Процессы и треды

Понятие процесса было введено для реализации идей мультипрограммирования. Необходимо было отличать термины «**мультизадачность**» и «**мультипрограммирование**».

Мультипрограммирование *- способ организации выполнения нескольких программ на одном компьютере.*

Мультизадачность *- свойство операционной системы или среды программирования обеспечивать возможность параллельной/псевдопараллельной) обработки нескольких процессов.*

Для реализации «мультизадачности» в ее исходном толковании, помимо процесса для мультипрограммирования, необходимо было тоже ввести соответствующую сущность, которая стала называться «**легковесными процессами**». Позднее оно трансформировалось в название «**потоки**» или «**нити**» (**threads, треды**).

Когда говорят о процессах (process), то тем самым отмечают, что ОС поддерживает их обособленность:

1) У каждого процесса имеется свое виртуальное адресное пространство

2) Каждому процессу назначаются свои ресурсы — файлы, окна, семафоры и т. д.

Все это необходимо для того, чтобы защитить один процесс от другого, поскольку они конкурируют друг с другом за ресурсы.

Процессы и треды

В общем случае **процессы** просто **никак не связаны между собой** и могут принадлежать даже разным пользователям, разделяющим одну вычислительную систему. Как итог: **ОС считает их совершенно несвязанными и независимыми**. При этом именно ОС берет на себя роль арбитра в конкуренции между процессами по поводу ресурсов.

В самих процессах возможно задействовать **внутренний параллелизм**. Цель: повышение производительности вычислительной системы.

Например, некоторые операции, выполняемые приложением, могут требовать для своего исполнения достаточно длительного использования центрального процессора. В этом случае при интерактивной работе с приложением пользователь вынужден долго ожидать завершения заказанной операции и не может управлять приложением до тех пор, пока операция не выполнится до самого конца. Такие ситуации встречаются достаточно часто, например, при обработке больших изображений в графических редакторах. Если же программные модули, исполняющие такие длительные операции, оформлять в виде **самостоятельных «подпроцессов»** (**легковесных или облегченных процессов — потоков, «задач»),** которые будут выполняться параллельно с другими «подпроцессами» (потоками, задачами), то у пользователя появляется возможность параллельно выполнять несколько операций в рамках одного приложения (процесса).

Процессы и треды

Легковесными задачи называют потому, что **операционная система не должна для них организовывать полноценную виртуальную машину**.

Потоки (треды, задачи) не имеют своих собственных ресурсов, они развиваются в том же виртуальном адресном пространстве, могут пользоваться теми же файлами, виртуальными устройствами и иными ресурсами, что и данный процесс. Единственное, что им необходимо иметь так это только процессорный ресурс (время).

В однопроцессорной системе треды (задачи) разделяют между собой процессорное время так же, как это делают обычные процессы.

В мультипроцессорной системе могут выполняться одновременно, если не встречают конкуренции из-за обращения к иным ресурсам.

Главное, что обеспечивает многопоточность, — это возможность параллельно выполнять несколько видов операций в одной прикладной программе. Параллельные вычисления (*а следовательно, и более эффективное использование ресур¬сов центрального процессора, и меньшее суммарное время выполнения задач*) уже часто реализуются на уровне тредов, и программа, оформленная в виде нескольких тредов в рамках одного процесса, может быть выполнена быстрее за счет параллельного выполнения ее отдельных частей.

***Например**, если электронная таблица или текстовый процессор были разработаны с учетом возможностей многопоточной обработки, то пользователь может запросить пересчет своего рабочего листа или слияние нескольких документов и одновременно продолжать заполнять таблицу или открывать для редактирования следующий документ.*

Процессы и треды

Впервые потоки (threads, треды) появились на однопроцессорных вычислительных системах, позволяя более эффективно организовать вычисления.

Для использования достоинств многопроцессорных систем с общей памятью треды стали просто необходимы, так как позволяют не только реально ускорить выполнение тех задач, которые допускают их естественное распараллеливание, но и загрузить процессорные элементы работой, чтобы они не простаивали.

Примечание: крайне желательно уменьшить взаимодействие тредов между собой, ибо **ускорение** от одновременного выполнения параллельных потоков **может быть сведено к минимуму из-за задержек синхронизации и обмена данными**.

Каждый тред выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Треды, как и процессы, могут порождать треды-потомки, поскольку любой процесс состоит, по крайней мере, из одного треда. Подобно традиционным процессам (то есть процессам, состоящим из одного треда), каждый тред может находиться в одном из активных состояний. Пока один тред заблокирован (или просто находится в очереди готовых к исполнению задач), другой тред того же процесса может выполняться. Треды разделяют процессорное время так же, как это делают обычные процессы, в соответствии с различными вариантами диспетчеризации.

Процессы и треды

Особенности тредов

1. **Все треды имеют одно и то же виртуальное адресное пространство своего процесса**, т.е. они разделяют одни и те же глобальные переменные. Поскольку каждый тред может иметь доступ к каждому виртуальному адресу, один тред может использовать стек другого треда. Между потоками нет полной защиты, так как это, во-первых, невозможно, а во-вторых, не нужно. Все потоки одного процесса всегда решают общую задачу одного пользователя, и механизм потоков используется здесь для более быстрого решения задачи путем ее распараллеливания. Кроме разделения адресного пространства, **все треды разделяют также набор открытых файлов, используют общие устройства, выделенные процессу, имеют одни и те же наборы сигналов, семафоры и т. п.**

2. **Каждый тред имеет собственный: программный счетчик, стек, рабочие регистры процессора, потоки-потомки, состояние**. Вследствие этого, **между тредами**, относящимися к одному процессу (выполняются в одном и том же виртуальном адресном пространстве), **можно легко организовать тесное взаимодействие**, в отличие от процессов, для которых нужны специальные механизмы обмена сообщениями и данными. *Программист, создающий многопоточное приложение, может заранее продумать работу множества тредов процесса таким образом, чтобы они могли взаимодействовать наиболее выгодным способом, а не участвовать в конкуренции за предоставление ресурсов тогда, когда этого можно избежать.*

Процессы и треды

Советы по использованию потоков при создании приложений

1. **В случае использования однопроцессорной системы множество параллельных потоков часто не ускоряют работу приложения**, поскольку в каждый отдельно взятый промежуток времени возможно выполнение только одного потока. Кроме того, чем больше создается потоков, тем больше нагрузка на систему, потраченная на переключение между ними. Если проект имеет более двух постоянно работающих потоков, то такая мультизадачность не сделает программу быстрее, если каждый из потоков не будет требовать частого ввода/вывода.
2. **Необходимо определиться для чего необходим поток**. Поток, осуществляющий обработку, может помешать системе быстро реагировать на запросы ввода/вывода. Потоки позволяют программе отзываться на просьбы пользователя и устройств, но при этом сильно загружать процессор. Потоки позволяют компьютеру одновременно обслуживать множество устройств, и созданный поток, отвечающий за обработку специфического устройства, в качестве минимума может потребовать столько времени, сколько системе необходимо для обработки запросов всех устройств.
3. **Потокам можно назначить определенный приоритет для того, чтобы наименее значимые процессы выполнялись в фоновом режиме**. Это путь честного разделения ресурсов процессора. Необходимо осознать тот факт, что процессор один на всех, а потоков много. Если в программе главная процедура передает нечто для обработки в низкоприоритетный поток, то сама программа становится просто неуправляемой.

Процессы и треды

Советы по использованию потоков при создании приложений

4. **Потоки хорошо работают, когда они независимы**. Непродуктивная их работа начинается когда они вынуждены часто синхронизироваться для доступа к общим ресурсам. Блокировка и критические секции отнюдь не увеличивают скорость работы системы (хотя без них сложно организовать механизмы взаимодействия).
5. **Надо помнить, что память виртуальна**. Механизм виртуальной памяти следит за тем, какая часть виртуального адресного пространства должна находиться в оперативной памяти, а какая должна быть сброшена в файл подкачки. Потоки усложняют ситуацию, если они обращаются в одно и то же время к разным адресам виртуального адресного пространства приложения. Это значительно увеличивает нагрузку на систему. Поэтому надо разрабатывать программы с учетом этого факта (так же, как организуется доступ к файлу).
6. Всякий раз, когда какой-либо из потоков пытается воспользоваться общим ресурсом вычислительного процесса, которому он принадлежит, **необходимо тем или иным образом легализовать и защитить данные процесса**. Одно из решений: критические секции, семафоры и очереди сообщений. Даже после тестирования приложения и отсутствия в нем ошибок синхронизации в реальной эксплуатации они могут появиться.
7. **Не надо возлагать на поток несколько функций**. Сложные функциональные отношения затрудняют понимание общей структуры приложения, его алгоритм. Чем проще и менее многозначна каждая из рассматриваемых ситуаций, тем больше вероятность, что ошибок удастся избежать.