Управление процессами и потоками

Одной из основных подсистем любой современной мультипрограммной ОС, непосредственно влияющей на функционирование компьютера, является подсистема управления процессами и потоками. Основные функции этой подсистемы:

- создание процессов и потоков;
- обеспечение процессов и потоков необходимыми ресурсами;
- изоляция процессов;
- планирование выполнения процессов и потоков (вообще, следует говорить и о планировании заданий);
- диспетчеризация потоков;
- организация межпроцессного взаимодействия;
- синхронизация процессов и потоков;
- завершение и уничтожение процессов и потоков.

К созданию процесса приводят пять основных событий:

1. инициализация ОС (загрузка);
2. выполнение запроса работающего процесса на создание процесса;
3. запрос пользователя на создание процесса, например, при входе в систему в интерактивном режиме;
4. инициирование пакетного задания;
5. создание операционной системой процесса, необходимого для работы каких-либо служб.

Обычно при загрузке ОС создаются несколько процессов. Некоторые из них являются высокоприоритетными процессами, обеспечивающими взаимодействие с пользователями и выполняющими заданную работу. Остальные процессы являются фоновыми, они не связаны с конкретными пользователями, но выполняют особые функции – например, связанные с электронной почтой, Web-страницами, выводом на печать, передачей файлов по сети, периодическим запуском программ (например, дефрагментации дисков) и т.д. Фоновые процессы называют демонами.

Новый процесс может быть создан по запросу текущего процесса. Создание новых процессов полезно в тех случаях, когда выполняемую задачу проще всего сформировать как набор связанных, но, тем не менее, независимых взаимодействующих процессов. В интерактивных системах пользователь может запустить программу, набрав на клавиатуре команду или дважды щелкнув на значке программы. В обоих случаях создается новый процесс и запуск в нем программы. В системах пакетной обработки на мэйнфреймах пользователи посылают задание (возможно, с использованием удаленного доступа), а ОС создает новый процесс и запускает следующее задание из очереди, когда освобождаются необходимые ресурсы.

С технической точки зрения во всех перечисленных случаях новый процесс формируется одинаково: текущий процесс выполняет системный запрос на создание нового процесса. Подсистема управления процессами и потоками отвечает за обеспечение процессов необходимыми ресурсами. ОС поддерживает в памяти специальные информационные структуры, в которые записывает, какие ресурсы выделены каждому процессу. Она может назначить процессу ресурсы в единоличное пользование или совместное пользование с другими процессами. Некоторые из ресурсов выделяются процессу при его создании, а некоторые – динамически по запросам во время выполнения. Ресурсы могут быть

выделены процессу на все время его жизни или только на определенный период. При выполнении этих функций подсистема управления процессами взаимодействует с другими подсистемами ОС, ответственными за управление ресурсами, такими как подсистема управления памятью, подсистема ввода-вывода, файловая система.

Для того чтобы процессы не могли вмешаться в распределение ресурсов, а также не могли повредить коды и данные друг друга, важнейшей задачей ОС является изоляция одного процесса от другого. Для этого операционная система обеспечивает каждый процесс отдельным виртуальным адресным пространством, так что ни один процесс не может получить прямого доступа к командам и данным другого процесса.

В ОС, где существуют процессы и потоки, процесс рассматривается как заявка на потребление всех видов ресурсов, кроме одного – процессорного времени. Этот важнейший ресурс распределяется операционной системой между другими единицами работы – потоками, которые и получили свое название благодаря тому, что они представляют собой последовательности (потоки выполнения) команд. Переход от выполнения одного потока к другому осуществляется в результате *планирования* и *диспетчеризации*. Работа по определению момента, в который необходимо прервать выполнение текущего потока, и потока, которому следует предоставить возможность выполняться, называется планированием. Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков. При планировании принимается во внимание приоритет потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращения к вводу-выводу и другие факторы.

Диспетчеризация заключается в реализации найденного в результате планирования решения, т.е. в переключении процессора с одного потока на другой. Диспетчеризация проходит в три этапа:

- сохранение контекста текущего потока;
- загрузка контекста потока, выбранного в результате планирования;
- запуск нового потока на выполнение.

Когда в системе одновременно выполняется несколько независимых задач, возникают дополнительные проблемы. Хотя потоки возникают и выполняются синхронно, у них может возникнуть необходимость во взаимодействии, например, при обмене данными. Для общения друг с другом процессы и потоки могут использовать широкий спектр возможностей: каналы (в UNIX), почтовые ящики (Windows), вызов удаленной процедуры, сокеты (в Windows соединяют процессы на разных машинах). Согласование скоростей потоков также очень важно для предотвращения эффекта "гонок" (когда несколько потоков пытаются изменить один и тот же файл), взаимных блокировок и других коллизий, которые возникают при совместном использовании ресурсов.

Синхронизация потоков является одной из важнейших функций подсистемы управления процессами и потоками. Современные операционные системы предоставляют множество механизмов синхронизации, включая семафоры, мьютексы, критические области и события. Все эти механизмы работают с потоками, а не с процессами. Поэтому когда поток блокируется на семафоре, другие потоки этого процесса могут продолжать работу.

Каждый раз, когда процесс завершается, – а это происходит благодаря одному из следующих событий: обычный выход, выход по ошибке, выход по неисправимой ошибке, уничтожение другим процессом – ОС предпринимает шаги, чтобы "зачистить следы" его пребывания в системе. Подсистема управления процессами закрывает все файлы, с

которыми работал процесс, освобождает области оперативной памяти, отведенные под коды, данные и системные информационные структуры процесса. Выполняется коррекция всевозможных очередей ОС и список ресурсов, в которых имелись ссылки на завершаемый процесс.

Как уже отмечалось, чтобы поддержать мультипрограммирование, ОС должна оформить для себя те внутренние единицы работы, между которыми будет разделяться процессор и другие ресурсы компьютера. Возникает вопрос: в чем принципиальное отличие этих единиц работы, какой эффект мультипрограммирования можно получить от их применения и в каких случаях эти единицы работ операционной системы следует создавать?

Очевидно, что любая работа вычислительной системы заключается в выполнении некоторой программы. Поэтому и с процессом, и с потоком связывается определенный программный код, который оформляется в виде исполняемого модуля. В простейшем случае процесс состоит из одного потока, и в некоторых современных ОС сохранилось такое положение. Мультипрограммирование в таких ОС осуществляется на уровне процессов. При необходимости взаимодействия процессы обращаются к операционной системе, которая, выполняя функции посредника, предоставляет им средства межпроцессной связи – каналы, почтовые акции, разделяемые секции памяти и др.

Однако в системах, в которых отсутствует понятие потока, возникают проблемы при организации параллельных вычислений в рамках процесса. А такая необходимость может возникать. Дело в том, что отдельный процесс никогда не может быть выполнен быстрее, чем в однопрограммном режиме. Однако приложение, выполняемое в рамках одного процесса, может обладать внутренним параллелизмом, который, в принципе, мог бы ускорить его решение. Если, например, в программе предусмотрено обращение к внешнему устройству, то на время этой операции можно не блокировать выполнение всего процесса, а продолжить вычисления по другой ветви программы.

Параллельное выполнение нескольких работ в рамках одного интерактивного приложения повышает эффективность работы пользователя. Так, при работе с текстовым редактором желательно иметь возможность совмещения набора нового текста с такими продолжительными операциями, как переформатирование значительной части текста, сохранение его на локальном или удаленном диске.

Нетрудно представить будущую версию компилятора, способную автоматически компилировать файлы исходного кода в паузах, возникающих при наборе текста программы. Тогда предупреждения и сообщения об ошибках появлялись бы в режиме реального времени, и пользователь тут же видел бы, в чем он ошибся. Современные электронные таблицы пересчитывают данные в фоновом режиме, как только пользователь что-либо изменил. Текстовые процессоры разбивают текст на страницы, проверяют его на орфографические и грамматические ошибки, печатают в фоновом режиме, сохраняют текст каждые несколько минут и т.д. Во всех этих случаях потоки используются как средство распараллеливания вычислений.

Эти задачи можно было бы возложить на программиста, который должен был бы написать программу-диспетчер, реализующую параллелизм в рамках одного процесса. Однако это весьма сложно, да и сама программа получилась бы весьма запутанной и сложной в отладке.

Другим решением является создание для одного приложения нескольких процессов для каждой из параллельных работ. Однако использование для создания процессов стандартных средств ОС не позволяет учесть тот факт, что процессы решают единую задачу и имеют много общего: работают с одними и теми же данными, используют один и тот же кодовый сегмент, имеют одни и те же права доступа к ресурсам вычислительной системы. А операционная система при таком подходе будет рассматривать эти процессы наравне со всеми остальными процессами и обеспечивать их изоляцию друг от друга. В данном случае это будет не только бесполезная, но и вредная работа, затрудняющая обмен данными между различными частями приложения. Кроме того, на создание каждого процесса ОС тратит определенные системные ресурсы, которые в данном случае неоправданно дублируются – каждому процессу выделяется собственное виртуальное адресное пространство, физическая память, закрепляются устройства ввода-вывода и т.п.

Из изложенного следует вывод, что операционной системе наряду с процессами нужен другой механизм распараллеливания вычислений, который учитывал бы тесные связи между отдельными ветвями вычислений одного и того же приложения. Для этих целей современные ОС предлагают механизм многопоточной обработки (multithreading).

Понятию "поток" соответствует последовательный переход процессора от одной команды к другой. Процессу ОС назначают адресное пространство и набор ресурсов, которые совместно используются всеми его потоками. В отличие от процессов, которые принадлежат, вообще говоря, конкурирующим приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в гораздо меньшей степени, чем процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство.

Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждый поток может иметь доступ к любому виртуальному адресу, один поток может задействовать стек другого потока. Между потоками одного процесса нет полной защиты, во-первых, потому что это невозможно, а во-вторых, потому что не нужно. Чтобы организовать взаимодействие и обмен данными, потокам не требуется обращаться к ОС, им достаточно использовать общую память – один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

Таким образом, мультипрограммирование более эффективно на уровне потоков, а не процессов. Еще больший эффект многопоточной обработки достигается в мультипроцессорных системах, в которых потоки могут выполняться на разных процессорах действительно параллельно.