Теория по ОСям для чайников от чайников

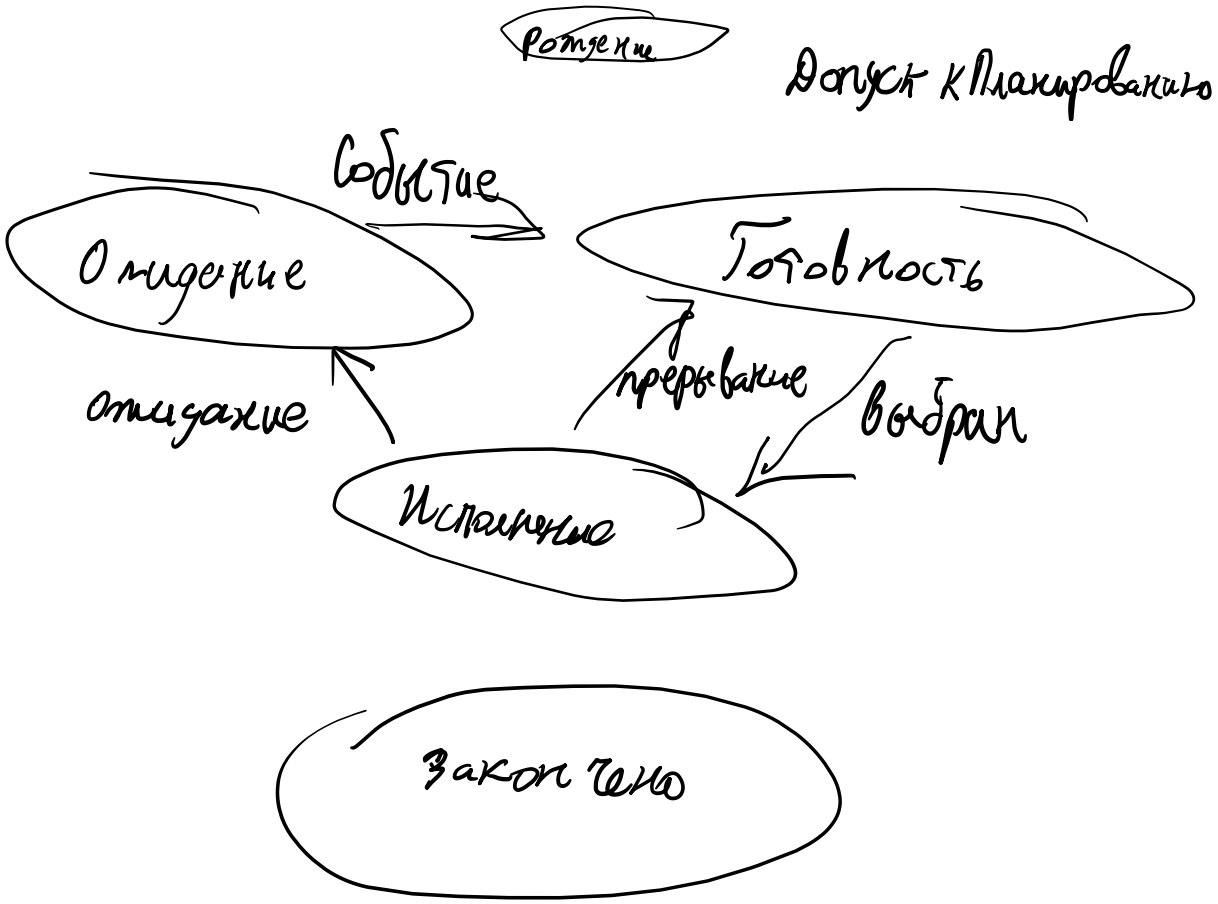
1. 3-й модуль

Термин процесс характеризует совокупность:

- набора исполняемых команд
- ассоциированных с ними ресурсов
- текущего момента выполнения

находящуюся под управлением ОС, но термины „процесс" и „программа" не эквивалентны, так как для исполнения программы может организовываться несколько процессов, в рамках одного процесса может исполняться несколько программ, в рамках процесса может исполняться код, отсутствующий в программе.

Жизненный цикл процесса можно представить в виде следующей схемы:



Каждый процесс UNIX имеет контекст, под которым понимается вся информация, требуемая для описания процесса. Эта информация сохраняется, когда выполнение процесса приостанавливается, и восстанавливается, когда планировщик предоставляет процессу вычислительные ресурсы.

Контекст процесса можно разделить на несколько основных частей:

1) PCB (Process Control Block) или системный контекст, который отвечает за состояние процесса, данные для планирования использования процессора и управления памятью, учётную информацию и сведения об устройствах ввода-вывода, связанных с процессом. Так же в состав системного контекста входит так называемый регистровый контекст, который отвечает за программный счетчик и содержимое регистров.
2) Пользовательский контекст, который отвечает за код и данные в адресном пространстве.

Операции над процессами

Однократные: создание процесса и завершение процесса

Многократные: запуск и приостановка процесса, блокирование и разблокирование процесса, изменение приоритета

Подробное описание операций:

1) Создание процесса:
    - Порождение нового PCB с состоянием процесса „рождение"
    - Присвоение идентификационного номера
    - Выделение ресурсов из ресурсов родителя или из ресурсов ОС
    - Занесение в адресное пространство кода и установка значения программного счетчика
    - Окончание заполнения PCB
    - Изменение состояния процесса на „готовность"
2) Завершение процесса:
    - Изменение состояния процесса на „закончил исполнение"
    - Освобождение ресурсов
    - Очистка соответствующих элементов в PCB
    - Сохранение в PCB информации о причинах завершения

3) Запуск процесса:

- Выбор одного из процессов, находящихся в состоянии „готовность"
- Изменение состояния выбранного процесса на „исполнение"
- Обеспечение наличия в оперативной памяти информации, необходимой для его выполнения
- Восстановление значений регистров
- Передача управления по адресу, на который указывает программный счетчик

4) Приостановка процесса:

- Автоматическое сохранение программного счетчика и части регистров (работа hardware)
- Передача управления по специальному адресу (работа hardware)
- Сохранение динамической части регистрового и системного контекстов в PCB
- Обработка прерывания
- Изменение состояния процесса на „готовность"

5) Блокирование процесса:

- Сохранение контекста процесса в PCB
- Обработка системного вызова
- Перевод процесса в состояние „ожидание"

6) Разблокирование процесса:

- Уточнение того, какое именно событие произошло
- Проверка наличия процесса, ожидавшего этого события
- Перевод ожидавшего процесса в состояние „готовность"
- Обработка произошедшего события

Кооперация процессов

Кооперативные или взаимодействующие процессы — это процессы, которые влияют на поведение друг друга путем обмена информацией.

Причины кооперации процессов:

- Повышение скорости решения задач

- Совместное использование данных
- Модульная конструкция какой-либо системы
- Для удобства работы пользователя

Категории средств взаимодействия:

- Сигнальные
- Канальные
- Разделяемая память

Сигнальные. 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.

Канальные. „Общение" процессов происходит через линии связи, предоставленные операционной системой, и напоминает общение людей по телефону, с помощью записок, писем или объявлений. Объем передаваемой информации в единицу времени ограничен пропускной способностью линий связи. С увеличением количества информации возрастает и возможность влияния на поведение другого процесса.

Разделяемая память. Два или более процессов могут совместно использовать некоторую область адресного пространства. Созданием разделяемой памяти занимается операционная система (если, конечно, ее об этом попросят). Использование разделяемой памяти для передачи/получения информации осуществляется с помощью средств обычных языков программирования, в то время как сигнальными и канальными средствами коммуникации для этого необходимы специальные системные вызовы. Разделяемая память представляет собой наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

Установка связи между процессами

Различают два способа адресации: прямую и непрямую. В случае прямой адресации взаимодействующие процессы непосредственно обмениваются друг с другом, при каждой операции обмена данными явно указывая имя или номер процесса, которому информация предназначена или от которого она должна быть получена. Если и процесс, от которого данные исходят, и процесс, принимающий данные, указывают имена своих партнеров по

взаимодействию, но такая схема адресации называется симметричной прямой адресацией. Ни один другой процесс не может вмешаться в процедуру симметричного прямого обмена 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 в системе, например ожидает получения информации от произвольного источника, то такая схема адресации называется асимметричной прямой адресацией.

При непрямой адресации данные помещаются передающим процессом в некоторый промежуточный объект для хранения данных, имеющий свой адрес, откуда они могут быть затем изъяты каким-либо другим процессом. Примером такого объекта может служить обычная доска объявлений или рекламная газета. 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 ее получить.

При использовании прямой адресации связь между процессами в классической операционной системе 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 и не требоваться. Информация, которой должен обладать процесс для взаимодействия с другими процессами, — это некий идентификатор промежуточного объекта для хранения данных, если он, конечно, не является единственным и неповторимым в вычислительной системе для всех процессов.

Направленность связи.

Под однонаправленной связью мы будем понимать связь, при которой 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 обмене, может использовать связь и для приема, 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.

Буферизация

Может ли линия связи сохранять информацию, переданную одним процессом, до ее получения другим процессом или помещения в промежуточный объект? Каков объем этой информации? Иными словами, речь идет о том, обладает ли канал связи буфером и каков объем этого буфера. Здесь можно выделить три принципиальных варианта.

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 соблаговолит ее получить, прежде чем заниматься своими дальнейшими делами (в реальности этот случай никогда не реализуется).

Буфер ограниченной емкости. Размер буфера равен n, то есть линия связи не может хранить до момента получения более чем n единиц информации. Если в момент передачи данных в буфере хватает места, то передающий процесс не должен ничего ожидать. Информация просто копируется в буфер. Если же в момент передачи данных буфер заполнен или места недостаточно, то необходимо задержать работу процесса отправителя 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 неявной адресацией под емкостью буфера обычно понимается количество информации, которое может быть помещено в промежуточный объект для хранения данных.

Поток ввода/вывода и сообщения

Существуют 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 100 байт из линии связи, не знает и не может знать, были ли они переданы одновременно, т. е. одним куском или порциями по 20 байт, приняли они от одного процесса или от разных. Данные представляют собой простой поток байтов, без какой-либо их интерпретации со стороны системы. Примерами потоковых каналов связи могут служить pipe и FIFO, описанные ниже.

Одним из наиболее простых способов передачи информации между процессами по линиям связи является передача данных через pipe (канал, трубу или, как его еще называют в литературе, конвейер). Представим себе, что у нас есть некоторая труба в вычислительной системе, в один из концов которой процессы могут "сливать" информацию, а из другого конца принимать полученный поток. Такой способ реализует потоковую модель ввода/вывода. Информацией о расположении трубы в операционной системе обладает только процесс, создавший ее. Этой информацией он может поделиться исключительно со своими наследниками — процессами-детьми и их потомками. Поэтому использовать pipe для связи между собой могут только родственные процессы, имеющие общего предка, создавшего данный канал связи.

Если разрешить процессу, создавшему трубу, сообщать о ее местонахождении 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, мы получим объект, который принято называть FIFO или именованный pipe. Именованный pipe может использоваться для организации связи между любыми процессами в системе.

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. 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) и т. д.

И потоковые линии связи, и каналы сообщений всегда имеют буфер конечной длины. Когда мы будем говорить о емкости буфера для потоков данных, мы будем измерять ее в байтах. Когда мы будем говорить о емкости буфера для сообщений, мы будем измерять ее в сообщениях.

Надежность средств связи.

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

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

Очевидно, что передача данных через разделяемую память является надежным способом связи. То, что мы сохранили в разделяемой памяти, будет считано другими процессами в первозданном виде, если, конечно, не произойдет сбоя в питании компьютера.

Нити исполнения (threads)

Нитью исполнения или просто нитью (в англоязычной литературе используется термин thread) называют абстракцию внутри понятия „процесс". Нити процесса разделяют его программный код, глобальные переменные и системные ресурсы, но каждая нить имеет собственный программный счетчик, свое содержимое регистров и свой 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. Такие процессы называются „традиционные процессы". Иногда нити называют облегченными 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 одного состояния в другое. Состояния нитей аналогичны состояниям традиционных процессов.

Планирование процессов

Планирование заданий используется в качестве долгосрочного планирования процессов. Оно отвечает за порождение новых процессов в системе, определяя ее степень мультипрограммирования, т. е. количество процессов, одновременно находящихся в ней.

Планирование использования процессора применяется в качестве краткосрочного планирования процессов. Оно проводится, к примеру, при обращении исполняющегося процесса к устройствам ввода-вывода или просто по завершении определенного интервала времени.

В некоторых вычислительных системах бывает выгодно для повышения производительности временно удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для дальнейшего выполнения. Такая процедура в англоязычной литературе получила название swapping, что можно перевести на русский язык как „перекачка", хотя в специальной литературе оно употребляется без перевода — свопинг. Когда и какой из процессов нужно перекачать на диск и вернуть обратно, решается дополнительным промежуточным уровнем планирования процессов — среднесрочным.

Для каждого уровня планирования процессов можно предложить много различных алгоритмов. Выбор конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, которых мы хотим достичь, используя планирование. К числу таких целей можно отнести следующие:

Справедливость — гарантировать каждому заданию или процессу определенную часть времени использования процессора в компьютерной системе, стараясь не допустить возникновения ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя фактически не начинал выполняться.

- Эффективность — постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов, готовых к исполнению. В реальных вычислительных системах загрузка процессора колеблется от 40 до 90%.
- Сокращение полного времени выполнения (turnaround time) — обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением.
- Сокращение времени ожидания (waiting time) — сократить время, которое проводят процессы в состоянии готовность и задания в очереди для загрузки.
- Сокращение времени отклика (response time) — минимизировать время, которое требуется процессу в интерактивных системах для ответа на запрос пользователя.

Параметры планирования

Для осуществления планирования алгоритмы должны опираться на какие-либо характеристики, или как их называют, параметры планирования. Параметры планирования существуют двух типов: статические и динамические.

К статическим параметрам вычислительной системы можно отнести предельные значения ее ресурсов (размер оперативной памяти, максимальное количество памяти на диске для осуществления свопинга, количество подключенных устройств ввода-вывода и т. п.). Динамические параметры системы описывают количество свободных ресурсов на данный момент.

Для краткосрочного планирования нам понадобится ввести еще два динамических параметра. Деятельность любого процесса можно представить как последовательность циклов

использования процессора и ожидания завершения операций ввода-вывода. Промежуток времени непрерывного использования процессора носит название CPU burst, а промежуток времени непрерывного ожидания ввода-вывода — I/O burst.

Вытесняющее и невытесняющее планирование

Невытесняющее планирование. При таком режиме планирования процесс занимает столько процессорного времени, сколько ему необходимо. При этом переключение процессов возникает только при желании самого исполняющегося процесса передать управление (для ожидания завершения операции ввода-вывода или по окончании работы). Этот метод планирования относительно прост в реализации и достаточно эффективен, так как позволяет выделить большую часть процессорного времени для работы самих процессов и до минимума сократить затраты на переключение контекста. Однако при невытесняющем планировании возникает проблема возможности полного захвата процессора одним процессом, который вследствие каких-либо причин (например, из-за ошибки в программе) зацикливается и не может передать управление другому процессу. В такой ситуации спасает только перезагрузка всей вычислительной системы.

Вытесняющее планирование. В этом режиме планирования процесс может быть приостановлен в любой момент исполнения. Операционная система устанавливает специальный таймер для генерации сигнала прерывания по истечении некоторого интервала времени — кванта. После прерывания процессор передается в распоряжение следующего процесса. Временные прерывания помогают гарантировать приемлемое время отклика процессов для пользователей, работающих в диалоговом режиме, и предотвращают „зависание" компьютерной системы из-за зацикливания какой-либо программы.

Алгоритмы планирования

First-Come, First-Served (FCFS)

Простейшим алгоритмом планирования является алгоритм, который принято обозначать аббревиатурой FCFS по первым буквам его английского названия — First-Come, First-Served (первым пришел, первым обслужен). Представим себе, что процессы, находящиеся в состоянии готовность, выстроены в очередь. Когда процесс переходит в состояние готовность, он, а точнее, ссылка на его PCB помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на его PCB.

Такой алгоритм выбора процесса осуществляет невытесняющее планирование. Процесс, получивший в свое распоряжение процессор, занимает его до истечения текущего CPU burst. После этого для выполнения выбирается новый процесс из начала очереди.

Raund Robin (RR)

Модификацией алгоритма FCFS является алгоритм, получивший название Raund Robin (Raund Robin — это вид детской карусели в США) или сокращенно RR. По сути дела, это тот же самый алгоритм, только реализованный в режиме вытесняющего планирования. Можно представить себе все множество готовых процессов организованными циклически — процессы сидят на карусели. Карусель вращается так, что каждый процесс находится около процессора небольшой фиксированный квант времени, обычно 10 — 100 миллисекунд. Пока процесс находится рядом с процессором, он получает процессор в свое распоряжение и может исполняться.

Shortest-Job-First (SJF)

При рассмотрении алгоритмов FCFS и RR мы видели, насколько существенным для них является порядок расположения процессов в очереди процессов, готовых к исполнению. Если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает. Если бы мы знали время следующих CPU burst для процессов, находящихся в состоянии готовность, то могли бы выбрать для исполнения не процесс из начала очереди, а процесс с минимальной длительностью CPU burst. Если же таких процессов два или больше, то для выбора одного из них можно использовать уже известный нам алгоритм FCFS. Квантование времени при этом не применяется. Описанный алгоритм получил название „кратчайшая работа первой" или Shortest Job First (SJF).

SJF-алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим. При невытесняющем SJF — планировании процессор предоставляется избранному процессу на все необходимое ему время, независимо от событий, происходящих в вычислительной системе. При вытесняющем SJF — планировании учитывается появление новых процессов в очереди готовых к исполнению (из числа вновь родившихся или разблокированных) во время работы выбранного процесса. Если CPU burst нового процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Приоритетное планирование

Планирование с использованием приоритетов может быть как вытесняющим, так и невытесняющим. При вытесняющем планировании процесс с более высоким приоритетом, появившийся в очереди готовых процессов, вытесняет исполняющийся процесс с более низким приоритетом. В случае невытесняющего планирования он просто становится в начало очереди готовых процессов. Давайте рассмотрим примеры использования различных режимов приоритетного планирования.

Механизмы синхронизации

Семафоры

Семафор представляет собой целую переменную, принимающую неотрицательные значения, доступ любого процесса к которой, за исключением момента ее инициализации, может осуществляться только через две атомарные операции: P (от датского слова proberen — проверять) и V (от verhogen — увеличивать). Классическое определение этих операций выглядит следующим образом:

P(S): пока S == 0 процесс блокируется;

S = S − 1;

V(S): S = S + 1;

Мониторы

Мониторы представляют собой тип данных, который может быть с успехом внедрен в объектно-ориентированные языки программирования. Монитор обладает собственными переменными, определяющими его состояние. Значения этих переменных извне могут быть изменены только с помощью вызова функций-методов, принадлежащих монитору. В свою очередь, эти функции-методы могут использовать в работе только данные, находящиеся внутри монитора, и свои параметры. На абстрактном уровне можно описать структуру монитора следующим образом:

monitor monitor_name {

```
    описание внутренних переменных ;
    void m1(...){...
    }
    void m2(...){...
    }
    ...
    void mn(...){...
    }
    {
        блок инициализации
        внутренних переменных;
    }
}
```

Здесь функции m1,..., mn представляют собой функции-методы монитора, а блок инициализации внутренних переменных содержит операции, которые выполняются один и только один раз: при создании монитора или при самом первом вызове какой-либо функции-метода до ее исполнения.

Важной особенностью мониторов является то, что в любой момент времени только один процесс может быть активен, т. е. находиться в состоянии готовность или исполнение, внутри данного монитора. Поскольку мониторы представляют собой особые конструкции языка программирования, компилятор может отличить вызов функции, принадлежащей монитору, от вызовов других функций и обработать его специальным образом, добавив к нему пролог и эпилог, реализующий взаимоисключение. Так как обязанность конструирования механизма взаимоисключений возложена на компилятор, а не на программиста, работа программиста при использовании мониторов существенно упрощается, а вероятность возникновения ошибок становится меньше.

Сообщения

Для прямой и непрямой адресации достаточно двух примитивов, чтобы описать передачу сообщений по линии связи — send и receive. В случае прямой адресации мы будем обозначать их так:

send(P, message) — послать сообщение message процессу P ;

receive(Q, message) — получить сообщение message от процесса Q.

В случае непрямой адресации мы будем обозначать их так:

send(A, message) — послать сообщение message в почтовый ящик A ;

receive(A, message) — получить сообщение message из почтового ящика A.

Примитивы send и receive уже имеют скрытый от наших глаз механизм взаимоисключения. Более того, в большинстве систем они уже имеют и скрытый механизм блокировки при чтении из пустого буфера и при записи в полностью заполненный буфер. Надо отметить, что, несмотря на простоту использования, передача сообщений в пределах одного компьютера происходит существенно медленнее, чем работа с семафорами и мониторами.