

Оси

Коченюк Анатолий

12 октября 2021 г.

0.1 Введение

В современном мире не представить вычислительный узел без операционной системы.

Пример. Супермаркет с одной кассой и тремя покупателями:

- с водой, человек очень хочет пить
- корзинка на ужин
- тележка на неделю

Если пропустим вперёд парнишку, его вклад будет крайне маленький. Но он выйдет из магазина и проходящий может зайти увидев очередь из двух человек.

Если же выстроить наоборот, то новые покупатели не заходят заходить в ваш супермаркет.

Хочется иметь монополию на власть в смысле порядка очереди

Многие процессы связаны с социальными процессами.

Определение 1. Операционная система – базовое системное программное обеспечение, управляющее работой вычислительного узла и реализующее универсальный интерфейс между аппаратным обеспечением, программным обеспечением и пользователем.

базовое – первое, что появляется и последнее, что умирает

системное – само пользу конечному пользователю не приносит, но без него не работает прикладное

управляет – достигает некоторых целевых показателей

Глава 1

Этапы эволюции ПО

1.1 Программы-диспетчеры

Конец сороковых, компьютеры работают строго по архитектуре Фон-Неймана. Есть устройство ввода и вывода.

4 принципа:

- однородность памяти – код и данные в единой памяти
- адресность – оперативная память это линейно-адресуемое пространство и мы можем обратиться в любой момент к любой её ячейке.
- программное управление – программа представляет собой набор инструкций в память. Процессор по ходу своей работы поочерёдно берёт и на такте выполняет следующую инструкцию
- кодирования – всё, и данные, и инструкции, кодируются с помощью двоичного кода.

Задача 1 (Повторное использование кода. Автоматизация загрузки и линковки). В те времена комп часто использовался для физических вычислений. Операторы заметили, что заново вводят одни и те же инструкции по многу раз. А почему бы не попробовать вынести куда-то.

Идея: вынести “подпрограмму” в конкретный адрес. МЫ переходим в него из своего кода.. но как вернуться, а ещё передать что-то хочется.

Появляются диспетчеры, которые управляют таким выводом

Задача 2 (Оптимизация взаимодействия с устройствами ввода-вывода). В классической архитектуре все устройства работают с памятью через процессор. Задача подкачки довольно простая, для неё не нужно всей мощности процессора.

Задача: осветлить картинку. Задача, которую можно делать независимо для всех пикселей

Идея: контроллер – связан с памятью, RAM и частично с процессором.

Предсказать время подкачки нельзя, потому что совершенно разные носители с разными гарантиями. Код не может знать подкачались ли уже данные или нет. А хочется уметь на это реагировать, т.е. ждать пока завершится процесс подкачки.

Обработчик прерываний – меняет значение флага, опрашиваемый в бесконечном цикле, пока контроллер не даст флаг, что всё сделано. Функционал диспетчера разрастется, теперь умеет обрабатывать прерывания.

SPOOL – simultaneous operation online

Определение 2. Прерывание – сигнал, поступающий от внешнего устройства к центральному процессору, приостанавливающей исполнение текущего потока команд и передающий управление обработчику прерываний.

Задача 3 (Однопрограммная пакетная <...>). Одно приложение – много программ. Кроме того может ещё набор констант.

Появляется термин пакет – как совокупность программ.

Пока исполняем одну, можем загружать другую или даже другие и тут возникает вопрос: вот мы завершили один процесс, а какой исполнять следующим. Возвращаемся к задаче про супермаркет. Если взять идею пропускать маленького и реализовать её вот так втупую, то мы рискуем попасть в программное голодание, когда постоянно подгружается что-то маленькое и проходит вперёд.

1.2 Мультипрограммные операционные системы

Что привело к их появлению: программы становятся более сложными и разнообразными. Неэффективно используем ресурсы, потому что исполняем программы от начала до конца.

Идея: несколько программ можно исполнять параллельно. Простая идея привела к огромным сложностям.

Программы обычно исполняются “псевдо-параллельно”

Задача 4 (Обеспечение разделения времени процессора). Сделал аппаратное решение, будет генерироваться прерывание каждый тайминг, которое будет запускать планирование того, что выполнять следующим.

Когда мы останавливаем процессор, в регистрах что-то есть и это что-то нужно. Идея: откатываться назад, но непонятно насколько, может я давно что-то туда положил. Забыть тоже нельзя, код дальше рассчитывает на эти регистры. Регистровый контекст приходится где-то сохранять (чтобы его потом найти), подгружать такой же контекст от другого процесса и запускать его.

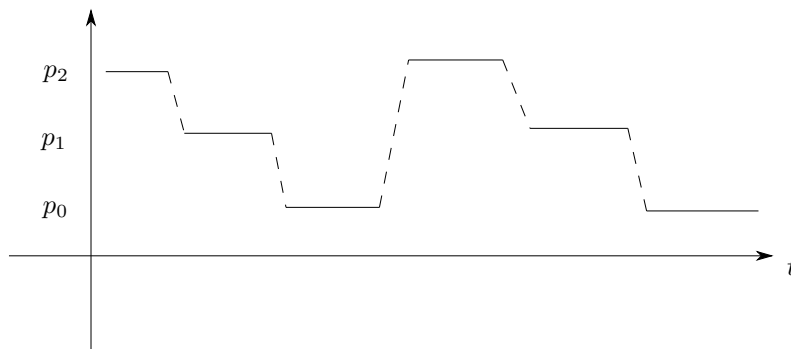


Рис. 1.1: pseudo-parallel

Задача 5 (Обеспечение разделения памяти). Когда мы пишем код, переменные заменяются на адреса. Программный ноль совпадал с реальным и всё было хорошо. Но у нас много программ и память может быть где угодно. Идея: виртуальная память – при исполнении подменять виртуальные адреса, считающие, что они в реальном нуле, на физические.

Задача 6 (Обеспечение защиты данных программы от деятельности других программ). Люди пишут код с ошибками – закон вселенной. Если программы многих людей, а мы своей ошибкой зашли в код чужой программы, то будет нехорошо.

Идея: защита памяти. аппаратное решение – при обращении к памяти понимать свой-чужой.

Но теперь не может работать наш диспетчер, он же должен быть изолированным. А ему нужно что-то взять, куда-то записать..

Идея: кому-то нужно разрешить. Привилегированный режим. Грубо говоря отключаем защиту памяти. Этот привилегированный режим приводит к пониманию современной системы:

System call – обращение пользовательской программы к ядру ОС с требованием предоставить ресурс или выполнить привилегированную операцию.

Теперь OS – универсальный интерфейс. У неё монопольная власть, на уровне неё мы пытаемся эффективно использовать ресурсы

Задача 7 (Планирование выполнения программ и использования ресурсов). У каждого контроллера своя очередь, у каждого жёсткого диска своя очередь, у сетевого узла, у процессора, ... А они ещё и связаны друг с другом

Хочется максимально эффективно всё заменеджить, что суммарно всё максимально быстро исполнилось.

Очередь и $re<...>$

Пример. Хотим напечатать что-то на принтере. Передаём данные, прервались, перключились на другую программу, а она тоже хочет печатать.. Неразделяемый ресурс.

Идея: ставим блок, чтобы только первый мог давать данные.

Но может быть дедлок.. 1 захватит ресурс 1, 2 захватит ресурс 2, в какой-то момент они хотят получить другой ресурс, не отжав первый – тупик.. Дейкстра занимался этим лет 15.

Задача 8 (Универсальный доступ к информации на внешних устройствах). Линейная адресация \rightarrow файл, каталог

Задача 9 (Обеспечение коммуникации между программами). Комфортная работа множества программ, а что если они будут передавать друг-другу данные.

ctrl+C ctrl+V – использование буфера, требует ручного управления.

Сигналы, передача из stdout одного в stdin в другого, ..

Появляется понятие виртуальной машины – приложение живёт отдельно и не знает, что есть другие приложения. С этим понятием появляется и термин операционной системы. Теперь делегирование всех операций лежит именно на ОС.

С первой ОС сложно, не понятно кого считать уже ОС.

1963 – компьютер B5000 с ОС MCP – Master Control Programm

1.3 Сетевые операционные системы

Компьютеры тогда – только большие и очень дорогие компьютеры.

Затраты на доставку программного кода до машины начинают превалировать

АМ – амплитудная модуляция, FM – частотная модуляция. Способы обозначать 0 или 1 в синусоиде.

Модем – модулятор-демодулятор. Теперь проблемы с безопасностью. Раньше был один конкретный подконтрольный оператор, с которого можно было в случае чего спросить. А теперь надо защищаться от людей. Появляются понятия учётной записи, аутентификации.

Появляются компании, специализирующиеся на предоставлении компьютерного времени. У такой компании могло быть уже несколько компьютеров.

Могла быть большая нагрузка в Чикаго и простаивать компьютер в Бостоне. Тогда строили линию АТМ между Чикаго и Бостоном, чтобы перенаправлять звонки.

1.4 Универсальные операционные системы

Мотивация: в 60-х все ОС были платформозависимые, невозможность повторного использования кода.

Идея: создать универсальную ОС на любую платформу

Парадокс: чтобы разрабатывать под такую ОС, на ней должен быть компилятор языка высокого уровня

Платформа переносимая, она сама написана на языке высокого уровня

В решении участвовалось подразделение компании AT&T – Bell Labs

Открыли реликтовое излучение, изобрели транзистор, открыли фотоэффект, матрица, “Математическая теория связи”

MULTICS – привязана жёстко к набору платформ. multiplexed

UNICS – uniplexed. Пишется полностью на ассемблере. Первая редакция запускается 01.01.1970

В 70 продолжаются разработки и разрабатывается язык B – интерпретируемый язык. UNICS переписывается на B

Керниган и Ричи разрабатывают C, встраивают его компилятор в UNICS (который ещё на B)

конец 75 – ed.4 ядро на C

Первая универсальная система

ed.7 1978 – последняя редакция UNICS, в ней появляется bash

В дело вступает антимонопольная служба США. AT&T весь код передала университет (первым – Бэркли). UNICS → UNIX.

Бэркли создаёт дочернее предприятие BSD (- software distribution)

Free BSD, Open BSD, ...

MIT, Berkley, Stanford – три университета

SUN (stanford university network), SUN OS .. Solaris

Проблема: коммерческие юниксы начинают патентовать решения. BSD лицензия защищает только имя автора. Многие пользовались этим, чтобы закрывать для других целые направления развития ОС

Манифест Столлмана – 4 свободы (0,1,2,3) программного изучения: использовать, изучать&адаптировать, распространять копии, публиковать

©→ ☺

Gnu is Not Unix

gcc – gnu c compiler

Студент Хельнского университета начинает интересоваться MINICS и преобразовывать. Появляется новая неожиданно-популярная система. Таненбаум делает пост: Линукс устарел

Студент – Линус Торвальдс.

ему остаётся только доказать, что его система качественная. Столлман предлагает ему подключиться к GNU. Линус соглашается, но с условием, что GNU переименуется в GNU/Linux – 1983-4

1989 – NeXT создаёт ОС NeXTSTEP

1997 – Darwin → MacOS. Apple хочет выйти на рынок компьютеров и покапает NeXTSTEP вместе со всем, что у неё есть. Что-то добавляет из FreeBSD, что-то сами дописывают.

Уровни:

- Функциональные – с позиции пользователя
- Информационная – потоки данных, структурированные информационные объекты
- Системная – интерфейсы: аппаратные, пользовательские, ..
- Программная – ООП, функциональная, ...
- Данных

Определение 3. Цель ОС – обеспечить производительность надёжность и безопасность исполнения пользовательского ПО, эксплуатации железа, хранения и передачи данных и диалога с пользователем.

Функции ОС:

- Управление разработкой и исполнением ПО
 - API
 - Управление исполнением
 - Обработка и обнаружение ошибок
 - Доступ к устройствам I/O
 - Доступ к хранилищу
 - Мониторинг ресурсов

- Оптимизация использования ресурсов. Хотим много всего, что противоречит друг с другом. Критериальные задачи $\hat{K} = \alpha K_1 + \beta K_2 + \gamma K_3$

Real-time ОС - гарантируется время отклика. Представим самолёт. При посадке ему нужно открыть закрылки. Если слишком сильно – мы перелетим, слишком слабо – упадём. Бортовому компьютеру нужно быстро посчитать этот угол.

Условный критерий $\hat{K} = \alpha K_1 + \beta K_2 |_{K_3 > z}$

- Поддержка эксплуатации. ОС должна иметь средства диагностики и восстановления на случай, когда железо ломается (inevitably происходит). Любая ОС умеет откатываться к последней удачной конфигурации.

-
- Поддержка развития самой ОС. ОС живёт дольше программного и аппаратного обеспечения. Может содержать ошибки и уязвимости..

Подсистемы:

- Управления процессами
 - Дескрипторы (PCB)
 - Планировщики. Для разных типов ресурсов выстраивают оптимальную очередь
- Управления памятью
 - Виртуальная память
 - Защита памяти
- Управления файлами
 - Символьные имена → физические адреса
 - Управление каталогами
- Управление внешними устройствами
 - Драйвера. Устройств сотни тысяч, для каждого должна быть подпрограмма, которая знает как с ним работать.
 - Plug&Play (Plug&Pray)
- Защита данных
 - Аутентификация и авторизация
 - аудит
- API
 - Разработка ПО
 - Исполнение ПО
- Пользовательский интерфейс
 - CLI
 - GUI

Вопрос: Что будет в ядре стал главным. Почему? Привилегированный режим, резидентность ядра (не меняет адреса после загрузки)

Принципы ОС:

- Модульная организация (в любом сложном ПО такое есть)
- Функциональная избыточность – функционал ОС существенно больше реального сценария её использования. Удобно для разработки, не удобно для эксплуатации

-
- Функциональная избирательность – всегда есть способ сохранить из всего многообразия функций только те, которые нужны для конкретного сценария.
 - Параметрическая универсальность – при разработке нужно максимально не загонять себя в константные рамки

1.5 Поддержки концепций многоуровневой иерархической системы

Модули:

- Ядра
- Работающих в пользовательском режиме

1.5.1 Монолитная Архитектура

Все видят всё. Преимущества производительности.

Три слоя:

- main program (software)
- Services
- Utilities (hardware)

Хорошая модель пока немного не очень сложного кода. Потом стало понятно, что три слоя это мало.

1.5.2 Многослойная архитектура

Не отдельная архитектура, а скорее концепция

1.5.3 Микроядерная архитектура

Часть слоёв вынесется в пользовательский режим.

Удобно: абстрагирование через системные вызовы, часть отдаём в подкачку и экономим память

Неудобно: небезопасно, в подкачке что-нибудь можно заменить и при подкачке оно исполниться как часть ОС; могут возникать проблемы с дедлоками и прочими тупиками.

Линукс – монолитное, но модульное ядро.



Рис. 1.2: many

1.6 Процессы

Определение 4. совокупность набора исполняющихся команд, ассоциированных с ним ресурсов и контекста исполнения, находящиеся под управлением операционной системы.

запускаем программу – последовательно исполняем набор команд. Если запустить vim в двух разных терминалах, это будут те же наборы команд, но разные процессы.

Поэтому ещё добавляет ассоциированные ресурсы. Но и здесь не ставят точку. Код ядра будет выполняться в контексте вашего процесса. Есть стек пользователя и стек ядра. Ещё есть регистровый контекст, который сохраняется при переключении процессов. команды + ресурсы + контекст

И наконец важно, что процессом управляет операционная система

PCB – дескриптор процесса

- – PID – process identifier
- PPID – parent PID
- UID – User ID
- ...
- ресурсы
- история использования ресурсов

Определение 5. Процесс – множество потоков

Поток – совокупность набора исполняющихся команд и контекста исполнения, находящиеся под управление операционной системы и разделяющие ресурсы некоторого процесса.

Поток – тоже находится под управление ОС. Планированием ОС занимается на уровне потоков.

Раз ОС занимается переключением потоком, значит она умеет сохранять контекст конкретного потока.

Если потоков много, то возникает ограничение на их количество. Очень сложно прогнозировать правильное количество потоков. Двух уровней мало, не можем управлять распределением времени на потоки

Определение 6. fiber (light-weight thread) (волокно в нити – thread'e) – набор команд, разделяющий ресурсы и контекст исполнения одного потока и находящийся под управлением пользовательского приложения.

Куча вопросов, а чего мы хотим? Не будет занимать в пространстве ядра никаких структур.

Кооперативная многозадачность облегчённых потоков. Внутри волокон есть безусловные переходы, вызывающие планировщик, чтобы он решил выполняться дальше или нет.

Поток в Erlang – fiber.

Корутина.

Но этих трёх уровнях всё ещё мало

Замечание. Хром создаёт процессы отдельно для вкладок. Но даже если их там, 200, другие приложения не помирают, получая $\frac{1}{200}$ ресурсов

Определение 7. job / C(ontrol)Group – настраивание квот. На количество подпроцессов, на интернет-трафик, ...

Процесс:

- Создание процесса
- Обеспечение ресурсами
- Изоляция
- Планирование

-
- Диспетчеризация
 - Межпроцессное взаимодействие
 - Синхронизация
 - Завершение

1.6.1 Создание

Создать процесс – создать структуры данных – ProcessControlBlock. В любой операционной системе процесс порождается другим процессом. Часто создание процесса называют рождением.

В Линуксе процессы образуют дерево и порождаются клонированием. Начальный процесс, $PID = 1$, `init/systemd`. Как создаётся первый процесс.. сложно, оно часто выглядит как набор костылей. у начального процесса $PPID = 0$. Есть два процесса, у которых $PPID = 0$: $PID = 1$, $PID = 2$, два дерева: пользовательские и ядерные процессы.

Чтобы завершить процессы, должны завершиться все его потомки.

Замечание. Зашли в баш по `ssh`, запустили веб-сервер. Хотим закрыть баш, но оставить сервер, ради которого мы собственно и зашли. Для такого есть механизм процессов-демонов. При завершении работы процесса-родителя, процесс переподвесился выше.

Если процесс завершился аварийно, и его дети не успели получить код завершения, они подвешиваются выше, чтобы код завершения уже их кто-нибудь прочитал.

`ctrl+C` – послать сигнал завершения

`ctrl+Z` – послать сигнал `SYGSTOP`

Если у приостановленного процесса завершился подпроцесс, от него осталась строка в таблице с его PID ’ом и кодом завершением. А PID ’ы штуки конечные. А если они закончатся, то даже команду `kill` нельзя будет послать, потому что это отдельные процесс. Такой подпроцесс называется зомби, а описанная ситуация – зомби-апокалипсис.

Порождение клонированием: `fork` (полная копия адресного процесса, со всеми ресурсами (указателями на их дескрипторы), подменяется значение PID), подменяется код на код подпроцесса.

Такая система не позволяет подпроцессу иметь большие права, чем сам процесс, потому что он их наследует, копирует

В Windows ситуация другая почти с начала: Есть Диспетчер/Менеджер процессов. Каждый процесс создаётся им. Может быть больше прав у дочернего процесса

Самая простая модель: два состояния: ожидание и исполнение. При рождении он попадает в ожидание.

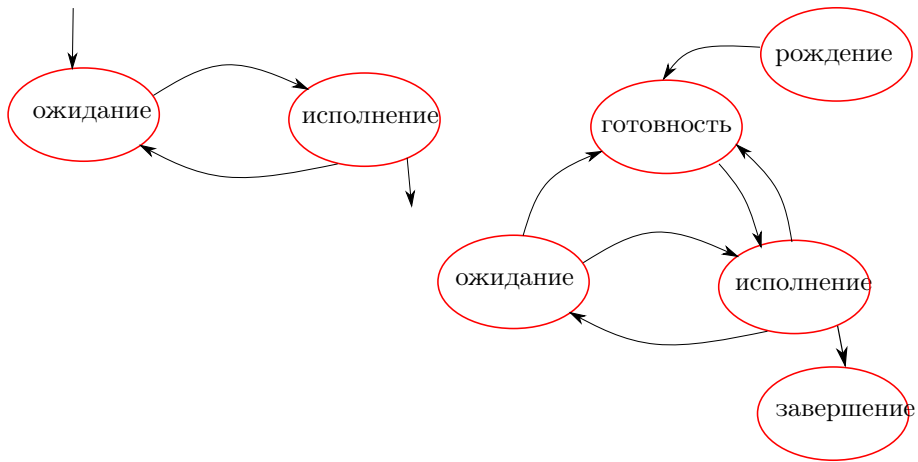


Рис. 1.3: 2states

Проблемы: процессы с исключениями не умирают сразу (для обработки ошибок) и иногда пытаются сделать то же самое (деление на ноль например) много раз, чем съедают ресурсы. Новое состояние – готовность

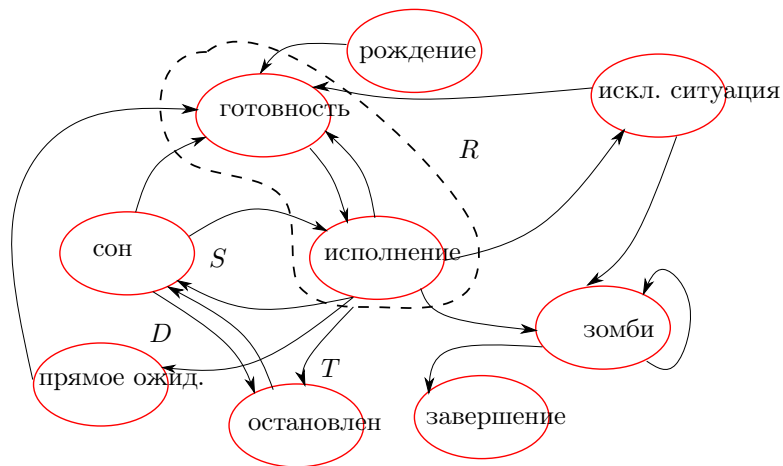


Рис. 1.4: manystates

1.7 Планирование

Планировать один шаг плохо, планировать надолго тоже много

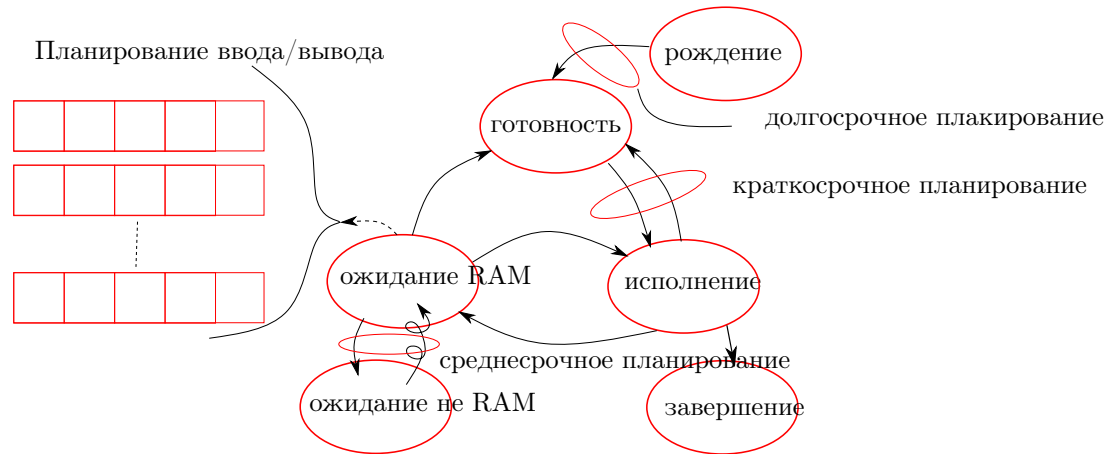


Рис. 1.5: plans

Критерии алгоритмов: справедливости, эффективность. Решения оптимальные по обоим критериям совпадают только если все процессы равны.

Новые критерии: полное время исполнения. Например при компиляции чего-нибудь большого хочется побыстрее получить результат и не важно что там по справедливости и ресурсам с другими процессами

Ещё один: для пользователя важно, чтобы не было ожидания

Последний: для пользователя важно время отклика. Например когда набираешь в vim'e хочется видеть каждую букву по мере того как она печатается, хотя с точки зрения ресурсов хорошо буферизировать строку и один раз её выводить.

Свойства:

1. Предсказуемость – на одинаковых параметрах одинаковый результат.
2. Минимальные накладные ресурсы
3. Масштабируемость. .

Параметры планирования:

- Статические параметры системы – не поменяются в этом экземпляре ОС
- Динамические параметры системы – свободные

-
- Статические параметры процесса – не меняются во времени
 - Динамические процессы:
 - CPU-burst
 - I/O-burst
-