Содержание

1 Определение задачи реального времени. Чем системы РВ отличаются от систем разделенного времени? Пример архитектуры ОС реального времени.

Режим реального времени – режим, при котором пользовательская программа получает управление в течение в течении гарантированного (обычно достаточно небольшого) времени после возникновения того или иного внешнего события [16]. Системы РВ – это те самые, которые могут гарантировать небольшое максимальное время реакции. Примеры ОС: QNX и VxWorks [36-37]. Примеры архитектуры: RT-11 [851], RSX-11 [854] Выделение памяти с помощью алгоритма парных меток [247-249]. Сборщики мусора неприемлемы [261]. Диспетчер памяти и виртуальная память не используются [288]. Спинлоки недопустимы [385]. Асинхронный ввод-вывход [почти 398]. Реализация многозадачности [438].

2 Компиляция и запуск.

2.1 Запуск задач в ОС семейства Unix.

Создание процесса осуществляется с помощью fork (дублирует текущий процесс со всеми потрахами, использую copy-on-write [336-338]) и последующего exec (заменяет образ текущего процесса на образ другого) [158]. Самый первый процесс в unix'е – init – запускается вроде бы загрузчиком, а вот все остальные получены как раз с помощью связки fork-exec [man init]. TODO: exec implemenation, возможно, из [Хевиленд/Грей/Салама]. Кратко: делается парой mmap'ов: один для кода, другой для данных, потом создаём стэк и передаём управление [160-161].

2.2 Что такое абсолютный и относительный загрузчики? Структура абсолютного и перемещаемого загрузочных модулей. Что такое позиционно-независимый код?

Абсолютная загрузка – программа всегда загружается с одного и того же адреса. Абсолютный заргузочный модуль хранит в себе сгемент кода и тупо сегмент данных, которые тупо загружаются в память [160-161]. Относительная заргузка – программа загружается каждый раз с нового адреса. Возникающая проблема: не знаем точные адреса, зато знаем относительные, решается с помощью добавления в загрузочный модуль таблицы перемещний и просьбой к программисту пользоваться этой таблицей. [162-167].

Ещё один способ формирования адреса – его вычисление на основании счётчика команд. Есть global offset table, в которой хранятся данные, прежде чем туда обращаться вычисляется её адрес [168-169, en.wiki/PIC]. Такой код удобно использовать в разделяемых библиотечках (ELF [320]).

2.3 Объектный модуль. Объектная библиотека. Структуры данных, содержащиеся в объектном модуле, в общих чертах. Алгоритм работы сборщика и выбора модулей из библиотеки.

По-простому: все файлы независимо компилируются в объектный модуль, каждый модуль содержит ссылки внутри, определённые символы и прочую ересь [174-177].

Объектная библиотека – файл, состоящий из заголовка и последовательного определения объектных модулей. Сборщик собирает в программу все объектные модули, переданные ему как параметры, далее перебирает все неразрешённые глобальные ссылки и на каждую ищет в библиотеки модуль, в котором находится требуемый символ [186-188].

2.4 Сборка в момент загрузки. Преимущества и недостатки этого метода. Чем отличаются DLL Win32 и разделяемые библиотеки ELF.

Объектные модули можно подгуражть не только во время сборки, но и во время зазгрузки программы. Недостаток: большее время зарузки, преимущества: экономия памяти, разделение кода, упрощается разработка программ. Потому и существуют разделяемые библиотеки [188-190].

Немного скучной теории про разделяемые библиотеки и про виндовый DLL в частности [193-199]. TODO: сравнение DLL [318+] и ELF[320]:

- У DLL есть проблема с загрузкой разных версий одной библиотеки, но решается путём поиска по полному путевому имени [318].

- DLL не являются позиционно-независимыми, для их перемещения используется специальная утилита [319], ELF являтся PIC за счёт наличия GOT и PLT [319].
- У каждого процесса своя копия DLL в памяти. С ELF все процессы разделяют не приватные части библиотеки.

2.5 Как происходит загрузка операционной системы? Что такое первичный загрузчик? Вторичный? Как происходит загрузка бездисковых машин?

Первое, что запускается – *загрузочный монитор*, который находится по фиксированному адресу в ПЗУ (для PC это, кстати, BIOS, ещё есть UEFI). Он как-нибудь определяет устройство, на котором находится ОС, и запускает *первичный загрузчик* с *загрузочного сектора*, которым обычно является нулевой сектор. Первичный, возможно, запускает вторичный загрузчик, дабы тот загрузил ОС, и т.д. [200-202]. Загрузка бездисковых машин несильно отличается от дисковых: по сети получаем вторичный зарзучик и вперёд... [204].

2.6 Загрузка ОС на PC-совместимых компьютерах.

Этапы:

1. Запускается BIOS.
2. BIOS инициализирует устройства, находит среди них загрузочное, копирует в память MBR, передаёт ему управление.
3. MBR копирует в память загрузочный сектор с требуемого отдела и передаёт ему управление.

вроде так [208, 211].

3 Память

3.1 Динамическое выделение памяти в ОС семейства Unix и стандарте POSIX.

Есть куча, при вызове malloc в ней ищется свободный блок. При отсутствии такового, у системы просится ещё память [216-220].TODO

3.2 Динамическое выделение памяти. Методы борьбы с фрагментацией. Основные алгоритмы выделения памяти.

Внутренняя фрагментациея – при выделении блоков фиксированного размера могут появиться дырки. Обычно куча блоков памяти представляется в виде двусвязного списка [220-225].

Существуют алгоритмы выделения:

1. first fit. самый актуальный, особенно, реализованный при помощи кольцевого списка [227].
2. best fit. увеличивает фрагментацию, т.к. после него остаётся много мелких кусков [226].
3. worst fit. имеется куча или посорченый список, быстро взять, но сложно вернуть [226].

3.3 Распределение памяти алгоритмами близнецов и парных меток Ограничения этих алгоритмов.

После выделения памяти её надо как-то освобождать. Проблема – объединение последовательных блоков.

Алгоритм парных меток. Для каждого свободного или занятого блока в начале и в конце хрантися его размер со знаком, знак означает знаят блок или свободен. Становится просто объединять блоки [228-230].

Алгоритм близнецов. Размеры блоков – степени двойки [247-249].

3.4 Алгоритм работы библиотечных функций malloc/free GNU LibC.

Для больших блоков используется first fit и при отсутствии нужного дополнительная память запрашивается у системы. Для маленьких выделяется блок размером степени двойки. Каждый большой разбит на фрагменты равного размера, равного степени двойки [230-233]. А вот как освобождается – не написано. TODO. Возможно, тут http://gee.cs.oswego.edu/dl/html/malloc.html есть что-нить интересного.

3.5 Сборка мусора. Основные стратегии сборки мусора, их преимущества и недостатки.

Память взяли, поиспользовали, но не освободили – вот и мусор. Как варинат, можно неявно освобождать память, для этого используется сборщик мусора [252-254]. Стратегии:

1. Подсчёт ссылок. Просто считаем кол-во ссылок на выделенный блок памяти. Недостаток: циклы [255-256].
2. Просмотр ссылок. dfs. Недостатки: на время dfs'а надо остановиться всю программу; неизвестно, когда вызовется дестркутор и освободятся объекты.
3. Генерационная сборка мусора. Разбиваем на поколения и внутри каждого делаем dfs [259-261]. Есть скромный хак для оптимизации, чтобы не блокировать потоки [264-265].

3.6 Методы реализации виртуальной памяти. Базовая адресация, сегментная и страничная виртуальная память.

Парам-пам-пам [273-289]. TODO: различия между страницами и сегментами.

3.7 Разделяемая память. Преимущества и недостатки по сравнению с другими методами межпроцессного взаимодействия.

Разделяемая память: используется 2 таблицы трансляции: одна для простых смертных процессов, другая – для ядра. Для передачи параметров в системные вызовы используются специальные команды [290+].

Альтернативное решение: отображать адресное пространство ядра на пространство пользовательского процесса. [298+].

3.8 Загружаемые модули и разделяемые библиотеки Win32 (PE).

Немного теории про динамическое библиотеки [193+, 315-318]. DLL Win32 – говно [318-320]. Что-то про внутренне строение PE, хотя много и убого [http://www.rsdn.ru/article/baseserv/pe_coff.xml].

3.9 Разделяемые библиотеки формата ELF.

Пра загрузке динамической библиотеки запускается редактор связей (по книге – ld.so.1, хотя у меня в системе конкретно ld.so нет, есть /lib/ld-linux.so). Есть global offset table (список символов внутри?) и procedure linkage table (ссылки на символы снаружи). [320-327].

3.10 Алгоритмы поиска жертвы при страничном обмене и кэшировании. Критерии выбора и влияние алгоритма на производительность. Что такое рабочее множество страниц?

[329-330]. Алгоритмы:

- Удалять то, что не было изменено.
- rand().
- Удалять то, что дольше всего лежит в памяти (FIFO), как в Win NT. [330-332].
- LastRecentlyUsed. Как его модификация – clock-algorithm [332-333].

Набор страниц, которые действительно нужны в данный момент процессу [334].

3.11 Организация страничного обмена в VMS, OpenVMS и Windows NT.

аналогично следующему?

3.12 Подсистема виртуальной памяти и алгоритм поиска жертвы в Windows NT.

Поддерживаем очереди, в между которыми гоняем страницы и в которых ищем жертвы. [330-332].

4 Внешние события, прерывания

4.1 Ввод-вывод в режиме опроса и по прерываниям. Преимущества и недостатки.

Режим опроса. Один из способов узнать, что где-то произошло событие – периодически опрашивать устройства. Недостаток: нужно устройство, которое будет навечно занято опросом [349-353]. Ещё недостаток: процессор, который производит опрос, должен всегда работать, решение как бы есть [352-355].

Альтернативный вариант – прерывания. Есть специальные входы, сигнал на которых намекает, что неплохо бы было его обработать [356]. Связь со словами ввод-вывод: работу с периферийными устройствами часто называют подсистемой ввод-вывода [471].

4.2 Прерывания в классических процессорах (PDP-11, 8086, x86). Внешние прерывания и исключения (exceptions).

В PDP-11 у прерываний появляются приоритеты. Наверное, потому что некоторые прерывания надо обрабатывать раньше, нежели другие. [356-358].

Исключения – внутренние прерывания. Разница: возврат из обработчика прерываний приводит к исполнению следующей команды, а исключений – повтору текущей [359].

5 Параллелизм и многопоточность

5.1 Реентерабельная программа. Техника реализации реентерабельных программ. Всегда ли это возможно? Что такое критическая секция?

Критическая секция – интервал, в течение которого модификация нарушает цлостность разделяемой структуры данных, и, наоборот, интревал в течение которого алгоритм нити полагается на целостность этой структруры. Решение проблемы с критическими секциями: либо тупо избавиться от них (что невсегда возможно), либо предоставить гарантии *взаимоисключения* (mutual exclusion). *Реентерабельная программа* – программка, в которой нет критических секций, для которых не обеспечено взаимное исключение. [378-380].

Реентерабельность системных функций и процего сишного добра в Solaris можно посмотреть в man'е в секции ATTRIBUTES, параметр MT-Level [man attributes].

5.2 Спинлоки и их применение. Их преимущества и недостатки по сравнению с другими средствами взаимоисключения.

Другие средства взаимоисключения:

- Поставить флажок перед каждой критической секцией. Недостатк: это неработающий костыль [380-381].
- Костыль, исправляющий костыль: у каждой нити свой флажок. Недостаток: всё сложно, особенно с увеличением кол-ва нитей [381-383].
- testandset. Одновременно проверять и устанавливать в холостом цикле – это и есть спинлок [381-385].

5.3 Мертвая и живая блокировки. Способы их предотвращения. Преимущества и недостатки каждого из методов.

Мёртвая блокировка (deadlock) – цикл взаимного ожидания нитей. Критерием блокировки является цикл (в том числе петля) в графе ожидающих друг друга задач [385-386]. Способы решения такой проблемы:

- Один из способов решения: проверять, ни приведёт ли попытка захвата ресурса к мёртвой блокировке, одна тут может возникнуть живая блокировка, которая ничем не лучше. Во избежания живой блокировке есть костыль: ждать случайное время, предварительно освободив все свои ресурсы [386-388].
- Разрешить каждой нити захватывать не более одного ресурса. Не всегда возможно [388].
- Захватывать в определённом порядке [388].
- Групповой захват [388]. Насколько я помню из лекций, Unix таких средств не предоставляет.

5.4 Семафоры Дийкстры. Мутексы, двоичные семафоры и семафоры общего вида. Мертвая блокировка и способы избежать ее.

Про мёртвую блокировку описано разделом выше.

Немного про синхронизацию и попытку замены бесконечно-ждущего цикла на сигналы [391-394]. В семафоре общего вида есть счётчик – кол-во нитей (или почти кол-во нитей), которые могут одновременно захватить ресурс. В двоичных семафорах (они же мутексы) это счётчик равен 1. [394-396]. TODO: реализация.

5.5 Флаги событий в RSX и VMS. Что такое AST?

Флаги вместо этих ваших мутексов и семафоров. На флаги можно ставить обработчики событий – AST.

5.6 Что такое гармонически взаимодействующие последовательные процессы? Средства для реализации этой дисциплины в существующих системах.

Концепция гармонического взаимодействия заключается в попытке таки исключить все критические секции, разбив программу на независимые по данным модули, и обеспечить взаимодействие за счёт магии. Бывают двухточечные, многоточечные симметричные и не симметричные. Бывают синхронными, буферизованными и без гарантии доставки. Бывают структурированными и потоковыми [405-409]. Кратко примеры реализаций в таблице 7.1 [408]. Подброно – дальше [409+].

5.7 Программные каналы (трубы) в системах семейства Unix.

Pipe – буферизованнй потоковый примитив передачи данных [409-413]. Замечания:

- нигде не сказано, что gcc, который указан как пример, по дефолту использует трубы, хотя умеет, если указать фалг `-pipe` [man gcc].
- `read` в дескриптор для записи, как и `write` в дескриптор для чтения приведёт к ошипке и вернёт -1.
- `mkfifo` просто создаёт файл, но ничего не открывает. Т. е. возможна ситуация, что процесс создаст трубу, а пользоваться ей будут другие [man fifo].

5.8 Почтовые ящики (mailbox) в VAX/VMS.

Оч много текста... [413]. TODO, естесна.

5.9 Линки в транспьютере.

Транспьютер – процессор со стэковой архитектурой [92-93]. Линки бывают физическими (реализованы отдельно на кристалле), логическими (для связи между процессами) и виртуальными [413-416].

5.10 Событийно-ориентированные системы. Обязательно ли такая система является многопоточной?

Генерируются события, они побуждают другие объекты реагировать на эти события и, возможно, генерировать свои события. Обязательна для асинхронной обработки событий [416-426]. Thread pool и worker thread [426-427].

5.11 Как реализуется многопоточность на однопроцессорной машине. Что такое контекст процесса? Какие особенности процессора влияют на скорость переключения процессов?

Переключением процессов [431+]. В контекст нити входит его стэк, регистры и прочая хрень... [438-439]. Немного про контекст процесса и его переключения [449-450]. TODO: как-то неполно...

5.12 Кооперативная и вытесняющая (preemptive) многозадачность. Преимущества и недостатки обоих архитектур.

Кооперативная многозадачность. Задача сама переключает себя на следующую. Преимущество: простая реализация, недостаток: повсюду надо пихать ThreadSwitch() со всеми вытекающими [432-437].

Вытесняющая многозадачность. Процессы насильно переключаются системой. Напр., по таймеру [437-444].

5.13 Планировщики разделенного времени. Динамическое управление приоритетами в системах разделенного времени.

Аналогично следующему. Конкретно про динамически приоритеты в ОС разделённого времени [452].

5.14 Приоритеты процессов и нитей. Управление приоритетами для нитей реального и разделенного времени. Где используется и для чего нужно динамическое изменение приоритета?

Нить, которой передаётся управление, определяется на основании приоритетов [450-456]. TODO: про реальное время немного всего.

5.15 Диспетчер задач в транспьютере.

Транспьютер [92-93]. Есть пара очередей [451]. Немного о контексте транспьютера [449-450]. TODO: мало.

5.16 Приоритеты процессов в OS-9.

Используется возраст и приоритет [457].

5.17 Инверсия приоритета. Способы ее предотвращения и способы обхода этой проблемы.

Возникающая проблема с приоритетами: нити с высоким приоритетом приходится ожидать нить с низким. Способы решения: наследование приоритета и потолок приоритета. Хреновые это способы, честно говоря [461-464].

5.18 Формирование запросов на ввод/вывод в системах фирмы DEC (RSX-11, VMS, OpenVMS). Какие преимущества предоставляет этот метод?

Асинхронный ввод/вывод: ставим запросы в очередь [465-467+]. Как работает драйвер при асинхронном вводе/выводе [610]. Чуть больше об асинхронном вводе/выводе [622, 619+].

6 RAID.

0 Просто объединение дисков.

1 Зеркалирование – данные тупо дублируются.

2 К данным добавляется информация для восстановления, под которую так же отводятся диски.

3,4 На отдельном диске хранятся контрольные суммы.

5 Аналогично 3-4, только контрольные суммы разбросаны по дискам.

[555-558, в ru.wiki явно описаны преимущества и недостатки].

7 Драйверы.

7.1 Драйвер устройства. Функции драйвера в ОС семейства Unix.

Драйвер – специализирвоанный программный модуль, управляющий внешним (и не только внешним) устройством [563+]. Функции драйвера [569+]. Немного про специфичное для Unix'а разделение на блочные и символьные [581].

8 Файловые системы.

Имена файлов [635+].

8.1 Структура файловой системы RT-11.

Оочень простая ФС, которая могла применяться на ленточных устройствах [648, 650-652].

8.2 Файловая система ISO 9660 (CDFS).

Чем-то похоже на предыдущую с траблами при мультисессиях [653].

8.3 Файловая система FAT.

File allocation table, всё такое... [654+]. Про устойчивость к сбоям у FAT [670].

8.4 Организация файловой системы HPFS.

B-деревья и прочие радости [657-661].

8.5 Понятия инода и связи в файловых системах ОС семейства Unix.

Основная инфа [661(-667), man stat]. Ошипки, связанные с инодами, при сбое [674].

8.6 Структура и особенности организации файловой системы UFS (FFS).

[661].

8.7 Структура и принципы работы файловой системы NTFS.

Описание на digit-life.com, первая ссылка при поиске по NTFS [667].

8.8 Устойчивые к сбоям файловые системы. Методы реализации устойчивых ФС.

Последствия сбоев, как с ними бороться [668+]. После журналируемых ФС идёт немного про сбои диска [691].

8.9 Журнальные файловые системы. Принципы работы. Для чего это нужно?

Принцип работы – наличие журнала :) [677-691].

8.10 Файловая система NetApp WAFL.

Copy-on-write [693+].

9 Сигналы в системах семейства Unix.

Регистрация обработки ошибок [733]. Что-то про драйверы [10.6.4].

10 Безопасность

10.1 Системы управления доступом. Полномочия и списки контроля доступа. Кольца доступа.

[772+].

10.2 Права доступа к файлам в ОС семейства Unix.

Основы [780]. Продолжение (setuit) [785].

10.3 Механизм setuid в ОС семейства Unix.

[785, 788].

11 Оставшиеся вопросы.

1. Аутентификация и проверка подлинности кода в Lotus Notes. (13.??)
2. Троянские программы и способы их внедрения. Меры по защите от троянских программ. (13.8)
3. Семафоры Unix System V IPC. Наборы семафоров. (IPC в книге не нашёл)