Процесс с точки зрения ОС

С точки зрения операционной системы на таком высоком абстрактном уровне у процесса есть два главных аспекта:

- Есть аспект **владения процесса** – тут речь идет о ресурсах, которые в данный момент доступны процессу. Процесс без доступа к ресурсам не имеет смысла. Он как минимум должен иметь доступ в какой-то момент к ресурсам центрального процессора, чтобы выполнять на нем собственные операция. Но чаще всего процесс имеет доступ к различным устройствам, чаще всего он, например, выводит что-нибудь на экран или читает что-нибудь с диска, или записывает что-нибудь в память и так далее. И, как мы знаем, операционная система является неким модератором: она управляет распределением этих ресурсов, она дает доступ и отбирает доступ. Поэтому, с точки зрения операционной системы, процесс – это некоторый агент, который постоянно запрашивает разные ресурсы и эти ресурсы нужно ему предоставить в таком количестве, чтобы этот процесс мог успешно выполняться.
- Другой аспект – это **планирование и исполнение**. Это всё, что касается запуска процесса, его выполнения, его завершения, переключения между процессами и так далее. Такие детали как, например, приоритеты – тоже относятся к этому аспекту процесса.

Операционная система, как правило, относится к этим двум аспектам независимо друг от друга. Есть аспект владения, есть аспект ресурсов, есть аспект планирования – и они не должны, в большинстве случаев, влиять друг на друга, а решения, которые принимает операционная система – должны опираться на какой-то один изолированный набор параметров.

Когда мы говорим о выполнении (о втором аспекте процессов), то именно в этом аспекте имеет значение понятия треда или потока. **Thread** (*поток*, еще один из переводов этого термина – это *нить*. Если весь процесс – это ткань, то ткань состоит из большого количества этих нитей, и в этом смысле аналогия похожа, но чаще всего в неформальной, а иногда и в формальной речи в русском языке применяется понятие *тред*.) – **это элемент выполнения процесса**. Когда мы говорим о процессе и имеем в виду его аспект владения, то мы обычно называем это – задачей (поэтому во многих операционных системах есть так называемый «Task manager» или «Менеджер задач», в котором мы можем увидеть все процессы, запущенные в данный момент в системе, и какие и в каком количестве эти процессы используют ресурсы компьютера, например, сколько какой-то процесс занимает времени центрального процессора в процентах или сколько памяти съедает какой-нибудь браузер). В этом плане мы видим эти процессы как задачи, которые исполняются в данный момент в операционной системе. То есть **task** (задача) – **это элемент владения**.

Тред (поток выполнения)

Тред – это поток выполнения внутри процесса, то есть, грубо говоря – это процесс внутри процесса. Естественно, он не такой же как сам основной процесс, иначе мы бы не называли его иным именем, но во многом он похож на процесс. **Это наименьшая единица обработки с точки зрения операционной системы.**

Многопоточность

Когда мы говорим о многопоточности (или multithreading на английском), то мы имеем ввиду **способность платформы или какого-то приложения, или виртуальной машины запускать несколько параллельных потоков (тредов) в рамках какого-то процесса**.

На рисунке 1 можно увидеть четыре возможных подхода к созданию и поддержке тредов в операционной системе.

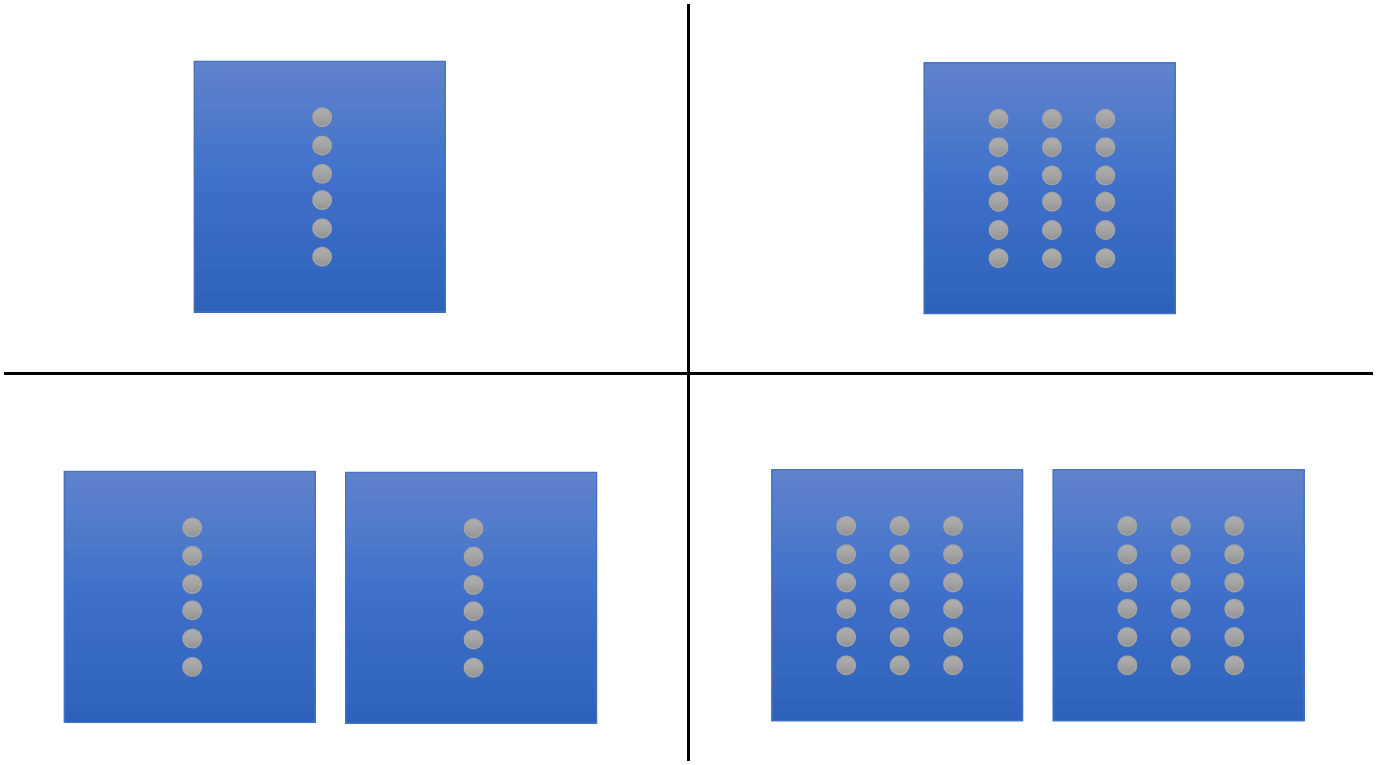

Рисунок 1. Четыре подхода к созданию и поддержке тредов

В верхнем левом углу в прямоугольнике мы видим синий квадрат. Этот синий квадрат означает процесс, а линия, состоящий из точек – тред. В этой ситуации у нас есть один процесс и один тред (в данном случае это операционная система, которая поддерживает лишь один процесс в любой момент времени и этот процесс может иметь один тред). Процесса без треда быть не может (как минимум один тред есть в любом процессе, то есть его основной путь выполнения). Примером такой операционной системы может быть старая версия MS-DOS, в которой можно было запустить только одну программу, а если вы хотите запустить другую, то нужно было закрыть текущую и эта текущая программа, запущенная в данный момент, имела только один тред исполнения. Это была по-настоящему однозадачная операционная система.

Пример в верхнем правом углу: как мы видим у нас также есть один процесс, но в этом процессе есть несколько тредов (здесь их три, но важно, что их больше, чем один). Это архитектура чуть лучше, у нас всё еще только один процесс, но зато внутри этого процесса существуют подпроцессы, которые выполняются виртуально одновременно. Но, как мы помним, процессы на обычной классической машине не могут выполняться по-настоящему параллельно (есть лишь иллюзия параллельности из-за того, что процессор очень умно переключается между процессами и делает это очень быстро). С тредами абсолютно та же история. Треды не нарушают никаких законов физики, они также не могут исполняться абсолютно одновременно на одном процессоре, но процессор может между ними переключаться также, как он переключается между целыми процессами. Примером такой операционной системы служат некоторые старые версии Unix. Они могли поддерживать один процесс, но несколько потоков внутри него.

Подход, изображенный в нижнем левом углу – это несколько процессов и каждый процесс имеет один тред.

Ну и то, что мы видим сегодня чаще всего в современных популярных операционных системах – это то, что изображено в нижнем правом углу. Несколько процессов, каждый процесс имеет несколько тредов. Это то, что мы хотим видеть, потому что это наиболее эффективная конфигурация, которая позволяет существовать нескольким сложным программам (они могут запускаться одновременно, они могут делать очень сложные вещи). Допустим, что у нас сейчас запущен процесс, который называется – браузер, а в нем несколько тредов и один из этих тредов может воспроизводить видео, какой-то другой тред может загружает какой-то другой сайт, третий тред что-то еще делает.

Потоки в процессе

Когда мы говорим о потоках внутри процесса или тредах внутри процесса, то чаще всего мы видим всё то же самое, что можно увидеть в процессах. У потока есть **состояние (state)**, потому что это, грубо говоря, процесс и в какой-то момент он будет приостановлен (если ему не повезет закончить свою работу за один раз). Это означает, что в какой-то момент к нему нужно будет вернуться, его нужно будет перезапустить, его нужно будет восстановить, а для этого нужно иметь его состояние. Также в нем должен быть какой-то **контекст, стандартные стеки выполнения, локальные данные, доступ к ресурсам процесса,** и доступ к ресурсу процесса – это одно из отличий.