутствия конкуренции) с целью избежать любых ведущих в ядро ОС системных вызовов.
- Семафоры представляют собой доступные ресурсы, которые могут быть приобретены несколькими потоками в одно и то же время, пока пул ресурсов не опустеет. Тогда дополнительные потоки должны ждать, пока требуемое количество ресурсов не будет снова доступно.
- События. Объект, хранящий в себе 1 бит информации «просигнализирован или нет», над которым определены операции «просигнализировать», «сбросить в непросигнализированное состояние» и «ожидать». Ожидание на просигнализированном событии есть отсутствие операции с немедленным продолжением исполнения потока. Ожидание на непросигнализированном событии приводит к приостановке исполнения потока до тех пор, пока другой поток (или же вторая фаза обработчика прерывания в ядре ОС) не просигнализирует событие. Возможно ожидание нескольких событий в режимах «любого» или «всех». Возможно также создание события, автоматически сбрасываемого в непросигнализированное состояние после пробуждения первого же — и единственного — ожидающего потока (такой объект используется как основа для реализации объекта «критическая секция»). Активно используются в MS Windows, как в режиме пользователя, так и в режиме ядра. Аналогичный объект имеется и в ядре Linux под названием kwait_queue.
- Условные переменные (condvars). Сходны с событиями, но не являются объектами, занимающими память — используется только адрес переменной, понятие «содержимое переменной» не существует, в качестве условной переменной может использоваться адрес произвольного объекта. В отличие от событий, установка условной переменной в просигнализированное состояние не влечёт за собой никаких последствий в случае, если на данный момент нет потоков, ожидающих на переменной. Установка события в аналогичном случае влечёт за собой запоминание состояния «просигнализировано» внутри самого события, после чего следующие потоки, желающие ожидать события, продолжают исполнение немедленно без остановки. Для полноценного использования такого объекта необходима также операция «освободить mutex и ожидать условную переменную атомарно». Активно используются в UNIX-подобных ОС. Дискуссии о преимуществах и недостатках событий и условных переменных являются заметной частью дискуссий о преимуществах и недостатках Windows и UNIX.
- Порт завершения ввода-вывода (IO completion port, IOCP). Реализованный в ядре ОС и доступный через системные вызовы объект «очередь» с операциями «поместить структуру в хвост очереди» и «взять следующую структуру с головы очереди» — последний вызов приостанавливает исполнение потока в случае, если очередь пуста, и до тех пор, пока другой поток не осуществит вызов «поместить». Самой важной особенностью IOCP является то, что структуры в него могут помещаться не только явным системным вызовом из режима пользователя, но и неявно внутри ядра ОС как результат завершения асинхронной операции ввода-вывода на одном из дескрипторов файлов. Для достижения такого эффекта необходимо использовать системный вызов «связать дескриптор файла с IOCP». В этом случае помещенная в очередь структура содержит в себе код ошибки операции ввода-вывода, а также, для случая успеха этой операции — число реально введённых или выведенных байт. Реализация порта завершения также ограничивает число потоков, исполняющихся на одном процессоре/ядре после получения структуры из очереди. Объект специфичен для MS Windows, и позволяет обработку входящих запросов соединения и порций данных в серверном программном обеспечении в архитектуре, где число потоков может быть меньше числа клиентов (нет требования создавать отдельный поток с расходами ресурсов на него для каждого нового клиента).
- ERESOURCE. Мьютекс, поддерживающий рекурсивный захват, с семантикой разделяемого или эксклюзивного захвата. Семантика: объект может быть либо свободен, либо захвачен произвольным числом потоков разделяемым образом, либо захвачен всего одним потоком эксклюзивным образом. Любые попытки осуществить захваты, нарушающее это правило, приводят к блокировке потока до тех пор, пока объект не освободится так, чтобы сделать захват разрешённым. Также есть операции вида TryToAcquire — никогда не блокирует поток, либо захватывает, либо (если нужна блокировка) возвращает FALSE, ничего не делая. Используется в ядре Windows, особенно в файловых системах — так, например, любому кем-то открытому дисковому файлу соответствует структура FCB, в которой есть 2 таких объекта для синхронизации доступа к размеру файла. Один из них — paging IO resource — захватывается эксклюзивно только в пути обрезания файла, и гарантирует, что в момент обрезания на файле нет активного ввода-вывода от кэша и от отображения в память.
- Rundown protection. Полудокументированный (вызовы присутствуют в файлах-заголовках, но отсутствуют в документации) объект в ядре Windows. Счетчик с операциями «увеличить», «уменьшить» и «ждать». Ожидание блокирует поток до тех пор, пока операции уменьшения не уменьшат счётчик до нуля. Кроме того, операция увеличения может отказать, и наличие активного в данный момент времени ожидания заставляет отказывать все операции увеличения.

Критика терминологии

Перевод английского термина thread как «поток» в контексте, связанном с программированием, противоречит его же переводу «нить» в общеязыковом контексте, а также создаёт коллизии с термином Data stream.

Однако термин «поток» связан с переводами иностранной технической литературы, выполненными в 1970-х годах издательством «Мир». В настоящее время в «академических кругах» (то есть в учебниках, методических пособиях, курсах вузов, диссертациях и пр.) он считается эталонным. Термины же «нить», «тред» и т. п. считаются техническими жаргонизмами.

См. также

- Hyper-threading

Литература

- *Kunle Olukotun.* Chip Multiprocessor Architecture - Techniques to Improve Throughput and Latency. — Morgan and Claypool Publishers, 2007. — 154 p. — ISBN 159829122X. (англ.)
- *Mario Nemirovsky, Dean M. Tullsen.* Multithreading Architecture. — Morgan and Claypool Publishers, 2013. — 1608458555 p. — ISBN 1608458555. (англ.)

Примечания

1. Jeffrey Richter. "Джеффри Рихтер. Windows для профессионалов. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows. 2001 год
2. MSDN http://msdn.microsoft.com/en-us/library/ms682530%28VS.85%29.aspx

Ссылки

- Введение в технологии параллельного программирования (http://software.intel.com/ru-ru/articles/writing-parallel-programs-a-multi-language-tutorial-introduction/) (рус.)
- https://web.archive.org/web/20050505155716/http://hardclub.donntu.edu.ua/projects/qt/qq/qq11-mutex.html#understandingmutexesandsemaphores
- Введение в многопоточность (http://www.codenet.ru/progr/cpp/threads.php)
