3. Процессы и потоки: создание и уничтожение, используемые структуры

Под **процессом** в современных ОС понимается системный объект, соответствующий запущенной на выполнение программе. Процесс создается системой при успешном запуске программы и существует все время, пока программа не будет завершена. В многозадачных ОС одновременно может быть запущено несколько программ и тем самым может существовать **несколько** процессов, совместно претендующих на системные ресурсы. Иногда такую ситуацию называют **внешней** многозадачностью. Все процессы разделяются на пользовательские и системные, создаваемые для поддержки нормальной работы самой ОС.

Кроме внешней, многозадачность может быть и **внутренней,** когда в одном процессе создается хотя бы два так называемых **потока** (thread). Поток – это относительно самостоятельный **фрагмент исполняемого кода** программы, который наравне с другими потоками претендует на время центрального процессора. Если программа (процесс) состоит только из одного потока, то она называется однопоточной, а иначе – многопоточной. Однопоточная программа содержит только один поток, называемый главным. Он автоматически создается системой при создании процесса и содержит весь исполняемый код программы. Разработчик программы может предусмотреть внутри своего кода любое число дополнительных потоков.

Необходимо подчеркнуть, что далеко не каждое приложение должно быть многопоточным, выбор должен быть обоснованным и целесообразным. Использование потоков дает целый ряд мощных возможностей, но в то же время создает и ряд проблем. В качестве типичных **примеров** потоков, которые целесообразно выделять в приложении, можно привести:

- **вычислительный** поток, которому из всех ресурсов в основном нужен только центральный процессор;
- поток **ввода данных** с клавиатуры, который, наоборот, в основном пребывает в состоянии ожидания нажатия очередной клавиши;

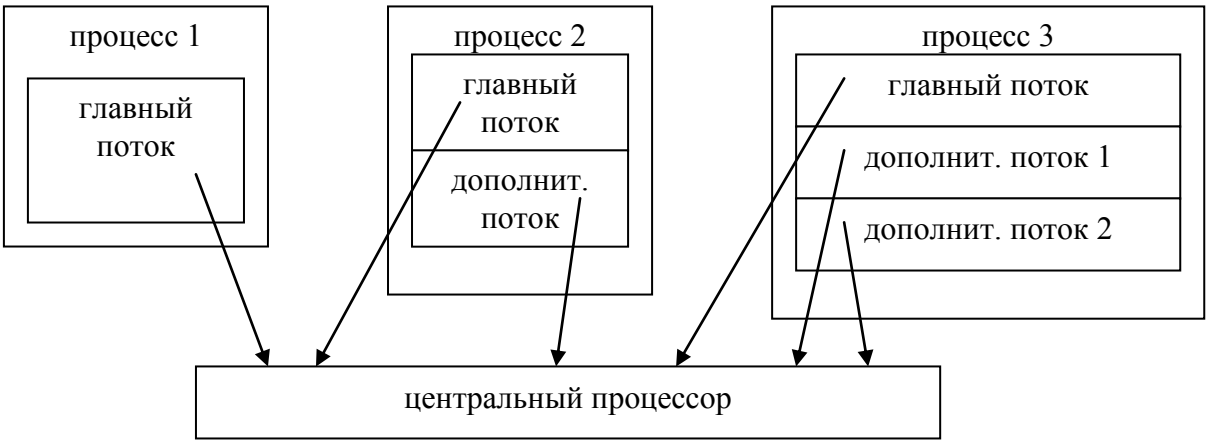
- поток **взаимодействия** с внешней (дисковой) **памятью;**
- **коммуникационный** поток (например, получение/отправка электронной почты).

Если приложение довольно часто выполняет перечисленные выше работы, то выделение в нем потоков позволит более рационально загрузить основные устройства компьютера и создать **иллюзию одновременного выполнения** приложением нескольких работ. На многопроцессорных системах дополнительно появляется возможность выполнения разных потоков на разных процессорах.

Примерами рационального использования потоков являются:

- мощные современные текстовые редакторы, в которых один поток отвечает за взаимодействие с пользователем, другой – за проверку орфографии вводимого текста, а третий – за автоматическое сохранение текста в файле;
- Web-серверы сети Интернет, обрабатывающие интенсивный поток заявок от клиентских машин с созданием для каждого клиента своего потока в рамках одного приложения.

Внешнюю и внутреннюю многозадачность можно проиллюстрировать следующим схематичным примером: созданы три процесса, в первом – только главный поток, во втором кроме главного имеется еще один дополнительный поток, а в третьем дополнительных потоков два. Тем самым за время ЦП максимально будут конкурировать шесть потоков.



Одно из важнейших отличий процесса от потока состоит в том, что процессу выделяются **все необходимые ему ресурсы** – основная память, файлы,

программные модули, внешние устройства, тогда как поток самостоятельно использует **только время ЦП,** а все остальные ресурсы у всех потоков одного процесса **одни и те же.** Можно сказать, что процесс – это единица распределения всех ресурсов приложения, тогда как поток – это единица распределения только процессорного времени. Следствием этого являются:

- создание нового потока **внутри** уже созданного процесса требует гораздо меньшей системной работы и выполняется значительно быстрее, чем создание нового процесса (в несколько десятков раз);
- все потоки внутри одного процесса используют **одну и ту же** область оперативной памяти, выделенную процессу в целом;
- взаимодействие потоков **внутри** процесса выполняется гораздо **проще,** чем взаимодействие потоков **разных** процессов;
- потоки внутри одного процесса практически **не защищены** от случайного или преднамеренного вмешательства со стороны других «родных» потоков, и поэтому ошибка в выполнении одного потока может нарушить работу всех остальных потоков и процесса в целом, что приводит к необходимости очень тщательной отладки многопоточных приложений.

Управление процессами и потоками включает в себя решение следующих основных задач:

- **создание** и **уничтожение** процессов и потоков;
- **планирование** порядка выполнения потоков;
- **синхронизацию** и **взаимодействие** потоков.

Операционная система создает объект «процесс» в ответ на запуск какой-либо программы на выполнение. Создание процесса начинается с формирования **специальной структуры данных,** содержащей информацию о запускаемой программе. Такая структура создается для каждого процесса и часто называется **дескриптором** (описателем) процесса. Набор дескрипторов образует таблицу, которая поддерживается системой и хранится в системной области основной памяти, недоступной для прикладных программ. Структура дескрипторов различна для разных ОС, но в основном содержит следующие данные:

идентификатор процесса, **время создания, полное имя** исполняемого файла, **приоритет** и **права** доступа процесса, используемые процессом **ресурсы** (память**,** файлы, устройства), **идентификаторы** потоков процесса. Таким образом, дескриптор процесса содержит всю информацию, необходимую системе для полного контроля над процессом.

Если ОС поддерживает **многопоточность,** то после создания процесса автоматически создается системный объект **«главный поток».** Его создание – это также формирование **своей структуры данных,** описывающей свойства потока. Эта структура, называемая **дескриптором потока,** создается для каждого потока и содержит следующую информацию: **идентификатор** потока, **приоритет** потока, **состояние** потока, параметры планирования потока, указатель на еще одну чрезвычайно важную структуру данных – **контекст** потока.

Контекст предназначен для **сохранения текущего состояния потока** в момент его прерывания и обычно включает в себя содержимое общих и системных регистров процессора (включая счетчик команд с адресом очередной подлежащей выполнению команды, регистр состояния процессора и регистр-указатель вершины стека). Информация из этой структуры используется для корректного возобновления работы потока. Схематично взаимосвязь описанных структур представлена на следующем рисунке:

Дескрипторы и контексты дополнительных потоков создаются динамически в процессе работы приложения в ответ на соответствующие системные вызовы. Например, **Win32 API** для создания процессов и потоков предоставляет системные вызовы **CreateProcess** и **CreateThread.**

После создания дескрипторов процесса и главного потока выполняется **загрузка** в основную память кода и данных запускаемой программы. Если все описанные выше этапы проходят успешно (необходимая память выделена, загружаемые файлы найдены и т.д.), то главный поток **включается в очередь** готовых к исполнению потоков и **начинает конкурировать** за время ЦП. Момент непосредственного начала работы главного потока зависит от реализованной в ОС стратегии планирования, текущего состояния вычислительной системы и параметров потока.

В целом видно, что создание нового процесса требует использования системных модулей управления памятью (выделение памяти для системных структур и запускаемой программы), а также взаимодействия с файловой подсистемой, что требует относительно больших затрат времени. Для ускорения и упрощения этих действий, в системах семейства **Unix** часто используется создание процессов и потоков **на основе** уже существующих, которые в данном случае называются **родительскими. Дочерние** процессы (потоки) **наследуют** все свойства своих родителей и при необходимости лишь изменяют некоторые из них. После этого дочерние процессы (потоки) могут существовать как зависимо, так и независимо от своих родителей. Зависимость процессов (потоков), например, может означать то, что завершение родительского процесса (потока) автоматически завершает все дочерние процессы (потоки). Подобная схема создания процессов реализуется в системах Unix с помощью вызова **fork.**

Возможные причины завершения работы потоков:

- нормальное завершение, т.е. выполнение последней команды потока;
- аварийное завершение, когда процессор не может далее выполнять команды потока в силу каких-либо ошибок;

- принудительное завершение со стороны других потоков (например, родительского).

В этих случаях происходит удаление всей связанной с потоком информации. При завершении главного потока происходит удаление и информации о процессе в целом.