**Тема 3. Общие сведения о процессах и потоках**

**Лекция 11 «Применение потоков. Классификация потоков. Реализация потоков»**.

**Поток** (thread) — это, сущность операционной системы, процесс выполнения процессором набора инструкций, т.е. программного кода. Планировщик ОС, руководствуясь приоритетом потока, распределяет кванты времени между разными потоками и ставит потоки на выполнение.

Потоки позволяют получить довольно большой прирост производительности приложений, выполняющих серьезные вычисления за счет параллельного выполнения различных задач и лёгкости создания. Особенно это актуально для систем, использующих несколько центральных процессоров, где появляется «реальная» возможность параллельных вычислений.

Процесс — это экземпляр выполняемой программы, а поток — это компонент процесса, который является самой маленькой исполнительной единицей и содержит: счетчик команд, отслеживающий порядок выполнения действий; набор регистров в которых хранятся текущие переменные; стек, содержащий протокол выполнения процесса. Процесс является тяжелым, а поток — легким. Поэтому один процесс делится на несколько потоков для эффективности. Процесс требует значительных ресурсов, а поток требует минимальных ресурсов. Поток не может существовать вне процесса, и каждый поток принадлежит процессу.

**Применение потоков.**

Допустим есть веб-сайт, куда поступают запросы на определенные веб-страницы, которые затем отправляются обратно пользователям. Большинство клиентских запросов приходится на главную страницу сайта. Данные «подборки» называются кэшем, который позволяет повысить производительность за счёт размещения часто используемых запросов на основной странице, то есть в основной памяти.

Один из потоков называется диспетчером, который «читает» входящие запросы от пользователей. Далее он анализирует запрос, подбирает «спящий» (то есть заблокированный) рабочий поток, которому и передает запрос. После этого диспетчер разблокирует данный поток и переводит его в состояние готовности к работе.

После перевода рабочий поток проверяет возможность получения запрашиваемых данных из кэша страниц (доступ к которому имеют все потоки). Если нужная информация не находится, то поток запускает операцию чтения с диска и переводится в «спящий» режим работы до завершения операции.

При блокировке одного потока из-за необходимости поиска данных с диска разблокируется другой рабочий поток и переходит в состояние готовности к работе.

Потоки позволяют использовать концепцию последовательных процессов, которые осуществляют блокирующие системные вызовы и в это же время позволяют «распараллелить» работу. Данные блокирующие системные вызовы позволяют упростить программирование, а работа в параллели даёт большую производительность.

Потоки можно **классифицировать** следующим способом:

* По отображению в ядро: 1:1, N:M, N:1.
* По многозадачной модели: вытесняющая многозадачность, кооперативная многозадачность.
* По уровню реализации: режим ядра, режим пользователя, гибридная реализация.

Классификация по отображению в ядро:

**Модель 1:1** — самая простая модель. любой поток, созданный в любом процессе управляется напрямую планировщиком ядра ОС, то есть один пользовательский процесс на один поток ядра. Такая модель реализована в Linux начиная с ядра 2.6, а также Windows.

**Модель N:M** - данная модель труднореализуема, но обладает серьезной производительностью, так как можно избежать значительного количества системных вызовов. Отображает некоторое число потоков пользовательских процессов N на M потоков режима ядра. Проще говоря имеем некую гибридную систему, когда часть потоков ставится на выполнение в планировщике ОС, а большая их часть в планировщике потоков процесса или библиотеке потоков.

**Модель N:1**.– это множество потоков пользовательского процесса, которые отображаются на один поток ядра ОС.

Классификация по многозадачной модели:

Кооперативная многозадачность означает то, что все потоки выполняют свою работу по очереди, с равным временем выполнения. Никакой другой поток, не может вытеснить текущий выполняющийся поток.

Вытесняющая многозадачность.

Поток с большим приоритетом «вытесняет» поток с меньшим. Современные ОС используют данный подход для выполнения поставленных задач.

Классификация по уровню реализации:

Реализацию потоков на уровне ядра можно сравнить с классической моделью 1:1.

Реализация потоков в пользовательском режиме. Такие тяжелые операции, как системный вызов и смена контекста требовали реализации поддержки потоков в режиме пользователя. Было сделано много попыток реализации, но данная методика не обрела популярности.

Гибридная (смешанная) реализация появилась при попытке совместить преимущества двух предшествующих реализаций, но задумка не удалась: получилось больше недостатков, чем достоинств.

**Реализация потоков**.

Потоки имеют два уровня реализации: пользовательские потоки — это потоки, управляемые пользователем и потоки ядра, поддерживаются и управляются ядром.

1. **Реализация потока в пространстве пользователя.** то есть потоки, управляемые приложениями; Ядро в общем случае не знает о существовании потоков. Все потоки работают поверх службы поддержки программ. Библиотека потоков просто содержит код для создания и уничтожения потоков, а также передачи сообщений и данных между ними для планирования выполнения потоков и сохранения (восстановления) контекстов потоков:

a. Thread -create создание;

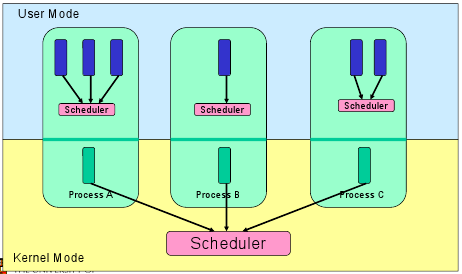
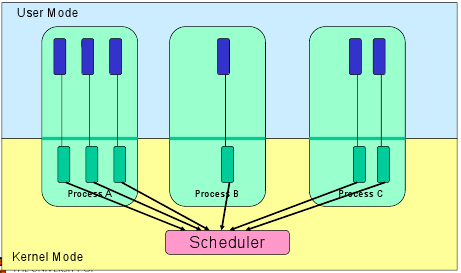
b. Thread-exit завершение;

c. Thread-join ожидание завершения некоторых потоков;

d. Thread-yield передача процессора другому потоку.

Преимущества: быстродействие и возможность иметь собственную политику потоков.

Недостатки: трудность реализации блокирующих системных вызовов и реализации виртуальной памяти.

1. **в пространстве пользователя b) в ядре ОС**

**2)** **Реализация потоков в ядре ОС**. то есть потоки, управляемые ядром операционной системы. Ядро выполняет создание потоков, а ещё планирование и управление в пространстве ядра. Когда потоку нужно создать новый или уничтожить существующий поток, он обращается к ядру, которое производит создание или разрушение путём обновления таблицы потоков в ядре. Потоки ядра обычно медленнее, чем потоки пользователей.

Преимущества: решены проблемы 1 системы.

Недостатки: уменьшение быстродействия. Нет собственной политики переключения.

**3)** **Способ активации планировщика.** Ядро не управляет потоками непосредственно, потоки работают поверх системы исполнения программ. Когда ядро определяет необходимость блокировки исполняемого потока (например: при вызове системного вызова), оно выполняет вызов системы поддержки исполнения программ, передовая в качестве аргументов номер потока и описание случившегося, при этом происходит «обратный вызов» (callback), что нарушает иерархию.

**4)** **Всплывающие потоки.** Основное назначение- обработка поступающих в систему запросов. При поступлении запроса создается новый поток для его обработки, что позволяет ускорить реакцию для написания события.

Преимущества: они создаются заново и не имеют прошлого – никаких регистров, стеков и всего остального, что должно быть восстановлено, создания таких потоков довольно быстро.

Недостатки: при превращении однопоточной программы в многопоточную отсутствует возможность повторного входа во многие библиотечные процедуры.