**Лекция 6. Применение потоков**

 **Цель занятия:** Сформировать знания студентов о потоках в операционных системах, о классификации и реализации потоков.

## Потоки

Концепция процесса, пришедшая из мира UNIX, плохо реализуется в многозадачной системе, поскольку процесс имеет тяжелый контекст. Возникает понятие **потока (thread)**, который понимается как подпроцесс, или *легковесный процесс (light-weight process)*, выполняющийся в контексте полноценного процесса.

С помощью процессов можно организовать параллельное выполнение программ. Для этого процессы клонируются вызовами fork() или exec(), а затем между ними организуется взаимодействие средствами IPC. Это довольно дорогостоящий в отношении ресурсов способ.

С другой стороны, для организации параллельного выполнения и взаимодействия процессов можно использовать механизм многопоточности. Основной единицей здесь является **поток**, который представляет собой облегченную версию процесса. Чтобы понять, в чем состоит его особенность, необходимо вспомнить основные характеристики процесса.

1. Процесс располагает определенными ресурсами. Он размещен в некотором виртуальном адресном пространстве, содержащем образ этого процесса. Кроме того, процесс управляет другими ресурсами (файлы, устройства ввода / вывода и т.д.).
2. Процесс подвержен диспетчеризации. Он определяет порядок выполнения одной или нескольких программ, при этом выполнение может перекрываться другими процессами. Каждый процесс имеет состояние выполнения и приоритет диспетчеризации.

Если рассматривать эти характеристики независимо друг от друга (как это принято в современной теории ОС), то:

* владельцу ресурса, обычно называемому процессом или задачей, присущи:
	+ виртуальное адресное пространство;
	+ индивидуальный доступ к процессору, другим процессам, файлам, и ресурсам ввода — вывода.
* Модулю для диспетчеризации, обычно называемому потоком или облегченным процессом, присущи:
	+ состояние выполнения (активное, готовность и т.д.);
	+ сохранение контекста потока в неактивном состоянии;
	+ стек выполнения и некоторая статическая память для локальных переменных;
	+ доступ к пространству памяти и ресурсам своего процесса.

Все потоки процесса разделяют общие ресурсы. Изменения, вызванные одним потоком, становятся немедленно доступны другим.

При корректной реализации потоки имеют определенные преимущества перед процессами. Им требуется:

* меньше времени для создания нового потока, поскольку создаваемый поток использует адресное пространство текущего процесса;
* меньше времени для завершения потока;
* меньше времени для переключения между двумя потоками в пределах процесса;
* меньше коммуникационных расходов, поскольку потоки разделяют все ресурсы, и в частности адресное пространство. Данные, продуцируемые одним из потоков, немедленно становятся доступными всем другим потокам.

### *Преимущества многопоточности*

Если операционная система поддерживает концепции потоков в рамках одного процесса, она называется многопоточной. Многопоточные приложения имеют ряд преимуществ.

* Улучшенная реакция приложения — любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке. Например, пользователь многопоточного интерфейса не должен ждать завершения одной задачи, чтобы начать выполнение другой.
* Более эффективное использование мультипроцессирования — как правило, приложения, реализующие параллелизм через потоки, не должны учитывать число доступных процессоров. Производительность приложения равномерно увеличивается при наличии дополнительных процессоров. Численные алгоритмы и приложения с высокой степенью параллелизма, например перемножение матриц, могут выполняться намного быстрее.
* Улучшенная структура программы — некоторые программы более эффективно представляются в виде нескольких независимых или полуавтономных единиц, чем в виде единой монолитной программы. Многопоточные программы легче адаптировать к изменениям требований пользователя.
* Эффективное использование ресурсов системы — программы, использующие два или более процессов, которые имеют доступ к общим данным через разделяемую память, содержат более одного потока управления. При этом каждый процесс имеет полное адресное пространство и состояние в операционной системе. Стоимость создания и поддержания большого количества служебной информации делает каждый процесс более затратным, чем поток. Кроме того, разделение работы между процессами может потребовать от программиста значительных усилий, чтобы обеспечить связь между потоками в различных процессах или синхронизировать их действия.

**ЗАДАЧА (TASK)** — блок программного кода, ответственный за обработку тех или иных событий, возникающих на объекте управления.

Задача может быть «оформлена» в виде:

* Отдельного процесса
* Потока управления внутри процесса (нити, легковесного процесса)
* Обработчика прерывания/подпрограммы

**РАБОТА ЗАДАЧИ (JOB)** — процесс исполнения блока программного кода в ходе обработки события.

Каждая работа задачи характеризуется следующими временными параметрами:

* ***r (Release Time)*** — момент времени, когда задача становится готовой к исполнению (например, процесс переходит в состояние готовности)
* ***d (Absolute Deadline)*** — абсолютный крайний срок, момент времени, к которому задача должна завершить очередную работу.
* ***s (Start Time)*** — момент времени, когда задача начала исполняться на процессоре
* ***с (Complition Time)*** — момент времени, когда задача закончила работу, обработав событие
* ***D (Relative Deadline)*** — относительный крайний срок. D = d — r
* ***e (Execution Time)*** — время исполнения задачи при выполнении ею очередной работы. e = c — s
* ***R (Response Time)*** — время отклика. R = c — r

Диаграмма ниже иллюстрирует эти параметры:

 r s c d

 \* |---------------| |

 | | | |

 | | | \*

--+---+---+---+---+---+---+---+---+---+---+---+---+---+----> t (у.е.)

 0 1 2 3 4 5 6 7 8 9 10 11 12 13

Приведенная на этой диаграмме работа задачи имеет следующие параметры:

* r = 2
* d = 11
* s = 5
* с = 9
* D = 11 — 2 = 9
* e = 9 — 5 = 4
* R = 9 -2 = 7

Упомянутые параметры определяются следующим:

* Времена перехода задач в состояние готовности, по сути, определяются природой управляемого объекта.
* Крайние сроки определяет разработчик СРВ, исходя из свойств управляемого объекта.
* Времена исполнения задач определяются архитектурой процессора, его тактовой частотой и конкретной реализацией того или иного алгоритма.
* Для определения последней величины можно использовать 2 подхода.
	+ Первый заключается в подсчете количества тактов процессора, необходимых на выполнение той или иной задачи. Отметим, что такой подсчет чрезвычайно усложняется в случае, если процессор содержит механизмы типа конвейеров и всевозможных кэшей.
	+ Второй подход более прост и состоит в том, что времена исполнения непосредственно измеряются. Опять отметим, что в случае процессоров с конвейерами и кэшами такие измерения не дают гарантии, что будет измерено именно МАКСИМАЛЬНОЕ время исполнения того или иного кода (???). Наконец, системы, использующие механизмы подкачки страниц, также являются менее предсказуемыми и поэтому считается, что такого рода механизмы являются «врагами» систем реального времени. Недаром в различного рода стандартах, касающихся СРВ, предусмотрены средства блокировки страниц памяти.
* Планирование процессов и потоков
* Одной из основных подсистем мультипрограммной ОС, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами и потоками, которая занимается их созданием и уничтожением, поддерживает взаимодействие между ними, а также распределяет процессорное время между несколькими одновременно существующими в системе процессами и потоками.
* Подсистема управления процессами и потоками ответственна 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 в совместное пользование с другими процессами. Некоторые из ресурсов выделяются процессу при его создании, а некоторые — динамически по запросам во время выполнения. Ресурсы могут быть приписаны 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.
* Когда в системе одновременно выполняется несколько независимых задач, то возникают дополнительные проблемы. Хотя потоки возникают и выполняются асинхронно, у них может возникнуть необходимость во взаимодействии, например при обмене данными. Согласование скоростей потоков также очень важно для предотвращения эффекта «гонок» (когда несколько потоков пытаются изменить один и тот же файл), взаимных блокировок или других коллизий, которые возникают при совместном использовании ресурсов. Синхронизация потоков является одной из важных функций подсистемы управления процессами и потоками.
* Каждый раз, когда процесс завершается, 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 списков ресурсов, в которых имелись ссылки на завершаемый процесс.

**Контрольные вопросы:**

* 1. *Что такое потоки и многопоточность?*
	2. *Выявите преимущества многопоточности.*
	3. *Что такое задача?*