ПОТОКИ

Потоки, причины использования

- В традиционных операционных системах у каждого процесса есть адресное пространство и единственный поток управления. Фактически это почти что определение процесса.
- Тем не менее нередко возникают ситуации, когда неплохо было бы иметь в одном и том же адресном пространстве несколько потоков управления, выполняемых квазипараллельно.

Аргумент 1

- Единое адресное пространство
- во многих приложениях одновременно происходит несколько действий, часть которых может периодически быть заблокированной.
- Модель программирования упрощается за счет разделения такого приложения на несколько последовательных потоков, выполняемых в квазипараллельном режиме.

Возможность использования параллельными процессами единого адресного пространства и всех имеющихся данных играет важную роль для тех приложений, которым не подходит использование нескольких процессов (с их раздельными адресными пространствами).

Аргумент 2

- Легкость (то есть быстрота) создания и ликвидации потоков по сравнению с более «тяжеловесными» процессами.
- Во многих системах создание потоков осуществляется в 10–100 раз быстрее, чем создание процессов.
- Это свойство особенно пригодится, когда потребуется быстро и динамично изменять количество потоков.

Аргумент 3

- Производительность.
- Когда потоки работают в рамках одного центрального процессора, они не приносят никакого прироста производительности, но когда выполняются значительные вычисления, а также значительная часть времени тратится на ожидание ввода-вывода, наличие потоков позволяет этим действиям перекрываться по времени, ускоряя работу приложения.

Аргумент 4

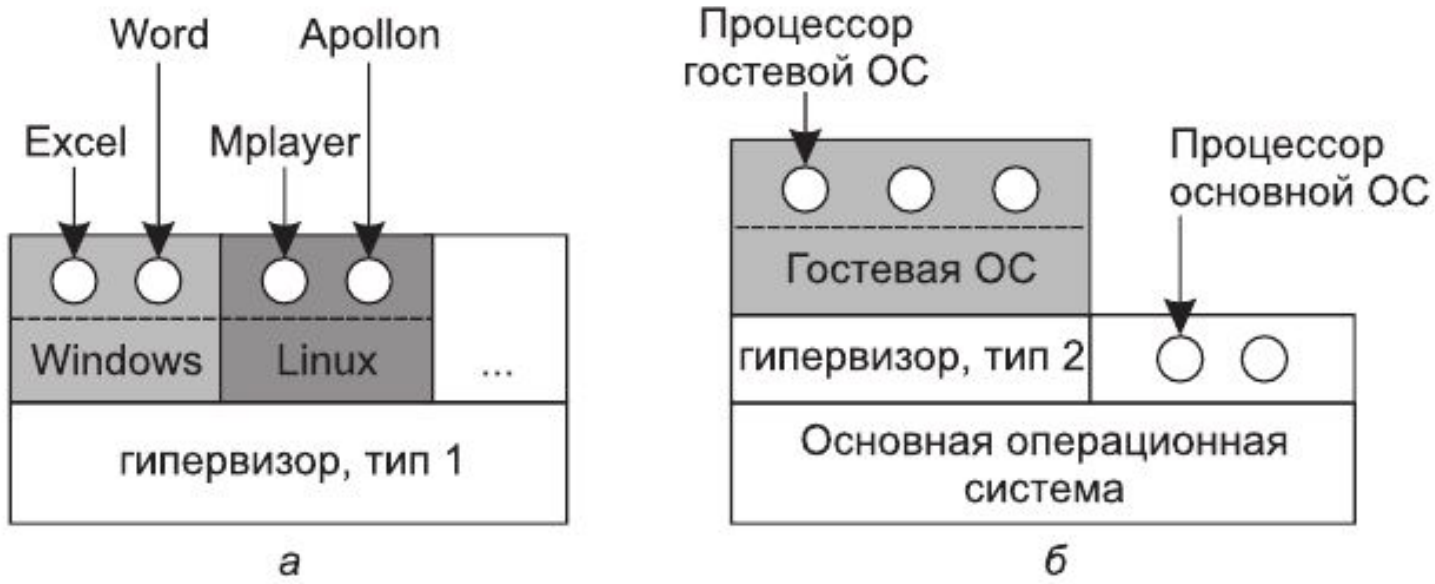
- Потоки весьма полезны для систем, имеющих несколько центральных процессоров, где есть реальная возможность параллельных вычислений.

Пример 1 (2 потока)

- Текстовый процессор как двухпоточная программа.
- Один из потоков взаимодействует с пользователем, а другой занимается переформатированием в фоновом режиме.
- Как только предложение с первой страницы будет удалено, поток, отвечающий за взаимодействие с пользователем, приказывает потоку, отвечающему за формат, переформатировать всю книгу.
- Пока взаимодействующий поток продолжает отслеживать события клавиатуры и мыши, реагируя на простые команды вроде прокрутки первой страницы, второй поток с большой скоростью выполняет вычисления.

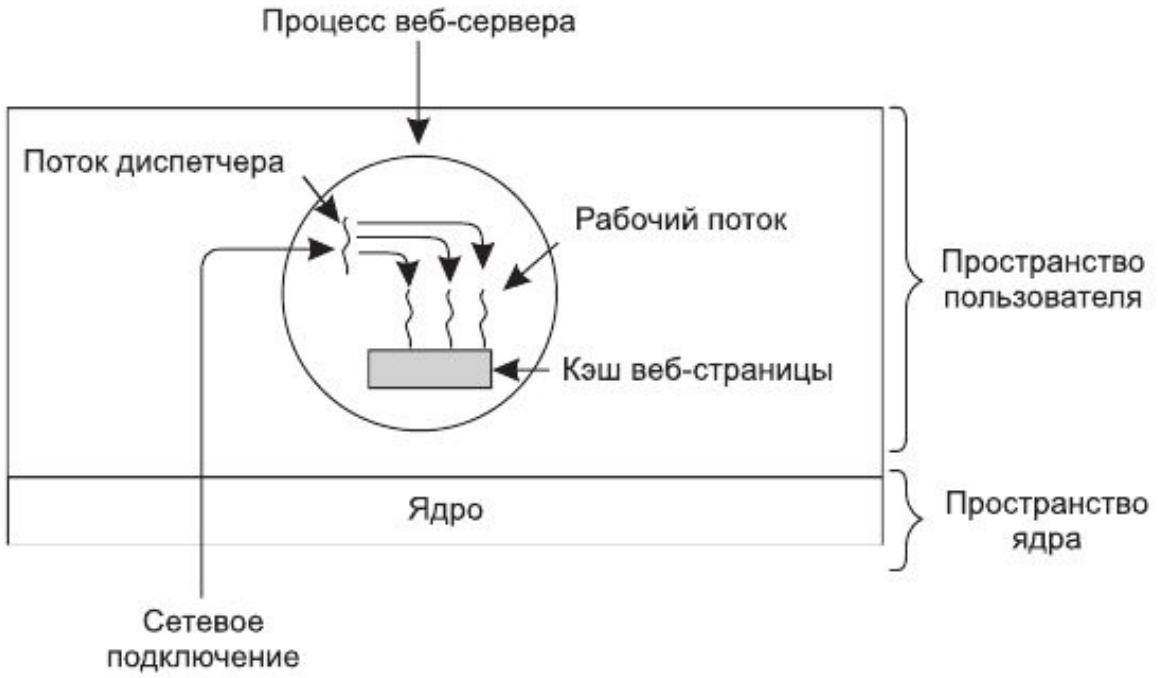
Пример 1 (3 потока)

- Третий поток может заниматься созданием резервных копий на диске, не мешая первым двум.
- . Три потока вместо трех процессов используют общую память, таким образом, все они имеют доступ к редактируемому документу. При использовании трех процессов та



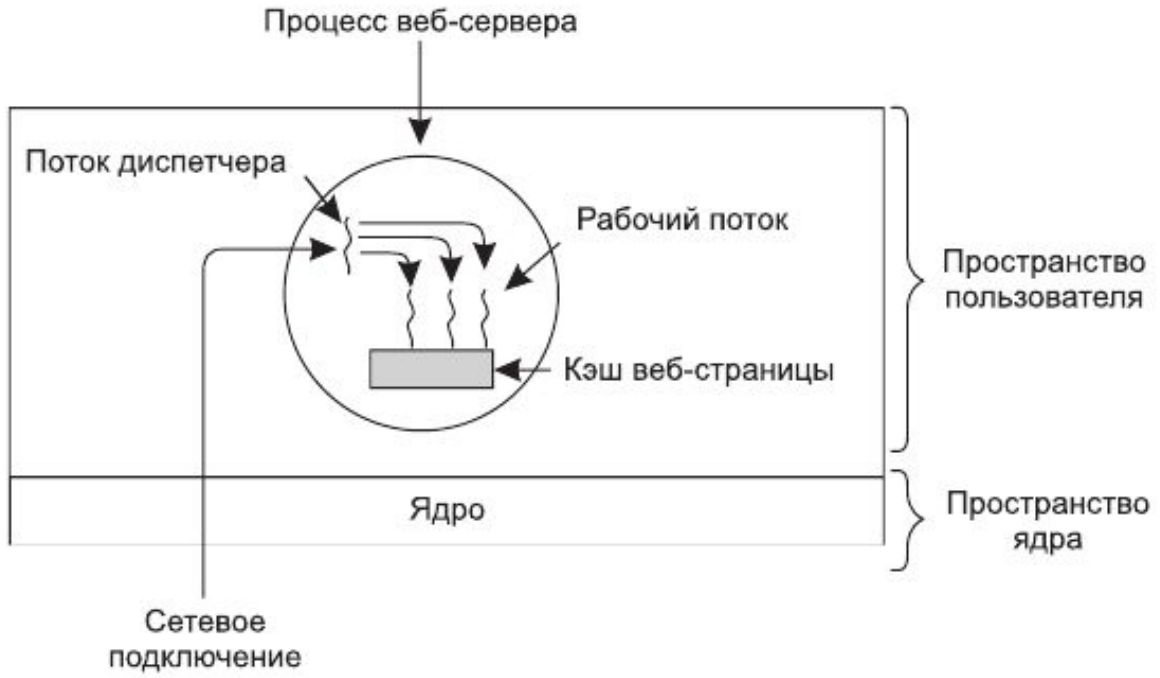
Пример 2. Многопоточный веб-сервер

- Один из потоков — диспетчер — читает входящие рабочие запросы из сети.
- После анализа запроса он выбирает простаивающий (заблокированный) рабочий поток и передает ему запрос, возможно, путем записи указателя на сообщение в специальное слово, связанное с каждым потоком.
- Затем диспетчер пробуждает спящий рабочий поток, переводя его из заблокированного состояния в состояние готовности.



Пример 2. Многопоточный веб-сервер

- Рабочий поток проверяет, может ли запрос быть удовлетворен из кэша веб-страниц, к которому имеют доступ все потоки.
- Если нет, то он приступает к операции чтения с диска и блокируется до тех пор, пока не завершится дисковая операция.
- Когда поток блокируется на дисковой операции, выбирается выполнение другого потока, возможно, диспетчера, с целью получения следующей задачи или, возможно, другого рабочего потока, который находится в готовности к выполнению.



Как можно было бы написать код веб-сервера в отсутствие потоков?

- Можно заставить его работать в виде единого потока.
- Основной цикл веб-сервера получает запрос, анализирует его и завершает обработку до получения следующего запроса.
- Ожидая завершения дисковой операции, сервер простаивает и не обрабатывает никаких других входящих запросов.
- Если веб-сервер запущен на специально выделенной машине, что чаще всего и бывает, то центральный процессор, ожидая завершения дисковой операции, остается без дела.
- В итоге происходит значительное сокращение количества запросов, обрабатываемых в секунду.

Машина с конечным числом состояний

- При такой конструкции модель «последовательного процесса», присутствующая в первых двух случаях, уже не работает.
- Состояние вычисления должно быть явным образом сохранено и восстановлено из таблицы при каждом переключении сервера с обработки одного запроса на обработку другого.
- В результате потоки и их стеки имитируются более сложным образом. Подобная конструкция, в которой у каждого вычисления есть сохраняемое состояние и имеется некоторый набор событий, которые могут происходить с целью изменения состояния, называются машиной с конечным числом состояний (finite-state machine), или конечным автоматом. Это понятие получило в вычислительной технике весьма широкое распространение.

Итог, потоки

- дают возможность сохранить идею последовательных процессов, которые осуществляют блокирующие системные вызовы (например, для операций дискового ввода-вывода), но при этом позволяют все же добиться распараллеливания работы.
- *Блокирующие системные вызовы упрощают программирование, а параллельная работа повышает производительность.*
- Однопоточные серверы сохраняют простоту блокирующих системных вызовов, но *уступают им в производительности*.
- Третий подход позволяет добиться высокой производительности за счет параллельной работы, но использует неблокирующие вызовы и прерывания, *усложняя процесс программирования*

Пример 3. Приложения, предназначенные для обработки очень большого объема данных

- . При обычном подходе блок данных считывается, после чего обрабатывается, а затем снова записывается.Проблема в том, что при доступности лишь блокирующих вызовов процесс блокируется и при поступлении данных, и при их возвращении. Возникает простой ЦП!!
- Проблема решается с помощью потоков. **Структура процесса может включать входной поток, обрабатывающий поток и выходной поток**.
- Входной поток считывает данные во входной буфер. Обрабатывающий поток извлекает данные из входного буфера, обрабатывает их и помещает результат в выходной буфер. Выходной буфер записывает эти результаты обратно на диск. Таким образом, ввод, вывод и обработка данных могут осуществляться одновременно. Эта модель работает лишь при том условии, что системный вызов *блокирует только вызывающий поток, а не весь процесс.*

Процесс

- является способом группировки в единое целое взаимосвязанных ресурсов (адресное пространство, содержащее текст программы и данные, а также **другие ресурсы**).
- Эти ресурсы могут включать открытые файлы, необработанные аварийные сигналы, обработчики сигналов, учетную информацию и т. д.
- Управление этими ресурсами можно значительно облегчить, если собрать их воедино в виде процесса.

Поток, у потока есть

- счетчик команд, отслеживающий, какую очередную инструкцию нужно выполнять.
- регистры, в которых содержатся текущие рабочие переменные.
- стек с протоколом выполнения, содержащим по одному фрейму для каждой вызванной, но еще не возвратившей управление процедуры.

- Хотя поток может быть выполнен в рамках какого-нибудь процесса, сам поток и его процесс являются разными понятиями и должны рассматриваться по отдельности.
- Процессы используются для группировки ресурсов в единое образование, а потоки являются «сущностью», распределяемой для выполнения на центральном процессоре.

Использование объектов потоками

Элементы, присущие каждому процессу	Элементы, присущие каждому потоку
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и обработчики сигналов	
Учетная информация	

каждый поток имеет собственный стек

- Стек каждого потока содержит по одному фрейму для каждой уже вызванной, но еще не возвратившей управление процедуры.
- Такой фрейм содержит локальные переменные процедуры и адрес возврата управления по завершении ее вызова.
- Например, если процедура X вызывает процедуру Y, а Y вызывает процедуру Z, то при выполнении Z в стеке будут фреймы для X, Y и Z.
- Каждый поток будет, как правило, вызывать различные процедуры и, следовательно, иметь среду выполнения, отличающуюся от среды выполнения других потоков. Поэтому каждому потоку нужен собственный стек.

Когда используется многопоточность,

- процесс обычно начинается с использования одного потока.
- Этот поток может создавать новые потоки, вызвав библиотечную процедуру, к примеру thread_create.
- В параметре thread_create обычно указывается имя процедуры, запускаемой в новом потоке.
- Нет необходимости (или даже возможности) указывать для нового потока какое-нибудь адресное пространство.
- Иногда потоки имеют **иерархическую структуру**, при которой у них устанавливаются взаимоотношения между родительскими и дочерними потоками, но чаще всего такие взаимоотношения отсутствуют и все потоки считаются равнозначными.
- Создающий *поток обычно возвращает идентификатор потока, который дает имя новому потоку.*

Когда поток завершает свою работу,

- выход из него может быть осуществлен за счет вызова библиотечной процедуры, к примеру thread_exit.
- После этого он прекращает свое существование и больше не фигурирует в работе планировщика.
- В некоторых использующих потоки системах какой-нибудь поток для выполнения выхода может ожидать выхода из какого-нибудь другого (указанного) потока после вызова процедуры, к примеру thread_join.
- Эта процедура блокирует вызывающий поток до тех пор, пока не будет осуществлен выход из другого (указанного) потока.
- В этом отношении создание и завершение работы потока очень похожи на создание и завершение работы процесса при использовании примерно одних и тех же параметров.

Сложности использования потоков При выполнении системного вызова fork (UNIX)

- Если у родительского процесса есть несколько потоков, должны ли они быть у дочернего процесса?
- Если нет, то процесс может неверно функционировать из-за того, что все они составляют его неотъемлемую часть.
- Но если дочерний процесс получает столько же потоков, сколько их было у родительского процесса, что произойдет, если какой-нибудь из потоков родительского процесса был заблокирован системным вызовом read, используемым, к примеру, для чтения с клавиатуры?
- Будут ли теперь два потока, в родительском и в дочернем процессах, заблокированы на вводе с клавиатуры?
- Если будет набрана строка, получат ли оба потока ее копию?
- Или ее получит только поток родительского процесса? А может быть, она будет получена только потоком дочернего процесса?
- Сходные проблемы существуют и при открытых сетевых подключениях.

Другая проблема: потоки совместно используют многие структуры данных

- Что происходит в том случае, если один поток закрывает файл в тот момент, когда другой поток еще не считал с него данные?
- Предположим, что один поток заметил дефицит свободной памяти и приступил к выделению дополнительного объема. На полпути происходит переключение потоков, и новый поток тоже замечает дефицит свободной памяти и приступает к выделению дополнительного объема.
- Вполне возможно, что дополнительная память будет выделена дважды.
- Для решения этих проблем следует приложить ряд усилий, но для корректной работы многопоточных программ требуется все тщательно продумать и спроектировать.

стандарт IEEE standard 1003.1c, пакет Pthreads

Вызовы, связанные с потоком	Описание
pthread_create	Создание нового потока
pthread_exit	Завершение работы вызвавшего потока
pthread_join	Ожидание выхода из указанного потока
pthread_yield	Освобождение центрального процессора, позволяющее выполняться другому потоку
pthread_attr_init	Создание и инициализация структуры атрибутов потока
pthread_attr_destroy	Удаление структуры атрибутов потока