**Многопото́чность** — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких *потоков*, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

Такие *потоки* называют также *потоками выполнения* (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) *[thread of execution](https://en.wikipedia.org/wiki/thread_(computer_science))*); иногда называют «нитями» (буквальный перевод [англ.](https://ru.wikipedia.org/wiki/Английский_язык) *thread*) или неформально «тредами».

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

К достоинствам многопоточности в программировании можно отнести следующее:

* Упрощение программы в некоторых случаях за счет использования общего адресного пространства.
* Меньшие относительно процесса временны́е затраты на создание потока.
* Повышение производительности процесса за счет распараллеливания процессорных вычислений и операций ввода-вывода.
* Поток в пространстве пользователя. Каждый процесс имеет таблицу потоков, аналогичную таблице процессов ядра.

**Состояния потока**

ОС выполняет планирование потоков, принимая во внимание их состояние. В мультипрограммной системе поток может находиться в одном из трех основных состояний:

* выполнение — активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
* ожидание — пассивное состояние потока, находясь в котором, поток заблокирован по своим внутренним причинам (ждет осуществления некоторого события, например завершения операции ввода-вывода, получения сообщения от другого потока или освобождения какого-либо необходимого ему ресурса);
* готовность — также пассивное состояние потока, но в этом случае поток заблокирован в связи с внешним по отношению к нему обстоятельством (имеет все требуемые для него ресурсы, готов выполняться, однако процессор занят выполнением другого потока).

**Создание и завершение потока**

Потоки подобно процессам работают с идентификаторами. Только эти иден-

тификаторы существуют локально в рамках текущего процесса. Для их хра-

нения предусмотрен специальный тип pthread\_t, который становится доступным при включении в программу заголовочного файла { XE "Заголовочный файл:pthread.h" } pthread.h.

Для создания потока и запуска потоковой функции используется функция

pthread\_create(), объявленная в заголовочном файле pthread.h следующим

образом:

int pthread\_create (pthread\_t \* THREAD\_ID, void \* ATTR,

void \*(\*THREAD\_FUNC) (void\*), void \* ARG);

Итак, pthread\_create() принимает четыре аргумента:

* По адресу в TTHREAD\_IDT помещается идентификатор нового потока (если таковой был создан).
* Безтиповый указатель TATTRT служит для указания атрибутов потока{ XE "Поток:атрибуты" }. Если этот аргумент равен NULL, то поток создается с атрибутами по умолчанию.
* Пугающий своим видом аргумент TPTHREAD\_FUNCT является указателем на потоковую функцию. Это обычная функция, возвращающая бестиповый указатель (void\*) и принимающая бестиповый указатель в качестве единственного аргумента.
* Аргумент ARG — это бестиповый указатель, содержащий аргументы потока. Если потоковая функция не требует наличия аргументов, то в качестве ARGT можно указать NULL

Чтобы лучше понять концепцию потоков в Linux, нужно представить себе,

что программа при запуске создает один поток, а функция pthread\_create()

позволяет создавать дополнительные потоки. Это означает, что основную

программу следует трактовать как "родительский поток".

Следует также понимать, что если один из потоков завершает программу, то

все остальные потоки тут же завершаются. Это еще одно важное отличие по-

токов от процессов.

**Завершение потока**

Известно, что программа обычно завершается посредством возврата из функ-

ции main() или через вызов exit(). Аналогичным образом работают и потоки,

которые могут завершаться возвратом из потоковой функции или вызовом фунции pthread\_exit().