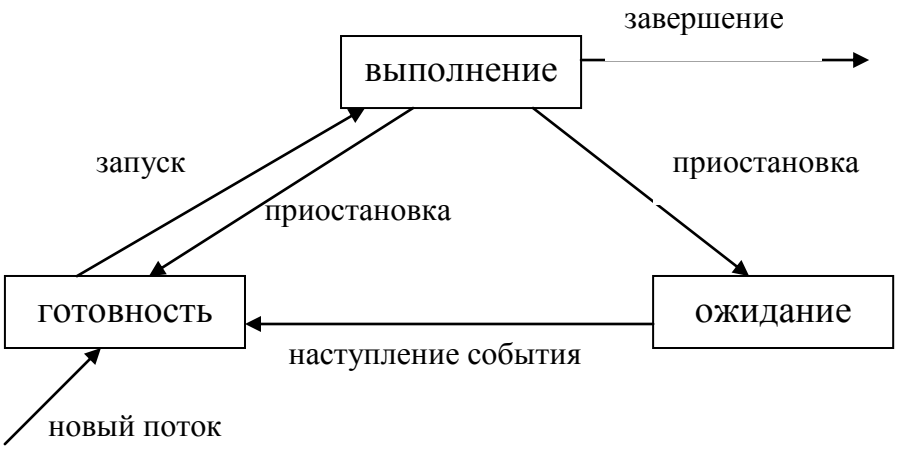
4. Состояния потоков и планирование их выполнения

Для каждого созданного потока в системе предусматриваются **три** возможных его состояния:

- состояние **выполнения,** когда код потока выполняется процессором; на однопроцессорных платформах в этом состоянии в каждый момент времени может находиться только один поток;
- состояние **готовности** к выполнению, когда поток готов продолжать свою работу и ждет освобождения ЦП;
- состояние **ожидания** наступления некоторого события; в этом случае поток **не** претендует на время ЦП, пока не наступит определенное событие (завершение операции ввода/вывода, освобождение необходимого потоку занятого ресурса, сигнала от другого потока); часто такие потоки называют **блокированными.**

Изменение состояния потока происходит в результате соответствующих действий. Удобно для этих целей использовать следующую **диаграмму** состояний и переходов.



Переходы между состояниями можно описать следующим образом:

- «готовность» → «выполнение»: система в соответствии с алгоритмом планирования выбирает для выполнения текущий поток, предоставляя ему ЦП

- «выполнение» → «готовность»: поток готов продолжать свою работу, но система принимает решение **прервать** его выполнение; чаще всего это происходит по следующим двум причинам:
    - завершается выделенное потоку **время владения** процессором;
    - в числе готовых к выполнению появляется **более приоритетный** поток по сравнению с текущим;
- «выполнение» → «ожидание»: дальнейшее исполнение кода текущего активного потока невозможно без наступления **некоторого события,** и поэтому активный поток прерывает свое выполнение и переводится системой в состояние ожидания (блокируется);
- «ожиданием» → «готовность»: в системе происходит некоторое событие, наступление которого ожидает один из блокированных потоков, и поэтому система переводит этот поток в состояние готовности (разблокирует), после чего он будет учитываться системой при планировании порядка предоставления ЦП;
- наконец, поток может нормально или аварийно **завершить** свое выполнение, после чего система удаляет его дескриптор из своей внутренней структуры, и тем самым поток перестает существовать.

В состояниях готовности и ожидания может находиться **несколько** потоков, поэтому система создает для хранения их дескрипторов отдельные **списковые** структуры. Организация этих списков зависит от тех принципов, которые положены в основу планирования потоков для данной ОС.

Цель планирования потоков вполне очевидна - определение **порядка выполнения** потоков в условиях внешней или внутренней многозадачности. Однако способы достижения этой цели существенно зависят от типа ОС. Рассмотрим сначала принципы планирования для универсальных ОС. Для таких ОС нельзя заранее предсказать, сколько и какие потоки будут запущены в каждый момент времени и в каких состояниях они будут находиться. Поэтому планирование должно выполняться **динамически** на

основе сбора и анализа информации о **текущем состоянии** вычислительной системы.

Для этого в состав ОС включается **модуль-планировщик,** реализующий выбранные алгоритмы планирования. Поскольку этот модуль представляет собой программный код, то для решения своих задач планировщик должен на некоторое время забирать ЦП. Отсюда следует, что алгоритмы планирования должны быть **максимально простыми,** иначе возникает опасность, что система будет тратить недопустимо большое время на решение своих внутренних задач, а на выполнение прикладных программ времени не останется.

Кроме вычислительной простоты, алгоритмы планирования должны обладать следующими общими свойствами:

- обеспечение максимально возможной загрузки ЦП;
- обеспечение равномерной загрузки ресурсов вычислительной системы;
- обеспечение справедливого обслуживания всех процессов и потоков;
- минимизация времени отклика для интерактивных процессов.

За время существования ОС было предложено и реализовано несколько принципов управления потоками. В настоящее время большинство универсальных ОС используют метод **вытесняющей многозадачности** (preemptive multitasking), который тоже имеет несколько разновидностей. В основе метода лежат два важнейших и достаточно понятных принципа:

- **квантование времени ЦП;**
- **приоритеты потоков.**

Квантование означает, что каждому потоку система выделяет определенный интервал времени (квант), в течение которого процессор потенциально может выполнять код этого потока. По завершении выделенного кванта планировщик принудительно переключает процессор на выполнение другого готового потока (если, конечно, такой есть), переводя старый активный поток в состояние готовности. Это гарантирует, что ни один поток **не захватит** ЦП на непозволительно большое время (как было в

более ранних системах с так называемой невытесняющей или **кооперативной** многозадачностью). Конечно, выделенный квант поток может и не использовать до конца, если в процессе своего выполнения он нормально или аварийно завершится, или потребует наступления некоторого события, или будет прерван системой.

Для эффективной работы ОС большое значение имеет выбор величины кванта. Очень маленькие значения кванта приводят к частым переключениям ЦП, что повышает непроизводительные расходы из-за необходимости постоянного сохранения контекста прерываемого потока и загрузки контекста активизируемого потока. Наоборот, большие значения кванта уменьшают иллюзию одновременного выполнения нескольких приложений. Некоторые планировщики умеют изменять кванты в определенных пределах, увеличивая их для тех потоков, которые не используют до конца выделенное время, например, из-за частых обращений к операциям ввода/вывода. Типичный диапазон изменения кванта – от 10 до 50 миллисекунд. При этом необходимо учитывать все возрастающие скорости работы современных процессоров: за 10 миллисекунд (т.е. за 1/100 секунды) процессор успеет выполнить около 10 млн. элементарных команд.

Можно связать величину кванта с приоритетом потока. Приоритет определяет **важность** потока и влияет на **частоту** запуска потока и, возможно, на величину выделяемого кванта. Интуитивно понятно, что потоки могут иметь разную степень важности: системные – более высокую (иначе ОС не сможет решать свои задачи), прикладные – менее высокую. Многие ОС позволяют группировать потоки по их важности, выделяя **три группы,** или класса:

- потоки реального времени с максимально высоким уровнем приоритета;
- системные потоки с меньшим уровнем приоритета;
- прикладные потоки с самым низким приоритетом.

Внутри каждой группы выделяется свой диапазон возможных значений приоритетов, причем эти диапазоны между собой **не пересекаются,** т.е. максимально возможный приоритет прикладного потока всегда будет строго меньше минимально возможного приоритета для системных потоков. Внутри каждой группы могут использоваться разные алгоритмы управления приоритетами.

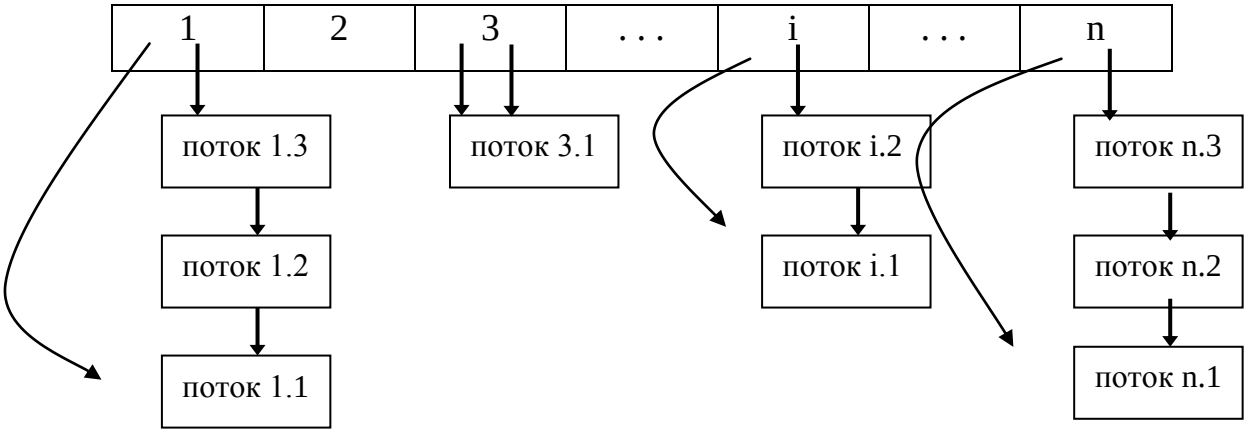
Если приоритет потока может меняться системой, то такие приоритеты называют **динамическими,** иначе – **фиксированными.** Конечно, реализация фиксированных приоритетов гораздо проще, тогда как динамические приоритеты позволяют реализовать более справедливое распределение процессорного времени. Например, потоки, интенсивно использующие внешние устройства, очень часто блокируются до завершения выделенного кванта времени, т.е. **не используют** эти кванты полностью. Справедливо при разблокировании таких потоков дать им **более высокий** приоритет для быстрой активации, что обеспечивает б**о**льшую загрузку относительно медленных внешних устройств. С другой стороны, если поток **полностью** расходует выделенный квант, система может после его приостановки **уменьшить** приоритет. Тем самым, более **высокие** приоритеты получают более **короткие** потоки, быстро освобождающие процессор, и следовательно, достигается более равномерная загрузка вычислительной системы в целом.

Довольно интересной и часто используемой разновидностью приоритетов являются так называемые **абсолютные** приоритеты: как только среди готовых потоков появляется поток, приоритет которого **выше,** чем приоритет текущего активного потока, этот активный поток **досрочно** прерывается с передачей процессора более приоритетному потоку.

Для реализации приоритетного обслуживания ОС должна создавать и поддерживать **набор приоритетных очередей.** Для каждого возможного значения приоритета создается своя очередь, в которую потоки (в виде своих дескрипторов) помещаются строго в соответствии с очередностью. Планировщик просматривает эти очереди по порядку следования

приоритетов и выбирает для выполнения первый поток в самой приоритетной непустой очереди. Отсюда следует, что потоки с меньшими приоритетами будут выполняться, только если пусты все более приоритетные очереди. Если допускается изменение приоритета, то планировщик должен уметь перемещать поток в другую очередь в соответствии с новым значением приоритета.

Схематично массив приоритетных очередей представлен на следующем рисунке, где для удобства более приоритетные потоки собраны в левой части массива, менее приоритетные – в правой, а сами приоритеты изменяются от 1 (максимум) до n (минимум). Условное обозначение «поток i.2» показывает, что данный поток имеет приоритет i и стоит вторым по порядку в своей очереди.



Для изменения приоритета и, возможно, кванта времени планировщику необходима следующая информация: базовая величина приоритета и кванта, время ожидания в очереди, накопленное время выполнения, интенсивность обращения к операциям ввода/вывода. Вся эта информация должна сохраняться в соответствующих структурах данных.

В итоге, планировщик включается в работу при возникновении одного из следующих событий:

- завершение кванта времени для текущего активного потока (сигнал от системного таймера);
- нормальное завершение кода текущего активного потока;
- аварийное завершение кода текущего активного потока;

- запрос активным потоком занятого системного ресурса;
- появление среди готовых потоков более приоритетного потока.

При этом запускается код планировщика, который просматривает приоритетные очереди и выбирает наиболее приоритетный поток. После этого происходит собственно само **переключение потоков:**

- формируется контекст прерываемого потока;
- с помощью контекста вновь активизируемого потока восстанавливается необходимое состояние вычислительной системы, в частности, загружаются необходимые значения во все регистры процессора;
- поскольку в регистр-счетчик команд из контекста заносится адрес очередной подлежащей выполнению команды активизируемого потока, то процессор переходит к выполнению кода нового потока точно с того места, где оно было прервано.

Планирование потоков в системах реального времени строится на других принципах. Поскольку для подобных систем наиболее важным показателем является **скорость работы,** то планирование выполняется **статически.** Для этого заранее строится так называемая **таблица переключений,** с помощью которой в зависимости от текущего состояния вычислительного процесса быстро и однозначно определяется запускаемый в данный момент поток.