ОПЕРАЦИОННЫЕ СИСТЕМЫ И СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Лекция № 05 — Кооперация процессов

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2024

2023.03.07

Оглавление

Некоторые важные термины, понятия и дополнения

Заблокированный процесс (или поток) [Blocked Process (or Thread)]

Процесс (или поток), ожидающий выполнения какого-либо условия (кроме доступности процессора), прежде чем он сможет продолжить выполнение.

ID родительского процесса (Parent Process ID)

Атрибут нового процесса, идентифицирующий родителя процесса. Идентификатор родительского процесса для процесса — это идентификатор процесса его создателя на время существования создателя. По истечении срока жизни создателя идентификатор родительского процесса является идентификатором процесса, определяемого реализацией системного процесса.

Дочерний процесс/процесс-потомок (Child Process)

Новый процесс, созданный (с помощью **`fork()`**, **`posix_spawn()`** или **`posix_spawnp()`**) данным процессом.

Дочерний процесс остается дочерним по отношению к процессу создания, пока существуют оба процесса.

`posix_spawn()` и **`posix_spawnp()`** используются для создания новых дочерних процессов, которые выполняют указываемый файл. Эти функции были определены в POSIX для стандартизации метода создания новых процессов на машинах, у которых нет возможности поддержки системного вызова **`fork(2)`**. К таким машинах, обычно, относятся встраиваемые системы без поддержки MMU.

Контролирующий процесс/Управляющий процесс (Controlling Process)

Лидер сеанса, установивший соединение с управляющим терминалом.

Если терминал впоследствии перестает быть управляющим терминалом для этого сеанса, лидер сеанса перестает быть контролирующим процессом.

Контролирующий терминал/ Управляющий терминал (Controlling Terminal)

Терминал, связанный с сеансом. Каждый сеанс может иметь не более одного связанного с ним управляющего терминала, а управляющий терминал связан ровно с одним сеансом.

Определенные входные последовательности от управляющего терминала вызывают отправку сигналов всем процессам в группе процессов переднего плана, связанной с управляющим терминалом.

Процессорное время (время выполнения) [CPU Time (Execution Time)]

Время, затраченное на выполнение процесса или потока, включая время, затраченное на выполнение системных служб от имени этого процесса или потока. Если поддерживается опция Threads, то значение часов процессорного времени для процесса определяется реализацией.

С этим определением сумма всех времен выполнения всех потоков в процессе даже в однопоточном процессе может не равняться времени выполнения процесса, потому что реализации могут отличаться в том, как они учитывают время во время переключения контекста или по другим причинам.

Действующий идентификатор группы [Effective Group ID]

Атрибут процесса, который используется при определении различных разрешений, в том числе разрешений на доступ к файлам.

Действующий идентификатор пользователя [Effective User ID]

Атрибут процесса, который используется при определении различных разрешений, в том числе разрешений на доступ к файлам.

Пустой каталог [Empty Directory]

Каталог, который содержит записи каталога не более, чем точка и точка-точка, а также имеет ровно одну ссылку на него (кроме собственной записи точка, если она существует) в каталоге точка-точка. Других ссылок на каталог быть не может.

Считается ли корневой каталог пустым, не указывается.

Исполняемый файл (Executable File)

Обычный файл, приемлемый в качестве нового файла образа процесса эквивалентом семейства функций **exec( )** и, таким образом, используемый как одна из форм служебной программы.

Исполняемые файлы могут создаваться с помощью стандартных утилит-компиляторов, но также могут быть предоставлены другие неуказанные методы создания исполняемых файлов.

Внутренний формат исполняемого файла не указывается, но соответствующее приложение не может предполагать, что исполняемый файл является текстовым файлом.

Специальный файл FIFO (или FIFO) [FIFO Special File (or FIFO)]

Тип файла со свойством чтения данных, записанных в такой файл, в порядке очереди записи.

Труба/пайп/канал (Pipe)

Объект, идентичный FIFO, но не имеющий ссылок в файловой иерархии.

Файл (File)

Объект, в который можно писать или читать, или и то, и другое. Файл имеет определенные атрибуты, включая права доступа и тип. Типы файлов включают обычный файл, специальный символьный файл, специальный файл блока, специальный файл FIFO, символическую ссылку, сокет и каталог. Реализация может поддерживать другие типы файлов.

Завершение процесса (Process Termination)

Есть два типа завершения процесса:

1. Нормальное завершение происходит возвратом из **`main()`** по запросу с помощью функций **`exit()`**, **`_exit()`** или **`_Exit()`** или когда последний поток в процессе завершается возвратом из своей функции запуска, вызовом функции **`pthread_exit()`** или отменой.

2. Ненормальное завершение происходит по запросу функции **`abort()`** или при получении некоторых сигналов.

Дескриптор файла (File Descriptor)

Уникальное неотрицательное целое число для каждого процесса, используемое для идентификации открытого файла с целью доступа к нему. Значение вновь созданного файлового дескриптора составляет от нуля до `OPEN_MAX - 1`. Дескриптор файла может иметь значение больше или равное `OPEN_MAX`, если значение `OPEN_MAX` уменьшилось (**`sysconf()`**) с момента открытия дескриптора файла. Дескрипторы файлов также могут использоваться для реализации дескрипторов каталога сообщений и потоков каталогов.

`OPEN_MAX` подробно определен в `<limits.h>`.

Порядок байтов хоста (Host Byte Order)

Расположение байтов в любом целочисленном типе при использовании определенной архитектуры машины.

Два распространенных метода порядка байтов — это big-endian и little-endian.

Big-endian — это формат для хранения двоичных данных, в котором старший байт размещается первым, а остальные в порядке убывания.

Little-endian — это формат для хранения или передачи двоичных данных, в котором младший байт размещается первым, а остальные в порядке возрастания.

Описание открытого файла (Open File Description)

Запись, содержащая информацию о том, каким образом процесс или группа процессов обращается к файлу. Каждый дескриптор файла относится ровно к одному описанию открытого файла, но на описание открытого файла может ссылаться более одного дескриптора файла. Атрибутами описания открытого файла являются смещение файла, состояние файла и режимы доступа к нему.

Группа осиротевших процессов (Orphaned Process Group)

Группа процессов, в которой родитель каждого члена либо сам является членом группы, либо не является членом сеанса группы.

Живой процесс (Live Process)

Адресное пространство с одним или несколькими потоками, выполняющимися в этом адресном пространстве, и необходимые системные ресурсы для этих потоков.

Многие системные ресурсы, определенные в POSIX.1-2017, используются всеми потоками внутри процесса. К ним относятся:

- идентификатор процесса;
- идентификатор родительского процесса;
- идентификатор группы процессов;
- членство в сеансе;
- реальный идентификатор пользователя;
- эффективный (действующий) идентификатор пользователя;
- сохраненный идентификатор пользователя;
- реальный идентификатор группы;
- эффективный (действующий) идентификатор группы;
- сохраненный идентификатор группы;
- идентификаторы дополнительных групп;
- текущий рабочий каталог;
- корневой каталог;
- маска создания файлового режима;
- файловые дескрипторы.

Кооперация процессов

Жизнь процессов в вычислительной системе состоит из постоянного ожидания в очереди к процессору и в постоянной борьбе за другие ресурсы.

Для нормального функционирования процессов ОС старается максимально изолировать их друг от друга:

1) каждый процесс имеет свое собственное адресное пространство, нарушение которого, как правило, приводит к аварийной остановке процесса (исключение).

2) каждому процессу, по возможности, предоставляются свои собственные дополнительные ресурсы.

Тем не менее, для решения некоторых задач процессы имеют необходимость объединять свои усилия.

Взаимодействующие процессы

Для достижения поставленной цели различные процессы (возможно, принадлежащие разным пользователям) могут исполняться ***псевдопараллельно*** на одной ВС или параллельно на разных ВС, взаимодействуя между собой.

Причины для их кооперации

1) **Повышение скорости работы**. Когда один процесс ожидает наступления некоторого события (например, окончания операции ввода-вывода), другие в это время могут заниматься полезной работой, направленной на решение общей задачи. В многопроцессорных вычислительных системах программа разделяется на отдельные кусочки, каждый из которых будет исполняться на своем процессоре.

2) **Совместное использование данных**. Различные процессы могут, к примеру, работать с одной и той же динамической базой данных или с совместно используемым файлом, изменяя их содержимое.

3) **Модульная конструкция какой-либо системы**. Типичным примером может служить микроядерный способ построения операционной системы, когда ее различные части представляют собой отдельные процессы, общающиеся путем передачи сообщений через микроядро.

4) **Просто для удобства**. Пользователь может желать, например, редактировать и отлаживать программу одновременно. В этой ситуации процессы редактора и отладчика должны уметь взаимодействовать друг с другом.

Процессы не могут взаимодействовать не общаясь

Общение процессов обычно приводит к изменению их поведения в зависимости от полученной информации.

Если деятельность процессов не меняется при получении ими любой информации, это означает, что они на самом деле не нуждаются во взаимном общении.

Процессы, которые влияют на поведение друг друга путем обмена информацией, принято называть кооперативными или взаимодействующими процессами, в отличие от независимых процессов, которые оказывают друг на друга никакого воздействия и ничего не знают о взаимном сосуществовании.

Различные процессы в вычислительной системе изначально представляют собой обособленные сущности — работа одного процесса не должна приводить к нарушению работы другого процесса. Для этого, в частности, и разделены их адресные пространства и системные ресурсы.

Поэтому для обеспечения корректного взаимодействия процессов требуются ***специальные средства и действия операционной системы*** – нельзя просто поместить значение, вычисленное в одном процессе, в область памяти, соответствующую переменной в другом процессе, не предприняв каких-либо дополнительных организационных усилий.

Категории средств обмена информацией

Процессы могут взаимодействовать друг с другом только обмениваясь информацией.

По объему передаваемой информации и степени возможного воздействия на поведение другого процесса все средства такого обмена можно разделить на три категории:

1) сигнальные;
2) канальные;
3) совместно используемая память.

Сигнальные (signal)

Передается минимальное количество информации — один бит, «да» или «нет».

Используются, как правило, для извещения процесса о наступлении какого-либо события.

Степень воздействия на поведение процесса , получившего информацию, минимальна — все зависит от того, ***знает ли он, что означает полученный сигнал***, надо ли на него реагировать и каким образом. Неправильная реакция на сигнал или его игнорирование могут привести к трагическим последствиям.

Канальные (pipe)

Общение процессов происходит через коммуникационные линии, предоставляемые операционной системой.

Объем передаваемой информации в единицу времени ограничен пропускной способностью линий связи.

С увеличением количества информации увеличивается и возможность влияния на поведение другого процесса.

Совместно используемая память (shared memory)

Два или более процессов могут совместно использовать некоторую область адресного пространства.

Созданием совместно используемой памяти занимается операционная система по запросу процесса.

Возможность обмена информацией максимальна, как, и влияние на поведение другого процесса, но требует «повышенной осторожности».

Использование совместно используемой памяти для передачи/получения информации осуществляется с помощью средств обычных языков программирования, в то время как сигнальным и канальным средствам коммуникации для этого необходимы специальные системные вызовы.

Совместно используемая память представляет собой наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

Вопросы логической реализации

При рассмотрении любого из средств коммуникации нас прежде всего интересует не их физическая реализация (общая шина данных, прерывания, аппаратно совместно используемая память и т. д.), а логическая, определяющая в конечном счете механизм их использования.

Некоторые важные вопросы логической реализации являются общими для всех категорий средств связи, некоторые относятся к отдельным категориям:

1) установление связи;

2) адресация;

3) информационная валентность процессов и средств связи;

4) направленность связи.

1) Установление связи

Можно ли использовать средство связи непосредственно для обмена информацией сразу после создания процесса или первоначально необходимо предпринять некоторые действия по инициализации обмена?

Сигнальные – **для передачи сигнала от одного процесса к другому инициализация не нужна.**

Канальные – передача информации по линиям связи может потребовать первоначального резервирования такой линии для процессов, желающих обменяться информацией.

Общая память – для использования совместно используемой памяти различными процессами потребуется специальное обращение к операционной системе, которая выделит требуемую область адресного пространства.

2) Адресация

Если передается некоторая информация, то необходимо указать, куда ее следует передать. Если необходимо получить некоторую информацию, то нужно знать, откуда ее можно получить.

Различают два способа адресации: ***прямую*** и ***непрямую***.

Прямая адресация

В случае прямой адресации взаимодействующие процессы непосредственно общаются друг с другом, при каждой операции обмена данными явно указывая имя или ID процесса, которому информация предназначена или от которого она должна быть получена.

Если оба процесса, передающий и принимающий, указывают «имена» своих партнеров по взаимодействию, то такая схема адресации называется ***симметричной прямой адресацией***.

Симметричный прямой обмен является изолированным — ни один другой процесс не может вмешаться в процедуру симметричного прямого общения двух процессов, перехватить посланные или подменить ожидаемые данные.

Если только один из взаимодействующих процессов, например передающий, указывает имя своего партнера по кооперации, а второй процесс в качестве возможного партнера рассматривает любой процесс, например, ожидает получения информации от произвольного источника, то такая схема адресации называется ***асимметричной прямой адресацией***.

Непрямая адресация

При непрямой адресации данные помещаются передающим процессом в некоторый ***промежуточный объект*** для хранения данных.

Этот объект имеет свой адрес, поэтому данные из него могут быть впоследствии изъяты каким-либо другим процессом.

При этом передающий процесс не знает, как именно идентифицируется процесс, который получит информацию, а принимающий процесс не имеет представления об идентификаторе процесса, от которого он должен или может ее получить.

Информация, которой должен обладать процесс для взаимодействия с другими процессами, — это ***идентификатор промежуточного объекта*** для хранения данных, если он, конечно, не является единственным в вычислительной системе для всех процессов

При использовании прямой адресации связь между процессами в классической ОС устанавливается автоматически, без дополнительных инициализирующих действий — единственное, что нужно для использования средства связи, это знать, как идентифицируются процессы, участвующие в обмене данными.

При использовании непрямой адресации инициализация средства связи может как требоваться, так и не требоваться.

3) Информационная валентность процессов и средств связи

Следующий важный вопрос — это вопрос об информационной валентности связи. Слово валентность здесь использовано по аналогии с химией:

- сколько процессов может быть одновременно ассоциировано с конкретным средством связи?
- сколько таких средств связи может быть задействовано между двумя процессами?

При прямой адресации только одно данное средство связи может быть задействовано для обмена данными между двумя процессами, и только эти два процесса могут быть ассоциированы с ним.

При непрямой адресации может существовать более двух процессов, использующих один и тот же объект для данных, и более одного объекта может быть использовано двумя процессами.

4) Направленность связи

Является ли связь однонаправленной или двунаправленной?

При *однонаправленной* связи каждый процесс, ассоциированный с ней, может использовать средство связи либо только для приема информации, либо только для ее передачи.

При *двунаправленной* связи каждый процесс, участвующий в общении, может использовать связь как для приема, так и для передачи данных.

В коммуникационных системах принято называть однонаправленную связь ***симплексной***, двунаправленную связь с поочередной передачей информации в разных направлениях — ***полудуплексной***, а двунаправленную связь с возможностью одновременной передачи информации в разных направлениях — ***дуплексной***.

Прямая и непрямая адресация не имеют непосредственного отношения к направленности связи.

Особенности передачи информации с помощью линий связи

Передача информации между процессами посредством линий связи является достаточно безопасной по сравнению с использованием совместно используемой памяти и достаточно информативной по сравнению с сигнальными средствами коммуникации.

Кроме того, совместно используемая память не может быть использована для связи процессов, функционирующих на различных вычислительных системах. Возможно, именно поэтому каналы связи получили наибольшее распространение среди других средств коммуникации процессов.

Буферизация

Возможность линии связи сохранять информацию, переданную одним процессом, до ее получения другим процессом или помещения в промежуточный объект.

Каков объем этой информации? Иными словами, обладает ли канал связи буфером и каков объем этого буфера. Здесь можно выделить три принципиальных варианта:

1) буфер нулевой емкости или отсутствует;
2) буфер ограниченной емкости;
3) буфер неограниченной емкости.

Буфер нулевой емкости или отсутствует

Никакая информация не может сохраняться на линии связи.

В этом случае процесс, посылающий информацию, должен ожидать, пока процесс, принимающий информацию, не будет готов ее получить, прежде чем продолжить выполнение.

Буфер ограниченной емкости

Линия связи не может хранить до момента получения более чем некоторый фиксированный объем информации, который определяется размером буфера.

Если в момент передачи данных в буфере есть место, то передающий процесс не должен ничего ждать и может сразу копировать информацию в буфер.

Если же в момент передачи данных буфер заполнен или не достаточно места для помещения информации, то процесс отправителя должен быть задержан до появления в буфере свободного пространства.

Буфер неограниченной емкости.

Теоретически это возможно, но практически вряд ли реализуемо.

Процесс, посылающий информацию, никогда не ждет окончания ее передачи и приема другим процессом.

При использовании канального средства связи с непрямой адресацией под емкостью буфера обычно понимается количество информации, которое может быть помещено в промежуточный объект для хранения данных.

Поток ввода/вывода и сообщения

Существует две модели передачи данных по каналам связи — поток ввода-вывода и сообщения.

При передаче данных с помощью потоковой модели, операции передачи/приема информации вообще не интересуются содержимым данных.

Процесс, прочитавший некоторое количество данных из линии связи, например, 100 байт, не знает и не может знать, были ли они переданы одновременно, т. е. одним куском, или порциями по 20 байт, пришли они от одного процесса или от разных процессов — данные представляют собой простой поток байт, без какой-либо их интерпретации со стороны системы.

Примерами потоковых каналов связи могут служить **`pipe`** и **`FIFO`**.

pipe

Одним из наиболее простых способов передачи информации между процессами по линиям связи является передача данных через pipe (канал, труба, конвейер).

Представим себе, что в ВС есть некоторая труба, в один из концов которой процессы могут сливать информацию, а из другого принимать полученный поток.

Такой способ реализует потоковую модель ввода/вывода.

Информацией о расположении такой трубы в операционной системе обладает только один процесс — ее создавший. Этой информацией он может поделиться исключительно со своими наследниками — процессами-потомками и их потомками (см. **`fork( )`**).

Поэтому использовать `pipe` для связи между собой могут только родственные процессы, имеющие общего предка, создавшего данный канал связи.

FIFO (именованный канал)

Если процесс, создавший канал, сообщит о ее точном расположении в системе другим процессам, сделав вход и выход трубы каким-либо образом видимыми для всех остальных, например, зарегистрировав ее в операционной системе под определенным именем, получится объект, который принято называть *FIFO* или именованным каналом.

Именованный канал может использоваться для связи между любыми процессами в системе.

Сообщения

В модели сообщений процессы налагают на передаваемые данные некоторую структуру. Весь поток информации они разделяют на отдельные сообщения, вводя между данными, по крайней мере, границы сообщений (примером границ сообщений являются пробелы между словами, точки между предложениями или границы абзаца в сплошном тексте.).

К передаваемой информации также может быть присоединены указания на то, кем конкретное сообщение было послано и для кого оно предназначено.

Все сообщения могут иметь одинаковый фиксированный размер или могут быть переменной длины.

В вычислительных системах используются разнообразные средства связи для передачи сообщений:

- очереди сообщений;
- sockets (гнезда или сокеты);
- и т.д.

Как потоковые линии связи, так и каналы сообщений могут иметь или не иметь буфер.

Когда идет речь о емкости буфера для потоков данных, обычно она измерятся в байтах.

Когда идет речь о емкости буфера для сообщений, обычно она измерятся в сообщениях.

Надежность средств связи

Одним из существенных вопросов при рассмотрении всех категорий средств связи является вопрос об их надежности.

Способ коммуникации называется надежным, если при обмене данными выполняются следующие четыре условия:

- не происходит потери информации;
- не происходит повреждения информации;
- не появляется лишней информации;
- не нарушается порядок данных в процессе обмена.

Самым надежным способом связи является передача данных через совместно используемую память. То, что сохранено в разделяемой памяти, будет прочитано другими процессами в оригинальном виде, если не произойдет сбоя в питании.

Для других средств коммуникации это не всегда верно.

Способы борьбы с ненадежностью коммуникаций в ВС

Для обнаружения повреждения информации можно снабжать каждое передаваемое сообщение некоторой контрольной суммой (message digest), вычисленной по посланной информации. При приеме сообщения контрольная сумма вычисляется заново и проверяется ее соответствие пришедшему значению.

Если данные не повреждены (контрольные суммы совпадают), то подтверждается правильность их получения. Если данные повреждены (контрольные суммы не совпадают), то сообщение считается утерянным.

ECC-кодирование

Вместо контрольной суммы можно использовать специальное кодирование передавамых данных с помощью кодов, исправляющих ошибки (ECC-кодирование.)

- блоковые коды (CRC — cyclic redundancy check, коды Хемминга, код Рида-Соломона);
- сверточные коды;
- каскадное кодирование;
- итеративное кодирование (турбо-коды, коды Галлахера).

Такое кодирование позволяет при числе искажений информации, не превышающем некоторого значения, восстановить начальные неискаженные данные.

Если по прошествии некоторого интервала времени подтверждение о правильности полученной информации не придет на передающий конец линии связи, то информацию считается утерянной, и она отсылается повторно. Для того чтобы избежать двойного получения одной и той же информации, на приемном конце линии связи должен производиться соответствующий контроль.

Для гарантии правильного порядка получения сообщений их нумеруют.

При приеме сообщения с номером, не соответствующим ожидаемому, с ним поступают, как с утерянным и ожидают сообщения с правильным номером.

Данные действия могут быть возложены:

- на операционную систему;
- на процессы, обменивающиеся данными;
- совместно на систему и процессы, разделяя их ответственность.

Операционная система может обнаруживать ошибки при передаче данных и извещать об этом взаимодействующие процессы для принятия ими решения о дальнейшем поведении.

Завершение связи

Важным вопросом при изучении средств обмена данными является вопрос прекращения обмена.

Здесь выделяется два аспекта:

- требуются ли от процесса какие-либо специальные действия по прекращению использования средства коммуникации;
- влияет ли такое прекращение на поведение других процессов.

Для способов связи, которые не подразумевали никаких инициализирующих действий, обычно ничего специального для окончания взаимодействия предпринимать не надо.

Если же установление связи требовало некоторой инициализации, то, как правило, при ее завершении необходимо выполнение ряда операций, например, сообщения операционной системе об освобождении выделенного связного ресурса.

Если кооперативные процессы прекращают взаимодействие согласовано, то такое прекращение не влияет на их дальнейшее поведение.

При несогласованном окончании связи одним из процессов, если какой-либо из взаимодействующих процессов, не завершивших общение, находится в этот момент в состоянии ожидания получения данных, либо попадает в такое состояние позже, то ОС обязана предпринять некоторые действия для того, чтобы исключить вечное блокирование этого процесса. Обычно это либо прекращение работы ожидающего процесса, либо его извещение о том, что связи больше нет (например, с помощью передачи заранее определенного сигнала).

Нити исполнения (threads, потоки управления)

Рассмотренные выше стороны логической реализации относятся к средствам связи, ориентированным на организацию взаимодействия различных процессов.

Однако усилия, направленные на ускорение решения задач в рамках классических операционных систем, привели к появлению совершенно иных механизмов и к изменению самого понятия «процесс».

В свое время внедрение идеи мультипрограммирования позволило повысить пропускную способность компьютерных систем, т.е. уменьшить среднее время ожидания результатов работы процессов. Но любой отдельно взятый процесс в мультипрограммной системе никогда не может быть выполнен быстрее, чем при выполнении в однопрограммном режиме на том же вычислительном комплексе. Тем не менее, если алгоритм решения задачи обладает определенным внутренним параллелизмом, мы могли бы ускорить его работу, организовав взаимодействие нескольких процессов. Рассмотрим следующий пример. Пусть у нас есть следующая программа на псевдоязыке программирования:

```
Ввести массив a (w)
Ввести массив b (w)
a = a + b
Ввести массив c (w)
c = a + c
Вывести массив c (w)
```

При выполнении такой программы в рамках одного процесса этот процесс четырежды будет блокироваться, ожидая окончания операций ввода-вывода.

С другой стороны, алгоритм обладает внутренним параллелизмом. Вычисление суммы массивов a + b можно было бы делать параллельно с ожиданием окончания операции ввода массива c.

```
Инициировать ввод массива a
Ожидание окончания операции ввода
Инициировать ввод массива b
Ожидание окончания операции ввода
Инициировать ввод массива c
Ожидание окончания операции ввода    a = a + b
c = a + c
Инициировать вывод массива c
Ожидание окончания операции вывода
```

Такое совмещение операций по времени можно было бы реализовать, используя два взаимодействующих процесса. Для простоты можно полагать, что средством коммуникации между ними служит совместно используемая память.

Тогда наши процессы могут выглядеть следующим образом:

```
Процесс 1                          Процесс 2
Инициировать ввод массива a           Ожидание ввода массивов a и b
Ожидание окончания операции ввода
Инициировать ввод массива b
Ожидание окончания операции ввода
Инициировать ввод массива c
Ожидание окончания операции ввода     a = a + b
c = a + c
Инициировать вывод массива c
```

Ожидание окончания операции вывода

Казалось бы, получен конкретный способ ускорения решения задачи. Однако в действительности дело обстоит не так просто. Второй процесс должен быть создан, оба процесса должны сказать операционной системе, что им необходима память, которую они могли бы совместно использовать с другим процессом, ну и наконец, нельзя забывать о переключении контекста.

Поэтому реальное поведение процессов будет выглядеть примерно так.

```
Процесс 1
Создать процесс 2                                          Процесс 2
                              Переключение контекста
                                                           Выделение общей
памяти
                                                           Ожидание ввода a и
b
                              Переключение контекста
Выделение общей памяти
Инициировать ввод массива a
Ожидание окончания операции ввода
Инициировать ввод массива b
Ожидание окончания операции ввода
Инициировать ввод массива c
Ожидание окончания операции ввода
                              Переключение контекста
                                                           a = a + b
                              Переключение контекста
c = a + c
Инициировать вывод массива c
Ожидание окончания операции вывода
```

:-(

Т.е, можно не только не выиграть во времени решения задачи, но и проиграть, поскольку временные потери на создание процесса, выделение общей памяти и переключение контекста могут превысить выигрыш, полученный за счет совмещения операций.

Для того, чтобы реализовать данную идею, вводится новая абстракция внутри понятия «процесс» — нить исполнения или просто нить (***thread***).

Нити процесса совместно используют его программный код, глобальные и переменные и системные ресурсы, но каждая нить имеет свой собственный программный счетчик, свое содержимое регистров и свой собственный стек.

Теперь процесс представляется как совокупность взаимодействующих нитей и выделенных ему ресурсов.

Процесс, содержащий всего одну нить исполнения, идентичен процессу в том смысле, который мы употребляли ранее.

Для таких процессов в дальнейшем будем использовать термин «*традиционный процесс*».

Иногда нити называют облегченными (легковесными) процессами или мини-процессами, так как во многих отношениях они подобны традиционным процессам.

Нити, как и процессы, могут порождать *нити-потомки*, правда, только внутри своего процесса, и переходить из состояния в состояние.

Состояния нитей аналогичны состояниям традиционных процессов.

Из состояния рождение процесс приходит содержащим всего одну нить исполнения.

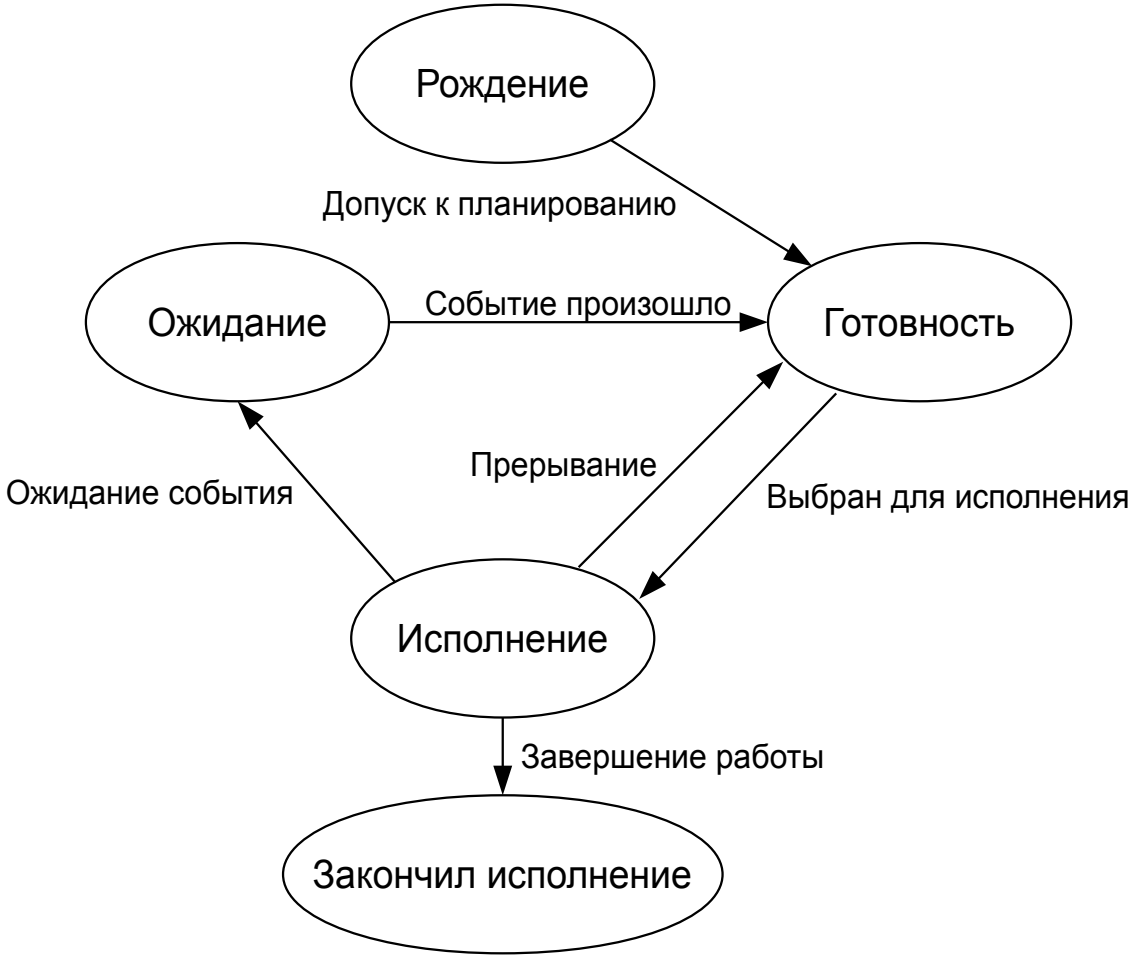
Другие нити процесса будут являться потомками этой нити.

Процесс находится в состоянии **готовность**, если хотя бы одна из его нитей находится в состоянии **готовность** и ни одна из нитей не находится в состоянии **исполнение**.

Процесс находится в состоянии **исполнение**, если одна из его нитей находится в состоянии **исполнение**.

Процесс будет находиться в состоянии **ожидание**, если *все его нити* находятся в состоянии **ожидание**.

Процесс находится в состоянии **завершил исполнение**, если все его нити находятся в состоянии завершили исполнение.



Пока одна нить процесса заблокирована, другая нить того же процесса может выполняться.

Нити совместно используют процессор таким же образом, как это делали традиционные процессы, в соответствии с используемыми в системе алгоритмами планирования.

Поскольку различные нити одного процесса совместно используют существенно больше ресурсов, чем различные процессы, то операции создания новой нити и переключения контекста между нитями одного процесса занимают существенно меньше времени, чем аналогичные операции для процессов в целом.

Предложенная ранее схема совмещения работы в терминах нитей одного процесса получает право на существование в следующем виде.

```
Нить 1
Создать нить 2                                          Нить 2
                          Переключение контекста нитей
                                                  Ожидание ввода a и b
                          Переключение контекста нитей
Инициировать ввод массива a
Ожидание окончания операции ввода
Инициировать ввод массива b
Ожидание окончания операции ввода
Инициировать ввод массива c
Ожидание окончания операции ввода
                          Переключение контекста нитей
                                                  a = a + b
                          Переключение контекста нитей
c = a + c
Инициировать вывод массива  c
Ожидание окончания операции вывода
```

Различают операционные системы, поддерживающие нити *на уровне ядра* и *на уровне библиотек*.

Все выше сказанное справедливо для операционных систем, поддерживающих нити на уровне ядра. В них планирование использования процессора происходит в терминах нитей, а управление памятью и другими системными ресурсами остается в терминах процессов.

В операционных системах, поддерживающих нити на уровне библиотек пользователей, и планирование процессора, и управление системными ресурсами осуществляется в терминах процессов.

Распределение использования процессора по нитям в рамках выделенного процессу временного интервала осуществляется средствами библиотеки.

В таких системах блокирование одной нити приводит к блокированию всего процесса, ибо ядро операционной системы ничего не знает о существовании нитей.

По сути дела, в таких вычислительных системах просто имитируется наличие нитей исполнения.

Процесс представляется как совокупность взаимодействующих нитей и выделенных ему ресурсов.

Нити могут порождать новые нити внутри своего процесса, они имеют состояния, аналогичные состояниям процесса, и могут переводиться операционной системой из одного состояния в другое.

В системах, поддерживающих нити на **уровне ядра**, планирование использования процессора осуществляется в терминах нитей исполнения, а управление остальными системными ресурсами в терминах процессов.

Накладные расходы на создание новой нити и на переключение контекста между нитями одного процесса существенно меньше, чем на те же самые действия для процессов.