

02.09.2022 г. Сессия 2.

Рязанова Николай Юрьевич

Процессы Unix.

Linux-Unix-подобная ОС, в которой реализовано ~~все~~ ^(искусственный) ~~абстрактное~~ Unix.

Процесс - программа в стадии выполнения.

Запускается на выполнение исполнительной подпрограммой.

Процесс - базовая абстракция ОС.

В Unix чтобы процесс создался системный вызовом fork() fork() создаёт процесс-потомок, который наследует код процесса-предка (процесс-предок - процесс, вызывающий fork()).

В том смысле, что процесс-потомок наследует код предка, дескрипторы открытых файлов, символьную маску, маску созданных файлов и т. п.

Классическая запись fork()

```
int pid, childpid;
pid = Process IDentifier.  
if((childpid = fork()) == -1)
{
    // ошибка fork(), создать процесс не удалось.
}
else if(childpid == 0)
{
    /* потомок */
}
else
{
    /* предок */ ид. потенцика.
```

По соглашению Unix, при ошибке выполнения системного вызова, возвращается -1. ($\neq 0$).

С момента вызова fork() начинает выполняться две новые одна программы.

Пакете присутствует системный вызов exec(), который

переводит
ОС на новый
неда они
задача на
В стар
стивенское
придка. У
одной м
Для р
Оптимиз
Для
трансвер
III. e. ги
менное
нее при
ка соот
ка (АП
томат
навис
зредок
страж
ситуа
тиши
боязь
переп
пара

8 Курсовая
занятие № 1
вопросы
и ответы

прекращает процесс на выполнение другого кода.
Он называется отладчиком, потому что в своё время занимался операторов — людей, управлявших компьютерами. Основная задача ОС — выполнить программы.

В старых системах для процесса-потока создавалось собственное адресное пространство, в котором находилась его память. Из-за этого в системе могло быть много несколько копий одной программы — незадействованные использованием памяти.

Для решения разработчики идут:

Оптимизируя fork()

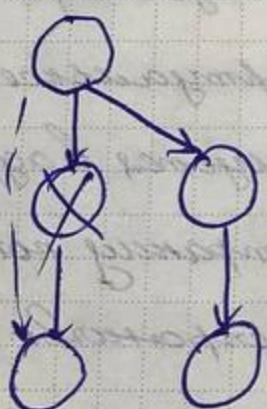
Для процесса-потока создается собственное картонажное адресное пространство (МРК) (в современных ОС — таблицы страниц). Т.е. для потока создаётся собственное виртуальное адресное пространство, но код процесса не помещается в адресное пространство потока, а таблицы страниц помечаются на страницу адресного пространства процесса (АП). Примерно где АП процесса права доступа меняются с общих Read-Write (WR) на Only Read и увеличивается флаг Copy-On-Write. После этого, если имеется запрос, или потоком попытается изменить какую-либо страницу, будет изменение по правам доступа. Такая ситуация в ОС будет существовать до тех пор, пока потоком не завершится, будь то exec() или же борется с собой exec(). В результате вызова exec() процесс-поток перейдет на АП программы, ком. передана в качестве параметра. В результате начнется выполнение этой программы.

В этих случаях для предка боязнь обычного права доступа завершения
файл Copy-On-Write обрашивается.

(*) Обработка это исключение (обрабатывает под его), ОС
обнаруживает файл Copy-On-Write и создает конец ограни-
чений в АП процесса, кем. попытка ее предотвратить.

Правильный Read-Write обнаружит синхронизацию, когда за-
писывать нечего.

Каждый процесс имеет чистое количество раз ~~записей~~^{байтами}
fork(), [fork()-блока потребляет ресурс ОС, из-за этого и
блока.] В ре-те синт. блоков fork() в Unix возможны дерево
процессов, отмеченные „предок - потомок“:



Если какой-либо из процессов аварийно завер-
шается, то его процесс - потомки ставят-
ся в остановку. ОС этого допустить не мо-
жет. ОС блокирует дальнейшую работу над
сопрограммией иерархии.

При завершении процесса ОС проверяет, не осталось ли у него
незавершенное помещение. Если также есть, то их уда-
ляют [процесс, открывший терминатор] (в Ubuntu - про-
цессор - посредник) с pid = 1.

Такие же поминки подтверждают указатель на нового
предка, а предок - указатель на новых помещиков. Процес-
сор предок получает статус завершил помещика
и продолжает работать, но как присоединенный. Например,
системной блок wait() заставляет процесса ждать

закончения процессов - помехов.

В системе всегда имеется процесс с pid = 0 и pid = 1.

Процесс с pid = 0 - процесс, запустивший систему.

Процесс с pid = 1 - процесс, отработавший terminal.

Все процессы, запущенные на данном terminal, составляют терминальную группу.

Ubuntu есть процессор - посредники (user int).

```
include < stdio.h >
include < unistd.h >
int main(void)
{
    pid_t childpid;
    if (((childpid = fork ()) == -1))
        perror ("Can't fork");
        exit(1);
    else if (childpid == 0)
    {
        /* child */
        printf ("child: pid=%d, ppid=%d, parent process %d\n",
            getpid(), getppid(), getpid());
        return 0;
    }
    else
        /* parent */
        printf ("Parent: pid=%d, child=%d, gid=%d\n",
            getpid(), childpid, getgid());
    return 0;
}
```

Обычно помехи имеют pid на единицу большее pid процесса. *Виртуальные помехи используют будем sleep*

Вашеёное понятие Unix обе. группы процессов, потому что процессов одной группы могут пользоваться одни и те же сервисы (базовое средство для обеих инф. о созданных DC).

Вашеёное содомие - законченные процессы.

03.09.2022г. 1,

Продолжение

Системный вызов waitpid()

```
int status;  
wait(&status); // статус завершения потоков
```

Статус завершения потока контролируется с помощью системных макросов.

Контроль за событием инициирует механизм надежно. int 2h - прерывание

т.е. в совершенном числе виртуальных памяти страницы, где процесса создается виртуальное АЛ, которое залогируется на страницу. Каждая описывается дескриптором.

Компьютером именем архитектуру Intel Ньюисона (см. лекцию), согласно которой выполняется момент первого кода, находящегося в оперативной памяти.

Расширенное же не является обозначенное в них. При запуске программы из файла превращается в процесс. Но-запускаемая программа должна выполниться, нужно загрузить в ОЗУ хотя бы страницу (однако это 3 страницы: DOS, код, данные, стек).

Страницчная память - поддерживается управление виртуальной памяти страницами по запросу т.е. когда генератор процесса загружается в память, когда процессор обращается к соотв. странице в реальное физ. адресе, работает башня.

exec() переводят процесс на АЛ программы, имена которых указаны в параметре. (см. след. слайд)

Книга: Вахидин "ЧИХ ИЛЮМРИ"

03.09.2022г. 1/р.

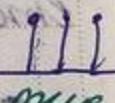
Прерывание 8 (interrupt)

Все современные ОС основаны на системе прерываний. В наст. время вносятся 3 типа прерываний: системное бояска (программные прерывания), исключений (исключ. инициации), аппаратное прерывания (обозначается именно словом interrupt).
int 21h - прерывание DOS.

Аппаратное прерывания делится на: прерывания от цикл. таймера (единственное периодич. прерывание в системе, в разных ОС период разный, называется таймокас.); прерывания от внешних устройств (харк. = прау. + ОП; основное - вспом. устройства); прерывания от действий оператора (в харк.). И появляются они, когда включалась консоль оператора, с помощью команды bootrec на программу, преследора: Ctrl+Alt+Del^(win) - Task Manager, Ctrl+C - заверш. процесс.

DOS - 16-битная ОС, ком. устанавливавшаяся на комп. с прау. 8086.

Таймер.

В состав любого харк. входит специальный микросхема - тактовый генератор, генерирующий импульсы (таймэр).  Удвоение частоты не добавят, поэтому микросхема обогащает работают по переднему/заднему фронту. Из этих импульсов с помощью микросхемы создается короткие импульсы  мк. Тик приходит на контроллер прерывания. (В старых ОС - один контроллер прерываний на шине памяти, когда подключение входит с устройств). В DOS (и далее) так работает $\approx 18,2 \text{ раз/сек}$.

DOS - однозадачная ОС.

В результате формируется вектор прерывания, с помощью кот. обращается к табл. вект. прерываний, где наход. физ. адрес обработ. прерываний (segment + offset). Так. одн. система не может адресовать обработчики прерываний.

Вызываемый int 8h будет в реестре V86, "режима языка DOS" - след. режима запущенного реестра.

16bit-реализм реестра. 16bit-программа - программы реестра нового реестра. 64bit хакер. могут работать как 16bit-реестра.

В 80. exe нул

поста наимен

тить временно

делает то,

Рутинные

1. Их врем

время. Их

вместе

ОС (ОС - пр

При за

"Python - это гибкость", потому что отдаёт предпочтение гибкости. Используется в программах потому, что более разработчикам удобно драматично.

Чтобы получить сог в виде ассемблера, нужно использовать команду汇编器. TI0 - source - sr.exe Выг получаемых данных:

1	2	3	4
Agnes	Main	Kog	Kon-
kog.	assem.	scenar.	metor
			:sr.exe

В коде видим обращение к нормал. (дискового и компр. прерываний).

запись

регистров

На о

номина

таймер

ко

с

В косм. существует два адресных пространства - ОП (унив. память) и норм (порт-адрес).

Ни единой адр. прообр. портов не хватит.

Взаимодействие процессора с внеш. устройствами возможно с помощью специальных устройств.

- memory mapping - работает видеокарта, исп. коякого подр. с памятью (mem)

- I/O mapping - обращение к внеш. устройствам через норм. I/O.

Контроллер lock - вспом. обратить внимание на назначение контроллера и принцип его использования. (послед?)

3.

Все это нужно указать beginning address и ending address: Каждое наименование иret, обычное подпрограммное - ret. / обратить внимание). Она заканчена по jmp, т.к. есть such code, который делает то, чего нет не знает, его пропускает.

Регистры 8-го прерывания

① Инициализация реального времени - подсчета текущего времени. Инициализация происходит по тайму.

* В состав хол. входит энергонезависимая CMOS-микросхема, управляемая от "материнки" (батарейка). CMOS-микросхема запускает OS (OS-программа, лежит на диске, но файлов не называется).

При запуске синхронизируется значение с CMOS-микросхемой в область данных BIOS.

② Инициализация счетчика времени до отключения ~~операции~~ дисковода.



← ^{диска}. Вставляется в дисковод.

Самые первые компьютеры не имели винчестера. Чтобы прочитать/записать данные с/на дискету, нужно разобраться до определения винчестера.

На оперирующей I/O необходимо комманду разобраться винчестеру, назначив нужную команду винчестеру и от дисковода выделена на винчестер. Это отведенное время.

Когда счетчик достигает 0, в карту дисковода поступает команда (в контроллер дисковода).

Существует один винчестер штотрека дисковода.

③ Входов пользовательского прерывания 1Ch.

В чистом DOS 1Ch - запущена, когда там есть.

1Ch вызывается коммандой так, на нем раб. регистр. программы.

напр., Norton Commander, по нашему ~~данным~~ сожалению.

Бофор ICh. возможен двумя способами (в зависимости от parity⁽¹⁾ queen).

Subroutine - базовое название подпрограммы.

Она запускается прерыванием.

По испр.: отрывок в будущеме буде (схема алгоритмов - 8-го прерывания и subroutine).

Несмотря на то, что существует ч. Планкенбахса, но её курс не читается. Всё же книга: С. Рана, статьи Чиркиева, М. Анонса.

Требования к схемам численик.

В алгоритмическом языке присутствование команды каскада.

В первые испр. (?)

05.09.2022 г. лекция. История вычислительной техники

Компьютер - программно-управляемое устройство.

В XIX в. Т. Бэббидж предложил концепцию программного управления.

Он поставил задачу построения универс. вычисл. машин, которая должна была: быть механической, исп. гелийм c/c, принимать каскада через перфокарты.

С помощью Бэббиджа сконструирована А. Чарльзом, которая считает первыми программированием (нап. определение год. восчисления чисел Бернули).

В XX в. возникла задача ускорения вычислений в военных целях. В 1940-х существовала британская лаборатория, занимавшаяся таблицами стрельбы, в 1940-х возникла задача

$$\partial \varphi(\varphi(y)) \cdot |\varphi'(y)|$$

построения
Эта машин
алгоритм
ENIAC
+
I AL.
I A C U
E N D O
L I N T E R
E C T R O N I C
I C P U T E R

ввода
(арифметик
Здесь п

число на
В это
числу В

делил
на
число

и число
делит
на

число
и число

и число
и число

построения машины, которая автоматизировалась до впечатления.
Эта машина получила название электронной цифровой вычислительной машины.

ENIAC ←→ Mark I — 1940-е г.

ENIAC
I I I I I I
E N J A +
E U M N O
L M T P M
E B E R P K
C R G T B
T I R A T R
R O N L O P
I C I

ENIAC была боя. машиной, в которой использовалось электронное лампы и коммутационное панели, последние присоединялись для управления — задавались ~~бесконечные~~ порядок вычислений. Для ввода и вывода использовалась перфокарта, где ввода — ячейки АИПР (арифметико-цифровое накапливающее устройство).

Здесь появилось понятие bug (туск) по неисправлению, приведшему к сбою машин и нарушавшему работу машины.

В это время Дон Нейшле создавал совершенно новую машину. В 1945 г. дон Нейшле опубликовал доклад, в котором описано основное концепция и принцип работы вычислительной машины. К этому времени существовало знание о двоичной системе счисления, сформированное Лейбницем. В 1946 г. дон Нейшле опубликовал статью, в которой описан цифровой организующий вычислительной машины, которую называли компьютером.

Фраза «цифровой компьютер»

1. Универсальная боя. машина должна содержать устройство арифметики, памяти, управления, связи с оператором. Так как машина вычислений работа не зависит от оператора.
2. Необходимо, чтобы машина имела функции не только цифрового, но и аналогового, т.е. машину с плавающим запоминанием.

ком. дешифровка профиль блокчейна.

3. Если представитель команды пишет в память число, когда и машина считает значение команды от данных, то будущем хранить профессия в памяти.

4. Помимо памяти, где команда пишет цифру 60, команда способна восстановить хранящееся в памяти команды сопле - оператора, управляемую.

5. В машине г. б. Арифметический орган - устр-бо, способен +, -, *, /.

6. Данные существовать устр-бо в виде битов под контролем оператора и машины. Машину можно было электрически, использовать двоичного в/о и последоват. ком. операции. Соблюдение варианта. (?)

7. Программа и данные хранятся в памяти в одном и том же пространстве. Команды расположены последовательно. Процессор восстанавливает программу ОП. Исп. адреса.

В состав процессора включается стек коман. Команда не содержит адреса след. команды. Последний восстанавливается.

II поколение.

В 1950-х появляются новые вычислительные базы - диоды и транзисторы, полупроводники. Развивается и блокинее устройство. Появляются математические библиотеки.

Первое программное реализовалось в машинных языках с использованием абсолютных адресов.

Первой языку программирования - это ассемблер. Позже наявутся языки высокого уровня.

ENIAC - первая серийная

Умодор нечто

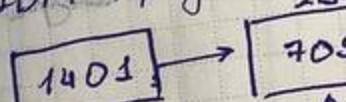
IBM 1401 - серий

10000 имп/с.

оператора,

расчет

IBM представил



709

услуги

использования

(и.е. ОС не требуется)

Создавалась

Появился язы

III поколение

Появление

1960 г.

многих обработ

и т.д.)

к III покол

кин. машин.

типов проп

Рядко уб

за честную

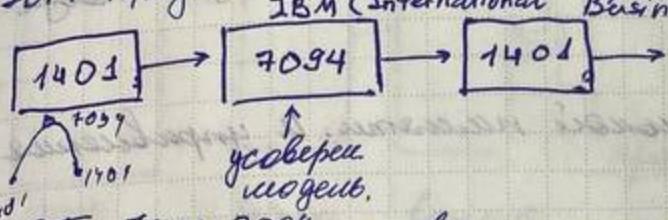
го контента

ENIAC - первая серийная В.м.

"Чтобы нечто можно стало серийным, ему нужно документация".

IBM 1401 - серийная вонгисит. машина от IBM, было продано 10000 штук. У таких машин появился обменивавшийся перенос - оператора, в зависимости которого вводилась загрузка программы.

IBM предложила вариантом работы с машиной IBM 1401:



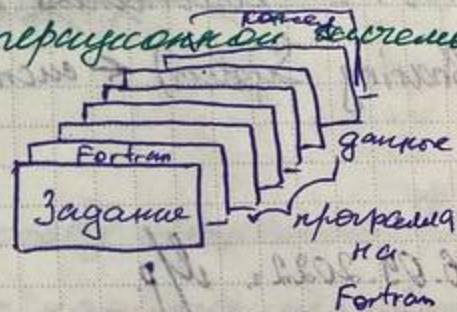
ОД IBM 7094 позволяла загружать несколько программ, оставляя загата переключение программ, которую вонгисит на специальное ПД, которое называли оператором (т.е. ОС появившееся в 1950-х).

Создавалась пакет - набор программ. Появился язык управляемых заданий.

Летом 1969 г. появилось интегрированное шистроохране, в которых обединились несколько элементов (диоды, транзисторы и т.д.).

К III поколению относят микропроцессорные вонгисит. машины. (при этом исчезает вспомогательное оборудование с архитектурой языка Нейблсац).

Резко увеличивается объем ОД (у-за ширер. exec.), за него идет о выполнении какой-либо программы от начала до конца потеряна актуальность.



В IBM 360 реализована идея распараллеливания под Синхрон.
усл. Управление всеми устройствами было на себе один процессор - канал. Это привело к падению производительности системы из-за перегрузки, т.к. отсутствовало необходимое информирование процессора о завершении операции ввода-вывода.

Нужно было реализовать специальное состояние процессора, в котором она не получает процессорного времени, но отдаёт завершение операции ввода-вывода. (Процессор-то 1).

Возникает идея виртуальной памяти и управления виртуальной памятью.

Unix. - открытой исходной сог.

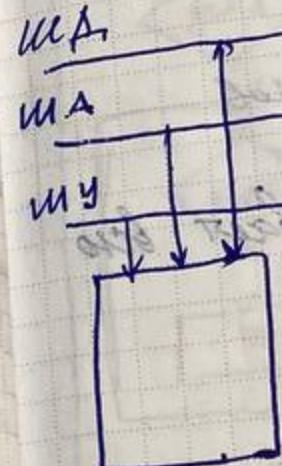
1964 г. - ~~кооперативная~~ MIT создает ОС CTSS (Compatible Time Sharing System) \leftarrow система пакетной обработки

06.09.2022 г. Илья

Уже в IBM 360 реализование распараллеливание. Управление устройствами - канал - специальное устройство, получающее каналную программу от процессора и выполняющее её.

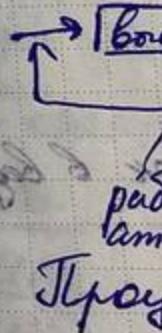
Контроллер - программируемое устройство. Помимо команд от ~~канала~~ процессора. Контроллер может находиться на устройстве, адаптере - на шине памяти.

Синхронизацию прерываний от внешних устройств можно добиться параллелизации процессора о завершении операции ввода-вывода.



IRQ - Int
IRQ0 -
IRQ1 -
IRQ(12) -

Все н
есор в



Процес

В Р

использование
на себя органов
и гормонов
и индифферент-
ного.

нашего про-
стого бы-
ка-бояка

рабочий

possible Time

Управле-
ние компь-

юнктора.

оника

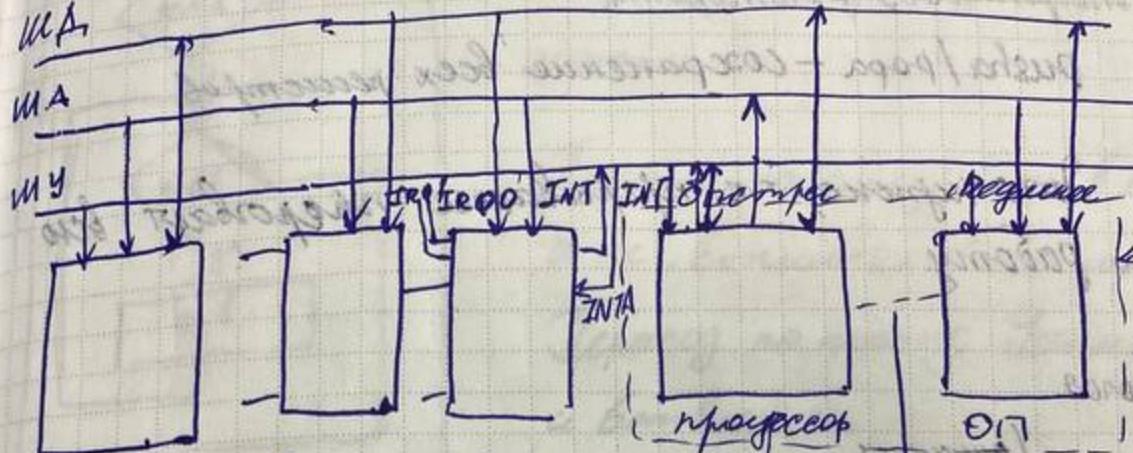
на зер-
ной блог

Синхрон:

1. Данные: команда и данные
2. Адреса
3. Синхрон управления

концептуально:

по каждому циклу
свой тип синхра



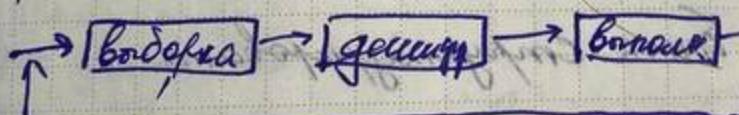
IRQ - Interrupt Request \Rightarrow int#h

- IRQ0 - системный таймер.
- IRQ1 - клавиатура (DS12).
- IRQ12 - мышь (PS12)

INT
INTA

ваше действие
через отдельную
восстановительную
функцию.

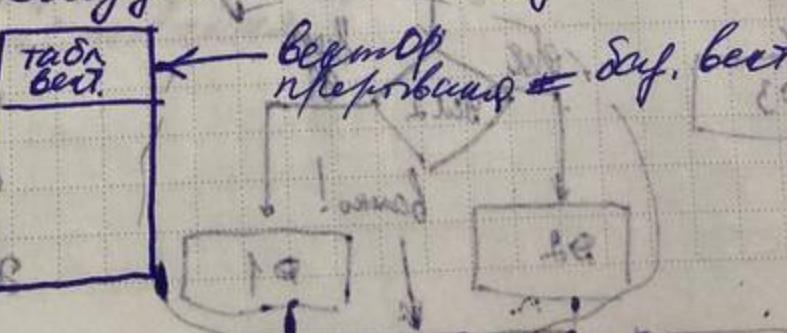
Все прерывания приходят на специальную линию процессора. Процессор в конце каждого такта проверяет наличие прерывания.



работа
компьютера

Процессор отправляет конфигурацию карты INTA, посып-
кий восстановление вектор прерывания.

В DOS вектор определяется как смещение в таблице
векторов прерываний.



$$2^{20} = 1 \text{ МБ} = 1024 \text{ КБ} = \text{fffff.}$$

READ, WRITE, INT, INTA - сигналы управления

В современных системах MSI - Message Signalled.

Interrupt. - о прерывании информирует сообщение.

Сохранение аппаратного контекста.

push es

push ds

push ax

push dx

pusha/popa - сохранение всех регистров.

DOS
сигн.
тактико
ре.
ремо
меновка

Это нужно, т.к. прерывание прерывает всю
работу.

pop dx

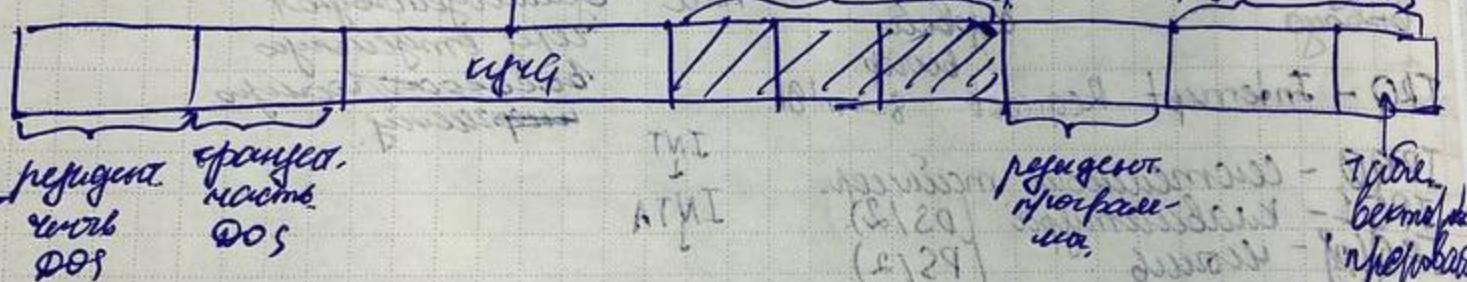
pop ax

pop ds

pop es

(диспетч.
режим)
сигн.
режим

режим
режим
режим



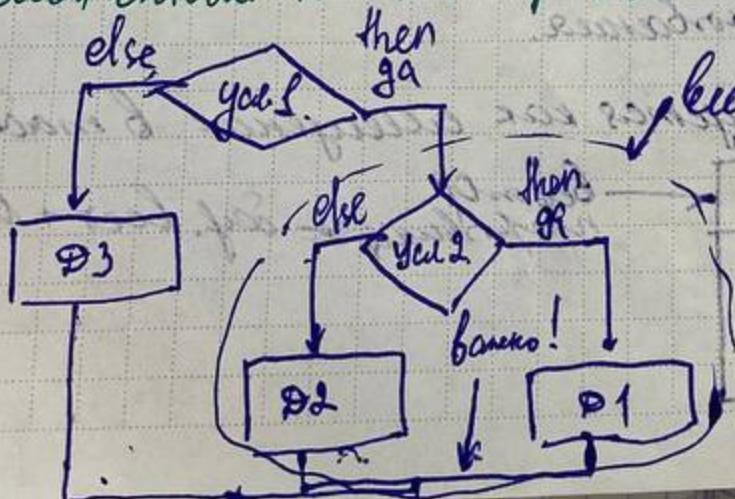
Прерывание обновляется чаще всего когда засыпает устройство.

Unix/Linux - куча I/O, Windows - LFS.

Схема аппаратная замена бомб. Структурированная.

Теорема: чтобы аппаратное менять бомбы предотвратить в базе
имеет конструкции: сдвигование, вставки, удал.

Выясненный if - как правильно рисовать?



Бесконечный if

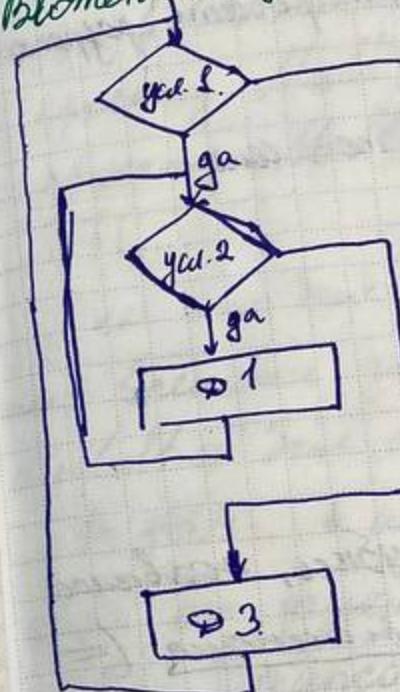
не реагирует, нет!

мало прошёл шаг!

запись преследует один
регистра (но не меняет)

if (y < 1) then
 if (y < 2) then D1
 else D2
else D3;

Блок-схемой удел -



10.09.2022. Ур.

20h - известной
к компьютеру

ИТА

ИТА

16g

ИУ

16g

ИУ

16g

ИУ

16g

ИУ

16g

ИУ

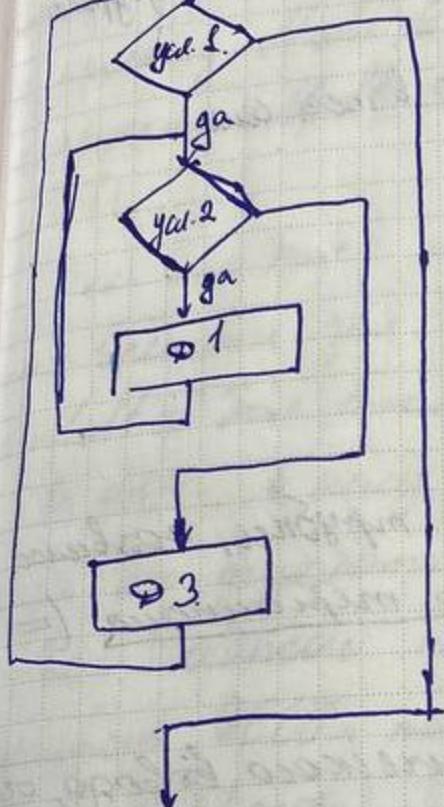
16g

ИУ

16g

if(yee₁) then
 if(yee₂) then φ₁
 else φ₃;
 else φ₂;

Блок-схема узла - как правильно рисовать?



Стрелки сбрасывают блоки обрабатываемые не забыть.

Надо подробно: в паре - команда, а не "вспомогатель" и т.д.

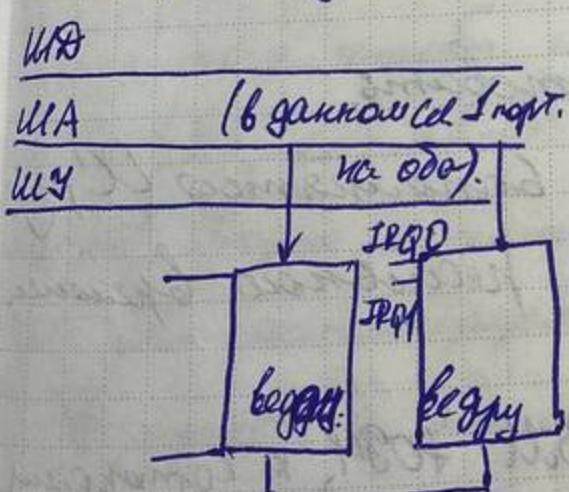
Перевод по просьбе: Установлен флаг... и бомбка.

Times New Roman ≈ 12. + 10px.

10.09.2022г. МР

20h - чувствительный порт

К контроллеру прерываний обращается по порту.



если 20h, 20h - сброс контроллера прерываний (шаски прерываний) маска прерываний запрещает некоторые прерывания.

команда приходит на бессущий контроллер, и это приводит к сбросу шаски прерываний.

(сдел. 1/p 06.09.2022)

И/р - часы 2. Функции обработчика системного таймера в системе
разделения времени.

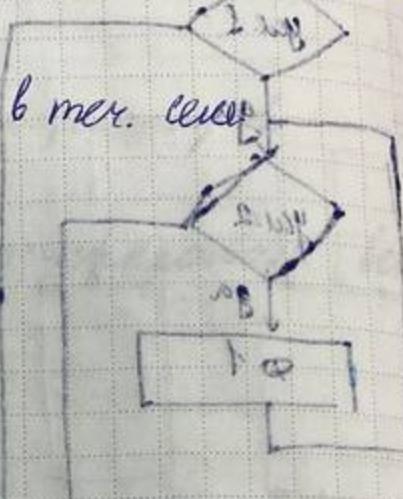
(небольшое ИРПУ)

Пингстайл - манипулятор с устройствами ввода на буферной памяти
Системное разделение времени должно гарантировать разработчику
время ответа

Написать фрагмент ИРПЧПУ. Пингстайл имеет в тек. сист.

Сообщение, Ресурсы, Установка

Входы



12.09.2022. Лекция.

Когда появилось ~~бесконтактно-механическое~~, трубки, ~~заявлено~~
возможность подключить удлиненное терминальное (=
экран + манипулятор).

Как появилась возможность динамического ввода, от-
ношение к интерактивному программированию.

Возможно, что человек не может отдать неопределенно
до долго, Т. Р. решив идти о ежедневной рутинной работе (из
базисе З сен.). \Rightarrow ОС г. Б. построена так, что у нее имеется
гарантированное время ответа.

Процессорное время ставится квотированием

На IBM 370 одновременно можно было иметь
несколько типов: пакетной обработке, реального времени,
разделения времени.

ОС CTSS разрабатывалась где JBill, Yogi, к которому
через терминальное окно подключалось удлиненное терминал

1965 г. - начало проектирования системы Multics. (Multiplexed Information and Computing Service).

В эту同一ную подконтрольную группе Bell Laboratories, в которую в 1963 г. присоединяется Дэниел Рамзи, который подключается к проекту Multiple Access Computer (MAC), а позже переходит к проекту Multics.

...BTL...

Все это было направлено на создание систем реального времени бывшего подразделения General Electric. GE645. Последний компьютер с Multics был введен в эксплуатацию 31.10.2000.

В 1969 г. в одну многоцелевую компанию Bell Laboratories вышла от разработки Multics.

Кен Томпсон писал игру "Space Travel," и ее хвастал ресурсов BEOS, поэтому он обратил внимание на PPP-7 (8 kB, 18-bit слова). Писал на макросы BE, перекодировав их на перфолентах.

К. Томпсон заново начал писать ОС под PPP-7, T.R. под g. б. имитатора, и начал с машинной подстановки. И начал писать такие образцы Unix \Rightarrow Unix ^(также, как).

Для того чтобы написать ОС, понадобились согдие гбы (B), который был интерпретатором и не имел инструкций. Поэтому началась разработка языка Си, которая была уже кастомизирована и предназначена для.

На языке Си написано ядро ОС Unix.

"Когда пишете большую программу..., ее дальнейшее развитие определяется, с которой работаете. Это невероятно

на Fortran, без структур данных".

IV поколение

Это поколение больших интегральных схем, которое ранее пользовалось обеднением нескольких одинаковых ячеек, а потом и в виде одной микросхемы.

Даны, к н.в. появившиеся сверхбыстрые интегральные схемы.

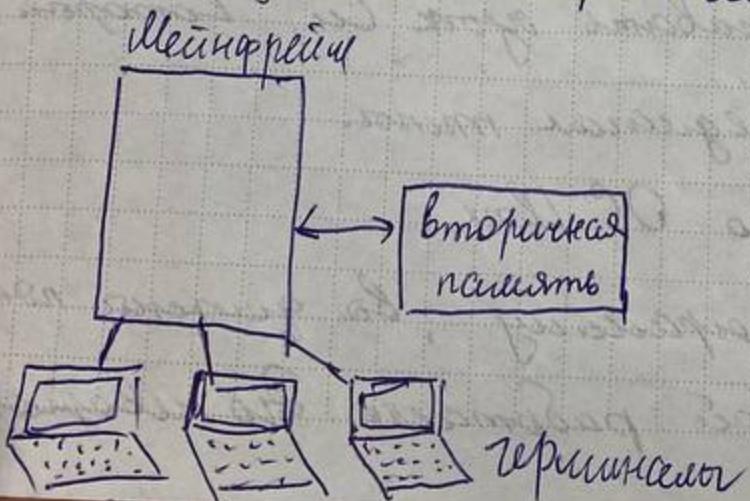
Процессор — это программируемое устройство управляемое от при помощи компьютера.

Для борьбы с быстрыми компьютерами используются ядра (L1, L2) при наделенном ядру, третий (L3) предназначенный для всей системы.

Сейчас стремится построить компьютер на основе сетевой архитектуры.

В курсе ОС видим:

- управление процессором;
- управление памятью;
- управление данными;
- управление внеш. устройствами;
- взаимодействие параллельных процессов.



Гарант Столбу

проект ОС GNU (1984 г.)

1985 г. - фунд. Бесплатного ПО

FSF - Free Software Foundation

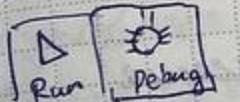
Текстовой редактор
документ Word

1991 г. - первая Linux

1992 г. - Система

Система

на ядре PDE



Операционные

управляемые

яущих эти

Ресурс -

если есть

Основные

иные ресурсы

Linux

если нет

К ресурсам

объектов

операторов

системах

не могут

Собраны

текстовой редактор GNU Emacs, GCC компилятор.
Линус Торвальдс.

1991 г. - первая версия ядра Linux для Intel 80386; распространяется под Linux согласно лицензии GPL.

1992 г. - Стабильные исправления Linux для завершения собственной системы.

Система GNU/Linux — многофункциональная ОС — большая написанная на языке PDP-11. Система должна поставляться с открытым кодом ядра.



Операционная система — это комплекс программ, ком. сокращенно управляет ресурсами большого числа устройств и процессов, использующих эти ресурсы при выполнении задач.

Ресурс — любой из компонентов большого числа и предоставляемое им функциональности.

Основная задача ОС — управление процессами и ведение списка ресурсов.

Linux позволяет получать информацию о ресурсах, принадлежащих процессу, — программы в стадии выполнения.

К ресурсам системы относятся процессорное время и ^{объем} оперативной памяти, объем вторичной памяти (на внешних запоминающих устройствах). В многопоточных ресурсах также имеются потоки.

Современное ОС делит время многоцелевым, системами реаги-

202

чение времени: в памяти может находиться несколько программ одновременно.

Применимостью к ОП часто говорят «дружеская».

В процессе разработки вспоминают ОС Видов:

- однозадачное пакетной обработки;
- многозадачное пакетной обработки;
- системные разрешения времени
- * (однозадачная ОС DOS, взаимодействие с программой инициативное);
- ОС реального времени.

ОС реального времени всегда специального назначения.

Обработка информации реальным времени - оптимизация и упрощение

Бол. системах, в тече., обслуге: обнуливание некоторого времени процесса, и зависящего от времени, состояния.

Режим реального времени характеризуется временным

требованием времени: $t_{\text{req}} \leq \frac{\text{Ряд. под.}}{\%}$ запрос

бюджет

Помоки,

типа

Пригод

Трансляция таких систем осуществляется системой управл. 1) система

использования и контроля технического процесса, системой управления 2) иссле-

дования и т. д. аппарата и транспортировки представления.

Известной системой реального времени является QNX (установка

бесплатная).

Сеймур Крейн — создатель первых коммерческих зондажных

суперкомпьютеров. Начало своего деятельности в 1957 г.,

а в 1960 г. на рынке были представлены 48-разряд. Компьютер

акции

1000

на Бе-франшайзерах — CDC 1604. В 1962 создан первый суперкомпьютер CDC 6600, который имел быстродействие 3 Mflops .

В янв. 1972 г. Крей создал фирму Cray Research, в которой

создан самое быстрое суперкомпьютеров

Cray-1 и Cray-2.

первый
комп.
успехи.
Север.
таким

Long
Instruction
cycle?

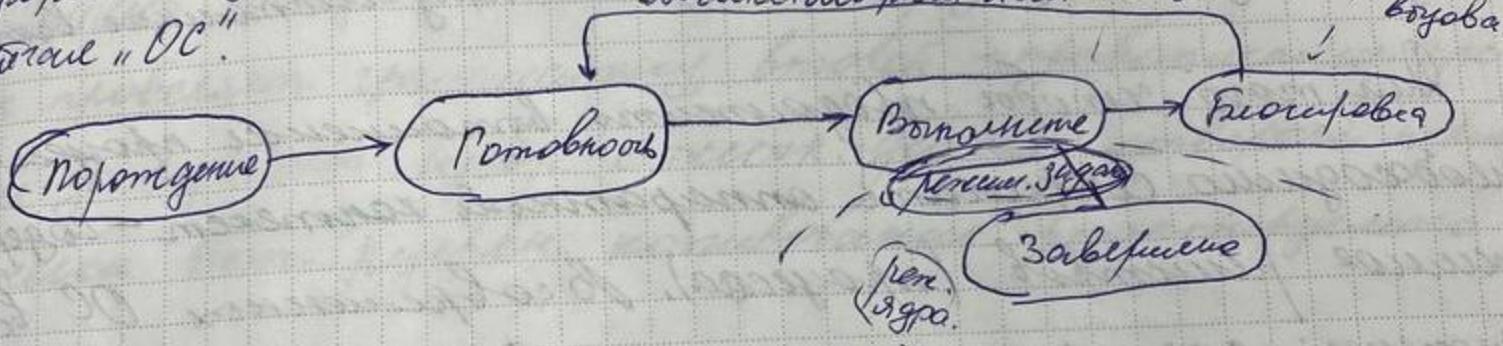
"Управляемые процессорами" — потому что процессор в микропроцессорных системах проектируется вперед к процессору за брешением.

Разработчики создают диапазон состояний для определяющих управляемых процессорами. Выделяют состояния и формализуют их.

система
с привлечением, с вовлечением.

в ре-
зультате
войска.

Ядро ОС.



26.09.2022 г.

Лекция,

Три вида
переключающих систем в реальности:

1) системное выключение; (программное прерывание)

2) исключение;

3) аппаратное прерывание

Система предоставляет привилегии над функциями, часто наз. API. Вызов функции приводит к переходу в процесс здра.

Число единиц \rightarrow управление (пример: информационное управление).

Число единиц \rightarrow неуправление (испр.) (испр.) приводят к аварийному завершению процесса? (если на 0, ошиб-
ка агр.)

Аппаратное прерывание → прерв. иниц. таймера
→ прерв. от упр. блока / ввода
→ прерв. от внешней операции

Пог ресурсов. когда производится чистая процедура, не исодействующая сама себе.

Помок - часть кода (кир) которая инициирует параллельно с другими частями кода. Задача, которая инициирует параллельно с другими задачами приложения. Помоки нужно запускать где надо, чтобы прилож. корректно работало.

Ни одна система не позволяет напрямую обращаться к упр. блока / ввода (запуска инструкций), все через системное ядро.

Для того чтобы продолжить выполнение процесса, необходимо сохранить аппаратной комплекс - содержащие регистров (процесса). В современных ОС всё сконструировано: где процесса создается виртуальное адресное пространство (создание соотв. структур данных машинуя структуру - машинной структур, содержащей адреса страниц).

Кроме аппаратного, г.д. сохранен полной комплексом - аппарат. комплекс + инф. о ресурсах, используемых процессом (адр. таб. страницы), инф. привязывается в дескрипторе. Несоответствия загруженой адр. таб. страниц, адр. блока, загруженной (уже несоответствия) полной комплексом.

Когда проиш. формирование / изменение квант, происходит восстановление полного комплекса, что приводит к некоторому переносу

бум доп. ресурсов системы.

Для того чтобы сократить эти накладные расходы и из соображений безопасности парашютного выпадения ход, боя введен потоки.

Некоторые приложения могут быть логически обоснованы разделяясь на потоки, некоторые - нет. Система не может иметь способ выполнения, поэтому введен поток однозначно и исполнительное при.

Потоки разделяют ход и данные процесса, но есть у каждого потока свой.

Пример пяти потоков: текст. редактор: первый поток проверяет грамматику, второй - правописание, третий загружает изобр. с текст. диска, четвертый переводит текста ком. редактируемого рабочего.

Всемирные исполнительности:

- 1) Отвественность (responsiveness): один поток может одновременно реагировать в то время как другие блокированы / или блокированы одновременно;
- 2) Разделение ресурсов (resource sharing): по умолчанию потоки разделяют ход, данные и другие ресурсы, что позволяет нек. заграждениях взаимодействия одновременно; потоки могут исп. один и те же ресурсы, где процессов это требует спецификации син. блоков;
- 3) Экономичность (economy): создание и управление потоками (переключение контекста) яв. занимает

быстрее, чем у процессов (переключение
иначе рора intel брэндов)
 и избирает сохранение места аппаратного
иначе интерфейса.

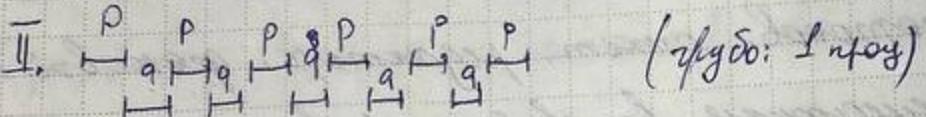
4) масштабируемость (scalability): использующее шинные
 процесорной архитектуру (одноплатной процессор с оди-
 наково процессора). Помимо быстроты в отработке, это
 важно для современного потребителя.

Уровень наблюдения.

I. Помадильное выполнение.



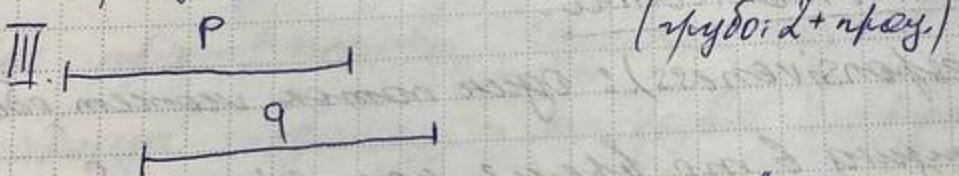
(одноданные системы, DOS)



(многоданные системы)

Квантовальное выполнение.

- каждая командой программы выполняется последова-
тельно, но подсов. консульт., что параллельно;
- процесса взаимодействуют друг с другом.



(многоданные системы)

Реальное параллельное выполнение.

- происходит иногда.

Общее представление о последовательных параллельных про-
цессах.

↑ дает доп. информацию о исполнительности

Проблемы программирования при исполнительности.

Возможность 5 областей, в которых исполнительных
приводят к нов. проблемам:

- выделение за-
дачи и находи-
тельство;
- балансиров-
ание обеспе-
чения обес-
печения;
- разделяемое
действие;
- зависе-
ние от ре-
альных обес-
печений;
- нестандарт-
ные ситуаци-
и радиаль-
ных обес-
печений;

Помехи

могут к обес-
печению на-
ходиться
данными

Помех

Сущес-
твует:

1) дата-
шилки

рпн.
обрабо-

- введение задач для параллельного выполнения (нужно убедиться, что задачу и находить где она, которое могут выполнять параллельно);
- балансировка потоков (поиск задач для параллельного выполнения, которые обеспечивают равное загрузки, потом не надо тратить на приводимую задачу время swap());
- разделение данных (data splitting) (нужно преобразовать единогодственные потоки друг с другом);
- зависимость данных (data dependency) / если одна задача зависит от рез-та другой, то они г.д. синхронизируются, чтобы обеспечить доступ к данным в нужном порядке);
- межпотоковые и отладка / возможен бессмысленные ситуаций, например, гонки (race conditions) - доступ параллельных потоков к разделенным данным, пересечения).

Потоки, так же как и параллельные процессы, могут обращаться к общим данным (**разделению**). Поэтому бессмыслицае одновременное обращение к общим данным, предупреждение. Необходимо обеспечить исключительный доступ к разделенным данным.

Типы параллелизма

Существуют два разных типа для распараллеливания на языках:

- 1) **data parallelism** - параллельность по данным: деление данных между множествами задач (потоками, threads), которое выполнением задачи для прп. чисел данных. (процессор - деление обращений на участки и обработка участков);

2) task parallelism - распараллеливание по задачам / функции на
значение: деление на разные задачи, которые можно
выполнять параллельно (+ одновременно) на разных машинах
(simultaneously).
один обр.

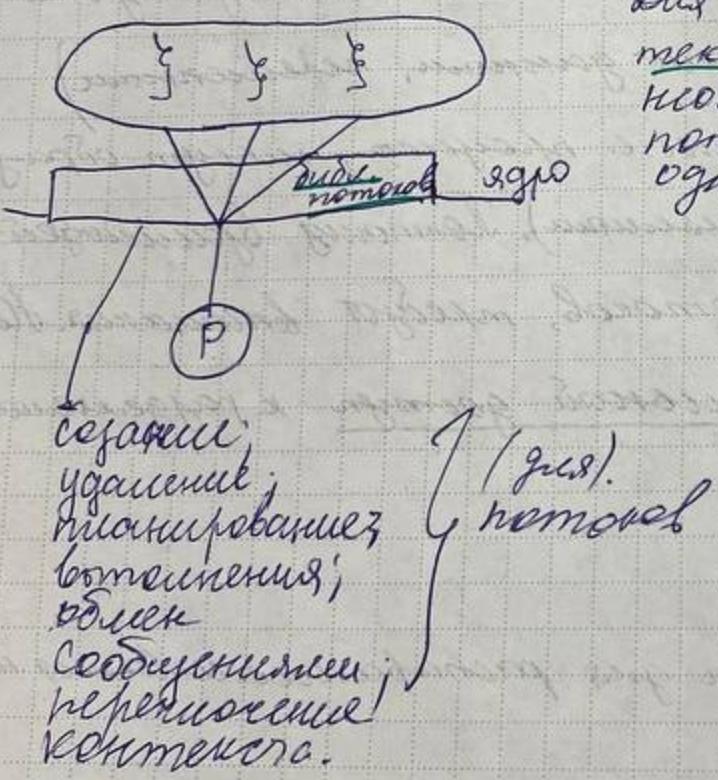
Модели многопоточности (**)

Существует два типа потоков:

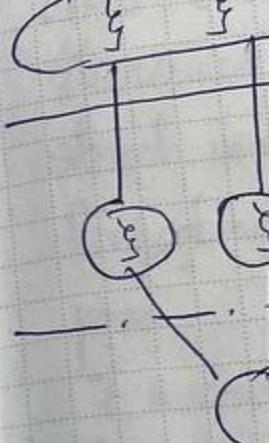
- потоки уровня пользования; } всё время
этого
крупного
- потоки уровня здравия.

Потоки уровня пользования.

Программный код делится на потоки, но о них системе
ничего не известно.



Для управления существует один поток, представляющий
исходящее сервисное. Для этого потоков не существует, это
один процесс. (**).



Переключение в режим здравия выполняется крайне часто, и это
крайне затратное действие. Идея бесперебойных потоков в со-
хранении этих действий.

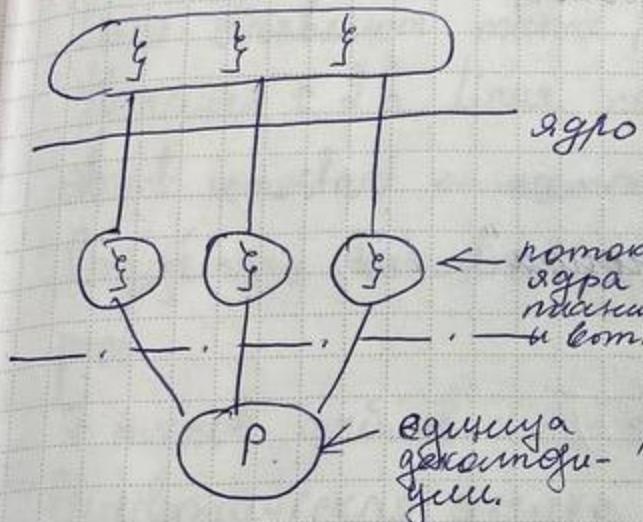
Проблема исчезает, когда поток выполняет системное

$f() \cdot f(y)$

т.к. о
появле

если поток запрашивает блок / беклод, блокируется
бескрайне в очереди (+). В системах разделяется времена исполнения
 потоков не учитывается, по истечении квоты времени про-
 цесса переводится в готовность. С исполнением про-
 токолов занята страницы базы данных помех. Вы-
 ширши - условной.

Потоки ядра ядра (в Unix / Windows), Linux.



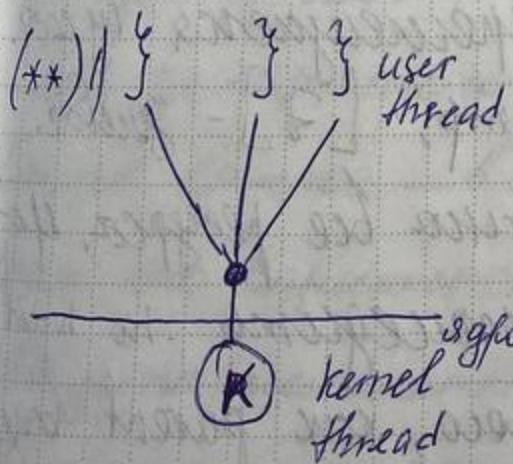
значит, что поток в Linux эквива-
 лентен процессу.

относит
присвоит
единст.
дающего
процесса.

называют
"поток
выдаёт
аппаратном
комплексом"

поток выдаёт регистрации имен

В системах нет дисциплины миграции, которая ба-
 рвала бы потока в очередь с "группировкой" по процессу. Вре-
 димости в очереди — потоки разных процессов. Если у по-
 токов один процесс, переключается аппаратный комплекс, если
 разные процессы — переключается ядерный комплекс.

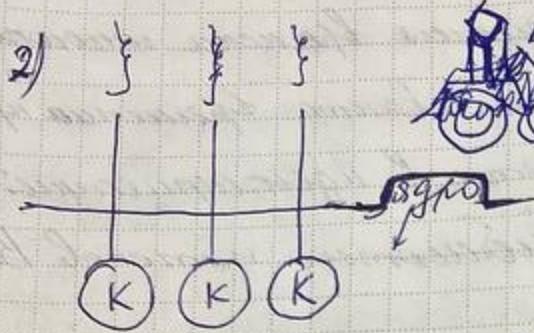


"один ко многим"

управление с помощью базы данных потоков
 уровня пользования; гарантируется то же
 что, пока не запрашивается иск. вызов. (*)

T.c. один поток ядра бол. на одн. процессе, может же
 несколько распределенных возможностей на нест. процессах

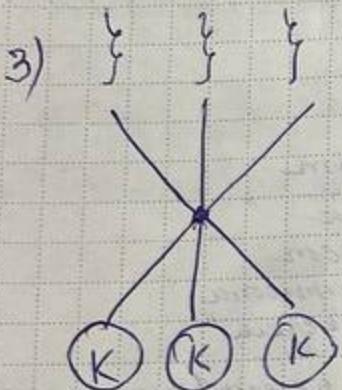
Green threads for Solaris and GM Portable Threads.



mis82
"ожидают"
"ожидают"

если нитка запрашивает сервис и не получает отклик, она блокируется, а группы потомки продолжают свою работу.

считается, что исчадающие процессы блокируются, поэтому при передаче между ними обратившись к исходным ниткам.



"множество множеств"

Linux обзор
Наследство
struct mm_struct
Соединение
proc.

F mono
Синхронизация

10.10.2022
Помогающая функция
execve ..
приводим

Процесс
Самое

(если з
Когда
ем сущ

T.R. о
нем

27.09.2022. Исп. (исследование F)

У parent'а было fork и exec, у child exec.

Процесс, отвечающий за fork-fork, не может быть ему хиробан.

[Running/Runnable - процесс постоянно переключается в статус готовности. [T] - temp, [Z] - зайде.

В состоянии зайди у процесса отображан все ресурсы. Не все дескрипторы. (Чтобы parent мог отослать no wait)

Потомок может завершиться до него, как parent boyo был wait(), или smudo и зайди.

10/10/2022

Два членов это пасо, т.е. процесса запрашивает ресурсы: машину, ядро (4-х уровневая структура), и.д. и в здр.

Parent - а нет, а zombie есть \Rightarrow система не контролирует, если все wait \Rightarrow в загаре процесс не блокит повторное загорание.

struct proc, struct task_struct {
 Unix } Linux } поддерживает
 Linux } повторение.
 если откроет list-head,
 так будем два загорания
 next и prev.

Linux обрабатывает mutex (list-head, pid_t) для контроля.

Начиная с 2.6 Linux перенесли hog иконодермство struct mm_struct \leftarrow memory management.

Структура task_struct основную иконодермацию такого же, как proc.

7 типов файлов: d - directory; - regular; l - symlink;

Символическая ссылка содержит строку - путь к файлу.

10.10.2022 г. лекция.

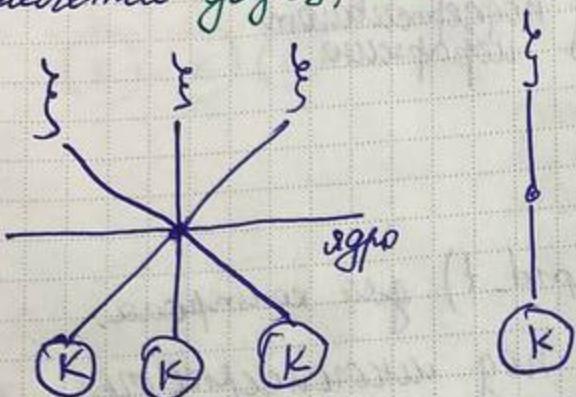
Приобретение не имеет ограничения на кол-во потоков. В числе „многие ко многим“ блокировка одного потока не приводит к блокировке остальных.

При распараллеливании трюк роль реального процессора самое важное - в отдельной потоке бордешть влог/свог. (если это возможно).

Квартильное \leftarrow в однопр. машинах; в них не делают синхронизацию по загаре (но группами), т.к. если загар может столкнуться с небольшим/периодичным присоединением.

Пример: там же текут. редактор, когда пользователь ~~логин~~
использует, подговаривает - наиболее вероятной в системе ("яркое")

Понятие параллельного кода "имеет ко идее"
активности двухуровневого кода:



Если в программе не создаются потоки, а merely зеркально
системы существует один главный поток.

Библиотеки потоков:

потоки → уровень языка.
→ уровень ядра.
(OS Solaris).

Любые системные вызовы в Unix предполагают библиотеки, бьющие
здесь для потоков.

Три основные библиотеки потоков:

1. POSIX PThreads.

Portable Operations System Interface (for Unix)

("Любой отсюда, свободен, иди туда")

POSIX стандартизирует системные вызовы, которые зависят
от поддерживаемого OS, чтобы I/O более переносимых.

POSIX охватывает потоки уровня подговаривания и уровня
ядра.

2. Win32 Threads

Библиотека потоков системы Windows (уровень ядра)

"ядрене потока бугоб")
3 Java threads.
(Java Virtual Machine)

```
int how-much-money-left-in-bank;  
int m;  
m=0;
```

QT создает потоки (фактически, QT Virtual Machine).
Python - просто однопоточное приложение, при этом же "использование, как потоки делают".

Нужно подготовить параллельную программу к использованию с потоками.

В и/р с динамич. вызовом `pthread_create()`.

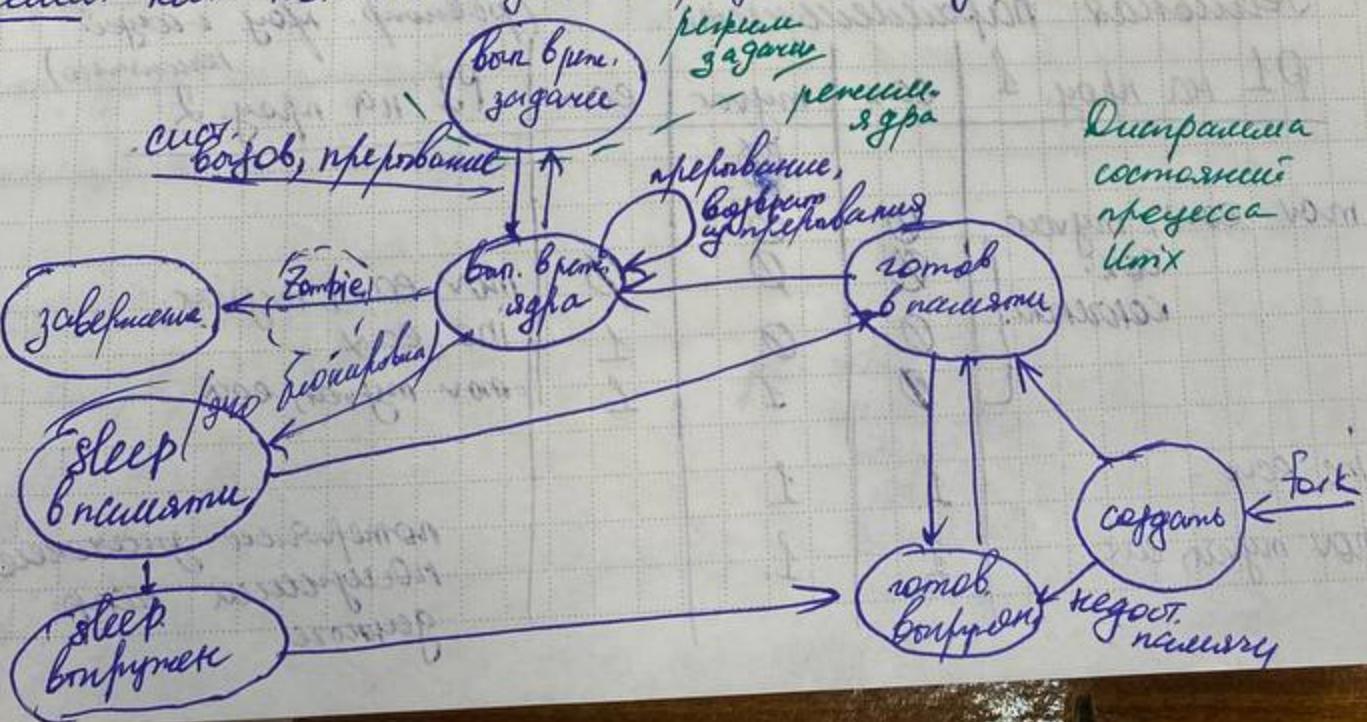
Все Java-программы используют потоки, кроме однопоточных (аналогичн. с QT).

JVM - основная часть исп. языка Java, в которой предусмотрено управление потоками (потоками). Любой класс, реализующий `Thread`, требует `public void run() {}`

OpenMP - набор директив компилятора для C/C++, Fortran, `#pragma omp parallel`.

```
{  
/* some parallel code here */  
}
```

Проблема: как потоки одного процесса получают сигналы?



Появление слова CDW привело к тому, что писать для страннички это решение проблемы генератора используемого в данной страннице.

Взаимодействие параллельных процессов.

Пример Пусть у бояка несколько фронтов. В момент t_1 фронт f_1 проходит границу на один и тот же временной отрезок $S = S + \Delta t$

P1: ...

$$S = S + \Delta t_1$$

P2: ...

$$S = S + \Delta t_2$$

Пусть P2 прошел $S, \Delta t_2$, потеряв счет.

P1 прошел $S, \Delta t_1$, ^{записав} ~~записал~~ в S .

P2 записывает в S , теряется Δt_1 .

Не имеет значения, решенная ли параллельность

P1: ...

mov eax, myvar
inc eax
mov myvar, eax

P2: ...

mov eax, myvar
inc eax
mov myvar, eax

Решение параллельности:

P1 на пруз. 1.

SMP-архитектур,

(равнопр. фронт садится

на место)

P2 на пруз. 2

	eax	myvar	eax	myvar
mov eax, myvar	0	0	0	0
inc eax	0	0	1	1
mov myvar, eax	1	1	1	1
inc eax	1	1	1	1
mov myvar, eax	1	1	1	1

потерянная засечка, полученная позже, дармоге.

Квазипараллельность. eax

P_1	$myvar$	P_2
	0	
<u>mov eax, myvar</u>	0	<u>бонусируется P_1</u>
<u>P_1 возвращает P_2</u>	0	<u>бонусируется P_2</u>
<u>CDR кратич. сна</u>	0	<u>mov eax, myvar</u> <u>inc eax</u>
<u>P_2 возвращает P_1</u>	1	<u>mov myvar, eax</u>
<u>inc eax</u>	1	<u>CDR кратич. сна</u>
<u>mov myvar, eax</u>	1	

также
проблема.

Переменная, подобное myvar, называется **разделенной**, часто говорят о разделенных ресурсах, связанных с обращением к тут же называемой **критической** (CR), просто код программы (PR).

Необходимо обеспечивать исключительный доступ к разделенной переменной (или критич. секции). Это обеспечивается методами блокировки/запирания. Если один процесс в крит. секции, другой не может войти в эту же крит. секцию.

Способы обеспечения исключительного доступа:

1. Программный
2. Аппаратный
3. С помощью семафоров
4. С помощью исключений.

Программный способ.

program exmp1;

flag1, flag2 : logical;
begin flag1=0; flag2=0;

P1: while (1) do; //проверка очищания

while (flag2) do; //установка очищания
flag1=1; CR & flag1=0; PR 1; end; //P1 end; //P2.

P2: while (1) do;
while (flag1) do;
flag1=0;
PR 2;
flag2=0;
PR 2;

parbegin
p¹, p²
parenend
end;

P2 проверяет, что flag1 собран, because в узле проверки помечено каким. P1 проверяет, что flag2 собран, because в узле проверки помечено каким, because в узле проверки, например, записано перенаправление, name f1, f2 : logical, then because; P2 установляет флаг, запускает перенаправление и заменяет значение, записанное P1. Продолжение.

Другой вариант

begin

P1:
while(1) do;
flag1 = 1;
while(flag2) do;
OP1;
flag1 = 0;
PQ1;
end; //P1

P2:
while(1) do;
flag2 = 1;
while(flag1) do;
PQ2;
flag2 = 0;
end; //P2.

parbegin
P1, P2
parenend
end

Тут же используется перехватом P2. Он устанавливает flag2 и теряет because. P1 устанавливает значение flag1 и удаляется в узле проверки всех because. (отбрасывается). P2 заходит в узле проверки и удаляется в узле because всех because и т.д. Это классический патч (patch). Каждый процесс передает обработкой, что неизвестно, но неизвестно это неизвестно.



Рисунок не работает.

Dekker (задача с
несколько задачами в
одном потоке)
program example;
f1, f2 : logical;
cycle : int;
p1 : while (1);
begin
f1 = 1;
while (1);
if (f1 == 1)
begin

end; //
begin
f1 = 0;
cycle = 0;
parbegin
P1, P2
parenend
end;

11.10.20
Некакая
exit();
return
receive

Dekker (двухголовый курильщик) алгоритм синхронизации
для загона башмачеков, но можно его и професии:

"бить горячего - орехи".

program exmpd;

f1, f2 : logical;

que: int;

p1: while (f1) do

begin

f1 = 1;

while (f12 == 1) do

if (que == 2) then

begin

f1 = false;

while (que == 2) do;

f1 = true;

end;

CR L;

f1 = false;

que = 2;

PR L;

end; end.

end; //p1

begin

f1 = 0, f2 = 0,

que = 1;

parbegin

p1, p2;

parend;

end;

11.10.2022, слр.

Базовые define-объявления Number

exit(...) - завершение процесса (нап, если избр. базов. режим - не оконч.).

return... - базовым управлением

receive signal (не finish signal).

P2: while (f1) do

begin

while (que == 1) then

begin

f2 = 0;

while (que == 1) do;

f2 = 1;

end;

CR R;

f2 = 0;

que = 1;

PR R;

end;

end;

$$Mx_1 + Mx_2 = Mx_1 \cdot Mx_2$$

Сист. вызовы exec(), fork(), pipe(), signal(), wait().

Что делают, для чего, и т.д.
в какой последовательности.

помощью
не надо
написывать
нен. кода
надо signal(),

14. 10. 2022 г. Семинар

Сигнал.

Процесс может сам определить реакцию на сигнал. Сигнал
- классическое средство идентификации процесса о событиях в системе (вспомогатель - завершение процесса). ~~При этом~~ и прерывания, сигнал инициирует установленное бессознательно действие. Процесс идет:

- инициировать сигнал;
- использовать обработчик по умолчанию (сигнала);
- определить свою реакцию на сигнал.

Вместе с тем сигнал пропущен. Каждому сигналу соответствует определенная константа. Для устойчивого сигнала исп. сист. вызов signal(), но он не входит в POSIX, тута входит sigaction().

Использование сигналов, можно изменять ход выполнения программы в зависимости от внешнего альтернативного события.
Все это процесс выполняет сист. вызов ~~kill()~~, но это будет синхронное событие.

pipe - буфер, созданный в области адреса системных подразделений. Использовать нужно с пандесатией, что это буфер.

Делонг.

ps -ajx — пример демонта из ядр 1 на Unix.
самая запутанная шар. в Unix делана с механизацией.

в Windows код не
использовался раз
вмест. всегда
применим)

в OS Solaris

Делонг не

S - interrupt

kthread -

kworker -

ksoftwarecall

системы

Программа

① Вход

данные

создавать

запомин

сингл. д

② fork

сирота

шдер

в S

редакт

3) set

Пл

в ко

изю

3) и

иор

4)

ен

6

• Windows код неструктурированный: маркированием время от времени да-
нился реальное участие ядра ядра.

• всп. всегда есть процесс с ~~process~~ id 0 (зан. системой), 1 fork.
множественное). • все собственное ОС поддерживает понятие уровня ядра.

• OS Solaris называют lightweight processes.

• Linux не имеет управляемого механизма, находящегося в состоянии

S-interruptable sleep. (S-идет сессии, l- многопоточного демона).

{ fthreaddmon - отвечает за запуск потоков.

kworker - ядро, сколько ядер, отвечает за всю работу.

kssoftwarecall? - отвечает за выполнение отложенных действий.

Системные демоны инициализируются при загрузке системы.

main() автозапускается предком, вложившемся.

Правила программирования демонов (все в пулках con. в группах демонов).

① Вход в функцию umask(0); // функция сбрасывает маску со-
дания файлов, маска устанавливается, с какими привилегиями будут
создаваться файлы.

{ (помощник наследует код, отработавшие, маску соф. файлов,
анал. маску от предка) }

② fork() и завершение процесса-предка. { помощник становится
суперпользователем, чтобы оставить вложенный помощник еще один
идентификатор сессии}.

{ в System V fork() рекомендуется возвращать значение, в Linux же нужно
одарить унаследованного предка - не давать процессу более одного вложенного процесса}

③ setsid(); - сист. вызов.

После вызова процесс-родителя становится лидером группы/сессии
в отличие от других. Пакине этом процесс управляет управ-
ляемый терминатор. (в parent, чтобы поддерживать, что наследуется)

{ umask(0) возвращается, чтобы процесс-демон мог создавать фи-
лы с избранными правами доступа. }

④ chdir("/"); - изменяет текущий каталог на корневой. Это же
в Unix базовое правило, применение обещается максимум

Windows под неструктурированные: маркированием время от времени за-
нимаемое различное участки хода здрава.

В syst. всегда есть процесс с ~~некоторым~~ id 0 (занесен в список), 1 fork
множества). Все современные ОС поддерживает понятие уровня здрава.

В OS Solaris присутствуют lightweight processes.

Далее не имеет управляемого таймсепча, находясь в состоянии
S-interruptable Sleep. (S - лидер сессии, L - многопоточного демона).

lthreaddaemon - отвечает за запуск потоков.

kworker - чисто, сколько здрава, отвечают за всю работу.

ksoftwarecall? - отвечают за выполнение относительных действий.

Системные демоны инициализируются при загрузке системы.
main() автономно предка, второй ир. тоже.

Правила программирования демонов (все в парамах бол. в группе
демонов).

① Всюду функции umask(0); // функция сбрасывает маску со-
здания файлов, маска устанавливается, с какими правами будут
создаваться файлы.

{(помощник наследует демониторов, обр. файлов
авт. маску от предка)}

② fork() и завершение процесса - предка. {помощник становится
сиротой}, это для того, чтобы окончательно поменять ~~зарегистрирован~~
лидером сессии}.

{В System V fork() рекурсивном порядке вложить файлов, в Linux же нет
односр. унаследованного - не даст процесс более одного демона/сессии}.

③ setsid(); - ист. вспом.

После вспом. процесс - лидер становится лидером группы/сессии
без которой он одни. Пакже этот процесс управляет управ-
ляемые терминалы.

{в parent, чтобы поддерживать, что наследует)
умask(0) возвращается, чтобы процесс - демон мог создавать файлы
с избранными правами до конца.}

④ chdir("/"); - изменяет текущий каталог на корневой. Это дела-
ется потому, что в Unix базовые приложения обладают избранными
правами

Сист. вызовов exec(), fork(), pipe(), signal(), wait().

Что делают, для чего, и т.д.
в какой последовательности.

помощью
не надо
им. конвейер
наго sleep();

14. 10.2022 г. Семинар

Сигналь.

Процесс может сам определить реакцию на сигналы. Сигналы - классическое средство информирования процесса о событиях в системе (вспышка - завершение процесса). ~~Блоки~~ и прерывания, сигналы именем установленное библиотекой stdlib.h.

- информировать сигнал;
- использовать обработчик по умолчанию (сигналы);
- определять свою реакцию на сигнал.

В качестве сигналов прописаны. Каждому сигналу соответствует определенная константа. Для устойчивого сигнала исп. сист. вызов signal()., но он не входит в POSIX, тогда будем sigaction().

Используя сигналы, можно изменять ход выполнения программы в зависимости от внешнего асинхронного события. Если из процесса вызвать сист. вызов ~~kill()~~, то это будет синхронное событие.

pipe - буфер, созданный в области адреса кучи поэтому использовать нужно с погаданием, что это буфер.

Делонор.

ps -ajx - пример демона из ядр. 1 на Unix.
самая замутная штук. в Unix делана с перманентной

В один корень и.д. вмонтировано базисное число разных
файловых систем. (смт. вогод mount()) /примера/. физически
дискета). После вмонтирования файлы системы доступны на
зываемом

Вставляемые диски - РС ^{*} физики монтируются в общем
дереве И.и.д., что диск запуска с физики. Переход в корне
всей системы для того, чтобы можно было отмонтировать
систему, с которой был запущен диски.

⑤ Закройте все ненужные файловые дескрипторы. Закрывается
крайттора всех файлов, которые открывались при -программа и
диски не нужны.

```
struct rlimit rl;
getrlimit(RLIMIT_NOFILE, &rl);
rl.rlim_max = 1024 / (1024 * 1024);
```

} нахождение
max. количества
дескрипторов.

2) проверка наружу значения на эквивалентность RLIM_INFINITY.

3) б удали вызывается close(); ей передается номер дескрипто-
ров. ^{соударя}

{ когда соединяется процесс, где него становится отработавшим
нум дескриптора 0, 1, 2 } Q&T

В программе ^{аналог} закрывается
все (0, 1, 2. more).

⑥ Все файлов с дескрипторами 0, 1, 2 становятся следующие
гением:

```
fd0 = open("/dev/null", O_RDWR);
fd1 } dup();
fd2 }
```

Все гением, которое делают возвращают, т.е. отправляют ^{аналог}
fdout/stderr, направляемое в "черную дыру".

Демон же обладает
именно ("запущен")
на текущем диске
запущен думиди

Демон в ег

бель думиди

У демона

исключительно

эксплорер,

Предлагает

все конца

file и process

(всюду куп

супремат,

Всё, что

С хамедом

по идей

В демон

передаетс

и SIGTERM

SIGHA,

В думиди

передаетс

и SIGTERM

SIGHA,

В думиди

передаетс

и SIGTERM

Device

Device

и передает

Device

Демон не сбрасывает с терминалом, исп. stdin/stdout/stderr - бессмыс-
лично ("защита от дурака", защищена от ошибок из-за "1").

В конце функции daemonize() происходит инициализация файлов
на терминале для журнализирования ошибок (syslog). Использу-
ются функции openlog и syslog.

Демон в единственном экземпляре. (н. 13.5).

Есть функция already_running()

история 13.3 брать за основу.
6 ипр)

У демона не г. б. упр. терминала, чтобы с консольной строки
исключить влияние на демона. Демон доступен бол. в единов.
экземпляре, чтобы исключить возможность конфликтов.

Предлагается прцес., связанный с файлами блокировки. Пере-
вая копия демона создает файл, устанавливая姆 флаг lock
file и режим lockmode. После этого создается функция lockfile()
(входит кусок сист. вызов, если брать из книги, то нужно по-
считать, что такое создается).

{Всё, что в коде, присоединим автомат}

{С каждой строкой тела надо разобраться. На второй вопрос
по какой строке нужно отвечать.}

В демоне создаём дополнит. поток (pthread_create()). Поток
передаётся функция thr_fn; обработка двух сигналов: SIGMP
и SIGTERM).

SIGMP - сигнал управляемого терминала;
В функции ^(sigaction) daemonize этот сигнал исклучается. (В насе sa-
handler структура struct sigaction' передается SIGIGN)
Поток восстановленный реализует зажигаю снова вызвать sigaction
и передать реализую по умолчанию SIGDFL.

Демон получает сообщение в syslog. врем.

{ К динамичные already-running }

Расщё блоакировки создается в директории /var/run/semlocks.
Доступ только у суперпользователя, (root), носящий имена
засечек с суперпользователем. Это для того, чтобы не
было его некоего удаления / Всех писем в другую директорию
все будет работать).

{ нечестные библиотеки подключать не надо. }

Всю потоков в демоне будет два: начальной и дополнительной.
Об этом косвенно свидет kthreadddemon.

Упр. 3.1, 3.3 из 13 шагов.

14.10.2022г. лекция.

P1:

```
fl1 = 1
do while (fl2 == 1)
```

```
{ if (que == 2) then // если очередь второго процесса,
    fl1 = 0; // первой сбрасывает свой флаг, что
    do while (que == 2), // гарантирует не заход в тупик
    fl1 = 1;
```

y

cr1;

fl1 = 0;

que = 2; // отдаёт очередь другому процессу

PR1

// end P1 // наше же взаимное блок. откладывается.

Асинхронное ожидание на процессоре: времид трансформе по ID.
верну адреса другого процесса.

Это приводит к управлением тупик и бесконечное откладывание.

Тонкое - (race condition) - попытка получить легких проф, к разд
переменных.

Аппаратура
в IBM
максимум
автоматиза
использова
новка), ко
устраняется
чается,
program
fl, c
pl:

||em

P2:

||em

{

Аппаратная реализация.

В IBM 360 называлась команда test-and-set. Она обозначает запись (амперсандом в асс.) и реализует как незаданное значение проверку и установку значения ^(байта) памяти. Команда test-and-set имеет значение B, копирует его A, а значение где B. устанавливает значение ид. 1. Использ. команды test-and-set изображается аппаратной реализацией (Важенна: простая блок-схема). Команды test-and-set есть также в Linux. Они могут устанавливать любое значение. При этом название сохраняется, а действие команды определяется пользователем.

program usete;

 fl, cl, cd: logical;

 pl: while(1) do

 cl = 1;
 while (cl == 1) do // начиная с 0 обнуляется переменная cl; while (cl)
 { test-and-set (cl, fl); }
 CR1;
 fl = 0;
 PR1;

 //end pl

 p2: while(1) do

 cd = 1;
 while (cd == 1) do
 { test-and-set (cd, fl); }
 CR2;
 fl = 0;
 PR2;

 //end p2

 void main()

 fl = 0; // начальное значение fl
 parbegin // практическое обозначение
 { p1, p2; }
 parend

Переменная fl имеет значение 1, если процесс входит в крит. зону, иначе 0. Пусть P1 хочет войти в крит. зону, а P2 уже находится в ней. P1 установлено в fl. Тогда, при выполнении в цикле проверки переменной fl волшебного test-and-set, fl == 1, test-and-set обнаруживает и устанавливает значение 1. В результате P1 находится в общей зоне крит. зоне, значение fl = 1, test-and-set обнаруживает и устанавливает значение 0.

Поскольку P2 находится в общей зоне крит. зоне, значение void spin-lock(spin-lock-t *c) while(test-and-set(*c) != 0) fl = 1, test-and-set обнаруживает и устанавливает значение 1. Если переменная fl имеет значение 1, то волшебное значение fl = 1, test-and-set обнаруживает и устанавливает значение 0. В результате P1 находится в общей зоне крит. зоне, значение fl = 1, test-and-set обнаруживает и устанавливает значение 0.

Команда test-and-set активно используется в системе: в цикле проверяется значение. Использование test-and-set в цикле используется для критической блокировкой (spin lock). Часто бесс test-and-set выражает предполагаемое значение некоторой или переменной (condition).

```
void spin-lock(spin-lock-t *c) // макро宏 замена на#define test-and-set
{
    while(test-and-set(*c) != 0)
        /* рекурсия занято */
}
void spin-unlock(spin-lock-t *c)
{
    *c = 0;
}
```

Реализация test-and-set в различных архитектурах включает с блокировкой ячейки памяти, отмывка очереди [test-and-set] [unlock]

приводит к засорению на длительное время систем памяти, что приводит к снижению производительности (так называемой "празднической") системы.

```
void spin-lock (spin-lock_t *c)
{ while (test-and-set (*c) != 0)
    while (*c != 0)
```

3.

Если переменная занята, начинается обратной (без захвата шина данных) цикл проверки. Атмосферная реализация ~~test-and-set~~ ^{использования} склонное ожидание — незадействованное время процессорного времени.

Семафор.

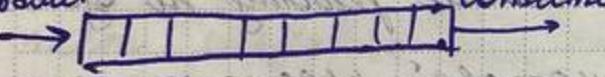
В 1965 г. J. K. Дайкстра (Dijkstra) опубликовал работу по взаимодействию параллельных процессов, в которой предложен семафор как средство взаимоискусения.

Семафор — это неотрицательная защищённая переменная, на которой определяются две исключительные операции $P(S)$ и $V(S)$, где S — это эта переменная, $P(S)$ — уменьшает значение S на 1, $(S=S-1)$, $V(S)$ — увеличивает значение S на 1. Декремент ^{passer} происходит в единицах, если $S > 0$. Если $S = 0$, то декремент невозможен, и процесс блокируется до тех пор, пока декремент не станет возможен. $V(S)$ — исправляет: $S = S + 1$, процесс будет разблокирован. К семафору и.д. обращаются операторы. Если семафор заблокирован, его заблокирована очередь. Если семафор свободен, его заблокирована очередь первого процесса в очереди. Если на семафоре определено два значения 0 и 1, он называется бинарным, если более ^(> 0) значений — сложенным.

В чём инновация? Продцес блокировкой (не нарушает производство), нет активного ожидания. Другой процесс, который consumer, while, p, n, v, в свободном семафоре, разблокирует процесс. Блокировка/разблокировка процесса и.д. выполнена только одного \Rightarrow конкурент. сист. языков, переход в ренессанс языка.

$P_1:$	$P_2:$	// var. ycm. $S=1$.
$p(s);$	$p(s);$	
$CR_1;$	$CR_2;$	
$v(s);$	$v(s);$	

Пример использования считающего семафора: решение для задачи производство - потребление (producer-consumer) на трех семафорах:

используется буфер размером n и два типа процессов:
 producer consumer

 (производитель и потребитель).

Производитель может только заполнять буфер, а потребитель - только вынимать данные из буфера.

Решение задачи с исп. трех семафоров: двух считывающих и одного бинарного:

Se - empty - число пустых

Sf - full - число заполненных

Sb - binary - бинарный.

program Semaphore;

$Se, Sf, Sb : int;$

producer:

{

согласно Janusz

$P(Se)$

$P(Sb)$

$n = n + 1; //$ добавление в буф.

$V(Sb); //$ обработка в буф.

|| нач. уст - х
 $Se = n; Sf =$
 Задача

реализует
 процессов.

ночь тоже
 используется

важно из
 потребитель

данное б
 ственный

в какую м
 ют сесси

В север,

то научба
 неактиве

На с
 прибору

5

4

3

consumer: while(1)

{

P(Sf);

P(Sb);

n = n - 1; // надо писать итерацию, дескремент

V(Sb);

V(Se);

}

// нач. ум-ки

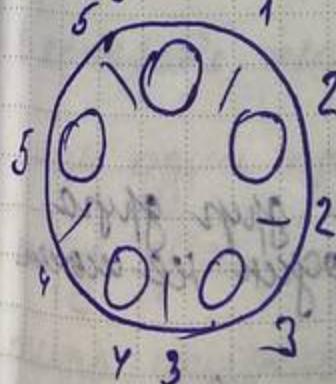
Se = n; Sf = 0; Sb = 1.

Задача приба-попр-а — это не такое загадка, в которой реализуется браческий механизм, но и в которой синхронизируются процессы. producer не способен позиционировать будущее для next nor, пока та же нет свобод. over. consumer, овободив очередь будущего, инициирует сессия Se. consumer не способен ничего вытащить из будущего, если в нём нет заполненных ячеек. т.е.

потребитель будет ждать, пока производитель не поставит данные в будущем. Процессор асинхронный, т.е. взаимодействие с будущей склонностью; несогласия предсказывать, когда какой процесс в какую точку придет. Простое браческий механизм использует синхронизацией, хотя это не предусматривает её.

Всё, что реализовано набором (считающим) сессий. Это то называется искусственное синхронизацией. Для присеяния распределенного массового задания (Дейкстры?) "Обедающее членство".

На столе стоят 5 тарелок, между которыми — по одному прибору.



Эти три действия: 1) как-то сразу попадают в две вилки, как-то средний есть, как-то блюдо вилки; 2) как-то нормально блюдо попадает вилки, удергиваю её, нормально блюдо попадает вилки; 3) как-то попадает вилка нормально блюдо, попадает нормально вилка блюдо, если ее поднимается блюдо попадает вилка блюдо, если ее

Эти загады заслуживают внимания семафоров.

program example;

var array

forks[1...5] of semaphore;

left, right; i: 1...5;

P1: left = (i+1) mod 5; // б-2
right = i mod 5; // 1-2.

- каждое единица
меняет биты
именно в
(выравнивание).

while(1) do
begin

<разрешение>
P(forks[left], forks[right]),
<если>
V(forks[left], forks[right]);

end;

// P1

P5:

left = 4;

right = 5;

while(1) do

begin

<разрешение>

P(forks[left], forks[right]),

<если>

V(forks[left], forks[right]),

end;

// P5.

Свойство набора семафоров: одновременное открытие всех семафоров может привести к значению всех/нисти семафоров набора. Семантически набор семафоров предполагается инициализирован (то есть набор).

В результате использования блокировок вместо разрывов семафоров возможна наложение преград в тупик.

P1: P(S1) | P2: P(S2)
P(S2) | P(S1)
CR1 | CR2
V(S2) | V(S1),
V(S1) | V(S2)

← оба преграда блокируются группой друга
переходят в тупик и не освобождаются
представляют блокирующие группы

Два отдельных преграда блокируют семафоры.

24.10.2022. № 1
При реализации
прогр. Б. дра "При
речия имен
Непр. исп.
и к Водник
↓ метод
метод агр
су.

Блокировка
именно сде
lock()
wait()
wait()

Нем пак
Монитор
мопр. як
(Concurr
Монитор
ко-тени
мопр. за
использов
в компаг
меш о
норм.
мона
(wait)

24.10.2022. Лекция.

При реализации решения и/р нужно создавать набор семафоров.

Б. Ари. Проблема с семафорами и инициированием ОС для обеспечения исключительного доступа в том, что они не структурированы. Исп. исп. может привести как к ползанию в тупике, так и к возникновению, называемых, семафорами проблем.

↓ монитор. -ср-бо более высокого уровня, чем "принципиальный ядро" - наименование ср-ба, предоставившего проес-

су.

системой предоставляемые функции команды, суть которых

является к здешним действиям - lock и unlock.

lock() - unlock()

wait() - post()

wait() - signal()

под ожиданием получается блокировка

Нем работают с семафорами. они рассм. отдельно.

Монитор ^{monitor} - именем предоставляемое именем в ОС, именем в ЗТ. Монитор как элемент включается в "Параллельный Паскаль" (Concurrent Pascal).

Монитор - набор процедур и данных; обращаться к данным можно только через процедуры монитора (говорят, что монитор "запирает" свои данные). Процесс, вызвавший процедуру монитора, - процесс, находящийся в мониторе. При этом в конец монитора включена процедура именем выработавшим один процесс; оставшееся ставится в очередь к монитору. Как правило, монитор оперирует переменные типа условие (condition) с помощью двух функций (wait(), signal()).

wait() блокирует процесс, signal() - разблокирует.

Простой монитор

Обеспечиваетование единственного ресурса несколькими процессами.

[this] {

 this->do();

 resource: monitor;

 var

 busy: logical;

 x: conditional;

 procedure acquire;

 begin

 if busy then wait(x);

 busy := true;

 end;

 procedure release)

 begin

 busy := false;

 signal(x);

 end)

 begin

 busy := false;

 end;

Для монитора характерно использование
записи переходной операции
to mutu conditional.

Когда к монитору обращается
процесс для захвата ресурса, он
вызывает функцию acquire(). Если
busy вертина, то обращение
какаша / неимеющей бояз-
нот, приводит к ожиданию не-
результатов на accessence. Если
значение busy - zero, то процесс
запрашивающий к монитору с
функцией acquire() получает
его, и процесс продолжает
свою работу, busy становится
базисом & free. Процесс, ждущий
ресурса

процесс, менеджирующий ресурс, вызывает оп-
тегицию монитора. signal(x), которая разблокирует гру-

нов процесс, который находится в ожидании синхронизаций. Для каждого отдельного блока памяти, на котором процесс имеет право блокир/перевед. В реч. описаниях, используется обозр. т.е. ev - обозначение сообр. ожидания.

Действие wait() и signal() подобно генерации P и V в hog семафорах. Это все равно системное действие.

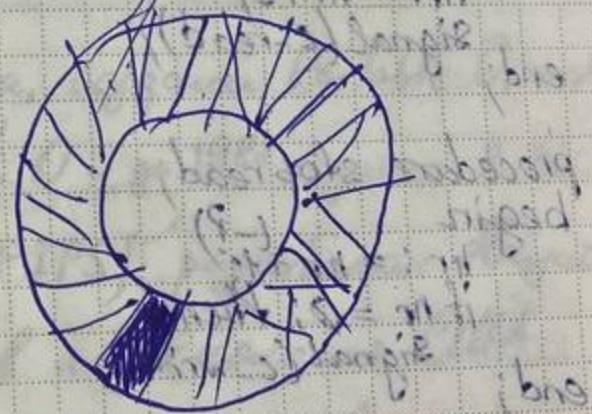
Монитор или vs. Семафор

Монитор - концепция языка

Две задачи прац. - напр. характеристика наименее занятых мест процессов: производитель - производитель (макс. занятость в буфере) и производитель - потребитель (макс. количество занятых мест в буфере).

```
resource monitor
var bufferempty, bufferfull: conditional;
buffer: array [0..n-1] of <type>;
pos: 0..n // текущий индекс;
j, k: 0..n-1 // запасенные
procedure producer (data: <type>);
begin
  if pos = n then wait (bufferempty);
  bcircle[i] := data;
  pos := pos + 1;
  j := (j + 1) mod n;
  signal (bufferfull);
end;
```

```
procedure consumer (var data: <type>);
begin
  if pos = 0 then wait (bufferfull);
  data := bcircle[k];
  pos := pos - 1;
  k := (k + 1) mod n;
  signal (bufferempty);
end;
begin pos := 0; j := 0; k := 0; f := 0;
```



Другой способ работы с задачами - это мониторы.

Он у/п не имеет никаких производительных задач и не имеет никаких потребительских.

Монитор читамент-пишамент (a)

Задача предполагает несколько двух типов процессов:
 Процессор - настолько склон к извлечению данных, когда
 и у они должны работать в режиме читамента
 то доступа к извлечению данных. При этом ку-
 какий другой писатель / читатель не может обра-
 титься к данным.

Процессор - считается когда только читатель имеет, то
 могут читать параллельно, не используя друг друга.

Писатель - программа бывает на транспорте, не параллельно.

monitor: resource; // no Deimaus

```
var:
    nr: integer; // читамент
    wrt: logical; // писатель.
    c-read, c-write: %conditional;
procedure startread;
begin;
    if wrt or turn(c-write)
        then wait(c-read));
    nr := nr + 1;
    signal(c-read);
end;
```

```
procedure stopread;
begin
    hr := hr + 1;
    if nr = 0 then
        signal(c-write);
end;
```

```
procedure startwrite;
begin
    if nr > 0 or wrt then
        wait(c-write);
    wrt := true;
end;
```

```
procedure stopwrite;
begin
    wrt := false;
    if turn(c-read)
```

then
 else
 end;
begin
 wrt := 0;
 wrt := false;
end;

Решение
 bakery algos
 (take a num

28.10.2022.

Inter-Pro

Система
 || проекта

System

Der mo

System

Federal

населен

POSIX

языка

JBill

языку

B E

Signal

боязь

21

```

    then signal (c-read),
    else signal (c-write);
end;
begin
nr := 0;
wrt_i := false;
end;

```

Данные условия обеспечивают правильное обновление битов. отмечается.

(if ... then ...).

Каждый активный элемент использует следующих ресурсов в очереди.

Решение должно предполагать лапортом (автором „Булфина“) bakery algorithm). Решение базируется на системе „взять номер“ (take a number).

|| Помаркное обновление

↑ когда же
программу
запускают

28.10.2022 г. Семинар

Inter-Process Communication (IPC)

Среда взаимод.

|| процессов

{ Unix

" — "

System V

очень много
семантических
различий
отличий

Unix (BSD).

Обычно методы избраны неизменными, называемые Portable Operating System Interface (for Unix) = POSIX. (1.0 - 90 1988, гаранс 2.0).

Federal Information Processing Standard (FIPS) разработаны Национальным институтом стандартов и технологий США.
POSIX => 1000 стандартов, из которых поддерживается при запуске персонального ПК.

И Bill разделил способность hardware и software: берет только аппарат, а не память написано.

Все они смешаны, неразделимой системе. Быстро, - X/Open. signal() не входит в POSIX, sigaction() входит. Существуют быстрее в смешанной, а инструкции - нет, они не организованы

Средства IPC System V

- 1) синхронизация
- 2) семафоров
- 3) пргр. каналов (шарж/чекшн)
- 4) очереди сообщений
- 5) семантическое разделение памяти.

В Unix BSD универс. сп-бами вспомогательные механизмы.

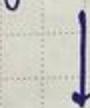
кемод.

Синхронизация. Активация базовых средствами интерактивизации процессов Unix о происходящих событиях.



асинхронное

запуск от
командного
процесса



не зависят.

Механизм сигналов вызывает прерывание микропрограмм на события. Как правило, наступление прерываний синхронно удаляет либо завершается, но пргр. несет on-reentry. Оно реагирует на микропрограммы, опр. по членам.

Мои расчл. сигналов (как в пргр. 75 к/п д/н Unix)
русской языке Unix (≤ 20) ($2^{20} - ?$).

Конечно сигналов в Linux значительно больше.

ал. картины в табл. по Unix.

#define NSIG 20
#define SIGHUP 1 — управление сессии
#define SIGINT 2 — управление CTRL-C

#define SIGKILL 9 — сущ. вороб kill.

#define SIGSEGV 11 — segment violation — борд за пределы памяти

#define SIGPOLL 30 —
#define SIGPOLLONESHOT 31 —
#define SIGPOLLIN 32 —
#define SIGPOLLOUT 33 —
#define SIGPOLLERR 34 —
#define SIGPOLLNVAL 35 —

#define SIGRTMIN 36 —

#define SIGRTMAX 45 —

В других
микропрограммах
процесса)

Сп-бами
kill и sigkill

int kill()

kill (ge

NSIG —

void (*su

сигнал
работы

• След. ОС поддерживает включение наименее стрессовыми по запросу, но показавшими сиcнem (кода, данных, стека) остановки.

#define SIGSYS 12 - ошибка выполнения сист. борьба
#define SIGPIPE 13 - запись в канал есть, чтение нет

Если нет читателя, то запись в канал бесконечна и будет блокирована.

#define SIGALARM 14 - alarm - будильник } - сигнал подсказки.

#define SIGUSR1 15
#define SIGUSR2 16 - пользовательские сигналы.

#define SIGCLD 18 - завершение процесса - потока.

При этом момент привести к борьбе двух процессов сигнала кода завершения, или других действий.

#define SIGPWR 19 - падение напряжения, разработчик может в ПО предусмотреть обработка данных / восстановление данных.

#define SIG_DFL (int(*)()) 0.
#define SIG_IGN (int(*)()) 1.

В функции демонтируется с помощью `sigaction` устанавливается импортированное значение `SIG_DFL`, а помехи в `main()` (в данном процессе) восстанавливаются реализацией сигнала по умолчанию.

С-базы посыпка и восприятие сигналов могут состоять из `kill` и `signal` (не входом в POSIX, но входом в ANSI C).

int kill(int pid, int sig);

kill(getpid(), SIGALARM); - Борьба прибегает к посыпке сигнала по будильнику, вызывающему `kill()`.

MSDN - базовый функцион

(как и manual в Linux)
manpage.

онтагард
(sig-handlers, еще базовая interrupt handlers).

void (*signal(int sig, void (+handler)(int)))(int);

Асимметричный борьб `signal()` выравнивает указатели на старые обработчики сигналов.

```
#include <signal.h>
int main()
{
    void (*old_handler)(int) = signal(SIGINT, SIG_IGN);
    /* обработка */
    signal(SIGINT, old_handler);
}
```

Возможна ситуация: определение в программе обработки потока притокоровано, хочется вернуться. Для этого используется указатель на старую обработку.

Для переносимого PID `signal()` использовать не рекомендуется.

запись в
коде демона.

```
int sigaction(int signal-numb, struct sigaction *action, struct sigaction
*old-action);
```

(изменение тела вот про)

Возможность реагации на сигнал расширена в POSIX.

Для этого есть два системных вызова

`sigsetjmp();` — устанавливает несколько точек перехода.

`siglongjmp();` — переходит на одну точку перехода.

PIPE-а (прог. кризис)

Классич. сп-во бдущедействия // процессов в Unixах обозн. ся каким-либо программским каналом. (так же можно)

{ в файловой системе можно видеть такое именованное обекта файловой системы}.

{ `mknod -p` — создание именованного программского канала — будера типа fifo).

Всё дескриптор — `inode` } `struct inode` — рассматривается как истреб. называемый index node.

{ `fd -ali` — показать inode } `i` { `i` } `i` { `i` } `i` { `i` }

{ Имена файлов — это строки символов }.

У Стивена, это "полудуплексный" (semi-duplex).

Банко: на прог. каналах определено бдущедействие.

Нельзя заменять канал экспем искажения. При

перехода наследу

Пруды буд

Уровень:

нашего. Для

переписывани

рамиши.

но труда

бывает до

прочитаног

буфер —

шер.

буфер

Обычное

и PIPE

При запус

кии процес

вопечатлющ

Размер

системе

31.10.2022

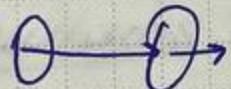
(Несколько каналов, пока пишут, и наоборот). Неписавшиеся при канале скопом используется только процессор - родительский. При вызове fork() создается процесс - потомок, который наследует дескрипторы открытых файлов, в т.ч. использованных при написании.

Проблема будгеризуется на трех уровнях:

1 уровень: в системной памяти. При перенаправлении потоков будгеров, имеющих идентификаторы процессов, переносится на диск сист. структуры группировки. Всех процессов записывается более 4096 байт, то проблема будгеризации во бредитаси: т.е. приводит к проблеме, когда до тех пор, пока все данные из канала не будут прочитаны.

Будгер - последователь байт, у которого есть нач. адрес и размер.

Будгер FIFO:



Обычный будгер: прям, членов,

у PIPE идентичны тут все удаляется.

При записи write() успешна блокируется, если сеть пуста, или процесс переходит в сост. ожидания. Конфликт, при котором выполняющий read(), будет блокирован, если данных нет, выполняется успешно, если данных нет.

Размер PIPE = размер страницы. Пересекаю в системе оптимизирована.

Пасмуро блокирующих пресмыканий (распределение существующим ресурсом, основанное на очереди). Проверяется в очередь к разделяемому ресурсу блокир. на производительность систем.

(5) динамит паспорта "бумага"

Взаимоисключение для n процессов.

Каждому приходящему клиенту выдается чистый паспорт. Новому клиенту — наибольший номер. Если чистого отсутствует, того и обслуживаю. Если клиентом прислан одинаковый номер, то я обслуживаю. Если чистого прислан однажды, то я обслуживаю одинаковое значение, но при обслуживании учитывается, например, номер паспорта.

При входе процесс проверяет все другие процессы и устанавливает, у которого меньшеший номер. Затем просор о реален-
зуется на одной машине. Но разрабатывается эта идея на следую-
щих распределенных вычислениях.

```
typedef char boolean;
shared boolean choosing[n];
shared int num[n];
for(j=0; j<n; j++)
{
    num[j] = 0;
```

// для i-го процесса
/* choose a number */

```
choosing[i] = TRUE;
num[i] = max(num[0], ..., num[n-1]) + 1;
choosing[i] = FALSE;
```

/* for all other processes */

```
for(j=0; j<n; j++)
```

```
/* wait */
while (choose)
/* wait */
while (num[i] == num[j])
/* critical */
num[i] =
```

Если i-й
попадется
участок

Состав

— Может

— Используя

в своем

compare

{ int

if (

) return

B code

сработ

Случа

l.a/n

онда

8

куро

Нашедо блокирующих пресетивов (расмотревшись рано
существом решения, основанное на очереди). Проверяется
в очередь к разделяемому ресурсу блок на производство
костюм системой.

(5) диаграмма логорта "бумага".

Взаимоисключение для n процессов.

Каждому приходящему клиенту выдается чистая санк-
так. Новому клиенту — наибольший номер. Если чистое
номер, того и обсуждают. Если клиентом присвоено одни-
менно, или выдаются одинаковые номера, то при обсуж-
дении учитывается, например, номер паспорта.

При входе процесс проверяет все другие процессы $\sim x$ —
бсе из них, у которого меньше номер. Здесь x — о ре-
зультате \sim — read-and-set . Часы
запуска на одной машине. Но разрабатываемая эта идея подходит
для распределенных вычислений.

```
typedef char boolean;
shared boolean choosing[n];
shared int num[n];
for(j=0; j<n; j++)
{
    num[j] = 0, // нач.
    // от.
}
```

// для i-го процесса

```
/* choose a number */
choosing[i] = TRUE;
num[i] = max(num[0], ..., num[n-1]) + 1;
choosing[i] = FALSE;
```

```
/* for all other processes */
for(j=0; j<n; j++)
```

```
/* wait if the p
while (choosing
/* wait if the
while (num[j]
}
/* critical secti
num[i] = 0
Если i-й п
помается в
 участник,
Есть амор
— Может,
— дядя
дядя
compare -
{
int old
if( old
}
}

В сокре
графов и
Синаси
Р. А. Му
меню
Для
курсов
```

```

/* wait if the process is currently choosing */
while (choosing[j]) { /* nothing */ }

/* wait if the process has a number and comes ahead of us */
while (num[j] < 0 and ((num[i], j) < (num[i], i))) { /* nothing */ }

}

/* critical section */
num[i] = 0;

```

Если i -й процесс не
имеет в себе участка,
то num[i] = 0.

Лексикографическое
отношение: если
два процесса имеют
одинаковое значение,
то первый в круг.
участок будет
процесс с меньшим
идентификатором.

$$(a, b) \prec (c, d) \Leftrightarrow \begin{cases} a < c \\ a = c \\ b < d \end{cases}$$

Более алгоритм Black-White Bakery Algorithm.

Нарисовать самосогласованное условие испр. о константах num test-and-set. Часто применяется команда compare-and-swap. Это
навязываемость генерации, ком. является опасной. Команда
буль это неделимая.

compare-and-swap (int *reg, int oldval, int newval)

```

int old-reg-val = *reg;
if (old-reg-val == oldval)
    *reg = newval;
return old-reg-val;
}

```

В современных ОС существует ком. багов mutex(). Для примера син-
хронизаций и семафоров можно показать особенности семафоров.
Семафор vs mutex

1. mutex имеет блокировку: это процесс, запрашивающий mutex,
может он момент одободжено / заблокировано mutex;

2. у семафора блокировка нет: любой процесс может забло-
кировать семафор;

P1 ... | P2 ...
 P(S) и. пасынк
коэффициент
коэффициент V(S)

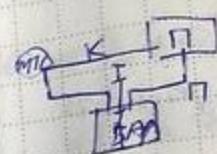
- 2) В отличие от сенсоров, на шиномехах определяется
она присоединяет;
- 3) Процесс, уделяющийся шинотех, не может быть (согласно
установлено), на сенсорах это же определяется, то определение
решения, заставляющие систему отключиваться могут сбоях,
что.

Это отличает сенсор от других средств блокировки
Проблема ставится парикмахером

В парикмахерской у парикмахера есть только один ра-
бочее место, в прическе - косметике стульев. Когда рабочая
вспомогательная уходит, а парикмахер идет в `break message()`,
стул за новым клиентом. Если никого нет, парикма- } В и/р чу-
хер сидит. Клиент проверяет, что ушел клиент парикмахера и пишет не
Если нет, клиент будет сидеть в кресле, если
работает, занимает место в прическе, если не
удалось, уходит.

Конечно, что парикмахерская работает только
так, но есть кое-какие конфликтные ситуации, где можно
рассмотреть более проблемную сторону, скажем
так, что генерация клиентом и парикмахера одновре-
менно. Случается, клиент идет в прическу, парикма-
хер проверяет клиентом, скажем, стул, какого-либо
года до прически, сидят на стуле и будет недопустимо

Прическа. Два
а то все время
Оба проверяют
чтобы и блокиров



Взаимодействие

распределен-

гому хосту (услу-
гами, все мо-

гие). Всё
запускает. Всё

вспомогательный

клиент уходит, а парикмахер идет в `break message()`.

Если нет, клиент будет сидеть в кресле, если

работает, занимает место в прическе, если не

удалось, уходит.

без
этих
данных
меньше
не
может
быть

все
запускает

При соот-

(P1)

send() —
переда-
вать
пара-
зит

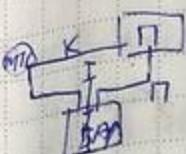
без
этих
данных
меньше
не
может
быть

все
запускает

без
этых
данных
меньше
не
может
быть

все
запускает

Призер. Два кислота могут пройти в присоединенное бодро
и то же время, когда в присоединенной единице. свобод. супре.
Оба проверяют, работает ли парижемахер, bevor в присоединенном и берут за единицу, свобод. чего.

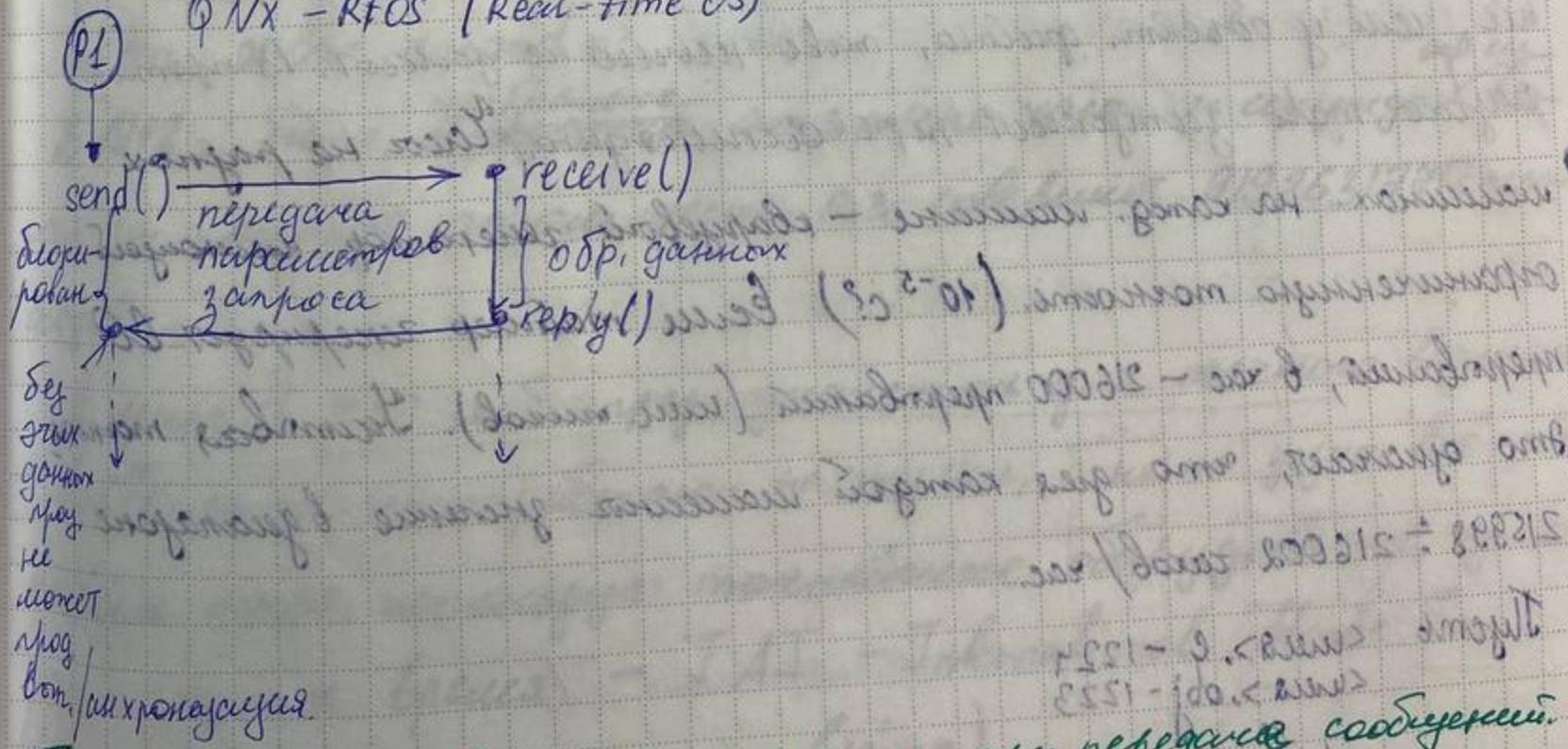


Взаимодействие процессов в распределенных системах.

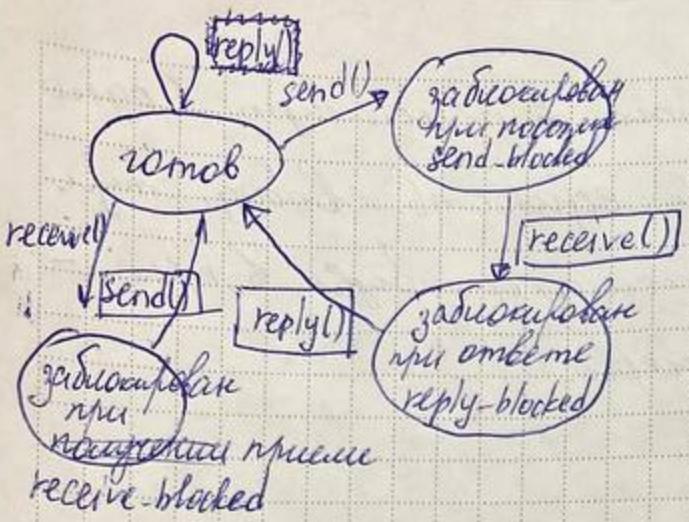
Распределенные системы — система с разделенным памятью. Каждый хост (узел) имеет собственную память. Средства, разработанные, не могут использоваться, т.к. там предполагается общая память. Взаимодействие организуется только на основе передачи сообщений, для чего г. б. систем. языка, (send message(), receive message()).

} Виртуально с буферами работают как с буферами, т.е. читать и писать не по индексу, а по указателям.

QNX - RTOS (Real-time OS)



Три состояния блокировки процесса при передаче сообщений.



в рамках обсуждения же действует, который включает в себя группу процессов.

4. 11. 2022 г.
1223 1224 1225
1222 1223 1224

Момент
перекон-
т. т. е. п

В 11

Для Unix
, включ

Про Си

разрабатывал на исх. языках, при изменениях переходил на другие

языки

и редакт.

на разных машинах. Для этого использовалась

таймер

типов

В 194

или, зо

ходов.

В с

Соб

бо бие

(исп

ти

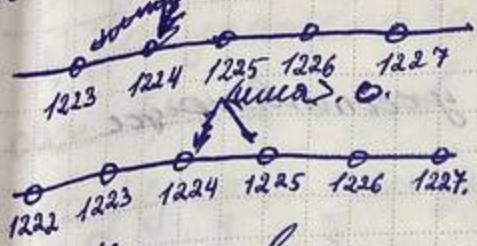
Когда процессу приходится посыпать (`send()`), а текущий процесс не готов принять сообщение, процесс блокируется. Когда получает сообщение — заблокирован при приеме. Если процесс выполняет `receive()`, а другой еще не отправил ответ, то он блокируется, make.

В них есть программа `make`. Большинство программного обеспечения разрабатывалось на исх. языках, при изменениях переходило на другие. В этом случае `make` заменял текущую машину — квадровый генератор, имеющий ограничительную мощность (10^{-5} с?). Если таймер генерирует 60 прерываний, в час — 216000 прерываний (или типов). Учитывая машину, это означает, что для каждой машины значение в диапазоне $215998 \div 216000$ типов/час.

Таким образом, если `c - 1224`
`obj - 1223`

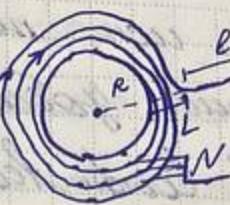
требуется перекомпилировать.

07.11.2022 г. Лекция.



1-й час.

$$\left\{ \begin{array}{l} = \frac{1}{\theta - a} \\ \end{array} \right.$$



Лекция - ITW

Может возникнуть ситуация, когда часы отстаивают, но перекалибровать не будем.

Т.е. невозможно ориентироваться на исходное часы компьютера.

В Unix есть часы реального времени (real time).

Для Unix также характерно наличие системы времени с включением часов.

Про Unix-микросистему было на семинаре, благодаря чему есть возможность пользоваться реальным (точным) временем.

Когда разбивается статика имеет смысл помнить реальное время. Интервал между последовательными переходами называется солнечной секундой \Rightarrow

mean solar second =

В 1948 г. были изобретены атомные часы, и с тех пор прошло, за которое atomic clock - 133, совершает 9192631770 переходов.

В мире $\exists \approx 50$ наборов, предназначенных для точного времени. Современное часы работают на титане, а не на уране. Все часы во всем мире, использующие оборудование, использующее, придают точное время - TAI - International Atomic Time (международное атомное время).

TAI - гипотетический часы, который не имеет расхождения.

Чтобы исправить, исп. потерянное сообщение. Когда Δ достигает 800 мс, добавляется 1 сес.

UTC - Universal Coordinated Time - время, усредняющее

потерянное сообщение.

В основе работают электрических компьютеров, имеющих реальное время. Числами отличие от других районов передается в другое, где потребность времени.

Ранее в часах присутствовало значение часов точного времени. Теперь во всем мире, но есть ограничение под

бога часов в час. единиц.

- алгоритм Кристмана;

- " " Берни;

- усредняющее алгоритм.

Для функционирования в часах единиц важно соблюдать отклонения "существующее" - "существующее норм". В единицах исп. ер-ва, исп. раз писать, небольшое - это ко времени сообщений. Очень важно соблюдать нормы регулирования времени сообщений П.

Исп. значение часов точного времени гораздо лучше и лучше всего применять, но если нужно более отн. Т. алгоритма полученных часов компьютера:

0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	54	72
48	60	80

Время отправки сообщения/его получения. \Rightarrow технологиче, наименование П.

Быть предупреждено о:

1) В отправляемых сообщениях

процесс удаляет временные ссылки;

2) Помогатель сравнивает времена свободного и ожидания
своих времена на 1 более временные ссылки.

В результате получаем бол. ошибки. П.

1	2
0	0
6	8
12	16
18	24
24	32
30	40
36	48
42	60
48	61
54	69
60	77
66	85
72	93
78	101
84	
90	
96	
102	

Отношение П наяву является пропорциональное
отношение.

Распределенное системе вносит в
сит. взаимоисключением свои око.

диаграмм, взаимоисключением в расп. системах.

1. Установленной (ан. задачи в Интернете)

Существует процесс - координатор (коорд. работу других
процессов). Процессы, т.е. время в критич. участок, посыпает
координатору сообщ. о начале, + инф. о критич. участке.
Все крит. учр. "свободны"; коорд. тут же посыпает сообщение - разрешение в ответ, иначе ставит запрос в очередь.
Если два запроса пост. одновременно, ищут единич-
ную проблему, ком. решаются с помощью штока врем-
яна исполнителя.

Всегда устанавливается максимум - время, определяем
об. координатора. (На случай, если коорд. нет). Процессы,
об. используя координатора, инициируют борьбу, подчи-
няют имелей номер (PID) всем процессам. Процессы,
не раб.

получивший сообз., сравнивает IP из сообз. с собственным ID. ID > ID в сообз. Этим процесс получает новые координаты. В итоге остается процесс с новым ID, который становится новыми координатами. Для решения задачи такой виде. на Аханк. г.б. ПО как участник взаимодействует, так и координатор.

2. Распределение.

Нем. процесс - координатор.

Когда проц. хочет войти в опр. прит. сенз., он проверяет свое местоположение (нахождение сенз + времени), и рассчитывает времена действующими процессами.

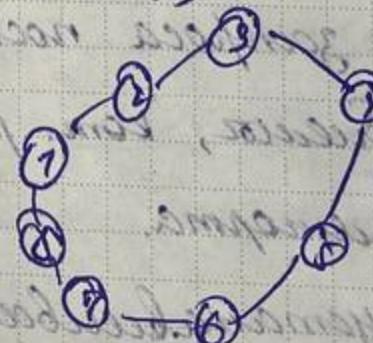
Процесс, получивш. такое сообз., действует так, что в какой-либо крат. сенз. он в дальнейшем исчезнет:

- не сидит, входит в крат. сенз. \Rightarrow разрешение
 - находит в крат. сенз. \Rightarrow запрашивает вперед
 - исчезает входит в крат. сенз. \Rightarrow останавливается
- если запрос и время вышли за пределы сенз., если его запрос не прошел \Rightarrow разрешение процессу, иначе, бросает ошибку.

Если нет ошибки, откладывается.

шаги:

3 Token ring



Процесса ждет разрешение здешнего администратора минимум полчаса, чтобы убедиться.

Когда крат. сенз. данного участка для него оконч.

Получив такое в крат. сенз. токен, токен дальше передается на один участок, токен необходимое нахождение нового координатора неизменено Token Ring, также RPC (Remote Procedure Call)

Процесс к

принимает

XXXX

данный

запрос

стандарт

соглашения

соглашения

для

на основе

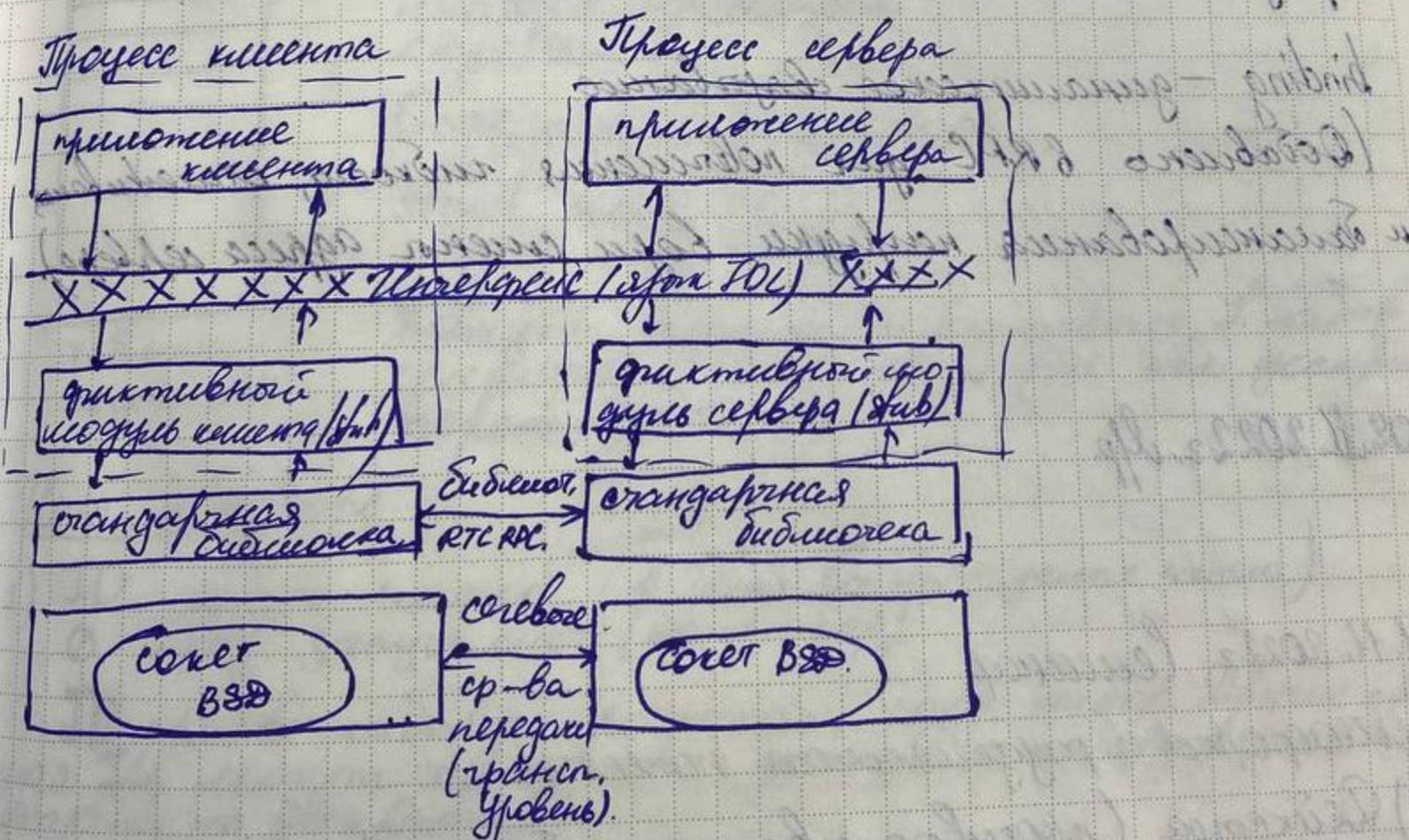
RPC соглашения

Получив токен, процесс проверяет, не является ли он своим в критической секции да, высодит из TokenRing, иначе выдаёт дальше по "каналу".
Если ни один из проц. не является своим в критической секции, токен достанет высодит по кругу.

Необходимо сравнить три архитектуры.
Наиболее надёжной — централизованной с поддержкой нового координатора. Принцип действия: если в процессе не получено пакета доступа к критической секции, так же и Token Ring, или восстановить свой токен, но скрыва.

RPC (Remote Procedure Call)

Используется туннелир. сп-ва для взаимодействия процессов.



Фиктивное модули — модули, генерирующие компоненты, на основе интерфейса, который пишется пользователем. RPC является как аналог виртуальных компонентов языка.

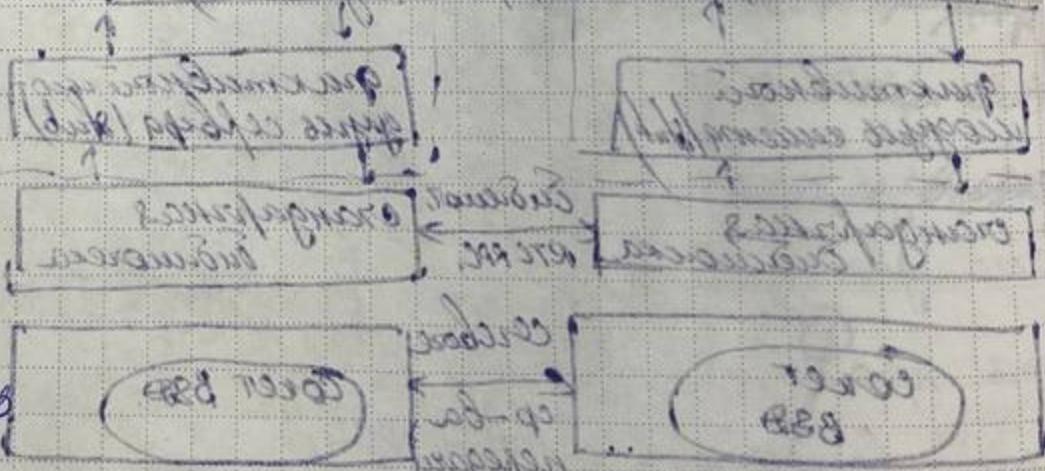
Когда клиент вызывает "локальную" процедуру, она
модуль преобразует (implements) входное значение
указанные при вызове, в пакет RPC
(сервис)
имеющий
запроса
отправка
отладки

Фиктивный модуль сервера отыскивает запрос клиента
когда он получает запрос по конкретному порту
(порт - это адрес), он преобразует содержимое упаковки
в соотв. параметров запроса. В результате при-
чески обрабатывает запрос, спорищет с ним
ком. диалог. модулем сервера будет преобразован
в пакет, передан по сети и, распакованной, будет
передан клиентскому приложению.

binding - динамическое связывание.

(Добавлено в RPC для повышения надежности, отслеживаний
и балансированием нагрузки, беря список адреса сервера).

08.11.2022 г. др.



11.11.2022 г. Семинар.

- Генератор и разделяемая память
- 1) Рейнхарта (преведомство - потребление)
 - 2) Коат (читатель - писатель).

В ре-те прокриптового блокчейна процесс
кумдела проверяет ~~старт/фин.~~ старт/фин. пер., что незадействовано.

Семафор - неотриц. числа переменная, над которой ресурс.
где операции $P(S)$ и $V(S)$.

В современных ОС реализованы наборы семафоров,
символич. - массив. Одной независимой ^{однотипной} единицей
может быть группа, одного или неск. семафоров.

На семафоре опр. для синхрониз.: $P(S)$ и $V(S)$, т.к. только
семафор может блокироваться и продолжать процесс.

$P(S)$ - деактивит. ($S = S - 1$, если это возможно), если ре-
активировано, прос. блок. на семафоре. $V(S)$ - инициирует
освобождение. Нет актив. очистки, т.к. процесс блок.
надо за это - переход в режим ядра \Rightarrow переход кон-
тактного ядра. В другом синхронизации можно использовать семафоры.



Таблица
символов.

Имеется базовая структура
(`sys/sem.h`)

Определена структура
`struct semid_ds`
иг. дескр.

Каждый элемент описывает 1 набор
семафоров, т.е. структура явно дескрипторизует набор.

Чир. о наборе:

- 1) ID - целое число ^(некотор.) (в Unix все ID - целые числа)
0 - прос. запуск син. (например).

ID присвои набору процессам, согл. набор. Другие проц
имеют некоторый доступ к набору (файл
указат. на дескриптор)

- 2) UID (user ID) (если это не кр. - тем ID
присваивается, когда новый родитель берет в управление.
PS-al-user ID показывает)

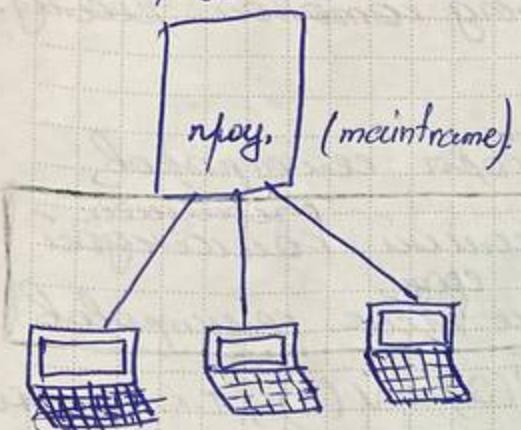
т.е. `ps aux` выведет

User ID.

символ
разделения времени

Есть еще одна вещь, это сразу же начнет
запуск с прос. UDP или Linux

Синт. разг. времечки:



• ID чуток содержит набор сеансов

сессий

Процессы, запущенные в ОЗУ которого
相伴 с ID состояния, имеют
один набор управлений упр.
параметров.

3) Права доступа:

Read
Write
Execute

4) Кол-во сессий в наборе

5) Время (установка знач. одного/каких сессий
последними прог.).
для т.ч. когда появляется, в как. времена сеанс. и прав. появляется
одновременно.

(Важно контролировать время, для ч. Эвакуации компьютера)

6) Время (установка упр. параметров набора
какими-л. процессами)

7) Указатель на набор сессий.

Сессионные наборы - массив, индексируясь ID.
Если в сессии. индекс. - изменение, то здесь - новый

На сессиях опр. тип. событий:

семфорд, блок, флаг, semget(), semctl()

создание набора

- semctl(); - управл. параметр.

- semop(); - ини. значение

int semop(int semfd, struct sembuf* opptr, int len);

блок

всё-флаги

операция

на сессии

На сессии System V определяются три операции
struct sembuf

ushort
short
short
sem-nr
sem-0
sem-1

При опре

1) дескриптор
(sem-op)
если не

2) прог.
(sem-op)

3) инициирует
(sem-

на се

IPC-

переходы

изменение

без при

сессии

нам в

установка

работ.

навсе

SEN

исева

и при

бюдж

short sem-num; - номер сессии
short sem-op; - операция
short sem-fl; - флаги.

При операции:

1) декремент (захват)
(sem-op < 0)

если нуль захвачен, прер. блок.

2) прер. переводится в состояние ожидания сессии
(sem-op = 0)
и соотв. ресурса (без захвата)

3) инкремент (освобождение)
(sem-op > 0).

На сессиях определяется статус умр:

тест F (сам один)
но если не менее
это умр.

IPC-NOWAIT - итератори. ядро ожидает прер.
переходит в сост. ожидания. (Ингл: wait - блокирован),
пакеты обмена исчезают из блоков блокировок
всех прер., кот. в очереди к сем. 3 в случае более выс.
сессии прер. завершается аварийно / получает статус kill, в связи т. что kill не влечет перехода умр
еще не см. Особ. захв. сессии, если не предприняты
специальные меры. Будут заблок. на сессии
исчезают;

SEM UNDO - итератори. ядро, что необходимо отменить
изм. стат. сессии в ре-те боязни земей
и при бою земей или захвата ядро должно инициализировать сессию. Но судя по всему исчезает.

Особенность: у семафора нет состояния, оно определяется в функции

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

```
struct sembuf sbuf[2] = {{0,-1, SEM_UNDO}, IPC_NOWAIT};
```

```
int main()
```

```
    int perms = S_IRWXU | S_IRWXG | S_IROTH | IPC_CREAT | perms);  
    if (fd = semget(100, 2, ID)) {  
        if (fd == -1)  
            perror("semget");  
        exit(1);  
    }
```

```
    if (semop(fd, sbuf, 2) == -1)
```

```
        perror("semop"),  
        exit(1);  
    }
```

монитор
наименование
семафора
первой сессии
запрашивается,
второй исп.
для открытия
освобождения

сигнал
бесконечн.
действия
вот
останов.

в а/р. процеф-норм.
три сессии
два семафора
один блокиратор.

Нужно объявить четыре массива структур, в каждой
две описываемые операции на двух семафорах,
(или буд. пнж., или буд. члж.)?

Обе задачи из а/р. — основное.

Разделение на сессии.

Сессии это разделяемый ресурс

ср-бо передача данных между параллельными проц.
(наряду с про. канальем), но нужно управлять паралл.
ресурсом

т.к. передача
стартовая опера-
ция синхрониза-

ботающие от про. каналов.

В Unix есть понятие stream (номок), в Linux управ-
ление это буферы.

Ило, это оно.
имением заменяется
т.е. для
исп. буфера
-shmget()
-shmctl()
-shmcat()
-shmctl()

Зада ID
может
его к сво-
лена. рабо-
тавший
иначе
(замес)

Сессии

АП при

также
про
(узн)

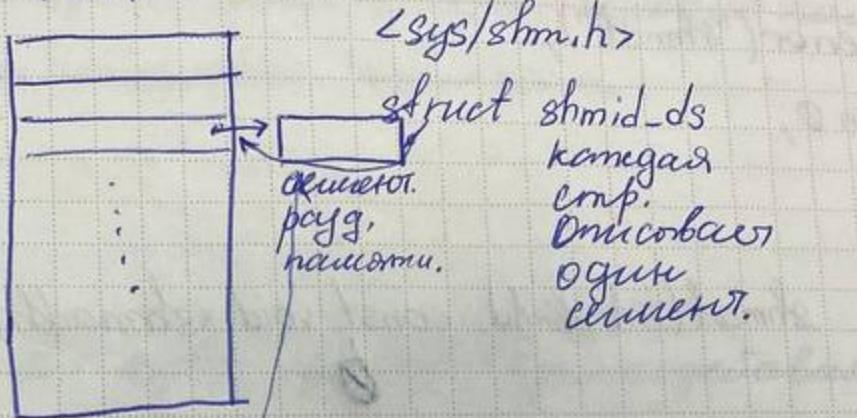
Ил., что омк. к про. ханчаке, омк. и к ср. пагг. наимене. Т.к. пагг. имеет значение. АП, ведущий, идентиф. себе 3-е АП, АП делает т.е. след. пагг. наимене соглашения в форме. Определение имен. борьбы:

- shmget(), control
- shmctl(), attach
- shmat(), "auto-detach." "auto-detach."
- shmctl()

Задаёт ID пагг. сечи, может ограничить его к своему АП.

Также пагг. наимене подавляется если инициатор комп. (запрошенное действие)

Используется пагг. наимене для обработки



`<sys/shm.h>`

`struct shmid_ds`

комерч
снп.
Описывает
один
сектор.

Секция пагг. наимене подавляется к АП процесса.

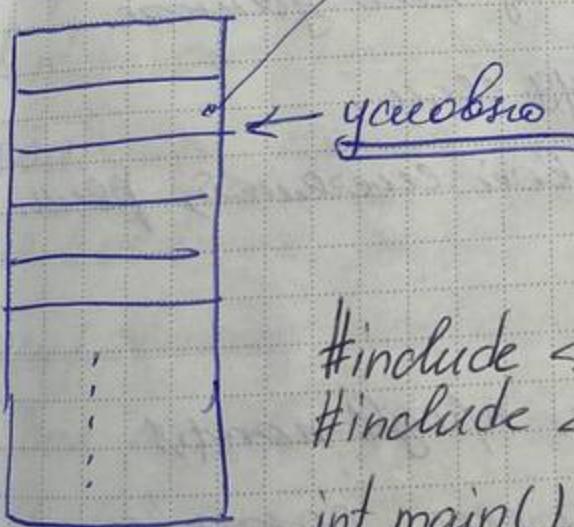


таблица
процессов
(указатель)

Идея. процессов не может быть системой, т.к. массив идет прилага к про. Несоответствие и подразумевается смешан (быть обнаружено).

`#include <sys/shm.h>`
`#include <string.h>`

`int main()`

```
    {
        int perms = S_IRWXU | S_IRWXG | S_IROTH;
        int fd = shmget(100, 1024, IPC_CREAT | perms);
        if (fd == -1)
        {
            perror("shmget");
            exit(1);
        }
    }
```

```

char *addr = (char *) shmat(fd, 0, 0);
if(addr == (char *) -1) perror("shmat");
    perror("shmat");
    exit(1);
}
strcpy(addr, "aaa");
if(shmdt(addr) == -1)
{
    perror("shmdt");
}
return 0;
}

```

void * shmat(int shmid, const void *shmaddr, int shmflg);

Θ
возвращает
метод
подходящий
адрес.

Такое место как професс записал данные и откликнулся
своим реаг. нашеами от АП, чтобы другой АП, зная ID, не
может подключить его и прочитать из него данные.

Было одно различие: проф. как | реаг. сис.

Проф. каким образом, где, когда, что

(реаг. нашеами — табличей вдру)

если проф.
каким
методом
или, то
что это
уверен
о каком-то
составе
таблицы
использован
ной проф.
каким).

Требование 16 байт превыше помрт.

В программе должно создаваться
не сколько ли 3 профов, и не сколько
3 помрбов; задержка уср. служеб
код.

Расшуповать (good)
таблица не через инд
кор, а через указатели.

Есть участие на разделенном семафоре, он не используется.
В будер кисть буфог по порядку, т.к. чтено видно.
Читатели писатели: не члены Зписок, не члены
Читателей. Разделом одну переданную типа int.
Писатели только информируют, начиная с 0,
(информируют каждый член, как говою поще-
дешун); читатели только читают.

14.11.2022г. лекция.

RPC - Remote Procedure Call - базовое средство взаимодействия, позволяющее достаточно просто, RPC с точки зрения реализации - очень сложной механизации. Для более тщательного взаимодействия используется binding (меняется хост, адрес и т.д.).

В RPC запрос клиента проиницирует выполнение процесса сервера.

rendez-vous (рандеву) - способ взаимодействия // процессов, находящихся同一 времени. В языке Ada, ком. создавалась для военных целей, изобретатель - Феликс. Модифицированное называется сокращение расхозов.

Был разработан специальным стандартом АИС.

Главной идеей rendezvous, реализованного Ada является запрос на синхронизацию "клиент - сервер" от одного процесса к другому, процесса последний синхронизует. Если запрос принят, то процесс синхронизируется и одновре-
менно получает результат выполнения действий

Есть упоминание на разделение симметрии, он же не описан.
В будор хватает букв по порядку, т.к. четко видно.
Читатели писатели: не меньше 3 чисел, не меньше
4 читателей. Разделением одну переменную типа int.
Писатели только инициализируют, начиная с 0,
(инициализирует конкретный писатель, как генератор доступа); читатели только читают.

14.11.2022 г. лекция.

RPC - Remote Procedure Call - базовое средство взаимодействия, называемое достаточно давно, RPC с точки зрения реализации - очень простой механизм. Для более глубокого взаимодействия используется binding (меняется хост, адрес и т.д.).
В RPC запрос клиента провоцирует выполнение процесса сервера.

rendez-vous (рандеву) - способ взаимодействия // процессов, находящихся同一 времени. В ЭП Ада, ком. создавалась для военных целей, исоджированием всем. Упрощение моделирования позволяет сократить расходы.

Был разработан специальный стандарт ANSI.

Главной идеей рандеву, реализованного Ада является запрос по адресу "клиент - сервер" от одного процесса к другому, процесса последний отвечает. Если запрос принят, то процесс синхронизируется и одновременно получает результат выполнения действий

$$\begin{aligned} \text{при } q(y) &= 0, \\ x(1+y^2) &= 0, \\ 1+y^2 &= 0, \\ y^2 &= -1, \\ y &= \pm i. \end{aligned}$$

Действия предполагают конструкцию, которая называется на языке называемое "обратка" ("guard"), имея которую более высокий уровень, параллельно на нем действует.

if $g_1 \rightarrow s_1$

$\square g_2 \rightarrow s_2$

$\square \dots g_n \rightarrow s_n$

fi

Например,

if $x >= y \rightarrow m := x$

$\square y > x \rightarrow m := y$

fi

it..fi - операт. скобки

g_i - обратка

s_i - список операт.

\square - разделяет операт.

В языке Ada под. е // проц. осущ. с помощью обходов, ком. называемое языковое словом task (задача). Задачи числом средств связи друг с другом, связываясь друг с другом с другими с помощью обходов.

Все одна задача обращаясь ко входу другой задаче запрос придет (асинхронность и синхронизацию RPC). В ре-те они уст-ком. у-
нависают раздеву. Задачи синхронизированы, но
бон. общие операторы. Задача несет как прямое
обращение, так и откоечить. Задачи числом мож-
ки входа (entrypoints), которое обозначается entry.

Пример:

task сумма is

entry <идентифик. входа> (дискрет. указатель) (формализованная
раскв.)

<группа обявленных входов>

end <имя>;

Необходимо, чтобы был оператор присеяния обращения
entry accept: <ID входа>...

до <последоват. операторов>; end;

Оно поддерживает распределенное выполнение. В Ada это предполагается, что результат выполнения последовательности операторов одновременно все находящиеся в rendezvous. Нет ожидания, блокировки ("блокировка - это зло"). Время блокировки предсказуемо, потому что не допускается где система реагирует, времена, где запрос g. обратим. строка опр. присеяния. времена.

Г. оператор?

Группа select, в трех видах:

- выборочн. ожидание;
- условн. выбор;
- таймеровский выбор входа;

В выборочн. ожидании допускается одна/ неск. альтернативы выбора запроса. В остальных сопоставляется RPC и rendezvous. RPC - это более нейтраль ^{член} rendezvous. Рассмотрим: RPC процесс запускается, а в Ada - часть программы, ком. уже работает.

Примечание. Неудавшееся транзакции.

Примечаний используется послег. операций над одним или неск. объектами БД (вставки, замены и т.д.), которая переводит систему из одного состояния в другое устойчивое состояние.

Процессы могут искать / удалять данные, но после них сущие транзакции никакие изменения неизменены. Примечание g. поддержка слож. логики выше, как минимум:

`begin`] transaction.
`end`
`abort`
`read/write`

Об-ба транзакции специфицированы

- 1) A - Atomic - атомарность, индивидуальность + независимость (контроль), какое значение
- 2) C - Consistent - согласованность, если все OK, коммит]
Commit - если все OK, откат]
Rollback - если Э не OK, откат]
- 3) I - Isolated - изолированность (от других транзакций и процессов)
- 4) D - Durable - устойчивость
по завершении транзакции (если сбоя нет, можно вернуться к нему для дальнейшего обновления)

Существуют разные способы транзакций.

CICS - Customer Information Control System (IBM) -
средство масштаб. реального обработки однокомандной
язык Z/OS, VSE/ESA и распределенных платформ.

EJB Container/Servers

Enterprise Java Beans.

Транзакции могут управляться контейнерами или
пользовательским кодом.

Два подхода к решению транзакций:

- каждый процесс, т.е. в транзакции имеет собств. простр., в котором имеется конец всех объектов (напр. файлов, тирнетах для реализации транзакций), лежащий в работе процесса, как только процесс выполнения кончается в ресурсах файлов. Проблема в том, что Э большое кол-во концов \Rightarrow тяжелодушный

- "список измерений": исодиаграммируются единицы объема, а не их единиц, но перед ними. При любом блоке единица счищена пропись запись в тетр. журнале регистрации, где указуя, какая регистрация, какой блок, блок, старое и новое значение блока; такого пишут после записи. запись в журн. регистрации, меняется регистрацией блоков; при синхронизации пропись запись в журнале, при списке изм. из журнала ит. дает приведение единиц в исх. состояние (откат).

Процесс возвращается на разные исходники, поэтому используется спуск прописки. Извлекают исп. прописки двумерной синхронии прописки, где один из проп. всп. роль координатора, начинает прописку, дальше запись в св. журнале, появляется нога координатора, всп. ту же прописку, сообщение. Получив сообщение, проп. начинает всп. прописку, через ногу. Время коорд. посыпает сообщение подготовленное к синхронии. Получив сообщение, прописка в своих журналах записывает, что готова к синхронии, посыпает сообщение.

Координатор

Запись Приматовского
к синхронии
Посыпка сообщения
"Приматовский к синхронии"
Запись "Собирает ответы"
Если все открыто "Томок"
Запись "организовать"

Подчиненное звено

Запись в журнале "Томок"
к синхронии
Посыпка сообщения "Томок"
к синхронии"

Последовательность
"запись в память"

Запись в память
"записывать"
Последовательность
"записывать"

(сбор отчетов
"записывать")

Когда координатор получит все отчеты, транслируя
считается зафиксированной

В распределенных шахтах возникает проблема: на
одной машине данные изменяются, на другой - остаются
в неизмененном состоянии.

Особенности:

1. Статическая непротиворечивость
2. Последовательная "—"
3. Принципиальная "—"
4. Непротиворечивость FIFO.

4) основное условие: оп. записи
одного процесса г. наименование
действий в порядке осуществления,
он. разных
процессов — в пределах

Указывается, что ресурсовать
в расп. системе невозможно
(требует больших начальных
расходов).

Критична к точности синхро-
низации времени. Указывается,
что атомарные, работают
с точной временем (криптаны,
беркли), приводят к замед-
лению действий не оп. язах
используя во сущесвии перево-
да часов назад \Rightarrow инвалид
синхр. времени невозможно, а
уордирующими они непригодны.
но, наиболее подходящий
метод - метод эти. времени логотипу

1) любое чтение до-та данных
г. выборку, значение, под-
помог, изменение это
и т.д.

2) реф-т в действии должна
быть такая же, как
если бы все операции
чтения/записи всех чуж-
есов выполнялись в и-
ном порядке.

3) где её бол. неодн., под
оп. записи, потому
своеволное присвоение передает
следов. свободно, когда
действия оп. прав в опре-
деленном порядке, если бол. ||
то недоп. в прав. порядке.

Предусматривает
синхронизацию
время Рутин
импорт

И вакансии под
последовательным
это для
криптографа.

Возьмет только
передок, другие
сообщения не
имеют значения.

Передок ~~передает~~ ^{получает} ~~получает~~ ^{гранич. валидности} не имеет доступа к ресурсу, ил. д. сооз-
сужд, что процесс находится в двух очередях и передок
обслуживается самим. Возможна разное распределение (про-
цессинг другое процессов и т.д.). Основная проблема в том,
что процесс может пропадать в очереди неопределенно
длительное время.

25.11.2022 г. Семинар.

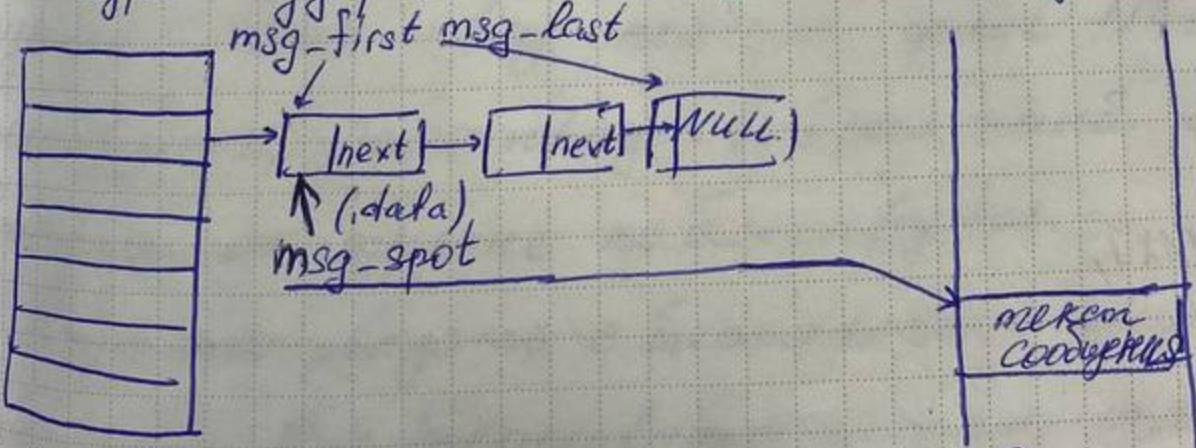
Очередь сообщений.

Это не буфер.

Вместо них с разг. пакетом буфером можно с помощью
передачи сообщений.

И нахождения следят, что это очередь, в которой находятся
сообщения.

В адр. поддерживается табл. очередей сообщений.



#include <sys/msg.h>
struct msg_id_ds

Сист. вызовы:

msgget()

msgctl()

msgsend()

msgrecv()

На сообщениях определена структура

struct msghdr

```
{ long msgtype;
  char msgtext[MSGMAX];
};
```

Сообщ. относ. к напр. ресурсам. Если сообщ. возвращено
нечисл. значение, то оно не существует и к буферам
макс FIFO.

Процесс и. с. создает очередь сообщений базовой msgget,
уничтожает упр. параметров базовыми msgctl(), отправляет
сообщ. через msgsnd(). Если процесс завершится, создае-
мый программой существование.

```
#include <string.h>
#include <sys/msg.h>
#include <sys/types.h>
#ifndef MSGMAX
#define MSGMAX 1024
#endif

struct msghdr
```

```
{ long mtype;
  char mtext[MSGMAX];
};
```

```
int main()
```

```
{ int fd = msgget(100, IPC_CREAT | IPC_EXCL | 0642);
  if (fd == -1) perror("msg");
  return 0;
```

Когда напи-
шет поддер-
жка комп.
данных
хенип. с
проц. и.
старт а
получен
из доступ
комп. с
комп.).

1. Взят
чтега;

2. Если
вмо
том

3. Проц
и. с.
богр.

На с
учен
блокир
богоф.

Нап
сообщ
reas

7. Ин

Когда приб. не передаёт сообщение в очередь, ядро создаёт для него поддельную запись и помещает её в конец. дальнейш. В конс. записи: там, число битом, указатель на обр. записях ядра, где наход. текст сообщения. Далее к пример. сообщ. из АП приб. в АП ядра. После этого приб. и. завершается (так в примере), т.к. в msgsnd() ставят флаг IPC-NOWAIT (процесс не ожидает пока не получится сообщ.). Отправляемое в очередь сообщение имеет в доступ. другими приб. Когда приб. получ. сообщ., ядро копир. сообщ. из АП ядра в АП приб.-получат (двойное копир.). Приб. и. возвращает сообщ. З способы:

1. Взять самое старое сообщение (использ. в голове очереди);
2. Если ID сообщения совп. с ID, которого указал процесс; (что такое ID, проверить в мануале); если нек. сообщ. имеет такой ID, берется самое старое;
3. Приб. и. возвращает сообщ., знае. что, которого пишет. из числовых или равно тому, если таких > 1, возвр. самое старое.

На текущий (31.10.2022). рассм. состояний при передаче сообщения. Если приб. передал флаг IPC-NOWAIT, это не блокируется, если приб. возвращает ошиб. блокировку способ возвращ. он также не блокируется.

Надо это связать с блокировкой процесса при передаче сообщений. Все блокированные режимы, если режим надежн-ней авт. передаче сообщений (с инф. о получении).

Три абсолютно разных передач в Linux.

чп. (1)

1) 2-я часть чп. 1.

"Регистры обработ. прерываний" отсчит. гайдера
бремени"; (зарезерв. регистра = 32bit)
Linux, Windows, Unix - система разделяет
время; (long-регистр времени для всего)
-квантование программы бремени;

2 часы:

- 1) фундамент обработ. прерываний отсчит. гайдера;
- 2) переход динамич. приоритетов.

Обе части пишутся отдельно для Windows & Unix.

Основной задачей обр. прер. в час. реа. бремени является декомпил. языка. Если вами написан, то обр. передает исходник ис. (в ПК) или другому прог.

(исходник - более поздний)

Для обеих систем:

Регистры таймера:

- по тому;

- по неактивному тому;

- по свинти.

18, 2 раза в с. не имеет отношения к (исходнику) самому слишком редко.

Глобальный таймер и.д. включает несколько режимов, так называемые, чтобы отдавать от простого таймера.

Windows: Стартовый, Режимовый, и т.д. о счет. гайдера распределенное по всему тексту;

Unix: Важность: есть паралл., но тоже не вероятно много. (напр., ~~один~~ бород таймфрейма).

Внуков - call - переход по адресу. Вызываемая программа не завершается. Для обработки прерыв. от сист. таймера это неподпустимо, т.к. он вон. посту на самом выс. уровне, Win-
Interrupt Privilege Level, в Unix - диапазон присортировок номеров
^{выше зон}
Этота ^{IPL, Power}

таймер находится выше всех прер. Работу не и. прерывать
ничто кроме падения напряжения. Т.к. он должен за-
верш. как можно быстрее. На сих. же одр. прерываний
также инициализируют от кон. генерации. Для Windows
DPC - Deferred Procedure Call.

То есть, ткну осуд. инициализ. работы пакетов
чика. Диспетчер ставит процесс в очередь ~~сам~~
процессов в очередь к процессору. Диспетчер заносится
как пр. возделанные пр. временн. Диспетч. не вон.
но имен. вставля, а заранее. (по имен. вставля подг-
ко), но ткну, ткну, так же, как пagedаemon, вытас-
кив. стр. на диск, заранее, до исполн. прерыв.
Пересчит. данные, присортирует.

Пересчитовь, только после присортирует. В (исчисл.)
системах есть два вида пр. решения времени
- аудио и видео. В Win аудио имеет диапазон
присортир. (аудио пр.). при блок на веди.
год).

Рис: Пример. Это можно находит пр. В Win нее
считывается тут же сканирование очереди процессов
Пересчит присортировок, обрату из Всехами, исчи. на 20,
Unix

что в Linux реализовано более сбр. дисп. переключ. В обоих системах чутк как простой в отладке и профиле, браузер. Прог. время неизв., присор ~~авто~~ тимируется инициируется и процессор, время. Прог. время неизв., присор ~~авто~~ RPC.

Вся 2-я часть сдается в виде ответа. В конце боят

- 3-4 циклов, отрабатывающие сумме проектированного.

лп2) Задача "Численно-механического" из Win по шаблону

Холода. (Реактив).

Программа пишется на языках.

Ти. 6.

Программное обеспечение по технологии реагир.

Чел. где функции:

InterlockedEvent

(Dec) C 242 где инд. борьба

WaitForSingleObject

WaitForMultipleObject.

Задача решается с помощью объектов ядра - event

(опр. 213).

об, при этом в Win 2 типа объектов ядра:

- со сбросом битами;

- с автосбросом.

Она транс. исп. в двух мод. исполнение. Код показывает свои ресурсы в и/р, задачи. Для отображения используется объект mutex (за него приглашается)

mutex никогда не занимается в одной функции и свободен в другой (в чот. / неч. Ч функции)

Код исполнения на Ada, а у нас нет turn(), а это исполь-

18. 11. 2022 г.
лп. 6 нр. 1
дипломные
тех. рабоч.

D1	5
P2	2
P3	4

В этом
направлении
не при

Сост. як

закрытие

→ синхронизация

закрытие
ресурса

конец

закрытие

Все

ресурса

закрытие

⇒ а

(8)

счетчики. Одна разделяемая переменная типа `integer`.
 Инициируется RPC, чтобы начать проработывание шестерни
 по RPC.

28.11.2022 г. лекция

(алгоритм блоксера).
 (инв. пр. иск.)

тек. расп.	заявка	Своб. рес.
P1	5	12
P2	2	4
P3	4	8

(опред. число процессов
 и опред. кол-во ресурса).

1.

В этой ситуации нет пасущ. проц, которое может гарантированно завершиться. Составное недeterministic, но может и не привести к тупику.

Сост. зал. безопасное $\in \mathcal{F}$ пасущ. проц.: 1-й проц. в пасущ. завершился, т.к. (он запросил max заявка кол-во ег. ресурса \rightarrow система удалил заявку), 2-й проц. завершился, если завершил первый и вернул все ресурсы, что в сущности свобод. рес. ит. использует удален. max напр. ресурс, i-й проц. может заверш., если проц. S_1, \dots, S_{i-1} завершил, освобод. свои рес., и в сущности со свободополож. рес. вернувшись. заявка i-го проц.

Важнейший факт, когда процесс делает заявку, менеджер ресурсов должен помнить устн. заявки пасущ. проц, когда заявка и.д. усовершенствован, нужно исходя из них. менеджер \Rightarrow алгоритм имеет теоретич. значение.
 (официальное закрытие, системное обновление).

Система реального процесса не должна попадать в
известную атаку, напр., склоняющуюся к атакам
~~безопасности~~ ситуацию (go^n^2). Бинарные
таблицы
неко
ческих
ситуаций

Наиболее известное алгоритмическое явл. алгоритм Процесс
запросов
Хабермана:

Менеджер ресурсов поддерживает массив

$S[0 \dots r-1]$, где r - число единиц ресурса.

$S[i] = r-i \quad \forall i : 0 \leq i < r$,

если $proz$, заявивши $S[i]$ ресурса и удерживавший k eg ,
запрашивает ещё одну eg ресурса, то $S[i]$ декре-
тируется $\forall i : 0 \leq i < (-k)$. Если некот. $S[i]$ отрицат.,
то это составляющее опасное отклик. туманка.

Обнаружение туманов

Система не ограничивается, процессом создания и
ресурсом взаимодействия по шире необх., \Rightarrow ставится
задача обнаруж. тумановой ситуации, где некон.
графовая модель Холла \Rightarrow двух-сторон. граф.

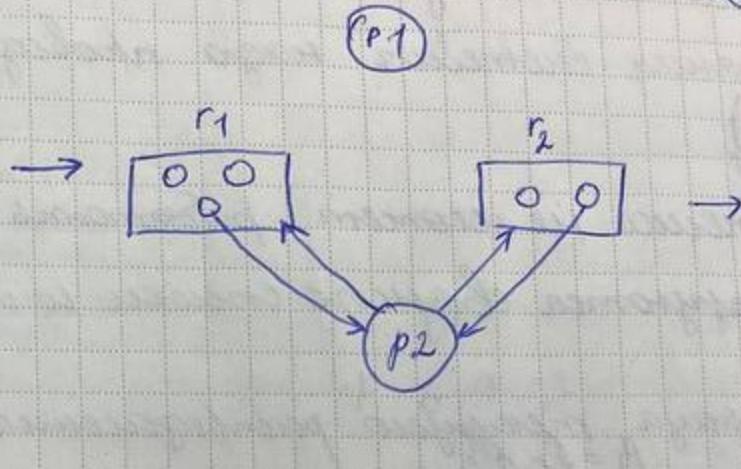
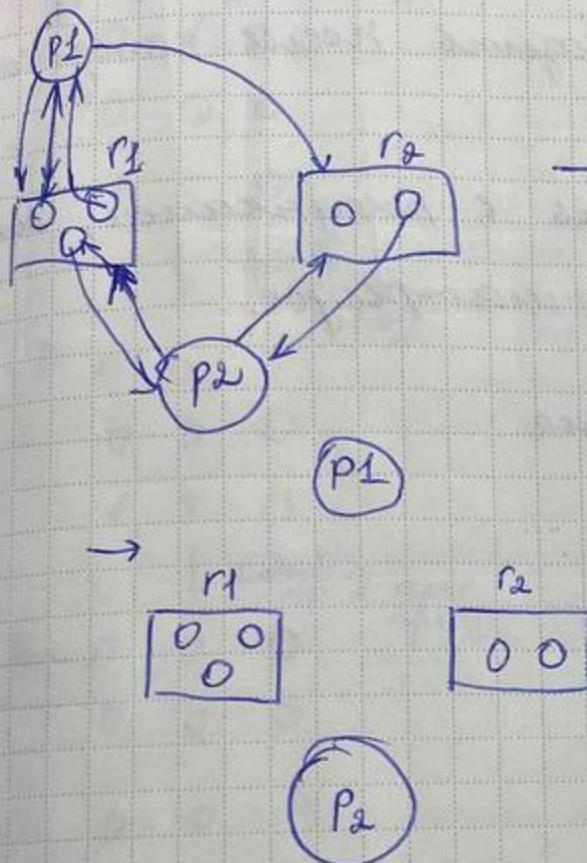
Она же - ба версии: одно - процесс, другое - ресур.
Версияног соед. драмы, имеющ. напротивостояние. Если
друг берёт из версии $proz$ в версии ресурс, это запрос,
в противоположном напр. - взаимодействие.

Процесс пост. делает запрос, если возможн., ресурс
взаимодействия \Rightarrow граф постоянно изменяется.

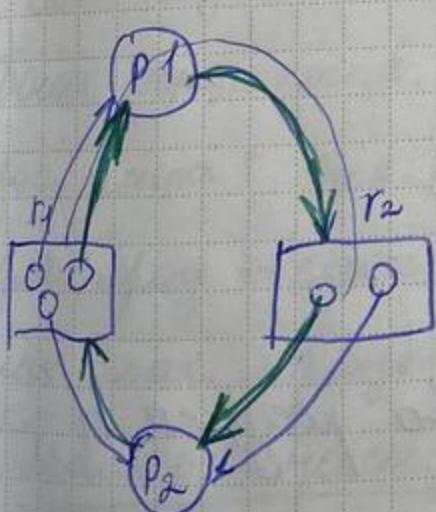
Если $proz$ не имеет такого же sol соотв. на путь
или путь неког., взаимодействия, или путь свобод. ресурса,
в тумане.

Обнаружить проф, попавший в тупик, искажающий путь
редукции графа.

Процесс P_i сокращается, если не яв. ни задек., ни ^{запро.}
путь удаления всех вынужденных и вынуждающих ребер. Так
мыс. сократу. P_i соотв. запросу и бывш. посещ. ^{быв. ресурса}
^{бывш. посещ.}



сокращающийся
граф



Учтъ запросов.
несокращающийся
граф.

Напомним, что
отправила Илоу

Теорема:

Теорема 1. Граф явл. посекостью сокращаемым, если
я неизгубим. сокрещущий, которая устранила
все дуги.

Теорема 2. Узлы (запущенные для запросов) в графе не необходимы для успешного выполнения тупика.

Теорема 3. Если S не авт. сост. тупика, Т авт. тупика, \exists переход $S \xrightarrow{?} T$, этот переход осущ. в рез-те запроса (тупик в системе может возникнуть только в рез-те запроса. Всие тупик возникают в рез-те запроса, а не в состоянии системы надо проводить после командного запроса)

Система не всегда работает с ресурсами, поэтому используются различные списки и мастерсфы.

Матрица текущего распределения

$$B = \{r_i, p_j\}$$
$$\begin{matrix} & \dots \\ & | \\ & b_{ij} \\ & | \\ & (r_i, p_j) \end{matrix}$$

Матрица запросов. $A = \{p_i, r_j\}$

$$\begin{matrix} & | \\ & a_{ij} \\ & | \\ & (p_i, r_j) \end{matrix}$$

Матрица распределений b_{ij} : количество ресурса r_i багажа по пр-ю p_j

Матрица запросов: кол-во запросов на ресурс r_j по пр-ю p_i .

Вектор свободн. единиц. ресурса F : сколько есть св. рес. r_i .

$$t_i = \sum_{j=1}^n b_{ij} + F_j$$

При этом при сокращении нет. все при об.

$$B \begin{array}{rrr} 0 & 1 & 1 \\ 1 & 3 & 0 \\ 1 & 5 & 0 \\ \hline 2 & 9 & 1 \end{array}$$

$$A \begin{array}{rrr} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 2 \\ \hline 2 & 2 & 3 \end{array}$$

$$F \begin{array}{rrr} 4 & 0 & 2 \\ R = 6 & 9 & 3 \end{array}$$

↓
сокр.
по 1-му
пред.

$$B \begin{array}{rrr} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 5 & 0 \end{array}$$

$$F \begin{array}{rrr} 5 & 3 & 2 \end{array}$$

↓
сокр.
по 1-му
пред.

$$B \begin{array}{rrr} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 5 & 0 \end{array}$$

$$F \begin{array}{rrr} 6 & 9 & 3 \end{array}$$

↓
сокр.
по 3-му
пред.

$$B \begin{array}{rrr} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}$$

$$F \begin{array}{rrr} 6 & 9 & 3 \end{array}$$

Это называется прямой алгоритмом обнаружения тумана. Более эффективн. алгоритм предполагает, что характериз. gen. инса: gen res. - сколько заявок, утвержд. по размеру gen пред - сколько отседаний - число ресурсов, поддающихся блок. процесса в отседании обобщ. ресурса.

Пример анализа состояния систем по алг. Хабермана.

Как узнать, что сост. системе безопасно?

1. Анализ состояния с текущим состоянием

2. Контролируется вхождение пред. max кол-во заявок ил. ресурсов.

3. Если все пред. заявки есть. Состояние.

⑥ Матрица
Request

⑦ Требова-
ние
Матрица

D
C
E
F

Запро-

бовані
дни

1. Всі

мако-

2. D

дни

3. E

в було-

Всі

запро-

1. F

дни

2. G

запро-

1. H

- рассчитывается и процессы и их ресурсов.
- матрица доступных ресурсов:
Max-Avail matrix $A = (a_1, a_2, \dots, a_m)$, где
 $a_i = t_i = |R_i|$

- матрица заявок

$$\text{Max-Claim matrix } B = \begin{vmatrix} b_{11} & b_{12} & \dots & b_{1m} \end{vmatrix} \xrightarrow{\text{но проек.}} \begin{vmatrix} B_1 \\ \vdots \\ B_n \end{vmatrix}$$

- матрица распределения $b_{n1} \ b_{n2} \ \dots \ b_{nm}$

$$\text{Allocation matrix } C = \begin{vmatrix} c_{11} & & & c_{1m} \\ & \ddots & & \\ & & c_{nm} & \end{vmatrix} = \begin{vmatrix} C_1 \\ \vdots \\ C_n \end{vmatrix}$$

возделенное и удовлетворяе-
щее прош. кол-во ресурсов C_{ni}

Ограничения:

① $\forall k \ B_k \leq A$.

Процесс не использует требование больше ресурсов, чем
используется в системе.

② $C \leq B$,

Проц. не использует запрашиваемые больше ресурсов, чем
указано в заявке.

③ $\sum_{k=1}^n C_{kj} \leq A$
по прош.

Никогда не берет больше ресурсов, чем доступны

④ Матрица доступных ресурсов D

$$D = (d_1, \dots, d_m) = A - \sum_{k=1}^n C_{kj}$$

по прош.

⑤ Матрица возможных запросов E

$$\text{Need matrix } E = \begin{vmatrix} e_{11} & \dots & e_{1m} \\ & \ddots & \\ e_{n1} & \dots & e_{nm} \end{vmatrix} = B - C = \begin{vmatrix} E_1 \\ \vdots \\ E_n \end{vmatrix}$$

⑥ Используя запросов F

$$\text{Request matrix } F = \begin{vmatrix} f_{11} & \dots & f_{1m} \\ \vdots & & \vdots \\ f_{n1} & \dots & f_{nm} \end{vmatrix} = \begin{vmatrix} F_1 \\ \vdots \\ F_n \end{vmatrix}$$

⑦ Предварит. состояние.

Пусть у нас есть все запросы
Начальное состояние.

$$D \leftarrow D - F_i \quad // \text{доступно} - \text{запрос}$$

$$C_i \leftarrow C_i + F_i \quad // \text{распределено} + \text{запрошено}$$

$$E_i \leftarrow E_i - F_i \quad // \text{передуется} - \text{запрошено.}$$

Запрос будет удобен, только если пред. состояние D не
безопасное.

допустимое пред. безопас. состояние.

1. Выделяем первого незаверш. проуз. P_i : $E_i \leq D$. Если
такого проуз. нет, переход к шагу 3.

2. $D \leftarrow D + E_i$. Отмечаем P_i как завершенное. Переход
на шаг 1.

3. Всем \uparrow все проуз. отмеченог как завершенное, система
в безопасном состоянии, иначе — в опасном.

Все сист. в недоп. состоянии, запрос блокируется, система
обращается:

Реш. $D \leftarrow D + F_i$

$$C_i \leftarrow C_i - F_i$$

$$E_i \leftarrow E_i + F_i$$

Процесс, ком. D отмечен как завершенный на шаге 2,
обращается. $D \leftarrow D - C$ и отмечается как незавершенный.

73.12.2022
RPC - CUI
бдмко
Моя 8
SVC -

Пример.

max-hail $A = (2 \ 9 \ 3)$

$$\text{max-claim } B = \begin{vmatrix} 1 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

$$\text{Allocation } C = \begin{vmatrix} r_1 & r_2 & r_3 \\ 1 & 2 & 0 \\ 0 & 1 & 1 \\ p_1 & 1 & 0 \\ p_2 & 0 & 1 \end{vmatrix}$$

$$\text{Available } D = (0 \ 1 \ 1) = A - \sum_{k=1}^3 C_k$$

$$E = \begin{vmatrix} 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix} = B_C - C$$

Пусть P_1 генерирует запрос

$$F_1 = (0 \ 0 \ 1)$$

Предварим, что запрос:

Доступно $D = (0 \ 1 \ 0)$, т.к. P_1 требует ресурса r_3 .

$$C = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{vmatrix}$$

$$E = \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix}$$

Процесс P_3 можно завершить, т.к. $E_3 \leq D$. Завершившись, он вернет запрошенное ресурсов в виде свободных ресурсов, т.к. $D = (1 \ 1 \ 1)$. Можно будет удовлетворить запрос P_1 и P_2 . В итоге, активу указывается на то, что система будет безопасной.

Возможен контроль состояния системы.

Сущест. сп-во останавливает выполнение цикла - timeout.

03.12.2022. вп.

RPC - синхронные задержки перед отправкой запроса, нем
важно

Указ $\&$ макросы Числа упр. прерываний...

SVC - supervisor call.

Программа ввода-вывода - драйвер

Функция будет на языке.

Блокирующий системный ввод-вывод.

Почему параллельное обновление?

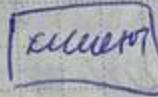
Потому что очередь между ними не одна

Сервер в исх. задаче однопоточного.

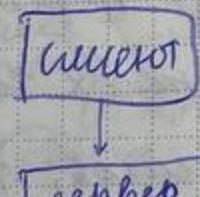
Параллельность:

- создает N потоков, но 1 на клиент.

- потоки дополнено получают доступ к портам клиента.



На сервере возникает список клиентов (pid + номер).

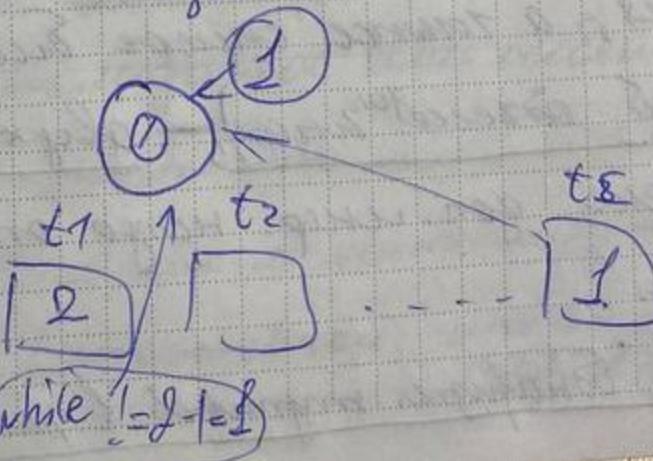
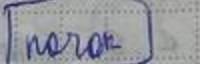


сервер



IP клиент

символ



05.12.2022. лекция.

На экзамене будут считаться чеки для только ручного
исполнения. Электронное признано не будет.

Типичные в расп. системах.

Чаще всего это транзакции. Речь идет о распределенных
базах данных. Всё, что говорится о транзактах, всегда работает
но с определенными особенностями, потому что транзакций
в расп. системах рассмотреть придется.

Специалист, упоминается, что в расп. системах транзакции
все, на неск. сайтах и исп. данных на этих сайтах. Даже
по сайтам расп. неравнозначно, время тоже выравнивается.
В ру-те транзакции и.д. активной на одних сайтах,
неактивной на других. Если на сайте 1 и 2 времепод. (если и 1 и 2
используют) транзакции, и.д. синхронные, когда одна
из них неактивна \Rightarrow возможно, где вон. транзакция вместо
вопроса о времени транзакции (*Transaction Location*).
(Problem)

Для решения проблемы предложен Daisy Chain:

- хранение доп. инф. о транзакции: при переходе от
запроса с 1 сайта на другой хранят список предыдущих
небуд. сайтов, видимых, куда обращалась, сайтов, где были
найдены, и.д. также список тадж./сайтов, которые
она посетила, а также список блокировок с помощью
(блокировок одн. доп. инф.) двухсторонней промежуточной
при открытии доп. инф. надо отправить на все запро-
шенные сайты.

Причина блокировки транзакций в расп. системе

Амортизация (Amortization)
Процесс сразу заменяется. Если при замене
последним его будем
1) номер проу.,
2) номер проу. / 0
3) номер проу., к
При погашении
Если падет, в
в 3)-е паде - не
Затем погасит
Сам проу. на
3) паде, он
Плане соодин
иначе оно

P1
создана
находясь
в транз.

(P1)

Нен. расп.
системы

Эти вопросы
рассмотрены

Управление

Алгоритм Chandy-Misra-Hoas

Процесс сразу запраш. неск. неодн. ресурсов \Rightarrow ищется транзакц. Если при запр. ресурсе занят, процесс теряя. сообщ. сообщаю о блокир. ресур. В сообщ. 3 поля - числа:

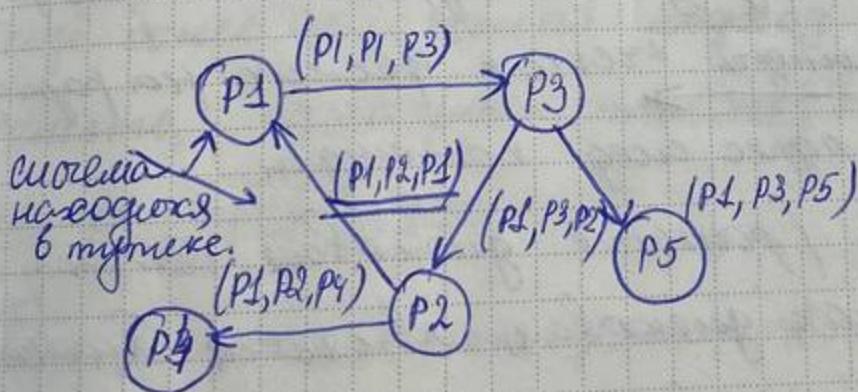
- 1) номер проу, отпр. сообщение.
- 2) номер проу. (помять неч. идущ. пишет свой номер)
- 3) номер идущ., котору сообщ.

При получении сообщ. проверка, пишет ли проу. ресурс. Если нет, в поле 2) этот проу. записывается свой номер, в 3)-е поле - номер проу, от кот. пишет освоб. ресурса. Затем посыпает сообщение дальше.

Сам проу. посыпает сообщ. и однажды. свой номер в 1) и 3) полях, от однажды, что числ. в группе,

Такое сообщ. наз. собщение-зап.

инициатор	отправление	получение
-----------	-------------	-----------



Как блокированный процесс может что-л. аномализовать, отпр., посыпать?

Вари. ответ: У проу. есть отдельный поток для сообщающей.

Чт. распределенных БД лимитируется ограничением ресурсами систем. Но это насыщивает свое особенности.

Эти вопросы не имеют однозначн. решения. Есть базовое, что рассмотрим, они чистые всего как-л. комбинируются.

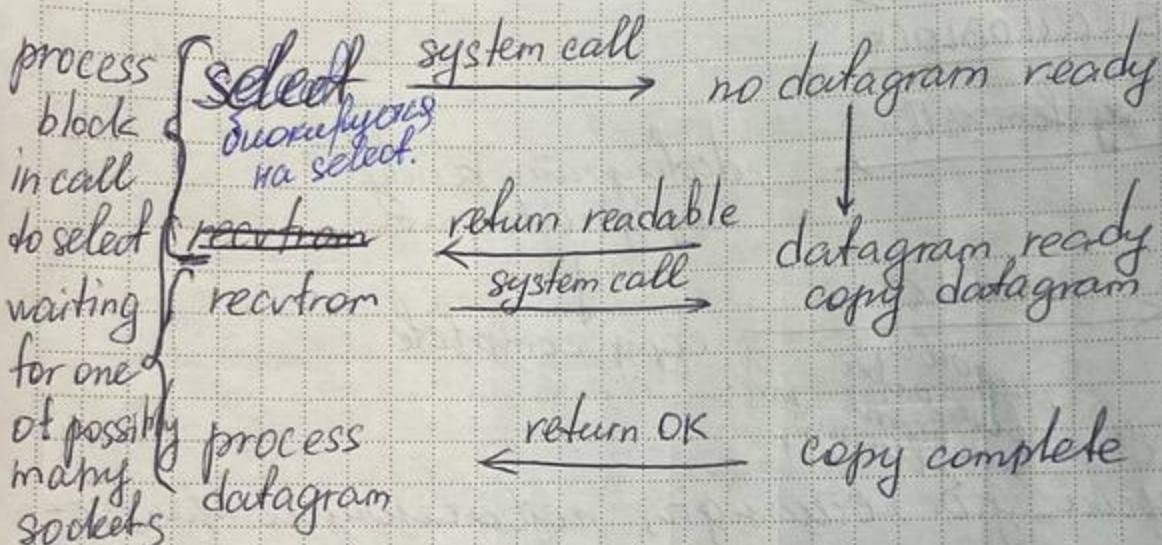
Управление памятью. (оперативной, чтобы память физич-
ки, в т.ч. оперативную

Применение

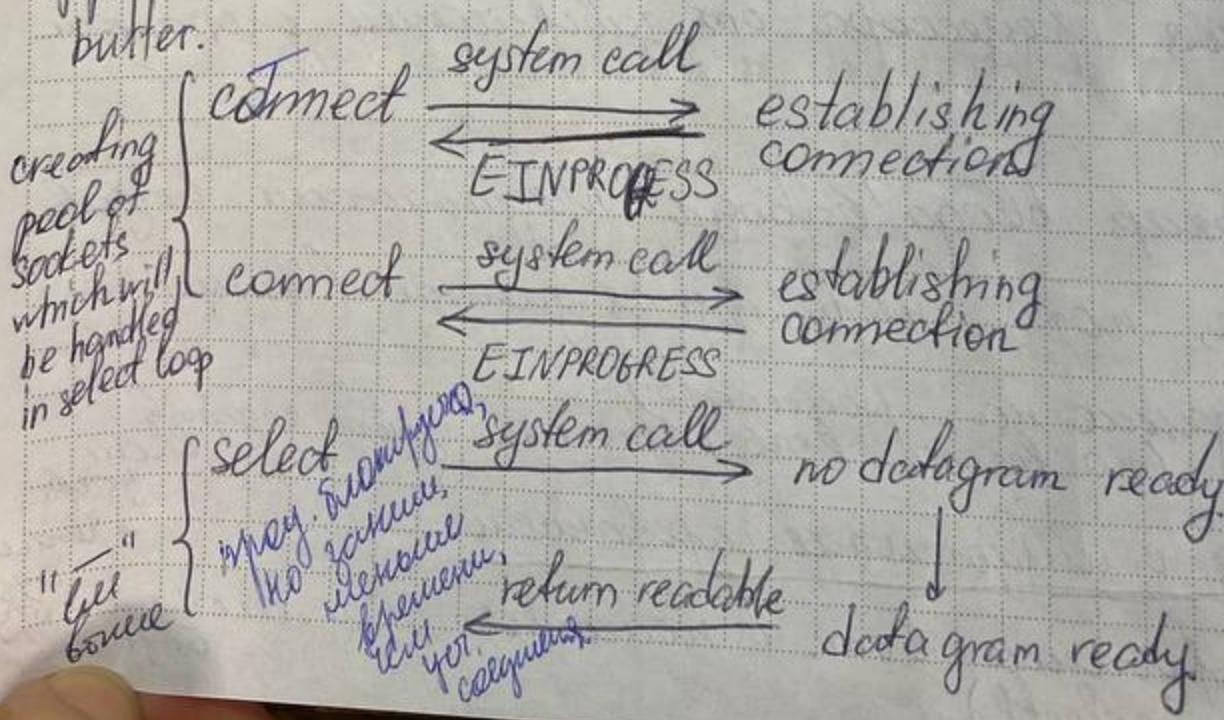
Языки

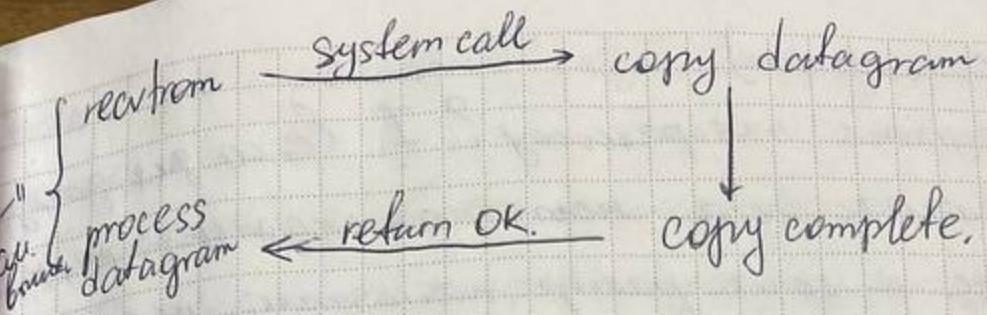
- языком программирования, основанным на сокетах. C++, Java, Python, Perl, Ruby, C, C#
- это адстрактная концепция машинного представления.
- реализуется на языке машинного представления. Гарантия правильности работы.
- ~~использование send() / receive()~~.

Использование языка программирования для автоматизации сетевого взаимодействия.



Соединение reply socket — это point 2 point, log emas reply over both socket. Для соединения можно использовать `socketpair` и `socket`, что более удобно.





12.12.2022г. Лекция.

Кэширование — это основной ресурс, вместе с процессором почти бесконечен.

В тупиках не рассмотрен вопрос о восстановлении работоспособности системы. Самый простой способ — перезагрузка.

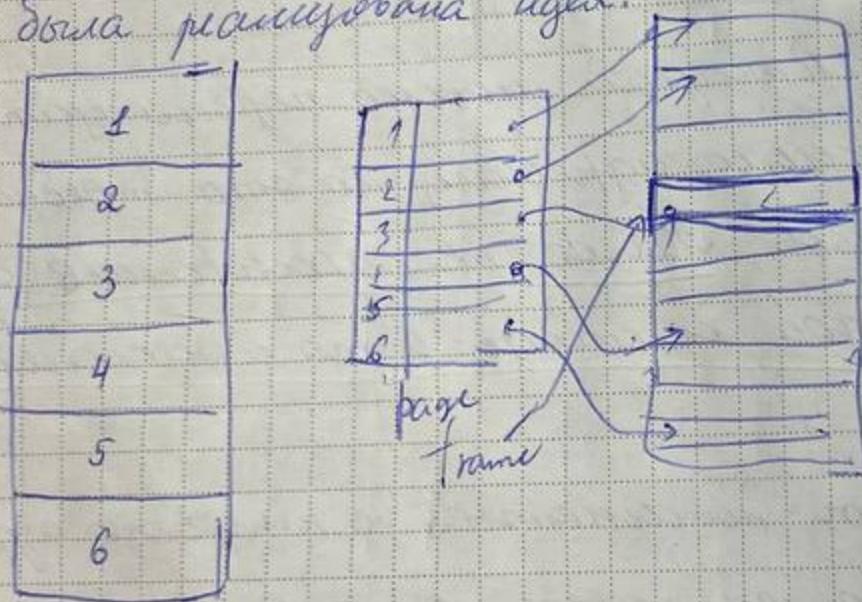
Она борется с ограниченностью памяти пересыпая программы к новым адресам, но это требует преобразования адресов. Такое преобразование выполняется на каждой команде, а то и несколько раз.

Программа может начинаться с нулевого адреса. Каждая программа считает, что она начинается с нулевого адреса.

Преобразование остается, и в программах изменяется. Системное программирование: 1) ОС, умножая ХС ; 2) транслятор, редакторов своей загрузки. Это приводит к недавнишним проблемам. Она прокладывает пересыпание — пересыпание начального адреса. Загружчик загружает прогр. в свободное АП, в первых кадрах — начальный адрес можно вбога.

(адрес раздела + смещение).

Когда перешел к программе? I. Если же зонами не
ки очередной процессор нет подсогрева, то
Все блоки отк. к испр. расп. памяти, пра-
дильна загружаются членами. Но программа
блочная, и горячий способ упр. разделения становится
бессмысленным. Важна идея введение памяти брау-
зера, ком. получающие память отрицают.
Все страницы процесса загружаются в ОП, если
и блока размежевана идея:



Это структурное распределение памяти.

I. Внедрение памяти клиентами

Разработчик сессии может изменить свой разработчик Веб-брау-
зера, арх. док. Неймана, запускаема клиентом браузера
ти программы, которая называемого загружена
в ОП.

От введения браузеров перешли к виртуальной
памяти. В которой процессор браузера г. д. загру-

зато зажигаю
его разве
програм
ммы бы
нашёл ке
брайтло
биско пы
берущий

меня же участки кода, к которым обращается про-
цессор.

Управляемое вирт. памятью.

1. Внедрение страницы по запросу
2. " — " сессии памяти по запросу
3. " — " сессии памяти, подгружаемой из памяти, по запросу.

I. Страницная память.

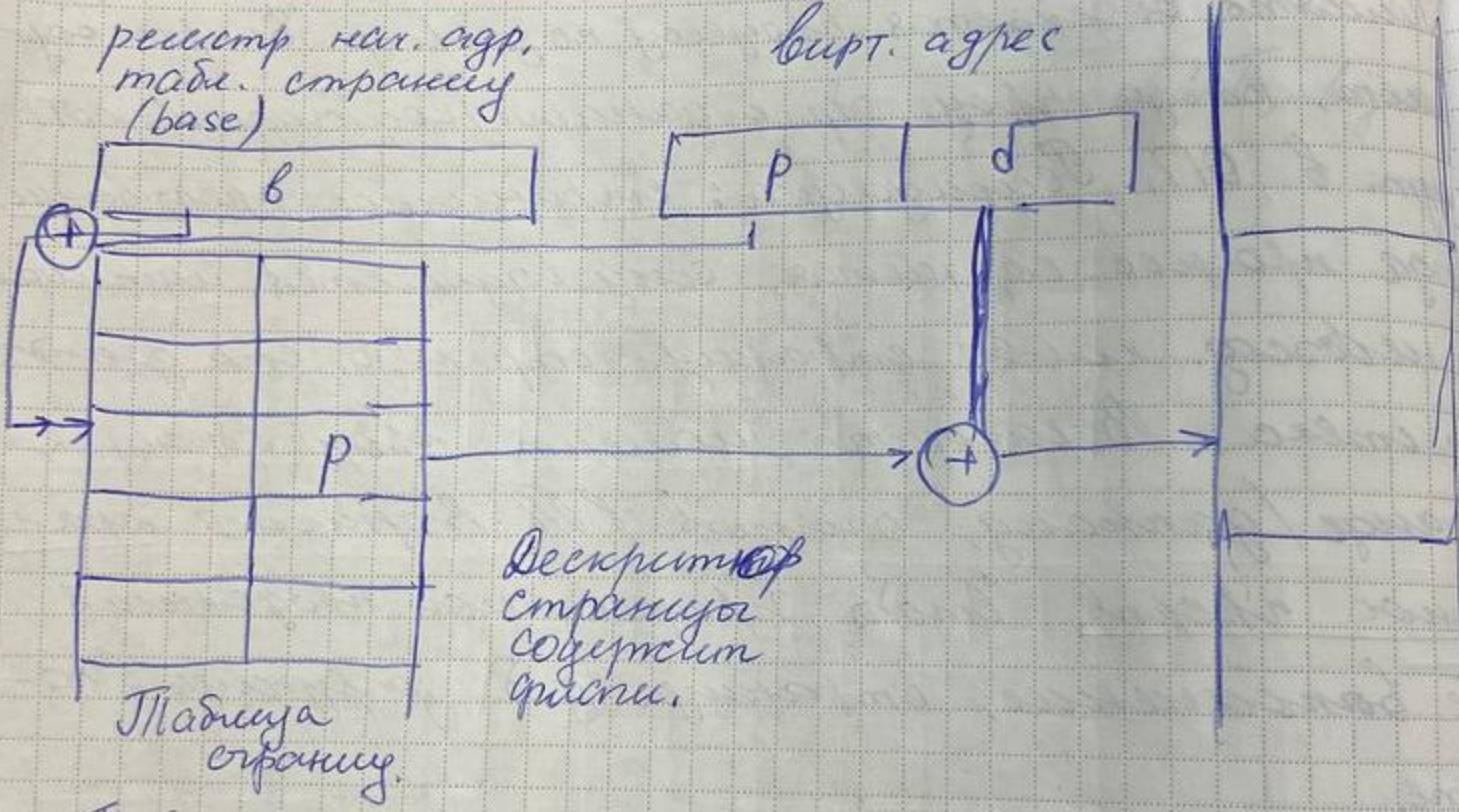
Память вовлекается процессу по запросу. Запрос вы-
никает, когда приу. обр к данному/командам, кот.
отсут. в ОП. Решается: страницное прерывание,
когда процесс создает, если вовлекается инициа-
тив. недоступ. или-бо страницу: стр. кода, стр. данных,
стр. стека. В какой-то момент приу обрну. к
данному /данным/, откуль в ОП вовлекает огра-
ниченное прерыв. Чтобы процесс мог продолжать
своё выполнение, страница я.д. загружена в па-
мять.

, Программа исчезает с первового адреса" — иск-
чесце АТЛ. Следующее берется из программы, Пр-
грамма находится на диске. Требует о том, что
АТL если не находиться, виртуальном. — это вовлекаем
сущей АТL. Когда приу. создает, где несоглас
виртуальное АТL. Программа вовлекается пр-бо,
и оно содержит сообр. сессии (кода/данных/текст),
кто пр-бо генерится на страницу.

Для того чтобы загрузить вирт.adr. в ячейку, на
и работать с адресами, необходимо дополнительное
образование.

Схема преобразования

- прямое отображение;
- ассоциатив. отображение;
- ассоциатив.-прямое отображение
- прямое отображение



Память страницы находится в оперативной памяти.

- ассоциативное отображение
предполагает наличие в системе блока ассоциативной памяти, которая всегда регистрирована; т.е. должна. Ассоциатив. память устроена таким образом, что за один такт проходит сравнение со всеми имеющимися и выборка подходящего знания.

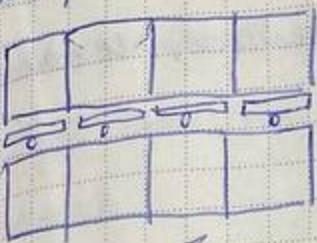
В 32-разрядном
регистре регистра

Число
ассоциативных
страниц

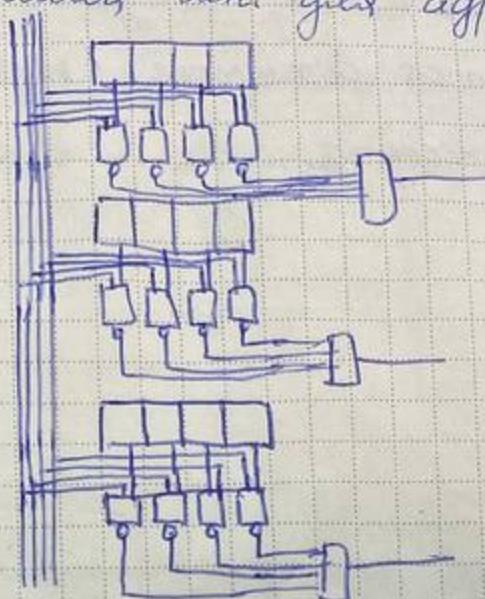
- ассоциативное отображение

В ке
ни, к
одноре
в ке
как ес
кеин

В 32-разрядных системах адрес 32-разрядный. 12 разрядов — разр. регистра, оставшиеся используются для агрегации.



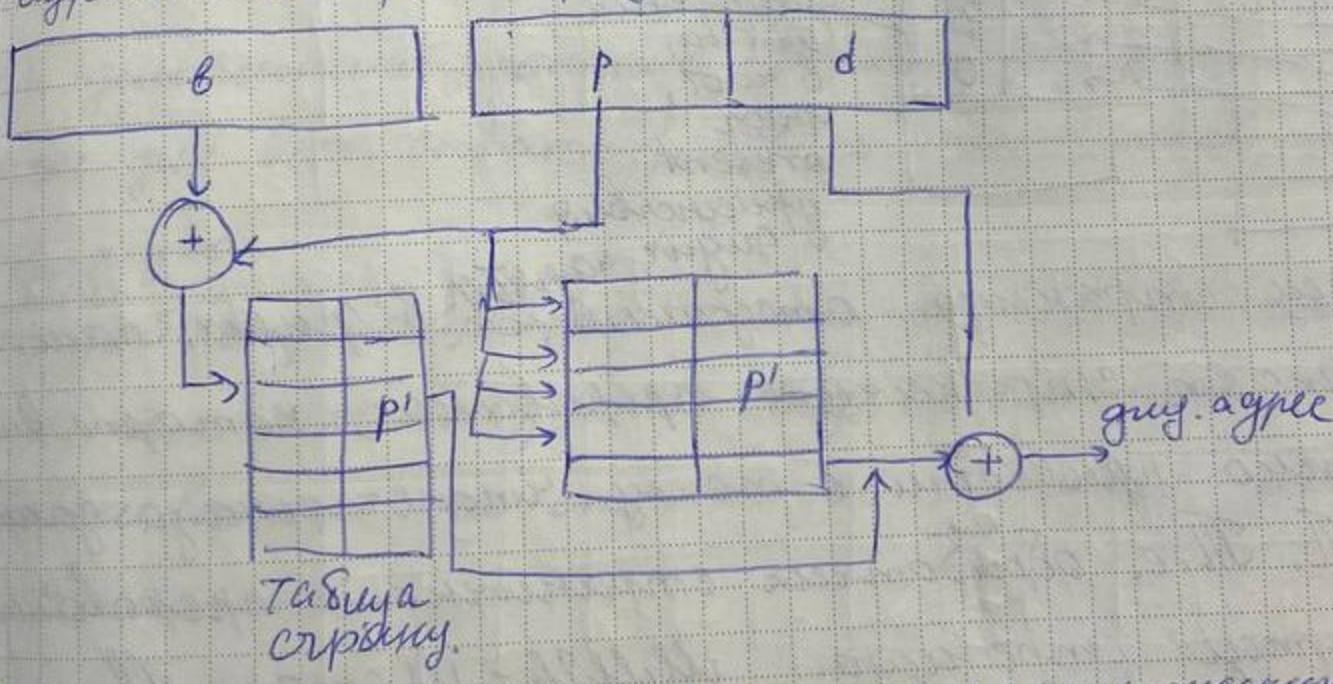
Имеет большую
академическую
нагрузку очень до-
рога.



- ассоциативно-прямое отображение

регистр базового
адреса табл. срв.

виртуальный адрес



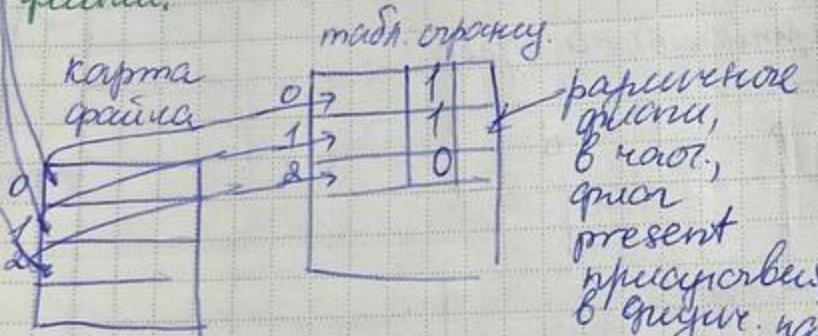
В хеше должны храниться адреса физических стра-
нищ, к которым были недавние обращения. Если более
обращения к странице, исходя из неё загружают
в кеш. Это обеспечивает до 98% ядер. обращений,
как если бы было обращение к количеству ассоциатив-
ных.

Страницы при прерывании предполагают блоки на работу. В ПД делятся на две первые части, блоки, которые хранятся обычное (regular) время, и блоки скомпакт/песцессия.

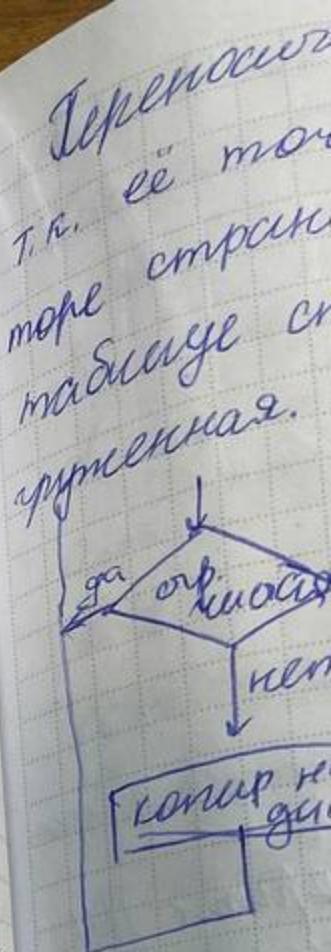
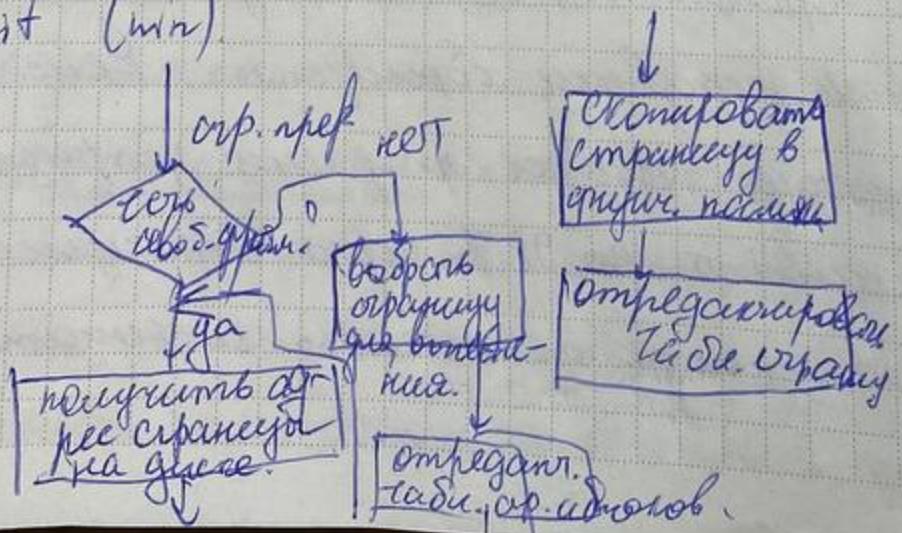


БЗУ

В системе должны быть таблицы, ком. изображаются карты страниц.



Всех страниц отсутствует в табл. памяти, возникает страницное прерывание, которое в свою очередь привести к тому, что страница загружена в ОП. Т.е. обработка страницного прерывания загружает страницу. Millib - Memory Management Unit (min).



анализ

1. Время

не забыть г

и ма все

напомин

2. Конфигурация

системы

Возмож

беско в

19. 12. 20.

	4	9
1	2	3
4	5	6
7	8	9

Факторы
теста

Переносить на диск вонднее шестую табличку означу, т.к. её точная конец уже находится на диске. В десктопе страницы есть один обращение и один dirty. В таблице страница страницу помечается как битоменная.



Физическое вытеснение страниц (page replacement)

1. Возвращающее спис. страниц. Плане наз. расход, не зв. дискассионное (вероят. вытеснение одна из них при всех страниц). Из-за недостатков применяется FIFO.

2. Каждой странице или присваивается временная адреса, или организуется сви. список типа FIFO. Возвращающееся ма страница, которая давнее всего в списке.

19.12.2022г. лекция.

	4	3	2	1	4	3	5	7	3	2	1	3
№3	3	4+	3+	2+	1+	4+	3+	5+	5	5	2+	4
F	+	+	+	+	+	+	+	3	3	3	5	2
macro tagger.	9	12	=	75%								

Для рассмотрения FIFO и. и. временн. страницы - организуетя список сортированный FIFO - удаляемые страницы сортируются по времени создания страницы. правильной (неправильной).

возвращающая страница неудачи

$$\frac{10}{12} \approx 83\%$$

Sinopumice LRL - Least-Recently-Used
12 ~ 53%.

Необходима временная штамп/свайное стяжки с регулируемой при А обрашаемой.

4	3	2	1	4	3	5	4	3	2	1	1	5
4+	3+	2+	1+	4	3	5+	4	3	2+	1+	5	
4	3	2	1	9	3	5	9	3	2	1		
4	3	2	1	9	3	5	9	3	2	1		
P	f	f	f	+ +	3	(2)	(2)	1	1	(1)	(1)	3

ЛНУ обнажены красные замерзшие.

Пограничные, 220 ЛРУ отмс.

Carex bromoides Stev. *Sp. nov.*

Ми спочатку сформуємо побічний ряд
з ряду a обмеженого $M = 3$ а $d = 4$.

Диноритик I РУ
насаждаемого
отвергает
одобренного
согласия
пропаганды:
— съ-бо и-
качиваемы,
Съ-бо доказ
соглашения
к огранич
находил
вероятно,
что съ-бо
вогородив
был к съ-бо
эти огранич
1

Следует о
стремлении
предъявлять
согласие
безусловно
всегда
небольшое
изменение

B m
gas p

Алгоритм LFU - least frequently used page replacement.

Наименее часто используемый страницей именем означает то, что заслуживает, т.е. не используется наименее как-то обрабатываемый.

Алгоритм NUR - not used recently. опровергается тем, что LFU.

Конк. виду прим. без обращение, вов. хот.

если при обращении к странице (использовано 0), с некот. периодом все биты обращения сбрасываются в 0.

Страница для замещения выбирается среди тех страниц, у кот. бит обращений = 0.

№ отр. бит. обр. Бит модификации

0	4	0
1	5	1
2	1	1
3	2	0

	бит обр.	бит модиф.
1	0	0
2	0	1
3	1	0
4	1	1

модиф.
бита до
последн
сброс всех
битов од-
ражения в 0.

Эксперимент показывает, что прогрессия никогда не
брал. во всем своем спектре.

1968 г. Детали предложеные в качестве искара производств.
вывод кот. во страницу, к кот. битам. прогрессия за
время t .

$$W(t, dt) \rightarrow$$

$t + \Delta t$
working set.

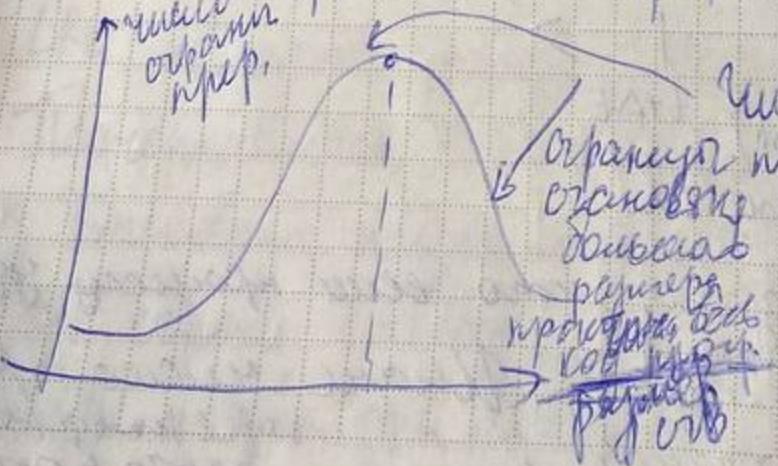
В ре-те эксп. было показано, что если процесс удачно
загружает в память working set, то не число обращен.
он бит. бит страницы превышает $\frac{1}{2}$ предполагает
переворачиваний



Страница прерывания возникает при обращении к
датчикам, встроенным в память. Регистровано (в Intel)
как исключение, обработчик ком. инициирует работу цикла
памяти, ком. загружает страницу с диска.

Все это (все эти рассуждения) дают возможность уменьшения числа заданных — как в программе, одновременно могут находиться в ОП.

Влияние разнотипа спринклера при сопр. обогрева ОН
на число спринклерных прерываний.



Число
из неправиль-
ных перво-

на
рекомендации
проверки
в ОН наложены меры
контроля ведения и гашения
и в том же время взысканы
суммы, взысканные взысканием

Drs. E. B. Vipond
He won \$300

Intel определяет битами разрядов 4КБ, где 00 значит
нон-действительный разряд. Т.е. недейств.
записи в кэш-памяти. Время записи превышает
время записи (уровень записи
запись)

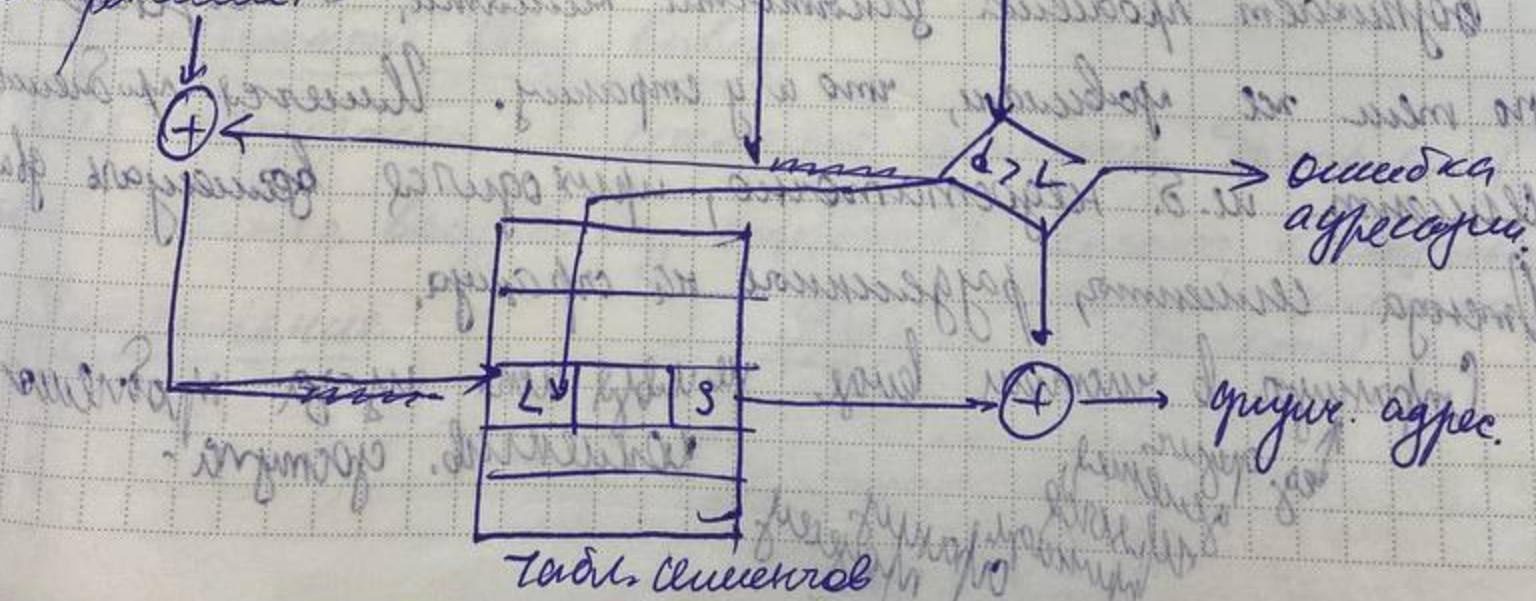
В собр. микросхемах нет общего строения. Приводится
управление памятью синхронизировано по заранее
установленному времени выполнения команды записи
памяти.

В прош. занятии находился регистр базового адреса
мак. модуля сегментов.

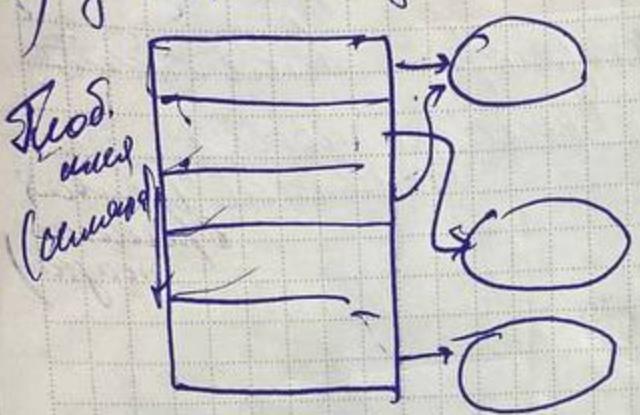


Баз. адрес записывается на эти ячейки. → сдвиг
мак. значения базового регистра сдвигается в сегмент
записи. — вагина в DOS, мак. размер 64KB (16-разряд.)

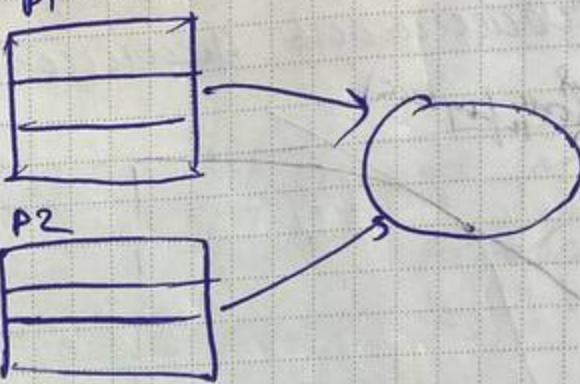
В процессе LONG нет сдвигов, она одна в сегменте
мак. записи.



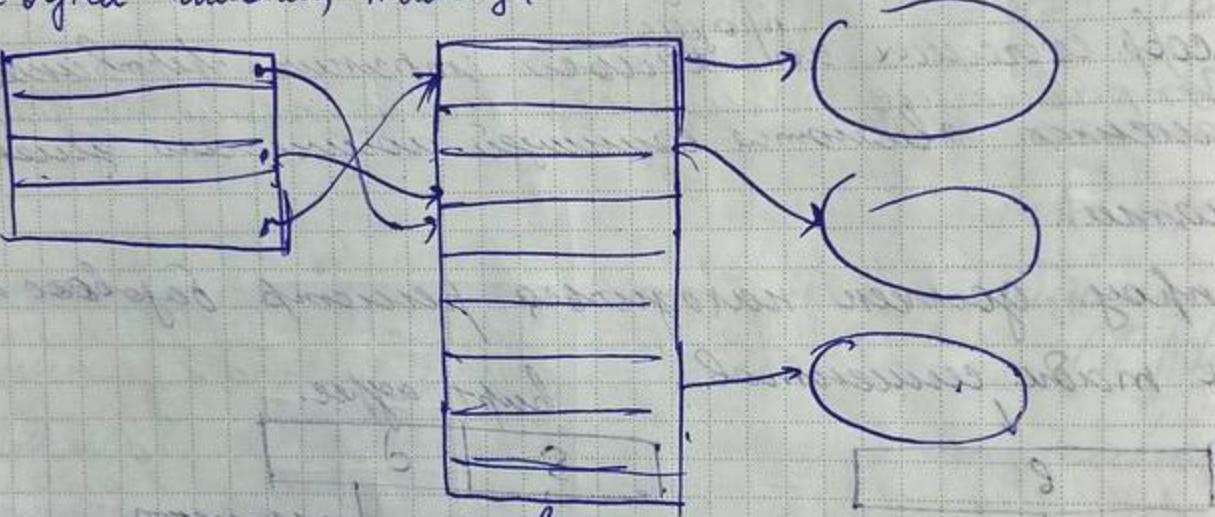
1) Единая таблица.



2) локальная таблица,
(унарный процесс)



3) локальное табличка
и огра. внешн. таблица.



3-я схема возможна в ре-ме синхронии реализующим
~~согласие~~ согласие коммита сессий. Все сессии (при)
запуски в памяти работают.

Возникает проблема запоминания записей. Запоминание вов-
но тем не проинициализированы, что и устраивает. Инерес проблема:
сессия и. д. недостаточно, приходит вспомогательные гла-
шатели сессий, разделяющие на сегменты.

Сравнив в чистом виде неизб. сесс. из-за проблем
сессий, проблема:

23.12.2022

Руководя

сиг. устр-
вова - вова

В наше
нашее ико

Для мого
что делает

При блок-
нока данных

4. Ввод - вов

Прином

establis.
Signal

process
continues
executing

Sign
Rec

process
blocks
while data
copied into
app
buffer

5. Асинхро-

Для ног
вовов. асин

Прином

23. 11 2022 г. Семинар.

Рукующая управляющая блессышия устр-вами передана
сис. устр-ву. - рогор. устр-в в нашем сис. с систмой арх.
Ввод-вывод.

В наших системах (у типичных программистов)
наиболее просто такого не сделать.

Для того чтобы применение можно получать данные,
это должно потребовать процессорное время.

При блокир. вводе-выводе мы. (примеч.) блокировано,
пока данные не будут получены в будущем применении.

4. Ввод-вывод, управляемый сигналами

Применение

establish SIGIO
Signal handler

Ядро

sigaction syscall }
return } wait
for data

process
continues
executing

signal handler ← deliver SIGIO
recv from → copy datagram
process datagram ← return OK ↓ copy complete

5. Асинхронный ввод-вывод

Для поддержки в системе должны быть сис.
ввода. Асинхр. ввод-вывод относится только к основному
средству ввода-вывода.

Ядро.

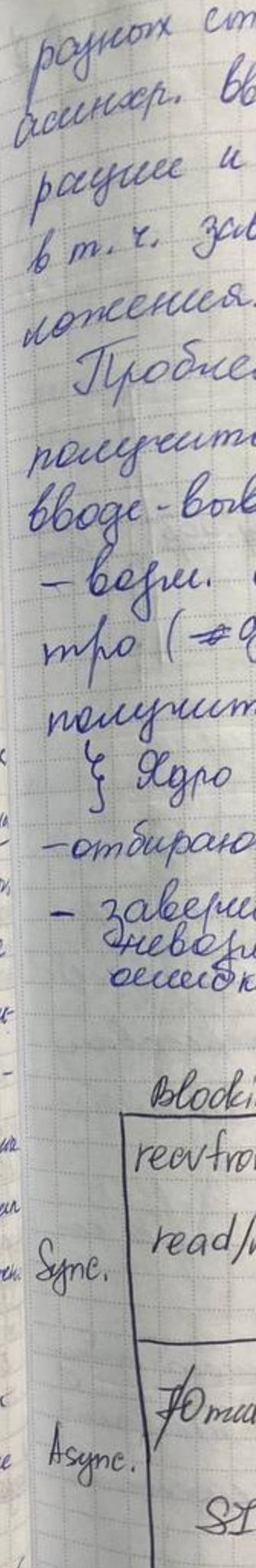
Применение



Дис. блоха-блоха, управл. синхрониз.

Блокируется sigaction(), устанавливается обработчик сигнала SIGIO, блок. буферам (и устанавливается сигнал SIGIO, блок. буферам) и устанавливается сигнал SIGIO, блок. буферам (и устанавливается сигнал SIGIO, блок. буферам). Это работает аналогично ядро. Это означает, что когда данные готовы, посыпается сигнал, в результате чего дис. блоха-блоха (callback-функция). Сигнал recvfrom можно блокировать в основном потоке программы дис. блоха-блоха и в основном потоке программы дис. блоха-блоха, назн. в качестве менеджера, т.к. это работает с единств. задачей. Для этого используется SIGIO, так как в ходе работы однодом. программа SIGIO, так что менеджер. Если sa-mask=NULL, то группе не будет менеджера.

Функция асистент. блоха/блока определена в POSIX. В POSIX синхронизирована, в отличие



разных стандартах, обладающих особенностью. Все асинхронные ввода/вывода сообщают ову о начале операции и уведомляют о процессе о завершении операции, в т. ч. завершение ~~ввода~~ ~~вывода~~, который в будущем не понесется. Отличие: сигнал приходит по каналу к копированию.

Проблемы асинхронного ввода-вывода: недостаток синхронизации асинхронного события. В случае в асинхронном вводе-выводе вспомогательное на двух каналах:

- перед. опред., что ввод-вывод можно вкл. бордюро (драматически задержка готовности, между теми получит);

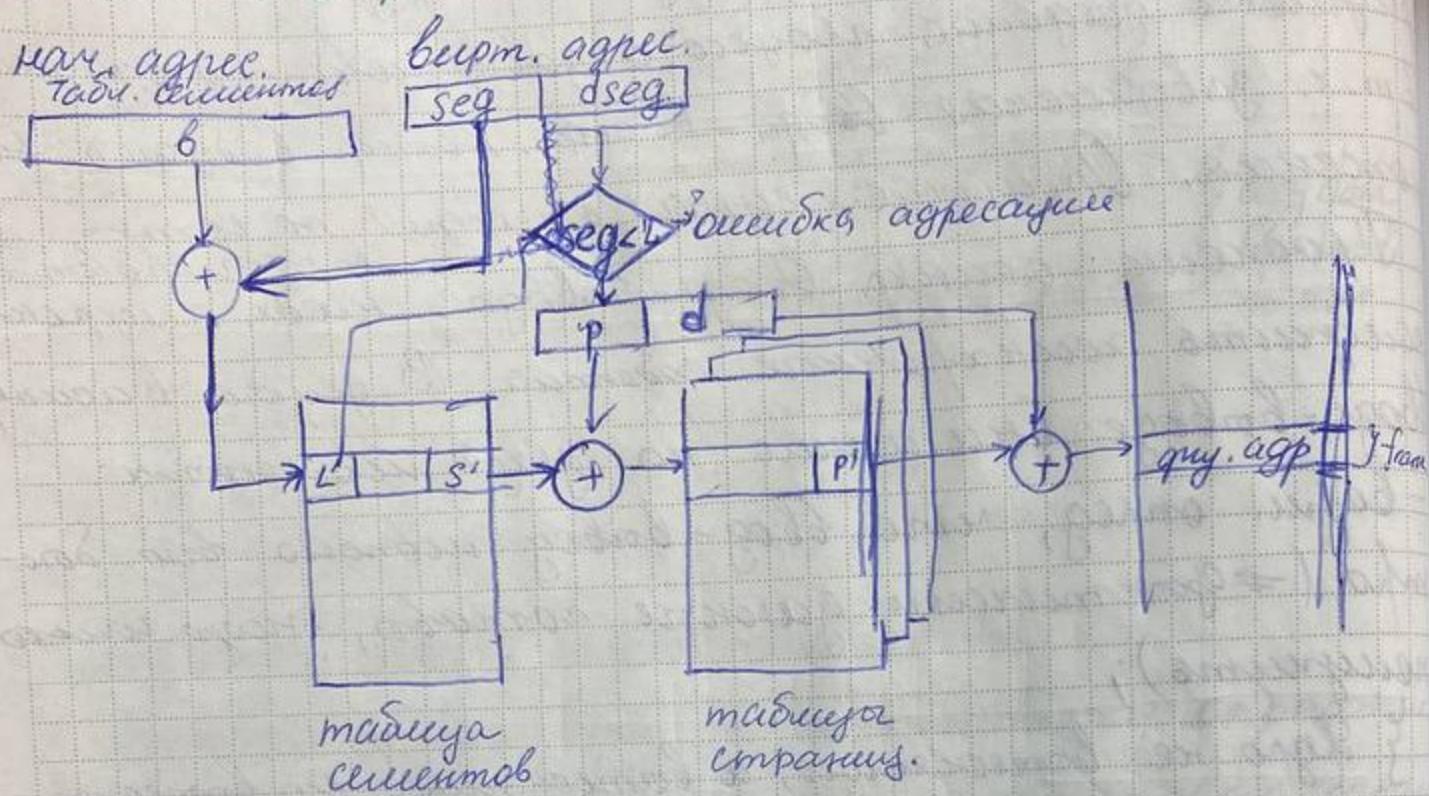
↳ Дело не виновательное, а виновательное; его виноваты отображают проф. время - а не он виноват.

- завершение оп. ввода-вывода в моменте сразу же возвращение. вкл. ввод-вывод сразу \rightarrow возвращение.

Blocking	Non-blocking
recvfrom/sendto read/write	polling
multiplexing SIGIO.	AIO.

26.12.2022г. лекция.

Сессионно-страничное преобразование



В таблицах находящиеся физич. адреса, комбайн параллельно - ищемение таблиц \Rightarrow переход к сессионно-страничному управлению назначено. (проблема сессий)

Коллективный доступ: один проц. заинтересован в общ. содержимом, другой – в изменение (проблема ординации).

Если проц. заинтересован в изменении страницы, ее страница будет создана в АП процесса.

Комбайн сессий делится на страницы \Rightarrow число под страниц, кол-во определяется кол-вом сессий программы.

Табл. сессий на рис. является некоторой, содержит инф. о сессиях, с кот. работает программа.

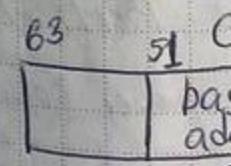
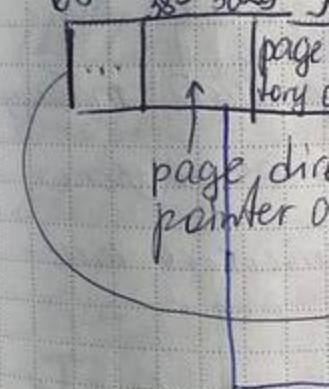
Сессии данных, когда, места оставшись, они находятся в АП процесса.

В игре отмечены
В ре-те исо
раст. шахмат. к
 \Rightarrow очень больш
Победа
табл. в общ
В табл. стр
машину р
Собр. комп
Рассматри

Регистро

ноге. В проце

$64k \times 4096$



нача
адр
4-го

{ В игре оптимизируя fork приведена схема 32-разряд. AT }

В ре-те можно откаститься от табл. сессийн. Последний фок. шахте. компютера, обеицав памяти, обеицав приложени
⇒ очень большое AT.

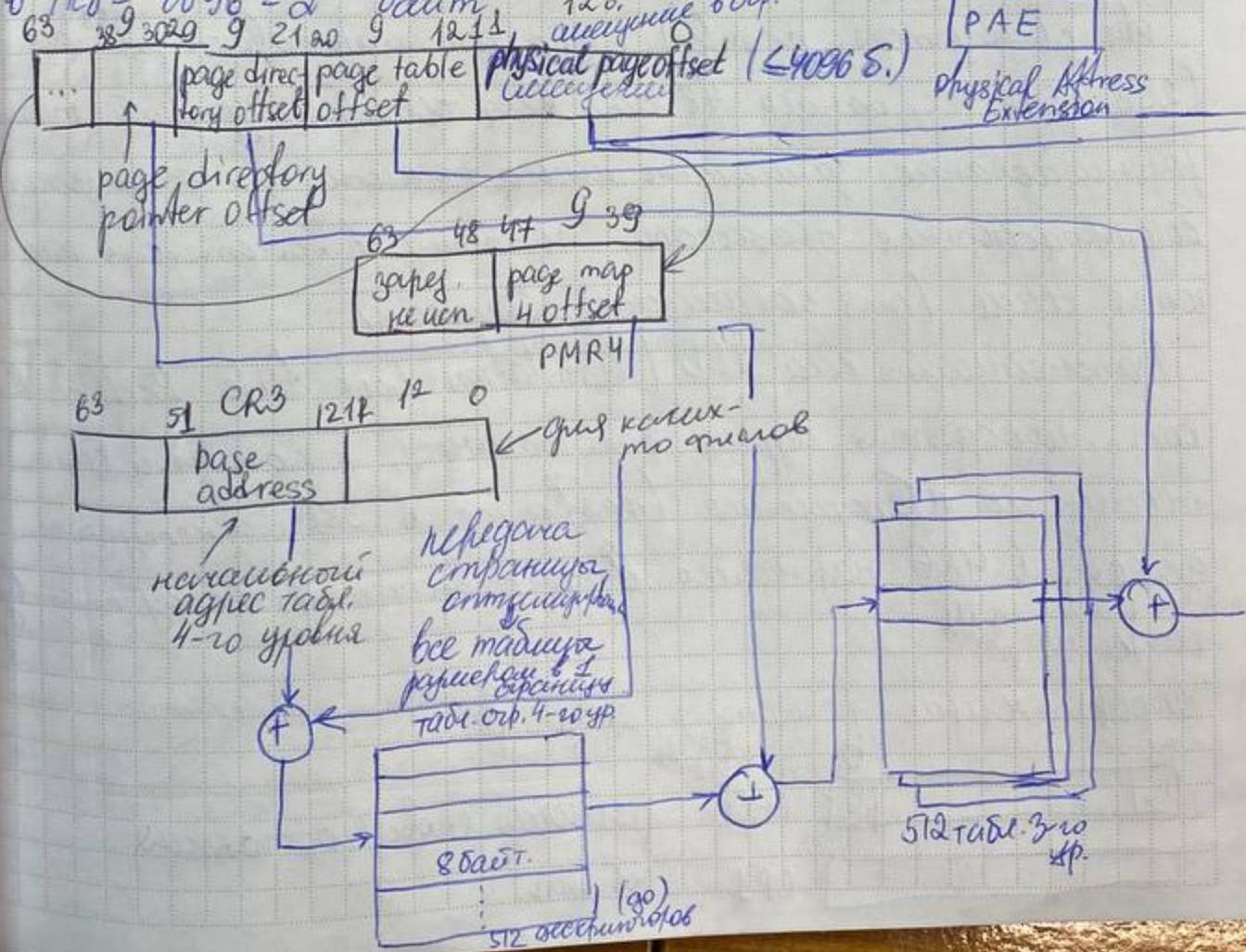
Память страницы — системная. Иного страницы — добавляю
табл. в области ядра. Присоединяя к инициал. табл. отрасли.
В табл. страницы хранятся ^{дескрипторы} табл., в кот. в данный
момент раб. процессор.

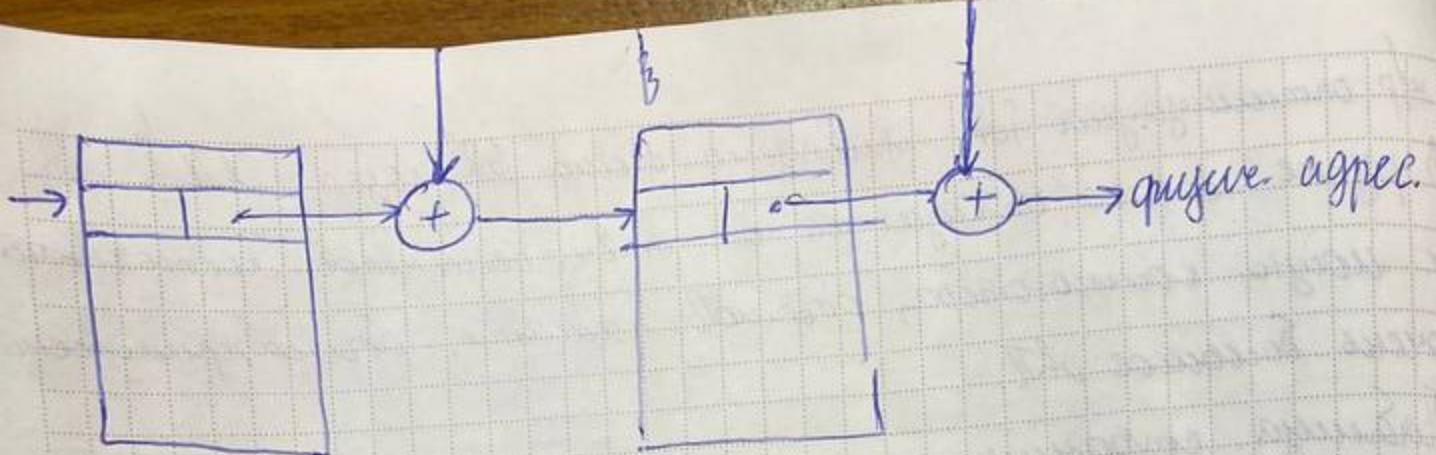
Собр. комп. работают в режиме long — 64-разрядном.

Рассматриваем x64.

Регистр 64-разрядное ⇒ адресное регистр 64-разряд-
ное. В процессе разбивки устанавливается развертывание страницы

$$8 \cdot 4 \cdot 5 = 4096 = 2^{12} \text{ байт} \quad 12 \text{ байт алюминиевая вор.}$$





262144
табличн.

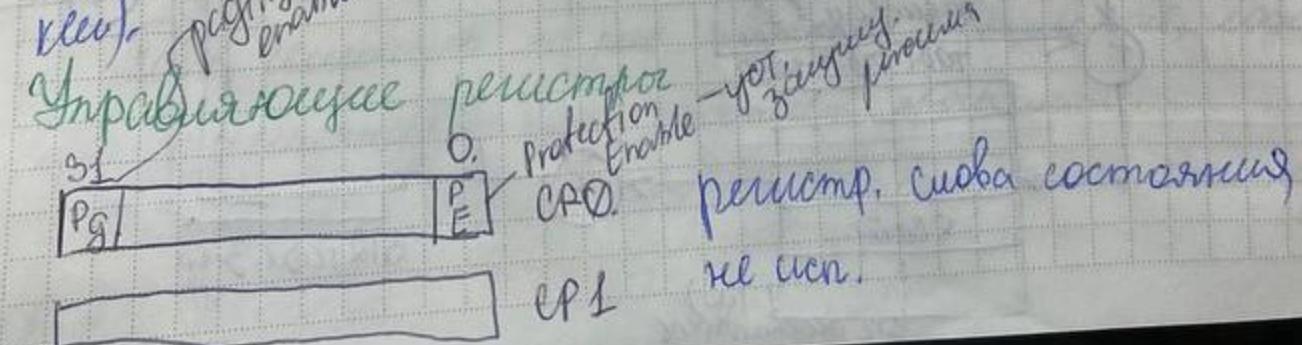
262144.512
табличн.
134217728

Все страницы сразу же создаются. Создаются страницы сразу же, сразу же создается. Создаются страницы сразу же, сразу же создается. Результатом преобразования является обработка страниц.

Результатом, за которым усиливанием Excel, за который называют "наибольшие расходы" — увеличение производительности.

Адрес (указатель, register) называется адресом 2^{48} бит (256 ТБ), т.к. размер 48 бит. Для некоторых уровней таблице размещения занимает много времени с точки зрения быстродействия процессора, поэтому в кристалле имеется кэш (для сохранения времени).

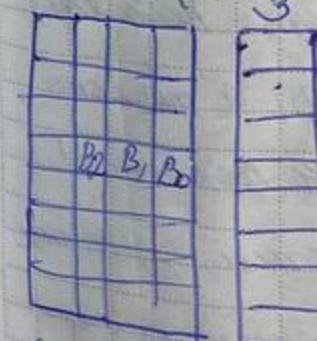
Важнейший кэш TLB (Translation Look-Aside Buffer), в котором находятся адреса физ. страниц, к которым были последние обращения (появился в 386, и используется до сих пор). Рассматриваемый кэш.



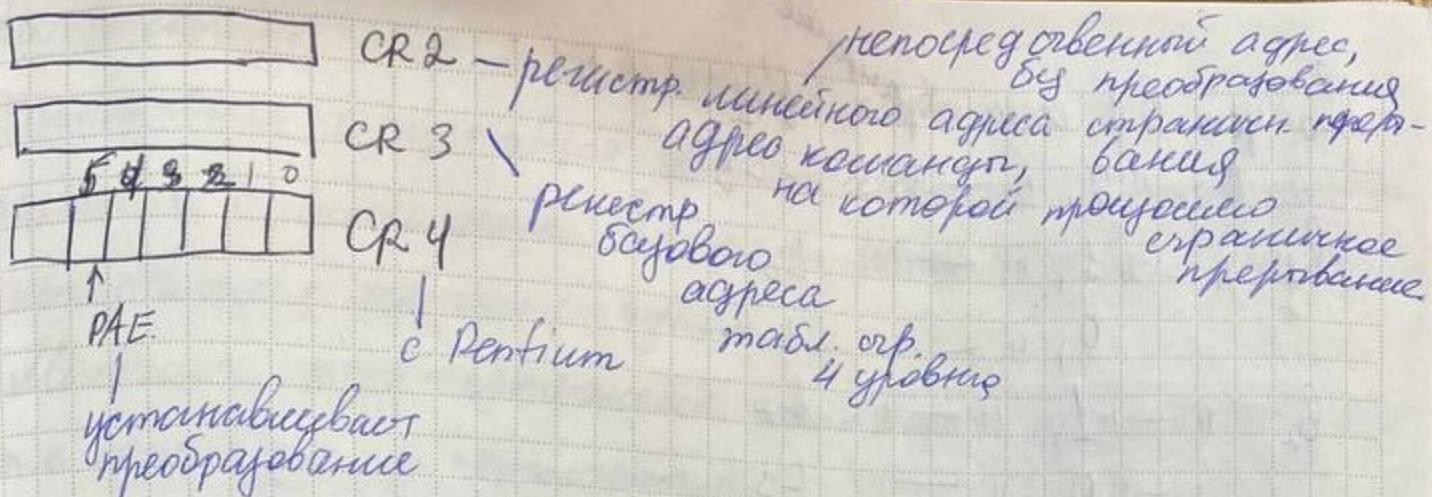
PAE
устаревший преобразователь
В регистре и PAE.
регистре, они же
в регистре страницы,
расширяя.

В простейшем
V86. Красиво
второго уровня
третьего уровня

Блок управления
и Р
TLB (486)



Блок управления
(двоичный)

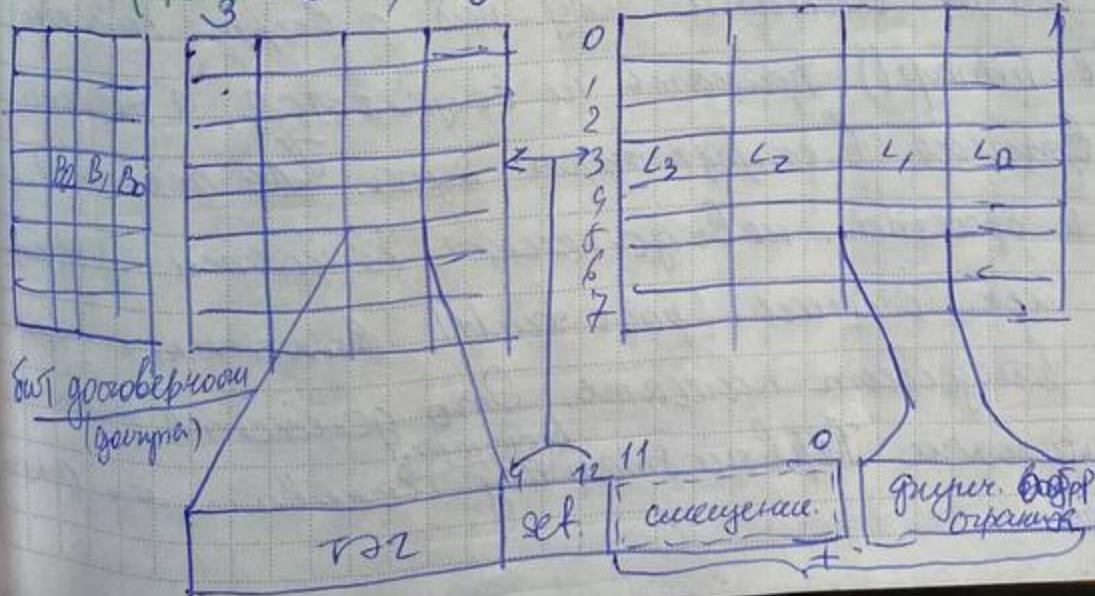


В реальном режиме имеются два преобразования: сегментация и PAE. Существуют архитектурно-закодированные регистры, они находятся в ОЗУ по известному адресу. Чтобы в реальном long mode был адресован, создается таблица страниц, на одной хомологичный уровня (т.к. ОС - более низкий уровень).

В простейшем случае TLB реализуется структурой RRU. Кроме TLB, который прилож. имеет кеша первого и второго уровня L_1 и L_2 и кешем обратного к кешу третьего уровня L_3 .

Блок дескрипторов и РУ общих для L_1 и L_2 .

TLB (48B) - частично ассоциативный кеш.



Был усовершенствован для при загрузки кеша и при обращении к сектору обратно

Был усовершенствован для при обращении к базовому избыточ.

б) учм. б1, если обратное
n - n0, " - " L2 L3

б) " - " 1, " - " L0

б) " - " 1, " - " L2
" - " 0, " - " L3

Благодаря алгоритму Pseudo-LRU всее обеспечивают
лучшую схему, чем в случае полного ассоциа-
тивного кешия.

(использование frame)
Descriptor entry (page table entry, PTE).



Черновая алокация
(чёрное выделение памяти)

В Linux определяет набор структур, которые работают с буферизованной памятью, которая создается для ядра. Текущее, что большую часть времени проходит с виртуальными страницами. При выполнении mmap() память не выделяется, а модифицируется в структуре ядра. При первом обращении к странице, неводимое обращение (page fault), происходит компоновка страницы. Использование (page fault), основанное на предварительном выделении памяти. Это делается для быстрого выполнения. Первый раз страница выделяется, он о

для системного
адреса и с
использованием
множеств
м.е. сопостав-
щимся
набором кеши-
стиве с ади-
ческим про-
цессором
сопостав-
щимся
набором
в рабо-
те процес-
са

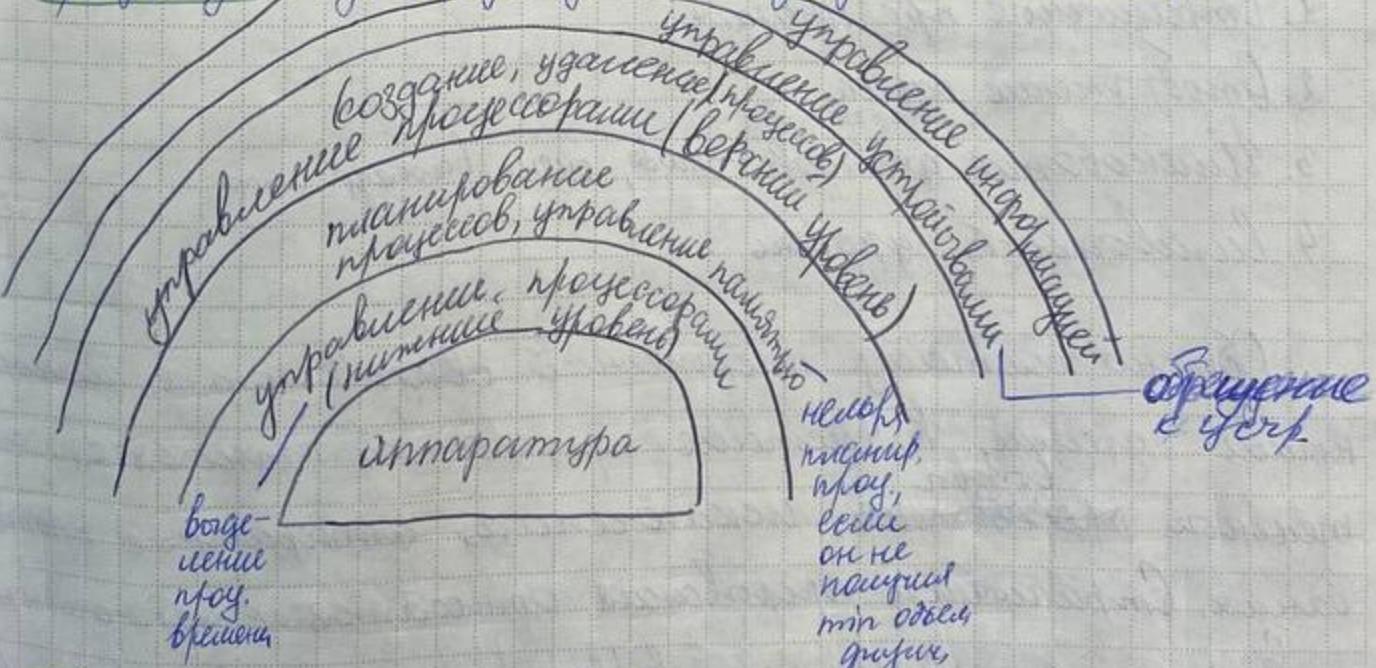
После
использо-
вания
запра-
вляем

архитектурой ядер операционных систем.
Практически существует две архитектуры: с именованием
адреса и с адресом. Системы Windows и Unix/Linux это
системы с именованием адреса.

Монолит, адро-программа, используя авт. структуры, м.л. состоящая из подпрограмм.

Микрагрегированная архитектура – исходясь издря, реализующая набор жесткоустановленных действий, т.е. неподвижных взаимодействий с аппаратом, частного системами и обесценчив. взаимодействий между процессами системы, ком., верисети, именем собственное АП и бол. на ур. нач. Исподчин микрагрегра обесцен. вращ. между. приу. пользоват. и приу. пользовает. и приу. оп. спот. с помощью передачи сообщений. Микрогрегра это можно

В иерархической машине иерархия построена по осям. всё речь о ре. здравоохранении?



Последовательность ввода-вывода
получаем от приемлемости
запросов на ввод-вывод и блокируется
комб. системами ввода (7).

ИБМ2Т

Структура ядра BSD версии 4.4.

для того, чтобы более конкретно определить, что находит
ся в ядре оставит. Для этого необ. чётко пред-
ставляет иерархию действий в системе.



1. Страницное прерывание
2. Отображение адреса
3. Именование файлов - mode, очень блокир. инф.
4. Символический уровень

Обычно систему прерываний связывают с машиной здравии. К системе прерываний относятся: машинное ~~прерывание~~^{ядро}, исключений, аппаратное прерывания. Страницное прерывание относится к исключению.

Единицей диспетчеризации в Unix/Linux является поток. Потоки не создаются дел. потоки, считается, что потоки машинной поток (или обработка).

Для под-
ключения
пользован-
ия инициа-
лизации.
Система
ядра обес-
печивает.
В бинар-
ном формате
аппарату-
ры разде-
лена на
части (в
иерар-
хии). Но
затем все
автомати-
чески син-
хронизи-
руется
и выдается
код (нап-
риимер
воспроиз-
водства).

В ини-
циа-
ции дейст-
вует в
моде

Над надо, чтобы в системе присутствовало множество ядр, но нужно перекомпилировать. Но в Windows и Unix/Linux можно компилировать ядро без перекомпилирования. В OC Windows это многоуровневое драйвера.

Система прерываний — это токи входа (в ядро). Плагин ядра обновляется инициализаторами, т.к. вызывающие различные функции.

В бинарном будет вопрос о исключительном ядре, надо демонстрировать знание о прерываниях, в част. об. аттракторах.

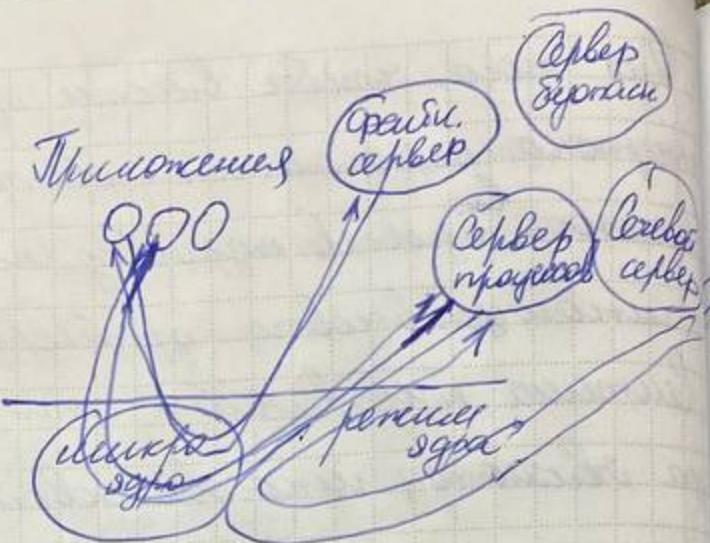
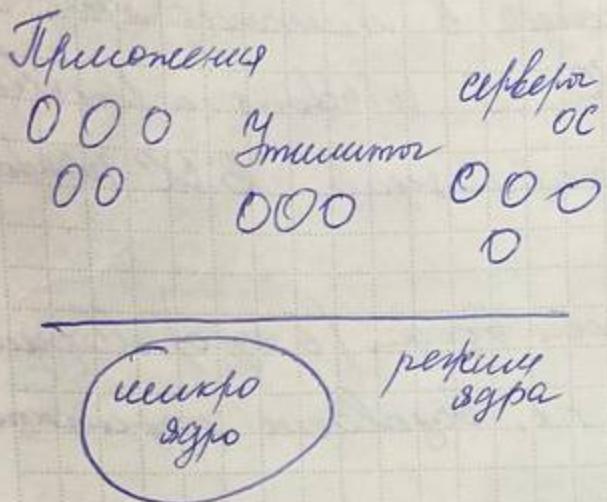
Микроподсистемы архитектура.

В ядре оставалось только самое многоуровневое действие (в част., диспетчируя — ведение проц. времени). Квант — отрезок времени, ком. определяется соответствием тактовых шиников. Многоуровневое действие обновляется ведение графической страницы. И в то же время давно не исп. страницу, например. Вирт. АП — более высокого уровня действие.

При работе с устройствами многоуровневые действия обновляются непосредственное обращение к внеш. устройству (напр., к внеш. диску). А, напр., получение под — более высокого уровня действия.

В микроядре оставляем самое самое многоуровневое действие. Остальное делаем в отдельное ядро, взаимодействие — с помощью сообщений.

ИБМ 2



Взаимодействие в такой системе должно быть надежно. Если сервер получит запрос на обслюживание, он должен обслюживать этот запрос. Здесь работает правило „Три сост. блокировки проф. при передаче сообщения при посыпке, при приеме, при ответе“. Возникает небольшое блокирование. Обращение к другому серверу происходит через ядро \Rightarrow это длительный процесс \Rightarrow эффективность микроядра?

Windows 2000 не авт. система на осн. микроядра (в класс. понимании). ОС Mac - пример ОС на основе микроядра. Микроядро включает сервисы напр. почтовых, подачи сообщений, воспроизв. папок и..., все облачное - на уровне почт. Система, построенное в чистом виде на микроядре, непривычно? с точки зрения эффективности? (Сенекон, Русаков?). В исполнит. зоне I переключения „ почт. \leftrightarrow ядро“, в микроядре - III зонами II переключения. Это можно оценить, но оценить блокировки невозможно.

Чтобы к микроядерной арх. увлечившись, это делают

сервисы
тропи
тишо
кою
Определение
Распол. врем
передающей ут
ческ. Кислор
демергинаци
е времена
передается в
в этом слу
допускаются
на действие
Регистрация
ОС в межд
бесп. проф
ответа сисе
на обслуги
характери
Это опред
„блокиро“
сов посту
обслуги с
тоб. Три в
в реальном

возможностью изменять кода серверов, не затрагивая ядра. При этом это все правило сиюминутного спу. назначения (всегда системы реального времени). Нет. ОС обычно настроен с штрафами.

Определенные ресурсы времени в ОС по стандарту POSIX.

Ресурс времени в ОС - это способность ОС обеспечивать предыдущий уровень сервиса в опр. промежутках времени. Ключевыми ограничениями ядра ОС РВ являются, откуда же демонстрированной, основанной на строгом соблюдении времени работы. Для выполнения одного сервиса ОС не имеет пределов времени исполнения задания. продолжает. В этом случае ОС обн. назначения не забывает, что же делает серверу и не допускает ему задержки \Rightarrow замедление работы на действующем пользовании.

Ресурс реального времени - организующий работу ОС в течение, обеспечивающий обеспечение непрерывности, не зависящий от выпол. инструкций. Время ответа системы \leq период получения запроса на обслуживание. Ресурс реального времени характеризуется временем ответа системы. Это определяется как правило, а не "бюджетом", "неделимостью" и т.д. Запросы время проходят поступают в контекст прерываний. Порядок обработки определяется системой адм. приоритетов. При обращении к ОС предполагается что ряд в реальном времени предоставляет только гарантии

к надежности и живучести

QNX - Unix-подобная ОС
с высокой

Основное концепции Mach.

Ядро Mach, подобно другим микроядрам, облегчает управление процессами и памятью, коммуникацию между ними, ^{вног. вида} предоставляет асинхронные ком. б/п. поиска. Основные алгоритмы:

- процесс;
- потоки (thread);
- объектов памяти;
- ком.
- сообщения.

В Mach процесс понимается в классич. смысле. Процесс является владеющим ресурсов, приоритета, АП (вирт.). В Mach процесс является единицей декомпозиции системы. Ресурсы запрашиваются потоками, но владеет процесс.

Поток (Thread) - классич. Является единицей выполнения (диспетчеризации), ему принадлежат все ресурсы (аттаченный контекст).

Процесс, не создающий потоков, недоступен проф. б/пам (без обуздости).

Объекты памяти (memory objects) - структура данных, кот. могут быть обл. в АП проф, могут занимать один/несколько страниц, являются основой систем управления памятью. Если процесс обр. к объекту памяти, не использует, в прит., можно, вероят.

страницы
других)
ком. обра-
кинг и. и.
страницу.
РАЕ.

IPC в ма-
- порт,
Порт соеди-
- управляемые
ресурсы. Уни-
верс. порт,
опр. свободу
порта, при-
нципы на-
ружки, осо-
бенности

Для ком-
муникации
создано бло-
к сообщений
без с. пос-
тавок. С-
ма (сарат-
права по-
ОС РВ
крыле зод-
го зондра

погодная ОС с сенсорами
и датчиками, обес печивающими
автоматизированное
управление, мониторинг
и диагностику
и т.д.

ИПС в Mach основана на передаче сообщений. Основа — порт, предоставляемый самим приложением, поставляемым фирмой. Порт создается в API языка и инициализируется оператором — упорядоченным списком сообщений. Оператор не имеет прямого доступа к нему, но для управления ими концепция порта устарела, поэтому, например, когда сообщение в очереди, если оно не было обработано, то оно не может быть удалено из очереди. Вместо этого оно будет оставлено в очереди. Для этого используется механизм называемый запросом/ответом: процесс, запрашивающий сообщение, получает его от оператора.

Задача оператора — это управление библиотеками, которые можно было бы считать компонентами протокола, сообщениями. В Mach протоколы представляются в виде языка запросов/ответов сообщений, который поддерживает различные форматы. Это реализуется с помощью языка запросов (query language). Всё вспомогательное относится к языку запросов.

ОСРВ предназначена для решения общих определенных проблем языка. Программа в такой системе выражается в виде запросов ресурсов.