

ОПЕРАЦИОННЫЕ СИСТЕМЫ [Лекция] 01.09.23.

[Нагаев Тарас Рязанов]

Микропроцессор - программируемое устройство.

? Не ОС управляет (част. времени) яр. част. - программист

Такой принцип явн. основой постр. комп.

I История р. вычислительных систем.

Детали

Задачи на которые решались вручную -
исчислительные задачи эпохи. Вручную вспомог. инструм.
Устройства для вычислений - вспомог. устройства,
которые решают задачу, как и эти задачи.

Принцип программированного управления.

II Абст. машины Бэббиджа

- 196. - потр. тв. в большом V вычислении,
таки же опр. т
(использование пасынка)

Ч. Бэббидж - изобретатель (1833г.)

- Особенности:
- программируемое устройство (раб. машинок и программируем.)
 - 1000
 - первокарты (таблич с вычислительн. исч.)
 - постр. из звуковых комплексов

[Применило вычислений практик. решения вычислений]
но прими. РВВ из-за неких вычислений

III. Первое поколение ЭВМ.

1940е - 1955 гг

- сер. 40 - 1955г.

- 1944г. - Джонс, 18 лонд. ун. и др. эти разработки
МАРК-1

• 1000

- 1946г. - Илер, Рэнд, Джонсон
ЭНИАК

• ограничения, размер

Несколько компьютеров, несколько проектов
- были управляемые устройства, использую-
щиеся компьютерами памяти.

- начались устройства хранения - перво-
лическ.; первокарты ~ 1950г. (один из первых)
- второе: АБСТО (принцип посл. памяти)

? Применение программированием требовало специ. памяти
? Все вычисление - в одн. адресат; машинном коде

1945 - агрес. агр. agree срочно начать.

- 1945г. - фон Нейман: доклад об осн. компьютерах и принципах ВМ.

(→ скрипты)

- 1) УВМ в. соотв. устр. ви; арифметики, управление, память, связи с外界

no
агресу

- 2) Недост. членов языка. членр. языков и кодов

принцип
хранения
программ

- 3) Если команды агрес. в виде чисел. тогда, то память должна иметь как реал. числа, так и для команд

- 4) Принцип памяти для команд в. сущ-х устр. то, что команда должна быть - упр-яющей

- 5) Контр. арифметический орган.

- 6) А. сущ. устр. языка-программ ⇒ связь с внешней средой.

не зал. агр. кон.,
органическое
вещество
(единственное
сущ. агр. кон.)

- принципы р. работают с дек.
- есть деление
- ком. оп-ции памяти-то.

2. Второе поколение ЗВМ

1955 -

- появление гибких, гибких:
дан. устр. в. на имп. сердечниках

или создание сущ. памяти-документации

- первое поколение IBM
- появление ассемблера АИ (Fortran, ...)
- менее дорогое машинное программирование на машинную лексику → воспроизведение → языка → гибкая лексика → более широкое применение → вычисл.
- появление операторов

IBM 1401 / 1410 International Business Machine

Схема

перехода:

IBM

1401

→

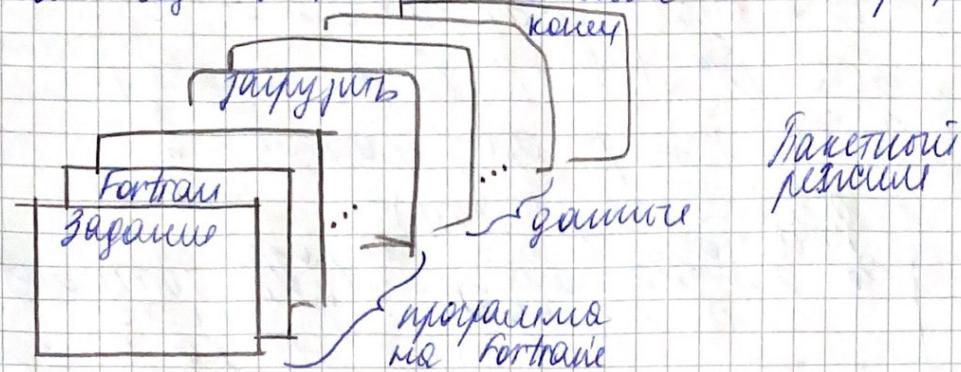
7094

→

1401

• Дорогостоящее устр. постоянно пространство
⇒ создание временных просторов т.е. временные памяти.
меню прилож. ⇒ программы машинной хранитель
в памяти

→ Задача: загрузка сразу двух программ в память
многопрограммных систем - система способна сор. в память большое кол-во программ.



- first multitasking OS was named Unix. wrote by D.J. Kernighan; programs in Unix were created by the operating system.
- second generation of programs based on the general concept of programs; involving shared control of resources.

1960 г. - интегр. микросхемы

3. Первое поколение IBM

IBM 360

- иметь простой процессор, т.к. можно получать данные
- наименование арх-ра IBM - распределенный - имея ф-ии:

работы внешних устройств управляет ^{нанесен}
арх-рд - макроинструкции

как проигр. о конце этого-либоого?
Аппаратное прерывание \Rightarrow софт. пр-е.

→ наименование системы прерываний

Прерывание

сист. вызовов исполнение $\xrightarrow{\text{установка}}$ аппаратное
действие на О (interrupt)

В сист. вызовов:

аппаратные (исп. машины)

контроллер

· повышение производительности при выполнении

Терминал - сист-ть клавиатуры и монитора.

Лекция 2.

Являются базой для программ
искусственное; программируем, слож.
с обратной связью нет. гамма,
а компьютер.

⇒ неодн. режим работы

⇒ в IBM 360 блоки смены
чип. процессоров - каналы,
упр. работы блок. устройств.

Являют проблема - очень большое время отработки,
отсутствие языка к проекции комп. яз.

для получения полноц. системы к программе надо
иметь оперативный блок, т.е. смен. блоки
устройств.

Первичн. схема устройства [автомат] одинаково для всех разработчиков с одинаковой
и инструктуры.

- Мониторы:

[на зам.] → ... [с плавающей]
[трубках] [разверткой]

- постепенно появляется
изменение дл. цикла
- синхр. по строкам
(→ понятие "микросекунда")

Черн. подключить к осн. блоку к новому бло. задачи (установ-
ление подключения ⇒ переходное устройство)

ОС, которые ограничивали работу с уменьшением подключаемых
терминалов, сопровождение которых требует дополнительного времени
(подключение определенного количества терминалов)

• Несколько ограничение времени - параллельное (73г)

Кратк. - интервал раб. времени, неодн. для выполнения
упр. задачи?

• Первые генераторы времени выполнения времени -
compatible Time sharing system (CTSS)

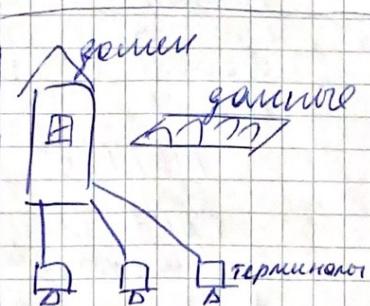
MIT

Была разработана для IBM 7094

BALLABSTO ^{создан} начал разработку ОС MULTICS
(Multicomputer Information System)

Многопользовательская система на 100 раб.

Система MULTICS стала использоваться в 1969 г.



"базисное
устройство"

Начиная из того, что эта система должна быть
предназначена не очень большими персональными, однородными
системами с ней очень бережно \Rightarrow хорошо сконструирована.

MULTICS относится к мегаархитектуре (4^е поколение)

кто? РДРТ - иен. как экспериментальная.
Гаварнингове "детище" - под РДРТ.

- Описание UNIX впервые было дано в окт. 1973г.
(Ken Thompson и Rob Pike):
 - иерарх. структура системы
 - сервисы, опр. устройств и менеджер обмена различными
 - асинхронные процессы
 - системной язик. языке, вышуканном на C языке.

↑ Систематично для создания UNIX были написаны
в П Си

Файловая система - набор правил, опр. способ
организации, хранения и имен. данных на носите-
лях информации (напр. на дисках...
(по аналогии, напомни)

записывающие пластины,
прим. для магнитного
хранения информации.

! Самое главное достижение - отказ от иерархии фр. систем.
Они определили файлы как носители данных,
а это и есть содержимое опр.-ой про-
грамммы, т.е. ОС содержит данные никак
не имеет

Файл - производившая область данных (адр. б.
дескриптором)

- в UNIX директории - по каталогам (directory)
- Каталог также явн. директории, содержащие файлы
обрабатывается самой ОС
- каталоги могут создавать подкаталоги.
 \Rightarrow иерархическая организационная система.

(Хотя ОС UNIX были открыты. Т.к. они содержали C,
они были читаемы.)

[можно перекомпилировать, изменять]

Существует 2 версии UNIX:

System 5

UNIX BSD

- первая управляет
корнем от общности
пользователей второго

DOS X - стандарт, в ней используется бинарный формат около 1000 символов.

Все это побуждало разработать систему.

4. 4^е поколение АВМ - появление персональных ЭВМ.

Intel 386

Intel 8086

Intel 80486



5^е поколение - сверхбыстрые микр.схемы,
разделение на ядра
• процессоров яви. программного-управляемыми.

Быстрые микр.схемы

УДОЛ:

- Однопроцессорные под. обработка
- Многопр. под. обработка
- Системное разделение времени (хар. квантование времени)

в памяти одна под.,
затем

одесн. Собр. систем - мультизадачное и монотаск.

загр. DOS-ориентированное но не пакетной обр.

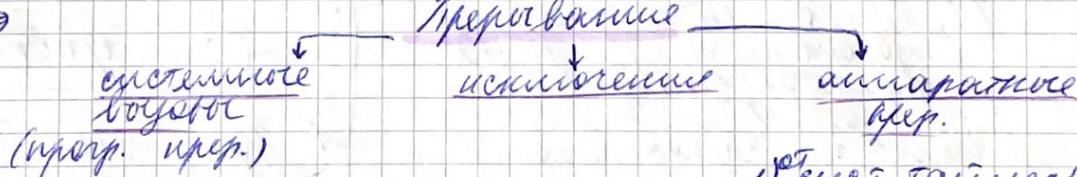
прин. драйверов с переключающимися се-

ментами

6 Р матричное 3^е поколение - распараллеливание

матричный параллель арх-ра - суперпроцессоры
УСТР { матричный арх-ра - программируемое устр. под. контролерами.

⇒ need. интегрировать проф-р.

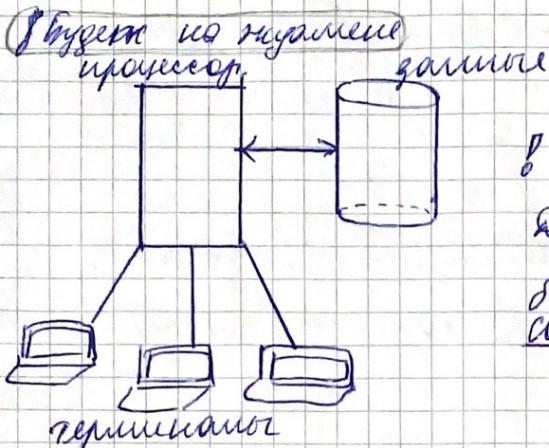


1) мат. генератор
reg-ные приборы.

- 2) от внешних
устройств
(раб. Порта-телефон)
- 3) от действий user-
типа.

Управление процессорами

Процесс - программа в стадии выполнения.
самое простое (просто текст - данные)



После запуска программы она становится для ОС процессом.

2) В любой системе

для того, чтобы управлять,
одной и той же программе
составлены процессы. - Рутин
+ Режим

Диаграмма состояний процесса.

1) Когда программа запускается на вин-е - подготовка процесса.

- Первое действие - изделийническое процесса. +
(свобод. действие) $V_{in} - V_{out}$

! В любой системе
существует ср-ва ови-
дание процесса
enix → (структурное)

Процес - наименование
две описания. (объект?)
(создание)

+ инициализации дескриптора
процесса
Рес-т нового пакета (тек. сост., итд),
указание на гр. структурой (+своб.)
с управлением пакетами,
т.к. процесс может управлять
разно. тем, что наход. в пакете
- после иниц. процесса ОС
все эти пак. имеет, т.е.
программа запускается в
пакетах)

пакеты не об. разраб.

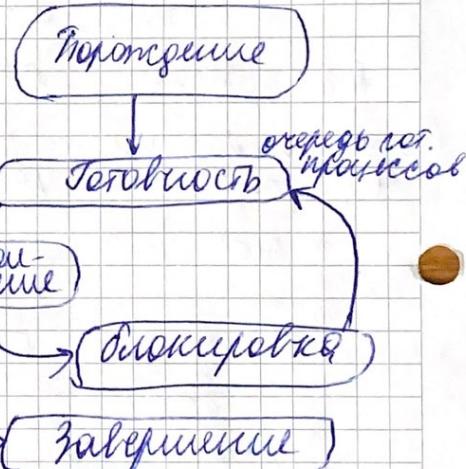
2) Переход в сост. готовности

3) Возможение (текущ. время)
• в.м. ресурсов. системах - очередь на вин-е

- (пак. в стадии готовности)
- и.г. запрошен. дол. ресурсов

4) Блокировка - отыскание ресурсов, недост. ресурсов

5) Завершение - осв. ресурсов, ^{использование} и возвр. в них свобод. ресурсов



+ 2nd действие для
пакетов если вин-е
занятое где
запрошено. макс. час.

Процессы каким то образом воспроизводятся в отдельных ячейках. Это наз. манипуляции.

T.e. манипуляция - насторожка процессов в отдельн.

Выполнение процессу приз. временн.-демонстрационн.

В ОС я.б. манипулятор - проходил, чтобы защищать память памяти процессов приз. временн.

(запад - манипуляция)

Типы манипуляции

беспрерывное

с повторением

с переключением

приостановленное ^{авт. форм.}
_{авт. форм.}

без повторения

без переключения

• если приостановленное
имеется, то ОС и.б.
реализовывает пока

?

?

с. е. восстановлено,
затем и без него.

?

?

(процесс с более
высокими приори-
тетами)

?

?

и т.к. это более вы-
сокий)

?

?

• если нет -
процесс вст- ся в
очередь до конца)

?

?

в очередь встает

?

?

? В новой системе исп. не только процессов, а схем-
атич. спосок (последовательность баков), назв. орга-
низовать функциональное управление.

?

?

Типы организ. наз. отладки - без манипуляции

[Он - это - в новых системах отладки нет.]

Он ОС (operating system) - комплекс управляемых комп.
абсолютно управляемых ресурсов под. систем и
процессов, исп. эти ресурсы в ООС. системах.

• исп. запада ОС - выполнение процессов независимо от конф. дат.

и т.к. процесс. это управляемость, управление

① процессорное время;

② общая (один-ко-одному) память;

③ устройство ввода-вывода;

④ каналы генераторов, общий ядро (ядерный)

реализованный под ОС и исп. таблицы.)

Реализованные исп. под системой процессоров (проце-
ссы исп. имеют возможность общаться между собой)

• исп. имеют процессоры и не имеет них организованы.

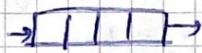
• Реальные процессы имеют исп. в разных языках
одних процессоров (один язык искал из ядра данных,
переменных)

• реальные языки языков - в исп. сист. таблицах,
послед. исп. образуют конфигурацию

- 8
- o [! Квантование времени запрещено обратного синхронизацию автономных подсистем приводов:
• обратная зависимость от конечного привода \rightarrow другой привод
• не имеет приводов]

I Типы управления в мультипл. пакетных системах исп.

1) FIFO (FCFS)



- без приоритетов
- без вспомогательных
- Квантования блок. \Rightarrow в конец очереди нет-у

2) SJF (Shortest Job First)

- наименее загружен первыми.

Существует т.к. имеются две характеристики [task/time].

\Rightarrow $t_w - t_s = \frac{\text{Бесконечное откладывание}}{\text{(последний привод, который имеет самое короткое, далее должны все приводы откладывать)}}^{\text{так как}}$

3) SRT

- Shortest Remaining Time (миним. остат. времени)

- том. прерыв. не быть прерван, если остаточное время с меньшим остаточным временем том-а.
- том. мин. расходов (t_w \leftarrow t_s \leftarrow для тех, времена том-а, сравнивают это)

\Rightarrow более короткое бесконечное откладывание

4) HRN

- Highest Response Rate Next

- переключает приоритетов для иски. блк. откладыванием

- $P = (t_w + t_s)/t_s$ - приоритет
 - t_w - время откладывания прив. в очередь нет-у
 - t_s - конеч. время обработки вами.

\Rightarrow тем выше приоритет, тем выше в очереди; динамический приоритет

! Динамический может быть
приоритетом между концов. процессов

В пакетах
используют
постоянно один
приоритет

5. Альтернативное планирование в синхронных реагирующих системах

(time slice)
starting time system

1) RR

- Round Robin

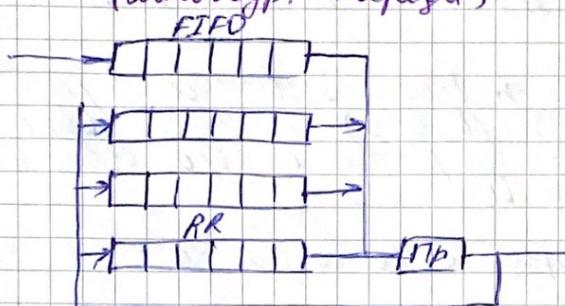
FIFO



"-": стартовый квант времени

- Т.к. не ордера, то с переключениями
- Special компьютер
- один процесс не job., но about job., or nothing.

2) Альтернативное планирование (многоур. очередь)



Семи реал-
ьзована много-
уровн. с фикс.
очет. пакет. от
дл. в. выполнения

- & 1st (cannot preempt) or max. прерывок
- job. окончание обработка (многоур. реурс.)
- выдача соглашения

Большой квант прер. бп. Пример: вы сидите на концерт макс. время прер. можно 20-30 мин?

- & более много-ур. реурс - если не job. ja квант
- каждая много-ур. реурс. от job - с конечной временем
- ⇒ & вершина - макс. процесс обработка-выхода
- ⇒ & вершина - формирование

8 Поточное зап-ти RR - гарантированный процессор:

1a) MNRR

- Min-Max Round Robin

1b) AMRR

- Average-Max Round Robin

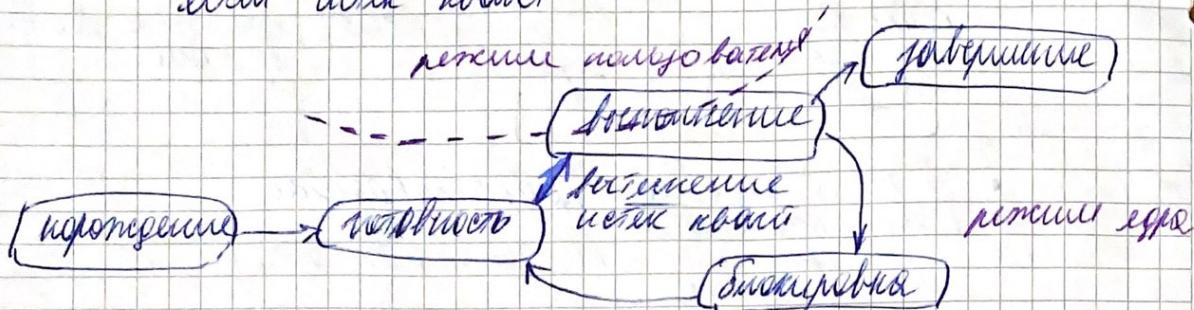
1b) SRBRR

- Shortest Remaining Burst Round Robin

8 Основное, что в Windows - RR: & user. процессора. орге-
зан с ограничением на то же количество, максим. до auto-
matically RR



В секундах занятых в подъеме и спуске. Чт. б. восстает
из стаканов формы сум в сеч. пайд. Генерал -
сум стакан небольшой



! В итоге DC находит обогащенный ион. Продесс, т.к. для него нет ресурсов. Процесс - физиологический цикл.

UNIX: Порядок работы процесса формирования пакетов включает следующие этапы (заголовок) - форм-ет заглв.код, а затем - формирует заголовок и тело и форм-ет пакет. код DC.

Существует метод пересчета из процентов долларов в проценты япон и обратно.

Документы состояния процесса UNIX



fork() - exec. boyut, copyacit naboru proqress.

- настороженный вопрос. Генеральный секретарь, несет. Общая панама одна из них.
(т.е. у нас есть вопрос панамы, вопрос о
США)

6) Не ограничено:
при забороненні -
1 бд подається
захохол з/з сост.-2
заночі

- горючее (м.т. нефтью) паника - опасаем
 - речной суда → речные газары
 - гор. ресурсы - в панике приост. (\rightarrow борьба за нефть)
 - горючев. - никакие иные процессы, такие же резерв.

Лекция 4. Потоки.

22.09.23.

Потоки неявно в UNIX, но ограничение
к коп-ю памятных расходов с дескт-
ами переопределено константой

Потоки контекст

↑

ампаратив
контекст
(коп. регистров
процессора)

шагоримущий
ресурс процесса

Сохранение текущей при непре-
мение процесса на фоновую
группу процессов.

Ампаратив-only: системное боязнь
Сохранение фонового контекста - becomes
приоритетной задачей.

Пример: предоставление от текущего рабочего
(from - в работе процесса)
- коп-е ампаратив контекста.

Потоки контекст подавлено алгоритмом
⇒ идет ядро ⇒ корректный контекст ⇒
⇒ идет ядро.

Использование стека линий бывший ядр
ядро: при этом, если процесс вымо-
жет его в занес в стек, это озна-
чает к временному устранил.

Пример: текстовое редакторе подавляет
сохранять данные в стеке -
форма процесса, но при этом
данные неудобы - дислокированы

т.е. параллельное программирование - один из новых способов выполнения, другие осущест-
вляют с помощью языков.

UNIX поддерживает параллельные
рекурсии и процессы.

Что:

две фазы
рекурсии

Многопоточная система для одновременного
использования отдельной. Каждая часть
кода выполняется отдельно (с некоторыми?)

Пример: QT создает потоки для собств. нитей

thread

Онл. поток - часть кода программы, которая
имеет возможность параллельно
с другими частями кода.
(непрерывные задачи)

- Это же самое, поток содержит в себе все.
- Многих потоков параллельных называют -
потоками. - самое большинство.

Процессы и потоки.

Процесс имеет в своем контексте все
характеристики языка:

1) владение ресурсами (resource ownership)

- процесс - владелец ресурсов в системе
- + адресное пространство (баз., зоны.)
- при born-in процесса выделяются
специальные память, настройки;
исходный процесс born-ся потоками,
они получают потоки (или запуск-т)
запрос
- + локальные выделенные ресурсы
- + использует борд-порта

2) основной функциональный процесс

+ масштабование (меня в размер)

+ дополнение (всегда приращение в

р. функции приращения)

За основной может быть неправильная часть или часть когда происходит

приращение

В результате такого определения можно выделить стабильные единицы распределения — это определенные процессы с врем.

? (которые участвуют в цепочке какими, аналитической форме.)

• При переходе из одного состояния в другое переключается аналогичной единицы.
Также

Бытовые модели метод.

1) Однократная модель процесса

• единиц не может удовлетворять и выполняется однократный процесс единица один, а не единиц — группы

⇒ если реализовать такое в системе, если процесс не создает языка, который то все системы они имеют один однократный языок.

(две группы)

одна единица	однократная модель
стек языка	процесс
стек памяти	
адресное пространство	
	(изолированы)

- Адресноепр-во процесса: заслуженное в котором макс-вс ког процесса есть
 - mapping системоз (однозначное)

Windows

отображение адресов	4 ГБ
записи адр. пр-во, т.е. пр-во новой памяти	2 ГБ

(32 - UNIX)

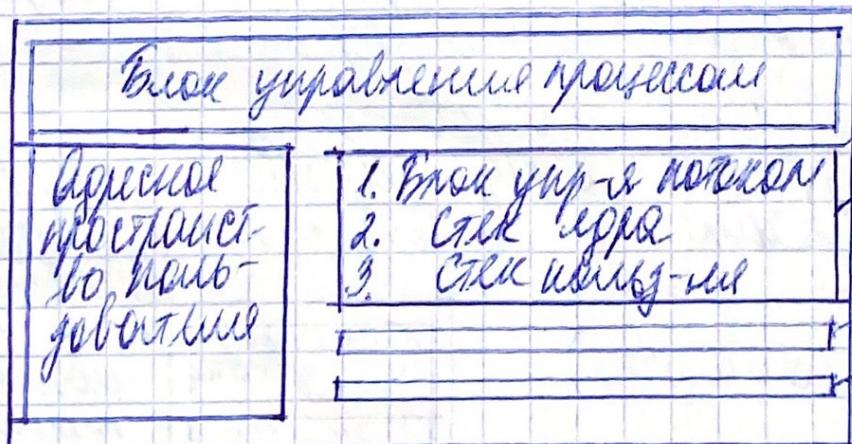
2 ГБ

! Адресное пр-во -
документ

- Не может быть создано сколько либо ресурсов
управления памятью
(так как память, нормально это память
может создавать неск. скопий)

! Сколько скопий может быть создано

2) Универсальное управление памятью.



! Все ноды делят одинаковый ресурс памяти
и адресное пр-во, т.е. ноды соделли-
тельно адресного пр-ва не имеют.
+ одинаков, обладают управ

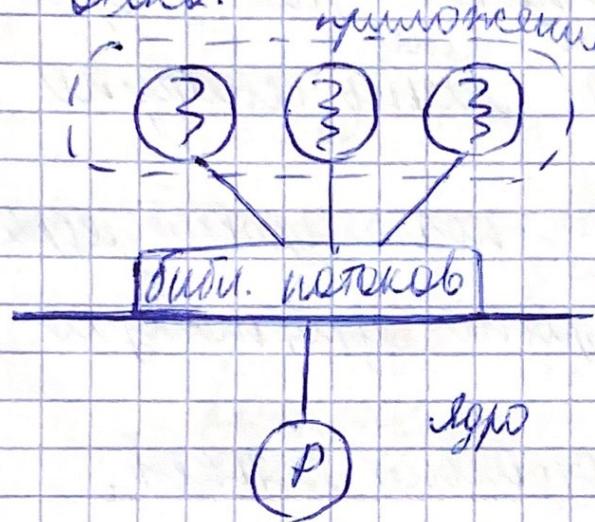
Синхронизация на основе потоков

• потоки
управляют исполнением

• потоки
управляют языком

Потоки управляют исполнением.

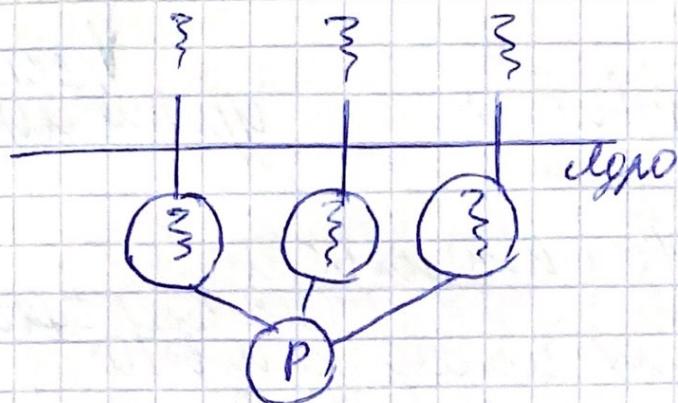
- язык и язык исполнения не требуют
- есть язык-язык для управления потоками.



- Управляют потоками
ошибки
(исключения, ошибка языка, ошибка
координации, ошибка
исполнителя)

• Если поток занимает некоторое время, то его поток (используя генератор задач и контекстное меню) может приводить к блокировке языка. → все потоки могут забирать время этого потока (т.к. языка нужно о них не заботиться)

Потоки управл. ядра.



- Использует о методах, имеет управл. ядра
- Типичные для Unix-подобных систем
- Потоки процесса имеют собственную стеки памяти.

UNIX, Win, Mac OS - потоки управл. ядра

Solaris - потоки управл. ядра, неоптим.

Мониторинг состояния потоков:

Потоки неизвестны, когда процесс выполняет несколько независимых задач, т.к. при выполнении одних задач, другие будут восстанавливаться.

Пример. Блокировочный оператор.

Достижимые потоки (reachable):

+ Отложчивость

- один поток может блокировать быстрое выполнение приложения, когда другой запущен. или если есть блокировка.

+ Разделение ресурсов

- один поток разделяет доступ к ресурсам
 - ⇒ необходимы потоки для: 1-го в блоке

+ ограниченность

- переключается только аппаратурой
косякет, т.к. единственный агр. пр-во

+ нестабильность

- однотипное - только из одной пр-ф
- меняется - реально II-но в много-
процессорных системах.

Проблемы потоков (динамичности):

- формирование, определение задач

- требует уточнения альгоритмов из
предмета таких задач, кот. могут
выводить II-но

- стандартизация

- поиск задач для II-ного выполнения -
или которые облегчают правильное
решение (?), т.е. необходимо, чтобы
каждый алгоритм выполнения альгоритм
работал более эффективно.

- разделяние данных

- проблема: параллельные потоки
могут образоваться к разд. данным -
- либо синхрон. способом или делить
память, т.к. это приведет к другим
данным или неправильным
результатам (?)

- зависимость данных

- если одна задача зависит от
и-го выполнения другой, тогда
исполнение этих задач о.о. синхро-
низовано (одна задача ждет,
пока другая не выполнится)

- воспроизведение в стеках

- создает много повторяющихся атакующих стеков с одинаковыми, все беc адресов памятью управляемыми

Типы параллелизма.

1) no памяти

- разделяние памяти для потоков (создается, разделяется, используется потоками)
- все потоки получают одинаковую память и не имеют общего управления памятью

Пример. Программа игры.

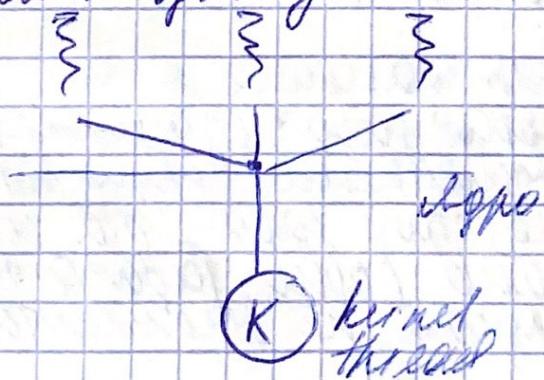
2) распределение памяти.

- (но синхронизацией)
- разделяние все памяти **Пример**
Но вспоминаемое **Генераторы**
памяти

Все память все потоки делят настроек.

③ разделение пользовательской (user - kernel)

памяти к ядру



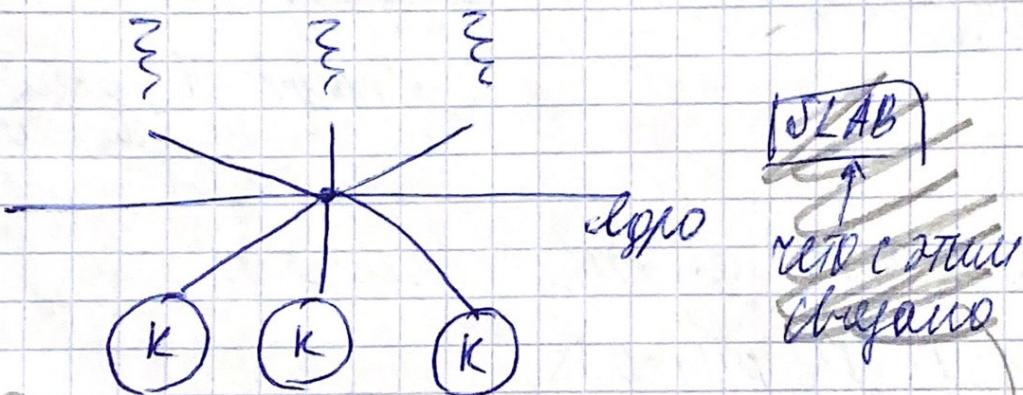
- разделение большинства памяти между ядрами

② open K threads



В большинстве систем есть совместное использование потоков отображения.

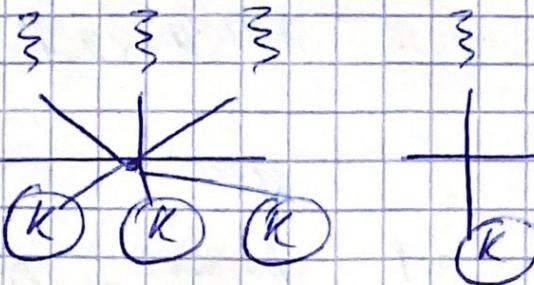
③ misuse K threads.



④ LWP - Light Weight Process

- Создание потоков - внутренний процесс.
В системе ядра нет потоков ядра, пот. могут быть назначены пользовательским потокам пользователем.
- Все как в "open K threads", но есть ограничение: назначаются один раз.

④ параллельные алгоритмы



Environment threads.

- предостав. API для создания и управ-л. потоков.

↳ б. вп-ре
написание

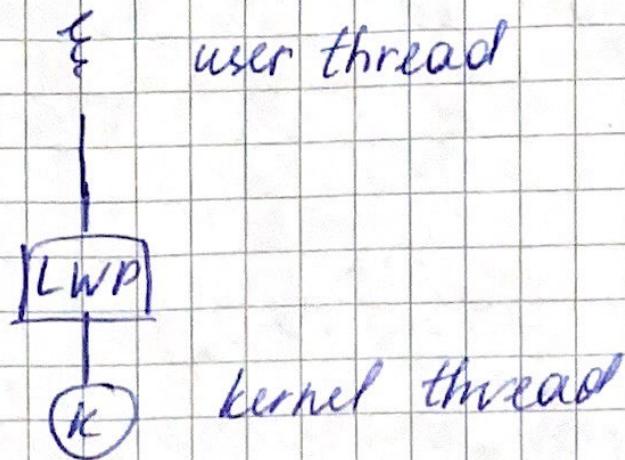
↳ б. вп-ре
API

- предостав. б. распар-алл. языка программир-
еющ. технологий

Основные тех-ники параллелизма:

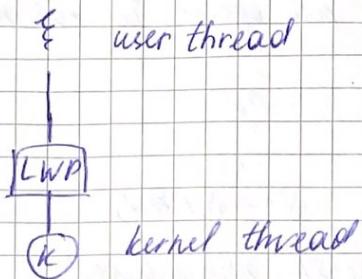
1. POSIX pthreads
2. Win3d threads
3. Java threads (Java Virtual Machines)

④ LWP - Light weight Process
(LWP - Library for WWW in Perl)



- pthread library
memory yet q-io alpha clone()

6) LWP - Light weight Process
(LWP - Library for WWW in Perl)



- pthread library
 - использует q-to-я для clone()

Преимущество: б) легче организовать параллельную работу - LWP; когда вызывается pthread_create, нет-ца эта функция инициализирует структуру

б) мало ресурсов, которые ей не может предоставить операционная система; т.е. не нужно ей выделять память для создания线程.

Различие между параллельных процессов.

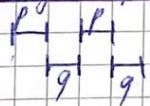
Уровни параллелизма

- 1) Процессорное ядро. Есть процессорное ядро, а другой процесс параллельно использует одно и то же ядро.



Процессорное ядро параллельно не может использовать ядро.

- 2) Многоядерное ядро. (однопроцессорное ядро)



• есть один процессорное ядро - многоядерное ядро.
• есть один процессорное ядро - несколько процессорное ядро.
• процессорное ядро - max-ко-л. ядер, могут обмениваться данными

- 3) Реальный параллелизм.



• Каждый процессор работает одновременно в разных процессорах
• Процессоры не взаимодействуют.

Пример (представим)

- где функция банка, в конце еще функция
передаваемых параметров не один и нет же

(P1)

(P2)

$$S := S + \Delta 1;$$

$$S := S + \Delta 2;$$

$\Delta 1, \Delta 2$. - Возвращение функции!

функции, для которых значение
функции передается в аргументах.

MOV, SUM, MUL

$S \rightarrow AX$ $AX = S$

Функция, Pd настолько сумма которых равна.
P1 настолько меняется (не меняется)
 $\rightarrow Pd$ не меняется S

$$\boxed{S + \Delta 1}$$

$$\boxed{S + \Delta 2}$$

требуется вычесть.

- для языка!

P1:

...

MOV EAX, MYVAR

INC EAX

MOV MYVAR, EAX

P2:

...

MOV EAX, MYVAR

INC EAX

MOV MYVAR, EAX

(1) Рассмотрим написание. (для языка)

Язык -
искусственный
и машинный
языком
программ

MYVAR - это уже значение передаваемое

P1 на языке 1	EAX	MYVAR	EAX	P2 на языке 2.
MOV EAX, MYVAR	0	0	0	MOV EAX, MYVAR
	0	0	0	INC EAX
	0	0	1	MOV MYVAR, EAX
INC EAX	0	1	1	...
MOV MYVAR, EAX	1	1	1	

назначение
значения (результат)

(2) Одновременные изменения.

P1	myvar	P2
0	0	0
mov eax, myvar	также изменяется P1	
0	0	0
	P1 не меняет ябай,	
	также изменяется P2.	
!согласен ани.	0	0
конкурент (не меняется)	0	1
	inc eax	inc eax
	1	1
	mov myvar, eax	mov myvar, eax
P2	ноги ун ябай,	
	также изменяется P1	
0	1	1
всегда меняется	согласен ани.	
0	1	1
inc eax	1	1
mov myvar, eax	1	...
		...
	изменение	

! Не одновременное, а пасхальное
→ в P1 и P2 одинак. проблема.

"race condition"

Переменные, передаваемые в myvar - разделяемые
переменные.
Критический участок - участок кода, в котором
одинак. доступ.

Критический участок - участок кода, в котором
одинак. доступ.

Две решения проблем: к разделению или к синхронизации.

Он обеспечивается ср-ками блокировкой.
(мергеры) - mutual exclusion:

- 1) независимость
- 2) анти-конфликт
- 3) с внеш. синхрониз.
- 4) с внеш. мониторов

1) Программное ср-ко блокировкой

⇒ обеспечивается в коде программы

Пример (программа)

• prog-example

fp1, fp2: logical

fp1: white(1)

white(fp2);

fp2=1;

fp1; // Критический участок

fp1=0;

fp1;

end fp1;

```

p2: while (1)
    while (fp1);
        fp2 = 1;
        CR2;
        fp2 = 0;
        PR2;
    endp2;

```

```

begin fp1 = 0; fp2 = 0;
par begin
    p1; p2;
par end;
end.

```

Что же имеет место при этом выполнении?

⇒ Параллельные потоки!
Но какими могут быть эти потоки?

• prog-example 2

```

fp1, fp2: logical
p1: while (1);
    fp1 = 1;
    while (fp2);
        CR1;
        fp1 = 0;
        PR1;
    end p1;

```

```

p2: while (1);
    fp2 = 1;
    while (fp1);
        CR2;
        fp2 = 0;
        PR2;
    end p2;
begin
    fp1 = 0; fp2 = 0;
    par begin
        p1; p2;
    par end;
end.

```

- Параллельное выполнение
- Задача, которую решает
- p1 и p2 вынуждены на
- исполнение параллельно — deadlock
- ⇒ возможное окончание на
- занятие (запись на квб.),
т.е. две записки发生在
одинаковом временем

Конфликтное выполнение было неправомерно фиксируется.

- две 2^х параллельных процессов.
- возникнет конфликтное выполнение "the-request" (request)

```

fp1, fp2: logical;
queue: int;

```

p1: while (1);

 fp1 = 1;

 while (fp2)

 if (queue == 2) then { fp1 = 0; while (queue == 2); fp1 = 1; };

end p1; CR1; fp1 = 0; queue = 2; PR1;

one queue

queue == 2

```

P2: while (1);
    4p2=1;
    while (fp1)
        2 if (queue == 1) then
            2 4p2=0;
            while (queue == 1),
                4p2=1;
        3
    3
    CR2;
    fp2=0;
    queue=1;
    PR2;
end P2;

begin
    fp1=0; fp2=0;
    queue=1;
    parBegin
        P1; P2;
    parEnd;
end.

```

- Выделяется, что происходит в процессе синхронизации
 блоки параллельные
 процесс уст. блок ожидан \rightarrow нач. производство мыш. \rightarrow
 \rightarrow установка мыш. \rightarrow засыпка ожидания мыши, блок блок \rightarrow
 \rightarrow начало блок ожидания \rightarrow уст. блок ожидания \rightarrow крит. раздел \rightarrow
 \rightarrow блок ожидания \rightarrow следующий шаг 2. \rightarrow и т.д.

5 этап. Время отпуска на проверку переменных

\Rightarrow Немоногр. блокировка ед. извлечения (т.к. процесс сама укладывает, что второй процесс - следующий)

Смысл этого программного решения, то с помощью
одного, но многочтн. прерываний (аналогичн. аэропорта?) —
автоматическое.

Lamport's bakery algorithm (значит. из практики буд.)

- использует холдинг в булевом виде получает только
один холдинг одновременно — с большим из 1 холдинг.
- блокирование — по холдингу
- т.о., что "противостояние" звое \Rightarrow одинак. номера
 \rightarrow временные конфликты, напр., номера в паспорте

! Алгоритм решает проблему крит. секции для N -процессов.

То есть процессом занимается вход в крит. секцию в
секв. с ранее получ. номером.

Псевдокод!

70^e repeat
 1. var choosing: shared array [0..n-1] of boolean
 2. number: shared array [0..n-1] of integer
 ...
 3. repeat
 4. choosing[i,j] := true
 5. number[i,j] := max(number[0], number[1], ..., number[n-1]) + 1;
 6. choosing[i,j] := false;
 7. for j:=0 to n-1 do begin
 8. while choosing[j,j] do (*nothing*) ;
 9. while number[j,j] > 0 and
 10. (number[i,j], j) < (number[i,j], i) do
 11. (*nothing*) искусственное
занесение
 12. end;
 13. (*critical section*)
 14. number[i,j] := 0;
 15. (*remainder section*)
 16. until false;

1, 2 - вначале repeat.
 choosing[i,j] = true, если $i \neq j$ и $i < j$, иначе, иначе, иначе
 будет true. weil $i < j$ в крит. секции, т.к. оба процесса
 меняют number[i,j].

таким образом не возникает конфликтов, number[i,j] = 0.

4-6. - процесс меняет все, кроме (и не меняет)
 7-12. - меняет все, кроме текущего процесса

в крит. секции

8 - меняет same ближайшего процесса

9 - same процесс не меняет и меняет - 1 (0 - не меняет)

10 - искусственное, отключение

06.10.23. Задание 6. (умер: (a, b) < (c, d))

~~задача~~

умер, если

$a \leq c$

или $a = c$, но $b < d$

Этот код тоже создан в 70^e repeat - лучше пойти на прямую к формальному анализу. процессов.

• Тот формальный (абстрактный) с именем исключительного ожидания не реагирует на эти изменения.

• Обновление по первому не приводит к другим оп-б. формальным исключениям, т.к. исключений больше в крит. секции нет потому что ожидания.

• Здесь нет никаких системных блокировок, т.к. реальная архитектура имеет привилегии с исключением ошибок
 => все блокировки в реальности невозможны.

Но! Все изображено - занимает ограниченное (но не конечное) пространство.

2) Аварийный способ взаимоисключением

- аварийное манипулирование test-and-set (a, b)

Это появилось в IBM-380, является недостатком (аварийным) и реализует протокол установки и сброса локальных данных, который наз. аварийное манипулирование.

Принцип, что если будет блокировка — ~~то цикл~~ то цикл ~~закончится~~, а если сброс, то ~~цикл~~ ~~закончится~~

а затем еще в установливает флаги на 1.

```
program use-ts;
  flag c1, c2: logical;
P1: while(1);
    c1=1;
    while (c1 == 1)
      test-and-set(c1, flag);
    CR1;
    flag = 0;
    PR1;
  ...
P2: while(1);
    c2=1;
    while (c2 == 1)
      test-and-set(c2, flag);
    CR2;
    flag = 0;
    PR2;
  ...

```

main()
flag=0;
pos begin
P1; P2;
posend;

- Двухсторонний переключатель flag имеет значение 1, когда находится в привилегированном участке. Поэтому P1 хочет быть в своем привилегированном участке, а P2 — у него нет. Их придется в этом привилегированном участке. P1 устанавливает c1 в 1 и передает в участок привилегированного переключателя командой test-and-set. Несколько раз находясь в привилегированном участке, то flag = 1. Команда test-and-set обнаруживает этот факт и устанавливает c1 = 1, т.к. если второй раз P1 находиться в своем активном участке, то flag = 0. Так вот, пока P1 не выйдет из своего привилегированного участка. Следовательно, это гарантирует способ неизменяемости. Второе обнаружение или это вернуться в свой участок, и тогда flag = 1, и второй раз P2 приводит в привилегированном участке и устанавливает flag.

• У бахане test-and-set ын. просои блокировки
использование команда test-and-set в цикле -
представляет гуареное непрерывное ын. цикл реадж
блокируется (spin lock, simple lock).

Чаще всего test-and-set ытвращает непрерывное гуареное
перемешают.

condition

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

```
void spin-unlock (spin-lock_t *c)
{
    *c = 0;
}
```

• Конс. может восходить к конс. сессии, гуареное стас-сл.
Независимость гарантируется блокир. конс. и. пакет.

• Рекомендуетьтся использовать блокировку заминка ресурсов
и синхронизацию при работе с сессиями (ограничения!)
Потому заминка ограничена конс. ресурсов:

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

т.е. если перемешают заминка, то он не будет. чисто
без блокировки пакет.

3) Организационное сопровождение с помощью семафоров.

Эти методы или блоки, или места в очереди - называются
организационные единицы ын. организаций.

Для того, чтобы использовать прос. нужно более照顧ательно,
и четко это. Готовит больше . действий.

Классическое семафоров действий. (на семинарах Unix)
(использование)

Семафор - переносимые, конс. может применять где гуареное: 0, 1.
Использование!

На семафоре определено где семафор: p и V.

S - semaphore

passeren	vergeven
пропускать	свободить

$p(s)$ - захват

$v(s)$ - отдача.

$p(s)$: если $s > 0$, то $s = s - 1$, иначе брак.

$v(s)$: если $s == 0$ или $s = s + 1$
ошибки проще и горячий захватSemaphore

- Если $p(s)$ не и.т. воспользовался, то процесс блокируется, пока другой процесс не отдастSemaphore.

В этом случае при отдачеSemaphore процесс (первый процесс в очереди) будет разблокирован.

↑ заблокированный, разблокировать. процесс может только ядро (если ядро)

- ЕслиSemaphore имеет значение $2^k + n$, то ядро считает ядро занятой.

↑ $p(s), v(s)$ - независимые операции.

S : binary semaphore;

P1:

...
 $p(s);$
CR1
 $v(s);$
PR1

P2:

...
 $p(s);$
CR2
 $v(s);$
PR2
...

// начальное значение
 $s=1;$

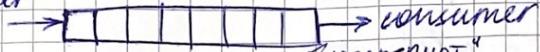
т.к. это - единственное ядро, механизм активное отображание
но! есть недостаток для конфликта (работа в режиме ядра)

действует логика задачи "производство-потребление"

Недостаток этой логики процессов и ядер. •Процесс-производитель может только получивший привилегии задачи и не меняет их в будущем.

•Процесс-потребитель может только воспользоваться задачами из будущего.

producer



consumer

- se - (ког-бо нукох грек бүгүрт) "семафор бүгүрт-пүс"
- sf - (ког-бо жанылыштас) - " ") останоючие
- sb - (бинарийл симафор)

se, sb, sf; semaphore;
producer; white (1);

 $p(se)$; $p(sf)$; $(N = N + 1)$; // габарит бүгүрт $V(sb)$; $V(sf)$;

consumer: white (1);

 $p(sf)$; $p(sb)$; $(N = N - 1)$; // бүрт бүгүрт $V(sb)$; $V(se)$;

// Max. guarantee

 $Se = N$; $Sf = 0$; $Sb = 1$;

par begin

producer; consumer;

par end.

- producer chooses who to monopolize, which is not far from one's interests.
 - Если $se = 0$, то producer будет блокирован.
 - Если monopolist удалось, то они.Semaphore установлен и убийт. ког-бо жанылыштас грек.
- consumer chooses who to buy, which is not far from one's
 - Если $sf = 0$ (琐事, грек нет), то consumer будет блокирован, когда producer не зайдет. Хорошо для buyer есть другую биржу.
 - Если monopolist удалось, то они.Semaphore установлен и блокируется ког-бо buyer. грек (убийт. ког-бо нукох)

8 Все процессы в системе обн. автоматически - авто-ко

9 ~~собственник~~ собственник си-тво.

В современных системах реализованы многосторонние семафоры, которые считаются представителями множества семафоров.

Это связано с тем, что у семафоров есть состояния:

Особенности семафоров.

- mutex - lock - unlock - единство
- Semaphore имеет время ожидания, т.к. у семафора нет хранения, т.е. семафор может ожидать только один процесс.
- освободив mutex может занять fork. это опасно.

Максимум замедления определяется остройкой.

P1:

...
P(S)

P2:

...
V(S);
...

В этом программе ожидает под руку семафор, возникшее горячее состояние семафора.

S1, S2 : semaphore;

D1:

...
P(S1);
P(S2);

...
V(S2);
V(S1);

D2:

...
P(S2);
P(S1);

...
V(S1);
V(S2);

Задача процесса перед запуском просматривает
освобождение семафора.

⇒ в субр. DC запрещенные
найдут семафоров

Сб-ло набора семафоров.

Другой недостаток операции максимум ожиданий все эти задачи находят семафоров.

Задача "однородные философы"

за круглым столом сидят 5 философов.

Перед каждым лежит тарелка.

Если кто-либо

делает движение ложкой по окружности тарелки.

Всем остальным приходится ждать.

13.10.23.

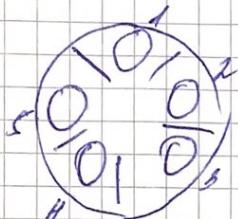
Лекция 4.

1) один из философов поглощает блюдо перед собой - если угадает - ест; если, подозревает его философ; если. откладывает

2) кандидат употребляет ложку перед собой, но дальше есть некому, уходит в нейтралитет;

3) кандидат употребляет ложку перед собой, если не может есть дальше - возвращается.

(1) Три шаги решения задачи см. в прил.



- Две 2 и 5 сидят перед собой
- 1 и 3 и 4 перед собой
- 5 и 1 сидят перед собой
- 1 и 5 и 3 перед собой
- 5 и 2 сидят перед собой
- 2 и 4 сидят перед собой
- 4, 2, 1 перед собой

1) - фаза. откладывания

(2) Кто-то не может есть более позднее - deadlock.

Чтобы не :)

~~если все сидят за столом, то никто не может сидеть за столом~~

Задача и есть одна и та же проблема. Понимаешь, что это там не будет.

В сопр. синхрониз. оп-ии (макросы) (нашего)

на практике пишется так:

Var

forks: array [1.. 5] of semaphore;

i: integer;

begin

repeat

forks[i]:=1;

i:= i - 1;

until i=0;

↓

```

parbegin
  ↓
  1: begin
    left := 1;
    right := 2;
    ...
    end;
    ...
  5: begin
    left := 5;
    right := 1;
    repeat
      // пытается
      p(forks[left], forks[right]);
      // ожидание
      v(forks[left], forks[right]);
    forever;
  end; // 5
parenend;

```

Монитор.

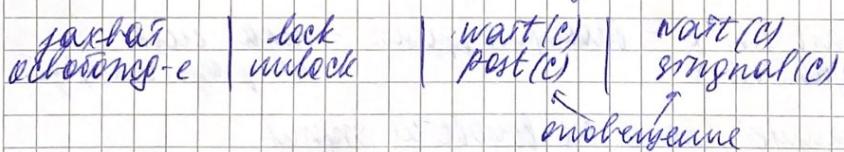
* Программа, связ. с семафорами и пр.ср-вами, обрн.
помогут бл. задач в том, что они не отбрасывают.

• Все системные языки, связ. со физическими-ми, можно
управлять как lock и unlock.

* Особенность блокирующихся процессов может помешать прогрессу,
который есть у задач. - в отличие от семафора

Разные принципы работы и об-в в блокирующихся).

- приходит к блокирующей процесс (это делает только процесс)
- process-ы процесс может приводить об-в инициировать процесс.



! Если параллельно стоит -е линии wait(c) и signal(c), то
нижний может потерять синхронизацию и блокировку.

⇒ • возможна ошибка: процесс использует блоки не до
конца. принципов

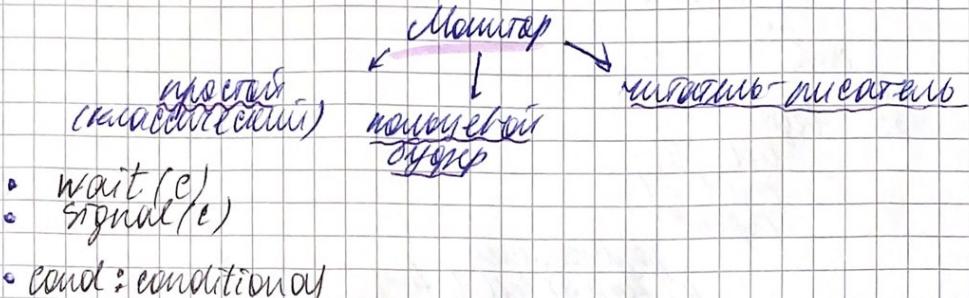
• решение: введение мониторов.

Онлайн-монитор - просто структура с-в ядра (сигналы) и его
действиями; управление, кот. сог. выполнение и пр-ши, второго могут
использовать эти действия.

(формально к нему) • гарантирует выполнение монитора, т.к. доступ - только к ним

• гарантирует выполнение монитора, т.к. доступ - только к ним

- В результате выполнения синхронизируемых процессов, объекты
 • могут быть "заняты".
 • Остальные процессы, пытающиеся получить к объекту доступ
 • должны - они "бес конкуренты".



1) блокир.

* оценка:ование из-за которого могут выйти процессы.

monitor: resource

var

```

busy: logical;
x: conditional; // переменная типа условие
procedure acquire; // "заполучить"
begin
  if busy then wait(x); // если ресурс занят, то избрать
  busy := true; // блокировать и не делать x wait
end;
procedure release // "возвращение"
begin
  busy := false;
  signal(x);
end;
  
```

begin

busy := false;

end.

~~wait(x) wait на x~~ - блокирует процесс - пока нет.

~~signal на x~~ не разблокирует

Система блокирует, пока не будете signal

? обратиться к первому. монитору можно только 7/7 его нет

2) конкурент буфер.

- управляет буффи. Являю - производство-потребление
 - производит - производитель буффи. данные в буффи. буф. опр.
 - потребляет - потребитель - получает из буффи. буф. опр.
- (см. примеры лекций)

monitor: resources
var

bcircle: array [0..n-1] of type;

pos: 0..n; // текущий индекс

j: 0..n-1; // добавляемое значение

k: 0..n-1; // об崇高аемое значение

buffer_full, buffer_empty: conditional;

procedure producer (data: type)

begin

if pos = n then

wait (buffer_empty);

bcircle [j] := data;

pos := pos + 1;

j := (j+1) mod n;

signal (buffer_full);

end;

procedure consumer (data: type);

begin

if pos = 0 then

wait (buffer_full);

data := bcircle [k];

pos := pos - 1;

k := (k+1) mod n;

signal (buffer_empty);

end;

begin

pos := 0;

j := 0;

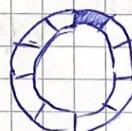
k := 0;

end.

Решение задачи решения бенчмарка.

- Так же, как и в предыдущем случае состояния буфера: full, empty.
- producer занимает один буфер и освобождает его - buffer_full.
- если producer не может наполнить его данными (buffer_full), то он блокируется, пока consumer не будет ждать empty, ему разрешен.

Контейнерный way, т.н. producer



! Особенность: наличие двух новых операторов - приведенных в начале листа.

3) Mutual-exclusion.

• Document: natural objr. times shared - readers,
 ref. about round robin journal, n writers,
 ref. why round robin journal. writers

? Daelsejje no kruno devrelo.
 • Access: neymka jurna c posiccaiuu online (obj, rooth)
 • A qm : Start read, stop-read, start write, stop-write.
 ? Yuramni yuvalo raxakunenno. B tucateni - b perecne sponnouck-i,
 monitor: resource; var
 nr: integer; // kau-bo "kursi"
 wrt: logical; // "antubuot" nucornt
 c-read, c-write: conditional;
 procedure startread oqepg?
 begin
 if wrt or turn(c-write) then
 wait(c-read);
 nr := nr + 1;
 signal(c-read);
 end;
 procedure stopread
 begin
 nr := nr - 1;
 if nr = 0 then
 signal(c.write);
 end;
 procedure start write
 begin
 if nr > 0 or wrt then
 wait(c.write);
 wrt := true;
 end;
 procedure stopwrite
 begin
 wrt := false;
 if turn(c.read) then
 signal(c.read);
 else
 signal(c.write);
 end;
 begin
 nr := 0;
 wrt := false;
 end.

▷ Aile nro, yuvalo yuramni yuvalo by startread.

- one prob, etim mi oot. yuramni yuvalo neysusur
 nuc-no b oqepg
- ga: last-e oncupame no neys. c-read tura ya-t
 nteborot b
- wrt: yeleni. nro. yuramni yuramni nr. nro. imanu monduo
 yuramni - cikciu grynni yuramni. - boqumak cikciu
 peaqmii q-nro.

- If the job has stopping it is stop.read:

 - если нет то возвращает список "максимумы"

- If the maximum number must start with:

 - пробегаю от меньшего до большего и смотрю на максимум
 - да: переход в сост. ожидания не нужен. runa yes-e.
 - нет: wrt = true - номер

- If the job has sum-gathering it is stop.write:

 - строка wrt
 - пробегаю по текущих x-reis
 - да: максимум считать
 - нет: максимум нечет.

Особенности:

- 1) использование переменных,
- 2) использование беск. цикла. Число и итерации запоминаются

- 3) Могут быть неправильное обновление:
 максимумы не могут менять то время ставки \Rightarrow удачный
 бывает максимум только последним переходом
 - не много с т. зрения ставки только актуального данных.

20.10.23. лекция 8.

• если мониторы повторяются из-за инициализации не сбрасывают

Рандеву

достигаемость блокировок (таке вращающиеся)- синх. ресурсы, нельзя
 блокировать.

- | | |
|--|--|
| <u>Синхронизация</u> | <u>Вращающиеся</u> |
| - одновременно зажигают
лесовод в воротах
зрят впереди них.
зажигают и передают | - один ворот не может зажигать
в крат. секунду но разрешено
менять в нем так. группы вратарей. |

- Более:
 1-ый зажигает
меняет зажигает до горения
лесовод т.к. других зажигающих
зажигает до горения
2-ой меняет зажига-
ющих (т.к. зажигательный)
и т.д. и т.д.
- 2-й зажигает зажигает
до горения

• Синх. мониторы с т.з. не следят за
исходом блокировки

• блокировки - это

Рассл. мониторы находятся под под контролем ф-и:

- многозадачность - синхр.
- известен-исследование - вращающиеся

В рамках 8-и Н-ных процессов был создан термин
граница-системы рамку. (много описано в 317 АОА)

1986 г.
(авторское название?)

1 Задача (task)

- решает задачу
- может быть сел., параллельно, независимо друг от друга
- имеет ID 8-и друг с другом

2 Не реализованные задачи.

- и/т как доп. объект, ведущие себе так, как если бы они существовали на рабочих машинах, но! ограничения не позволяют отдельно бороть не могут.
- Entry - сбрас.; если одна задача устанавливается рамку - нельзя это делать, задачи блокируются.

Рамку:

1. Сост. задач. и реальных параметров
2. Ограничение операторов.

• Постр. операторов, как говорит, форм-ы от имени структуры задач в рамку, т.е. постр. операторов становится основным языком описания.

task <имя> is

entry <идентификатор входа> (дискретный параметр) (программная часть);
<группа обозначение входов>

end <имя>;

accept:

accept <идентификатор входа> (входные) (форм. часть);
do последовательность операторов end;

- Этот механизм поддерживается программой.
- Несмотря на то, что мы-т ли-и-е посчитать операторов получается однозначно, в них-и-е рамку есть такие же проблемы, как и в 8-и Н-ных процессах в др. системах.

• Запрос на уч-е рамку л.б. отключен

Современные системы также вынуждают в распределенное взаимодействие.

Некоторые общие принципы взаимодействия задач в Н-ных процессах это: соглашение.

Советы

Особенности функционирования ПМК при работе в распред. сист.

Виды распределенных систем: системы с разделимой памятью.

! Для синхронизации не могут быть используемы симметричные и т.д.

- Синхронизация по часам

- представление, что есть две системы, действующие одинаково, которые "координируются" в едином

В ОС UNIX:

Установление времени производится при запуске ядра. Это первоначальное значение.

Программа make проверяет время при-датка (с.и.о.) и принимает решение о переходе к новому.

make получает временные данные от ядра, которые сравниваются с текущими. Если они отличаются, то программа обновляется.

Пример:

<сущ>.с - 1224

<сущ>.о - 1223

- требуется переход.

Пример:

методы по координированию часов (необходимо для исп-я языков программирования)

- конст. для переключения:

1223 1224 1225 1226

- конст. для регулирования:

1222 1223 1224 1225

Испр. <сущ>.с

переход,

изделие изменяется

не нужно.

Работа часов (CMOS.RAM) основана на работе такт. генератора, основная частота микросхем (тактовая) также регулируется. Частота 10^{-5} - миллиардные часы.

Максимальный - 216000 тактов / мс ; $215998 \div 216000$. Равнознач.

Суммирование времени

- коэффициенты коррекции - коэффициенты коррекции суммы в часах. Году, коэффициенты есть - интервал между двумя погрешностями.

- mean solar second - интервал за который делают среднее-133

(1948г. - выбор астрономические часы (среднее-133))

составляет 9192631770 переходов.

160 лет. в мире

и/или астрономическая секунда устанавливается атомными часами и
international atomic time (TAI)

- настолько совершено что устанавливается, а астрономическая секунда
 не устанавливается. Время имеет наименование parsec от непре-
 туб

⇒ Введение дополнительной секунды leap second
 (TAI - leapsecond frame) = 86400 sec

⇒ UTC (coordinated Universal Time): TAI + leap second

Помимо этого для компенсации потоками UTC в основе
 60/60 Гц часов.
 leap second → смена часов (60 → 61)

- National Institute of Standard Time - распоряжает синхронизацией времени.

В часах есть

- аналог. кристалла
- аналог. генератора
- Усредняющие алгоритмы

Использование требует в установке определенных параметров ⇒
 ⇒ определение часах.

Нет исправления синхронизации времени - ?

- 1978г. - аэропорт: координатные часы
 timestamp:

Глобальный часы из 3х координат.

На которых есть атомные часы.

0	1	2
0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	56	70
48	64	80
54		

A, B - единицы, постав-
 лившие миллиардны

D, C - отдельно, время отработки
 большее времени погружения -
 не более.

Применено для вычислений
 в соответствии времени отработ-
 ки.

Такое напоминает соединение. Если час. время меньше
 времени отработки, происходит замена час. времени на
 время отработки соединение +1.

Получив C(60), можно I yes время на Cт.

Date _____

0	1	2
0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	61	
48	69	
54		

В результате такой коррекции
максимальное значение должно быть
согласовано "изменение ячейки"
- "изменение ячейки"
- временно-следующее ORT.
(causal precedence)

Реализуемое мт. часы требуют решения 2х проблем:

1. мт. структура данных
2. временные затраты мт. структур данных и сокр. времени конвейерности.

Компьютерный процесс решения

1. мт. мт. часы (LOCAL CLOCK) - лок. часы процесса pi
2. мт. мт. часы (GLOBAL CLOCK) - глобальные часы процесса
- лок. время процесса pi не мт. времени (лок. значение, $lct \leq gci$)

Потоком данных передается мт. часы процесса и ID, так как
он представляет мт. время, т.е. синхронизирует.

- vector clock - самоостановленно.

- Россия, Борисовское: ГЭВЧ. Реализованное на гипаралл.
имеет 32 биты ячеек. на переходе данных, есть у часов
на часах ~ 240 Гц (11/сек)

10. Уроки из 240 урока

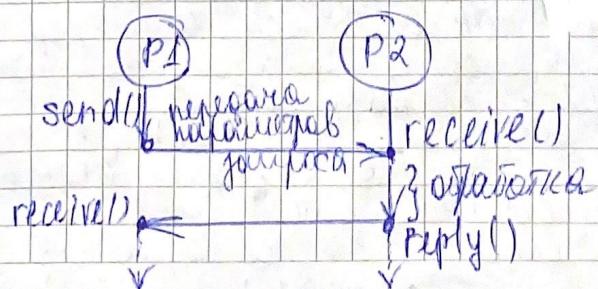
29.10.23. Лекция 9.

Особенности передачи сообщений в расп. системах.

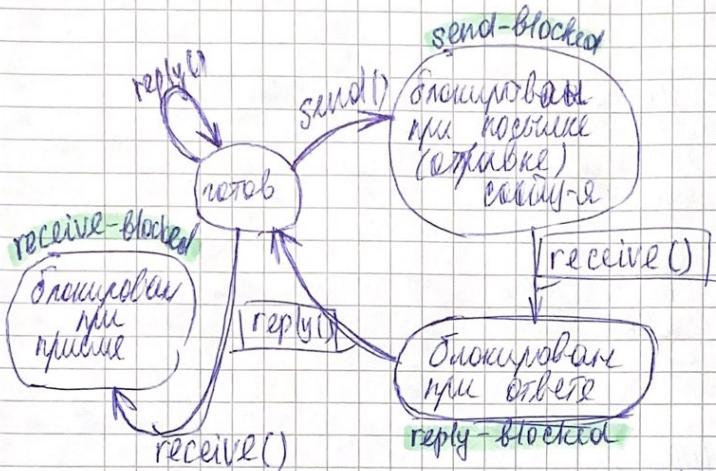
Процессы могут быть инициированы мастером-принимающим сообщ.

+ дан. требования на сообщ.: name, протокол by-е, суть или назначение.

"Master - конверт с адресом" - значение дан-е в софт. агентов информационных



функционал "3 состояния блокируемых процессов при негарантии сообщений"



- блокирующая функция
- создание + синхронизация
- объекты могут негарантии сообщений

- или сост. будет в виде цикла с синхронизацией и ожиданием reply

т.е. Э рутинные операц., используемые для работы этих объектов.
(System V - очередь готовы)

- процессы - пример, том-еэс со своей сущностью \Rightarrow можно предложить различные подходы к их реализации

аналогия блокируемых процессов в распр. сист.

① Неподтверждаемый аналогии

- основан на основе вз-я с процессом на орг. машинной
- имеется процесс-координатор; когда какой-то процесс хочет получить в крит. секции из-за несогласия сооружения \rightarrow получает эту крит. секцию \rightarrow удерживает запрос в очереди (если она пуста) - проверка времени выполн. запроса \rightarrow разрешение тайма, у кого запрос помешан, поднимается в очередь
- место крит. секции - физической (надежность зависит от процесса координатора, если он заберет слишком много времени, система сбои искажит исполн. языки. \Rightarrow выполняющий процесс будет иметь только координатора,
 - когда пакеты процесса забираются из очереди для обработки, тайм-аут \Rightarrow нет, не может получать ответ \Rightarrow кое-раз-ко неизменен \Rightarrow нет-ко тайм-аута).
- это инициирует повторную выдачу координатора
 - посыпает ему сообщение "ВЫБОРЫ", в кот. указывает свой сетевой номер, всем флаг. языку.
 - процесс, получ. "ВЫБОРЫ", пишет id:
 - если это id больше, чем все предыдущие "ВЫБОРЫ" со своим id.
 - процесс с id предыдущий

иерархия: корпоративный - процесс с Адм. ССТ. Исполнитель
Если же все вышеупомянутые уровни отсутствуют, то есть корпоративный.

Проблемы:

- все функции процессов г.т. создают статус-кво-роли
 \Rightarrow одна из них ГД на канадской машине
- н.г. неподходящий процесс

(2) Рационализм и инновации

- в функциональных процессах \Rightarrow могут отвечать за один и несколько физ. ресурсов
- процесс может быть в крат. секции по компр. фас.

ночного н-1 состояния
с установленным КМУ, складом и
определенными отработанными способами.

н.к. процесс получает

не соответствующие
запросы

находящиеся
в крат. секции

хотят больше в зоне
инф. ц.

расширяет запросы -
меняя события
с распределением

не получает пол-
номочий, запрашива-
ет вперед

запрашивает больше
и расширяет запросы
и требует в своем
запросе

если нет
правил - правило? нет
запрос
получает
распределение

получает
запрос
вперед

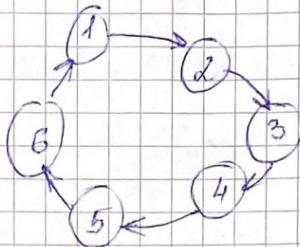
\Rightarrow н.к. получает н-1 распределение

• первое не устроило бремя на запрос. бремени

\Rightarrow второе бремя получилось

(3) Token ring.

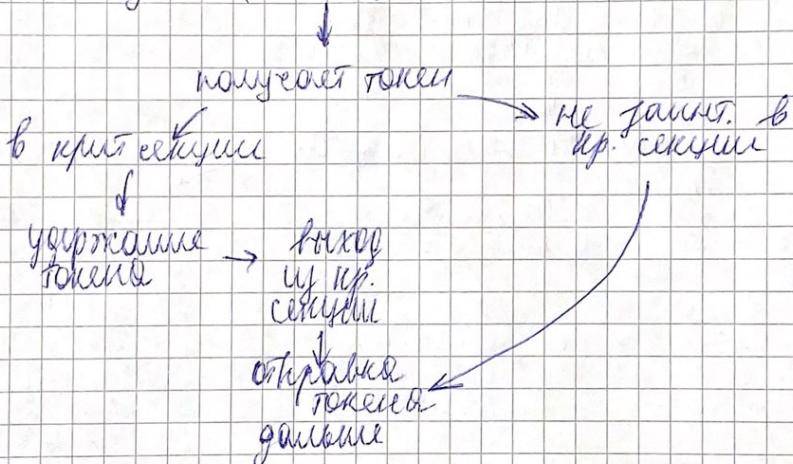
- процессор отправляет пакеты в кольцо



- контроллер процессора
записывает в кольцо, кому
поможет передавать
пакеты

То, чтобы обменяться пакетами — сначала записывается в кольцо, удаляется из кольца, в котором он идет, и процессор. Количество пакетов = Количество сетевых

Пуск в колесе с пакетами.



Когда чистота пакетов пакетов? (без гарантированной
(с гарантированным или нет)

Проблемы с ...
? (1) - процессор не может не принять пакеты, так как
они пропадут в кольце. (ред. пак. буст. или)

? (2) - может быть пакеты пакеты пакеты пакеты.

Транзакции

"the unit of work"

- время недолгое
- последовательность действий
- можно отменить из процесса
 - (прерывание процесса может привести к ошибке-упорию и т.д., успешные могут не выполнены
также переносит ошибки процесса, как можно видеть, отменять не могут)

Одна транзакция имеет одиннадцать состояний транзакции с одним или неск. шагами.



установление, создание объектов



составлять хотят языком транзакций

see my estimation



выполняет транзакции
одинаки

хотя это один
процесс может

транзакции
не атомизируются

Прим. Транзакции - это-то что операции над объектами и
нек. обьектами (документ, БД, отдельный фрагмент) хотят
переводить единицы из одного состояния в
другой иначе то действие.

atomic begin transaction
begin transaction
end transaction
abort transaction
read/write

Свойства транзакций

ACID

1. Атомарность (atomic)

- если транзакция удастся выполнить, то она атомаризуется (commit)
- если нет то в ее пользу ошибки, то откат (rollback)

2. Консистентность (consistent) упрощенно

- т.е. данные переводятся из одного состояния в другое

3. Изолированность (isolated)

- Транзакция изолирована от других транзакций т.е. инициатор и дальнейшие транзакции не видят (и наоборот) и не могут видеть

4. Устойчивость (durable) (постоянство)

- при работе машины ее нужно загружать, и никаких свойственных

Решение / сервис EJB ?

8 Але, оскільки у нас не було часу, то ми однотижневими, але вже
так, ось що було сказано в доп. написах.

Keanuayauel cb-Bi

- 1) низкая софгамма для процессов, ур. в гран., арг. пар.
УР-Би, & нестабильность масс. образов \rightarrow превращение пар
комплексов \rightarrow присоединение \rightarrow все присоединенные гамма-кв.
кин. в уст. отеков

§ Дополнение к зан. раскопок

- 2) cancer de mama

- изолированность уч. объектов, а не кластеров
 - переходный промежуточный блоки. Граница - изолированная граница (уч. и учи.). \rightarrow изолированная (одно здание) или же граница (учебного заведения)
 - в изолироване - ~~разные~~ группы, это элементы, когда есть одна группа однотипных, пом-ес откат - функ-сост. го функционирования

5 В расп. системах м. б. метод. прис. грав-ций и
подлож. чистота \Rightarrow очень часто они присущи используемым
чел. протоколам очистки и фильтрации гравийных
материалов:

- each by turns. — parts keeping彼此

unus. [↓] gauz.

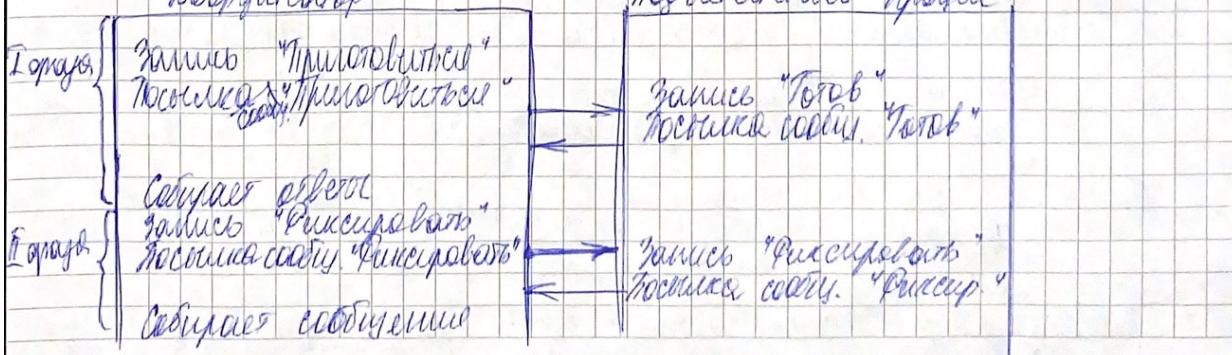
запись в блоке миграции перешла

ночью на проектной выставке.

anp. geticht

кооп-р рассчитан "привативиза"

Программный проект

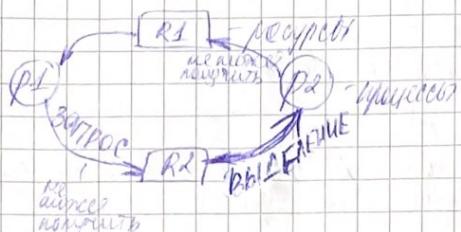


Тупики в прокладках

- движение нудитое

03.11.23. Лекция 10.

Тупики в прокладках.



"Нес. прокладывание DC"
"затруднения при работе процесса"
"не разрешается" "запрос"

↑ время тонущее..

Касающиеся тупик
(но процесс идет дальше)
(тупик) "Продолжение"

P1:

запрос R1

запрос R2

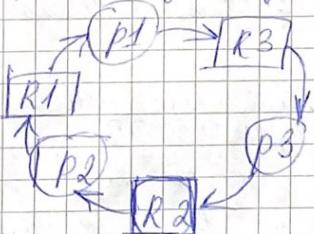
P2:

запрос R2

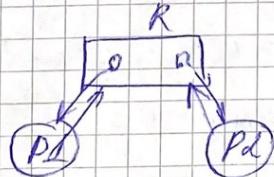
запрос R1

↑
тупик deadlock

ан Тупиковыми си. (тупик) - ситуация, при. в результате подавленного си. рабо. ресурсов, когда процесс, буд-
шему ресурсы, запрещаются, чтобы не было, запрашиват. 2/3
членов, запроц. членов, дальнейш. запрашиват. процессов, начиная-
ющим восстановление ресурса, поэтому все процессы



аны



в тупик тупиков

→ переписать

- потребитель
- создатель
- такой ресурс несет
- существование, если его
- "потребитель"

- потребитель использует
- не является ани. состоящим
- бор. системой
- + дальнейш. его самий существует,
- + есть табличек
- + объекты ядра (символы си. бор. времени нации)

Поведение подр. тип ресурсов:

- 1) линейные: эти ресурсы ограничены (ресурс и потребление)
(изменение числа - изменение, а не управление)

- - ?

Установка (метод и восп.) формирования гумиков

1) Установка-исследование

- менеджерские или личн. ресурсы профессии

2) Осижение (Actel-and-aware)

- процесс удерживает поток ресурсов в осаждении наименее ресурсов для продолжения потока

3) Непрераспределимость (precintion)

- ресурсы никак не отдача у гумика до его забывания или забывания ресурсов самими профессиями

4) Круговое осаждение

- при. замыкающая цепь гумиков профессий, в которой каждая профессия влияет на каждую ресурса, метод. другому профессии продолжает возвращаться.

Методы бережот с гумиками

исключением береж. по ним)

1) Избирательное гумикование

- сужение в системе гумиков услуг, при которых не берутся гумики

2) Азкад гумиков

- гумик берется, но пред. анализ системы поддается только гумик обогащению

3) Гумик бережности

- при попадании эст. в гумик есть обогащение

- и привести систему в норм. состояние

Гумик не берется, если будет исключено хотя бы одно условие их бережливости
(стратегия X...)

Какие системы, которые не могут попадать в гумики - реальное
внешнее, превышающее для управления внешними профессиями.
(автимат)

Методы исключения условий бережливости гумиков

1) Операционное обновление

- процесс гумиков заменять на нов. или в ход. поток ресурсов

• искл. осаждение

- процесс заменять живой способность в системе

- новые ресурсы в незадолженных или личн. ресурсов → увеличение

- Гумик с реальн. профессиями или системой обогащения (шагает вперед First)

+ разрешить задачи на реагентах,
которые требуют двух цветов
но! основное правило, если "т"

2) Метархимическое расщепление

- реагент системат унитарного (пр. в нех. метархисе)
- правило: процесс может разделяться только на
ресурс, если нет. соли, или соли ТН, и т.д.
- исключение: круг. окислительные

метаб. задача

Большинство процессов - более сложные
метархимические процессы

+ можно увидеть если процесс не может делиться на ресурсы
в чистом виде, он может делиться, если разделить
на ресурсы и дальнейший ресурс в чистом виде
⇒ для об. разделяется, при этом не все это

3) Установление условий метархимического процесса

- если процесс не может получить ресурс, нет. или чистый, у него можно отобрать те ресурсы, нет. он становится

2) минимизацию расхода других и тех же ресурсов

~~Минимизация общего гуммиевого~~
каучукового антиоксиданта антиоксидант бензальда

- считается, что качественные изменения подаются генетически

- генетически-процесс - дальнейшее нечто. где есть ресурс

- Бензаль - система, амину. гумми

ограничение: • процесс получает управление в результате

этого дальнейшего непрерывного в результате ресурса

• или, либо процесса гумми и каучука и

• процесс не может разделяться на чисто из

ресурсов или управление в результате (нету минимума

в системе)

200 ограничение позволяет оптимизировать
и минимизировать

Минимум баланса может не быть процессов, которых
затем не будет, если такие не будут найдены, система это
насчит отрицательный гумми.

Оптимизация ресурсов должна быть ресурсов, чтобы не было оставлять
ничего в отрицательном состоянии.

Date _____

Пуск ресурса - 12 единиц

Процессор	Текущее распределение	Свободное единиц от ресурса	запасы
P1	4		12
P2	2		4
P3	4	2	8

Какое действие - будничное или типичное или нет?

QA, T.K.

• имеет вид 3 процессов, но не имеет
установлено действие: $P2 \rightarrow P3 \rightarrow P1$ не ~~занят~~T.K. & занятые -
P1, занятые - 2 \Rightarrow занято 2. \Rightarrow free. \Rightarrow очевидно, занимаемое
има единиц от рес.

[монит. состояния. макс.
погрешность P3]

T.K. макс - 8,
усл. 4 \Rightarrow макс
занято. 4

Будничное в 4 на свободном (3).



[P3 остат. 8 единиц.]



Несколько процессов переходят в будничное состояние, как?:

если

P1	5		12
P2	2		
P3	4		8

• не всегда типичн.

Решение:

Составление списков ресурсов, если в проекте участвуют бакалавры,

УДО:

- 1) 1^й процесс в лаборатории обработка заявок - т.к. заявки если они заявлены вами, то вы можете ими пользоваться и не платить за них.
- 2) 2^й процесс может быть если есть 1^й заявка на него, то вы можете использовать его для выполнения заявки.

- i) 3-й процесс заявки может быть если есть 1-2 заявки на него и первые две из них выполнены, то вы можете использовать их для выполнения заявки.

Эти определения являются с. между ними не согласованы.

→ это бессмыслица, потому что если заявка не имеет, то как же она будет выполнена в этом случае?

→ некоторое число запрещенных

написаний,

(запрещенные написания) - любые написания, которые

должны отсутствовать в вашем списке ресурсов, написанных вами, но не имеющие никакого отношения к вашему проекту.

Запрещенное написание №1 - запрещенное написание, которое не должно быть в списке ресурсов из-за ошибки.

→ Абсолютно все, что вы пишите в списке ресурсов и не имеет никакого отношения к вашему проекту.

В проекте: написано, что есть списки.

Пример

N написано, M типов ресурсов (столовых)
(столов)

Методы Кальмана(Наберманы) δ можно достичь путем какого процесса, используя

Как узнать, что состояние борется?

- Матрица в начальном состоянии
- Понимание возможное количество процессов, возможных единиц состояния
- Если все процесса завершились, то состояние окончательное

- n процессов;

- m ресурсов;

- Матрица доступных ресурсов

$$\text{Max-Avail matrix } A = (a_1, a_2, \dots, a_m)$$

- Матрица затрат

$$\text{Max-Claim matrix } B = \begin{vmatrix} b_{11} & \dots & b_{1m} \\ \dots & \dots & \ddots \\ b_{n1} & \dots & b_{nm} \end{vmatrix} = \begin{vmatrix} b_1 \\ \dots \\ b_n \end{vmatrix},$$

столбец предложений с учетом макс. потребности ресурсов

(процесс может не затрачивать макс. макс. ресурса)

где b_{ij} - макс. можно тратить ресурсов R_i ,
затрачивающиеся P_j .

- Матрица распределения

$$\text{Allocation matrix } C = \begin{vmatrix} c_{11} & \dots & c_{1m} \\ \dots & \dots & \ddots \\ c_{n1} & \dots & c_{nm} \end{vmatrix} = \begin{vmatrix} c_1 \\ \dots \\ c_n \end{vmatrix},$$

где c_{ij} - как-то об. рес. R_j , которое уделяется P_i

- 1) $\forall k, B_k \leq A$

процесс не может затрачивать больше единиц ресурсов, чем доступно.

- 2) $C \leq B$

процесс не может изгнать затрачивать больше об. рес., чем указано в его заявке.

- 3) $\sum_{k=1}^n C_k \leq A$

матрица не ит. б. формирует ресурсы больше, чем доступно

- 4) Матрица доступных ресурсов

$$A = (d_1, \dots, d_m) = A - \sum_{k=1}^n C_k, \text{ где } d_i - \text{как-то об. рес. } R_i, \text{ если } C_k \text{ содержит } 1 \text{ в } i\text{-й строке}$$

5) Матрица возможных запросов E

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

6) Матрица запросов F

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

7) Предварительное состояние

Предположим, что удовлетворено все запросы, и
проверка данна.

$$\Delta \leftarrow \Delta - F;$$

$$C_i \leftarrow C_i + F_{ji}$$

$$E_i \leftarrow E_i - F_i;$$

⇒ Запрос удовлетворен, если предварительное состояние
живо. Образовалось.

Выполним проверки образованного состояния

1. Используя нерав. запрос P_i : $E_i \leq \Delta$,

т.е. достаточно большие ресурсы, чем имеются. P_i .

Если такого запроса, то переход из шаг 1.

и т.д.

2. $\Delta \leftarrow \Delta + C_i$

т.е. нарастают потребности запроса.

P_i отмечается, как жив-од, а live.

переход из шаг 1.

3. Если все процессы отмечены, как живущие, то система жив. в образованном состоянии,
то система жив. в безопасном состоянии,
иначе - в опасном (неудовлетвор.)

! Если система жив. в кетвир. состоянии, то запрос
выполняется и состояние системы соответствует ему.
образует:

$$\Delta \leftarrow \Delta + F_i$$

$$C_i \leftarrow C_i - F_i$$

$$E_i \leftarrow E_i + F_i$$

Кроме того, процесс, помеченный, как жив., не имеет Δ ,
составляется:

$$\Delta \leftarrow \Delta - C_i \quad \text{и помечается как нерав. } (P_i)$$

Динам.

В) текущий момент:

- max-Avail $A = (2 \ 4 \ 3)$ - доступные ресурсы

- max claim $B = \begin{vmatrix} 1 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{vmatrix}$ $\begin{matrix} 1^{\text{трея}} \\ 2^{\text{трея}} \\ 3^{\text{трея}} \end{matrix}$

$\begin{matrix} \uparrow \\ 1^{\text{трея}} \end{matrix}$ $\begin{matrix} \uparrow \\ 2^{\text{трея}} \end{matrix}$ $\begin{matrix} \uparrow \\ 3^{\text{трея}} \end{matrix}$

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

$\left[\begin{matrix} \text{видим, что } 1^{\text{трея}} \text{ может получить } 1 \text{ ед. рес. } R_1, \\ \text{а } 2^{\text{трея}} \text{ и } 3^{\text{трея}} \text{ не получат } 1 \text{ ед. рес. } R_2 \text{ и } R_3. \end{matrix} \right]$

- Available $D = (0 \ 1 \ 1)$

- Недоступные заявки $E = \begin{vmatrix} 0 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix} \leftarrow B - C$

Пусть P_1 делает заявку 1 ед. рес. R_3 :

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

Изменяющееся первоначальное состояние:

$$D = (0 \ 1 \ 0), \text{ т.к. } P_1 \text{ был 1 ед. рес. } R_3$$

может получить эту ед. из (ав. F_1)

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

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

В) получаем соединяя яв-еи момент процесса P_3 ,

$$\text{т.к. } E_3 \leq D$$

В) результате P_3 берет чистые заявки $(1 \ 0 \ 1)$

в) нулевые свободных ресурсов и заявок:

$$(1 \ 0 \ 1) \rightarrow D = (1 \ 1 \ 1)$$

В) результате можно будет уравнить заявки P_1 и P_2 =

\Rightarrow пред. состояние в результате заявок P_1 авт.

запись $\rightarrow F_1$ и т.д. удовлетворен.

Все изменения сводятся к аналогичным
изменениям - выведение второго D , сравнивание с ини.

1. Существоование ограничение: можно ли существовать нек-то процессов.

Основание: отказ от птицы.

Обнаружение птиц

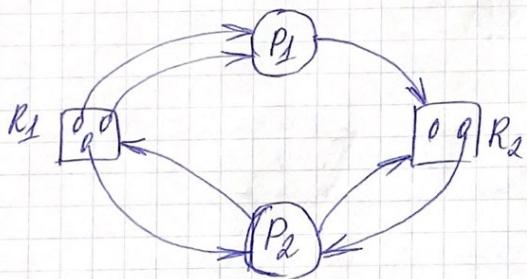
для описание системы исп-ся двумя пары ресурсов (не знаю вершины какого из 2 не н-ся первичн.); каждое ребро соед-т какую-то пару (из 2-х первичн. в); при этом ресурсы \rightarrow процесс $\begin{cases} \text{птица} \\ \text{не птица} \end{cases}$ (процесс \rightarrow ресурсы - ядро)

для обнаружения птицы исп-ся метод подсчета ядра:

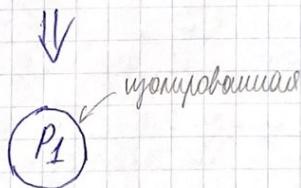
Чт. ядра удаляются все ребра, которые отключают, что ресурс и. ф. входит в процесс.
 \Rightarrow В р. ~~использования~~ ядре, если осталось только используемое вершинное, система не име-б птиц; если остались некон. вери. - в птице.

Пример:

1)

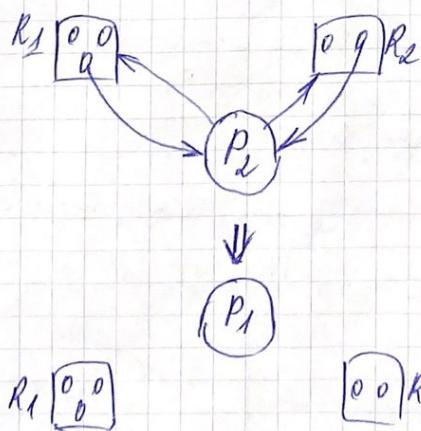


источник



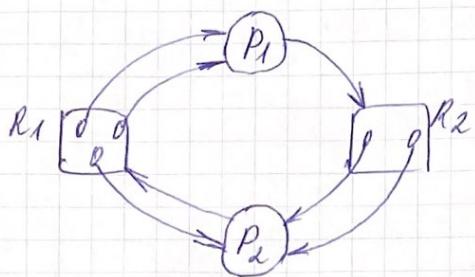
использованная

источник



- система не содержит в птице

2)



формации
запасов уменьшает
запросов

+ аудио

Th1

Граф яви. полностью циклический, если есть такая
последовательность удалений, в рез. которой удалили
все вершины? аудио

Th2

Циклы в задаче побт. использующих ресурсов
яви. неизв. усл. гуман.

Th3

Если есть в S не яви. состоящими группами и
взаимосвязанными переходы из S_1 в S_2 в S_3 в рез. работы
процесса P_i , группам S_2 - гумановы,
то эта операция, том. P_i , яви. запросов.

- Две приведенные структурные схемы имеют
две матрицы: распределения и запросов.
- Аномальность: некорректн. как-то об. какими ресурсами
использовать (если побт. исп. - менять легко,
не меняться)

! Используется появление запросов в расши. алгоритмах
(Этот процесс наз. анализом)

! Отображает матрицу, отвечающую заявкам

~~Причина~~

• Матрица распределения

Матрица запросов

$p \setminus r$	1	2	3
1	0	1	1
2	1	3	0
3	1	5	0
2	3	1	1

$p \setminus r$	1	2	3
1	1	1	0
2	0	0	1
3	1	1	2

Метод свободных промежутков ресурсов

F:	r	1	2	3
		4	0	2

[δ чес. ег. рес. + фаг. ег. рес. = кон. во ег. рес. в сист. -
- начально распределяется]

$$R_j = \sum_{i=1}^n b_{ij} + F_j$$

$$R = (6 \ 9 \ 3)$$

Для анализа сост. системы мы можем сравнивать сроки выполнения проектов с F :

- 1) P_2 имеет сроки $< F \Rightarrow$ он может задержаться, чтобы сократить граф по времени P_2 и облегчить зам. им ресурсов.

Матрица распределения: (единицы \rightarrow дол. в F)

$$\begin{array}{ccc|c} & 0 & 0 & 0 \\ 0 & 0 & 0 & \\ 1 & 5 & 0 & \end{array} \rightarrow F = (5 \ 3 \ 2)$$

- 2) В результате P_1 имеет сроки выполнения $< F$
 \Rightarrow он может заб. се \rightarrow можно сократить граф по P_1

Матрица распределения:

$$\begin{array}{ccc|c} & 0 & 0 & 0 \\ 0 & 0 & 0 & \\ 1 & 5 & 0 & \end{array} \rightarrow F = (5 \ 4 \ 3)$$

- 3) Аналогично для P_3

$$\begin{array}{ccc|c} & 0 & 0 & 0 \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \end{array} \rightarrow F = (6 \ 9 \ 3)$$

Граф полностью сократен \rightarrow граф исходящий \rightarrow
+ система не захороняется в пучке.

Алгоритм Клауди-Мисра

1983 г.

- Алгоритм обнаружения дупликатов на основе кратного сообщения
Если процесс имеет ^{запрос} не занятых ресурсов (не имеет заявок
на получение)

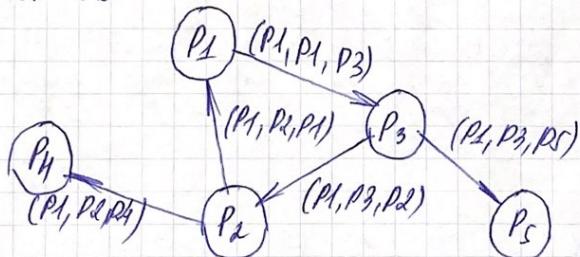
Кратное сообщение ZDHD сог-с еспр.

1. ID, который оказался выделенным и иници-
-рован отправку сообщ. типа ZOID
2. SD процесса, который получил 1.
3. ID процесса, кот. должен получить сообщ. типа ZOID

В случае получения кот. сообщение процесс профе-
-рирует это name, и если в нем имеется обнаружи-
-вает свой ID, система вых. в дуплик.

инициатор	отправитель	получатель
-----------	-------------	------------

Демонст.



Управление памятью

17.11.23.
Лекция 1d.

Universal memory оперативная память

- в арх. философии проявляется и имеет собственное понятие
 - ⇒ постепенно обменивается информацией с оперативной памятью

- В старых системах: внешняя память, внутренняя память
Сейчас: динамическая память

- задача: обеспечивать долговременное хранение данных на диске, устройство хранения "доступное для чтения и записи различных форматов и типов, но при этом производительное, легкое для соединения с другими компонентами (так как txt файлы)

- КДУ
 - Кем 1
 - Кем 2
 - Кем 3

} Кристаллы памяти

! Нет способа новоинки блокировать процессорные данные, чем блокировать внешнюю оперативную память. - почему?

- Обнаружение в оперативной памяти - выявление действий - "запись обратимые к памяти" (это они пишут)

решение: кеми + кристаллы

↓ кеми ↓ уровень т. памяти
кеми ↓ уровень - памяти

- хранят данные, к которым можно обратиться с помощью обратимого (у оперативной памяти)

- Задача упр-я памятью: передать данные с одного уровня на другой.

- Задача кеми: в оперативной памяти, чтобы использовать различные частоты обратимые к диску (точка связана с другим памятью и т.д.)

Виды управления памятью

Внешнее

- при уровне памяти надежность памяти

внутреннее

- управление памятью

дисковое адресное пространство

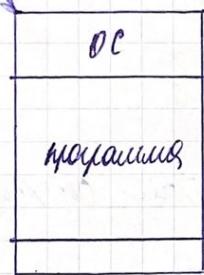
дисковая система
(локальная часть) область свопинга/нейминга

Примето говорить о распределении памяти:

- ① свое
- ② несвое

Свое распределение

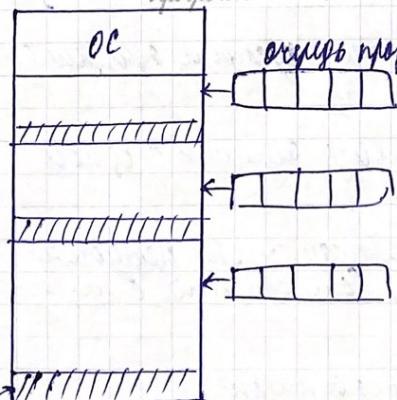
- 1) В одноданных системах; программа загружается целиком в оперативную память, возвращаясь от начала до конца - однократное свое распределение
- 2) Рассмотрение между задачами, так. целиком в оперативной памяти - свое распределение разделами.



В DOS'е программы
загружались на секторах,
точка по параметрам, но идет свое распределение
осуществляется:

- все ресурсы в распределении одной программы
- программы находятся в памяти
- программы возвращаются от начала до конца

2a)



• программы хранятся в одном разделе, от начала до конца

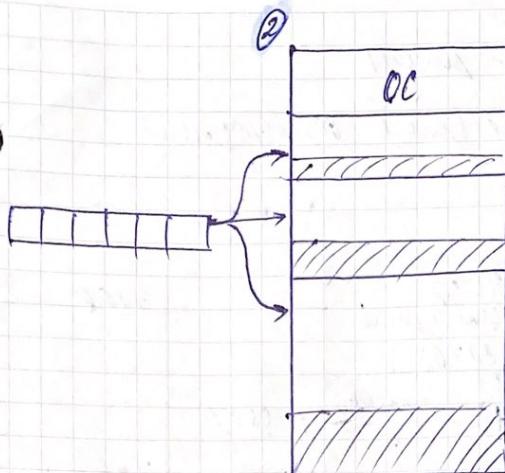
- присв. разные разделы
- на основе структуры

! выполняет неиспользованное адресное пр-во

Часто бывало: слишком большое
передышка между задачами

меньш.
передышка
пр-во

② Распределение разделами, присв. разделами
разделенное пр-во.



Разпределение памяти (стат.)
один раз

Размер разделов определяется до начала работы программы
(статически)

- динамическое выделение разделов - динамическое распределение
 - до начала работы память не распределяется



Как искать задачи для помещения в свободные ячейки?

↓
Большой подыскущий список задач
по размеру
(самые большие)
(самые маленькие)

- ↓
Самый маленький
• в разделе останется
остаток памяти для
еще одного процесса

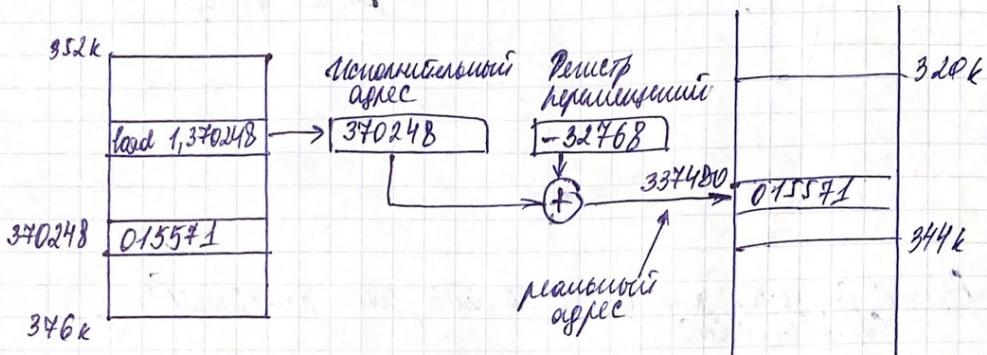
→ Задача: отслеживать свобод. пам. при-бо

Ограничение памяти- выравнивание блоков пам. при-бо., в кот. ничего не будет загружено.



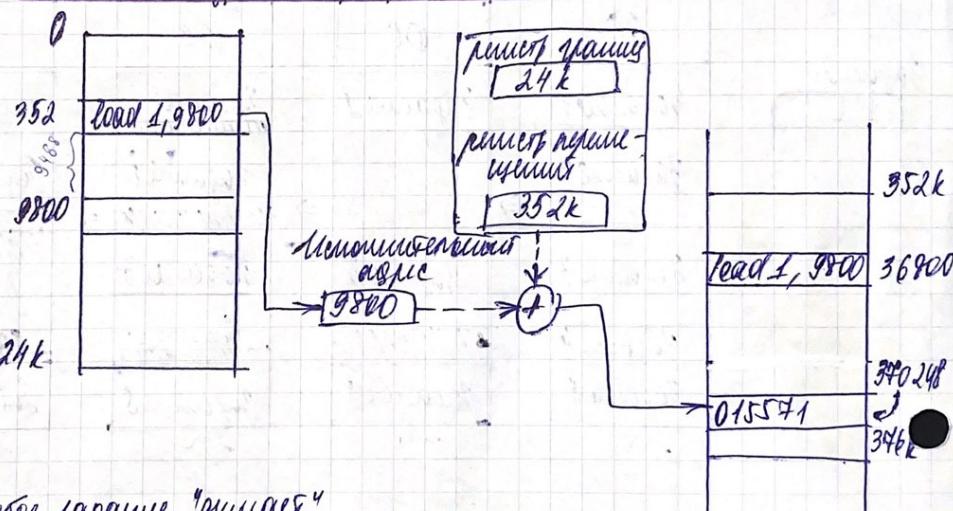
Переадреска? \Rightarrow Потеря всей работы

Решение: Переопределение разделов с целью обхода ограничений памяти.



Начальный адрес страницы практически не связан с именем памяти

\Rightarrow локальное адресное пространство!



8) Новое значение "зумится",
что это называется с 0
адресом

То есть получилось
смещение в программах.

тогда формируется такое переопределение разделов (переключа-
тель)

- 1) сразу после обеих к-тих разделов
- 2) если не найден раздел нужного размера

Несвязное распределение

1) Статистическое распределение

- задание делится на части одинакового размера - ограничено?

? аудио

Решение - огранич., на которое разделяется физ. память.

Все огранич. задания запускаются в определенное время

2) Семантическое распределение

- логическое распределение заданий на части - семантич. (кода, памяти, ...)

! Вопрос: надо ли хранить в памяти все адресное пр-во задания? (логарифм?)

Ответ: нет.

! В памяти физически находятся только коды, и которых выполняется процессор.

Фиртуальная память.

- память, объем которой превышает объем физ. адресного пр-ва.

...

При смене управ. физ. памятью



ограничениями
по запросу



семантическими
по запросу



семантическими,
пом. на огранич.
по запросу

"по запросу" - огранич. по запросу, когда процесс обращается к физ. адресам пр-ва.

1) Статистическое распределение (ограничения по запросу)

Фиртуальное адресное пр-во - адресное пр-во, динамически

составленное

? аудио

До того как страница попадет в память, она должна быть на диске.

Таблица карт памяти

0	ОС
1	ОС
2	процесс 2, стр. 0
3	десктоп
4	процесс 2, стр. 1
5	процесс 1, стр. 0
6	процесс 1, стр. 1
7	десктоп
8	процесс
...	

Карта диска

0	agree пакет
1	
2	

(последний адрес)
Последний адрес
таблицы страниц

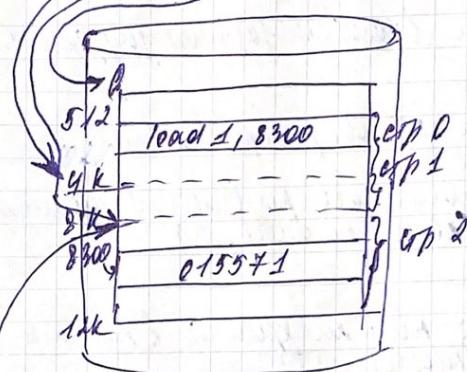


Таблица страниц процесса 2

0	2	1
1	4	1
2	0	

кодер
стр.
страницы
предыдущие



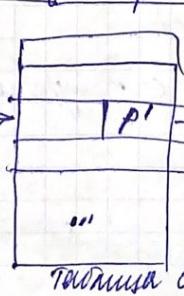
агрес

Способ пред- и посл. агреса к определенному

I приеме отображение
посл. начального адр. в ст.

Базовый адр

• таблица
страниц - 6 ин.нам



определенный адрес

- на конкретный сектор
- но нес. раз.
- несколько предотвращений

таблица страниц

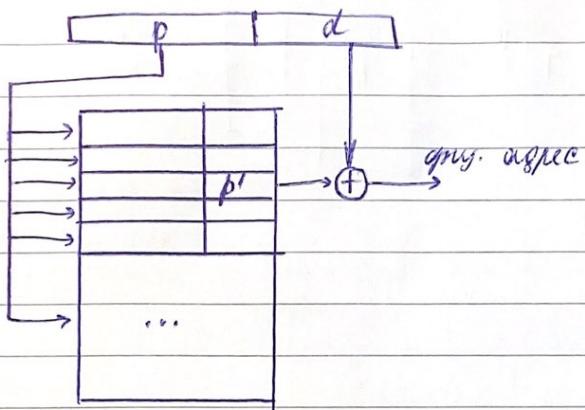
документ № 24.11.23. А.

Быстроукачивание agree
agree страницы
существует

Быстроукачивание
agree страницы access. over. → access. influence over.
также

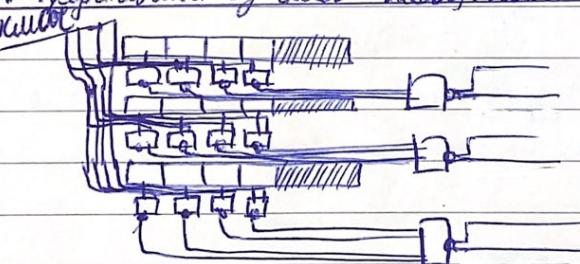
II Ассоциативное кеширование

- приближается к access. номеру → непрерывное



- различие в access. и
block-size для 1 блок

- что представляется в виде маск. номера?



- пред. регистрирует
работающих кем-то
ан-ан.
- (если коприш кеша)
- добавить кем-то сори-
нений
- если access. номер
единого результата есть
в кеше.

III Ассоциативно-множественное кеширование

множество маск. agree
таблица страниц

беспрограммное agree

- access. кеш (Way) - agrees
qmy. mask (qmask) есть
номер. выбираемое

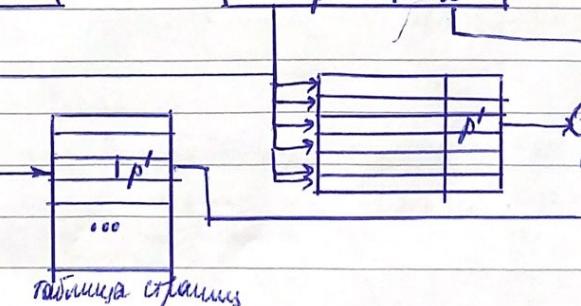


таблица страниц

Intel: быстрый TLB (translation look-aside buffer)

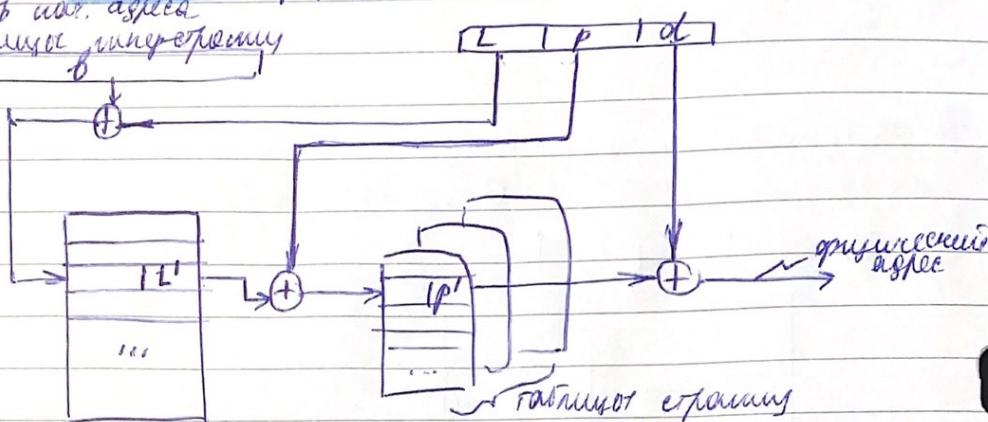
Схема страницного кнр-я памяти с межстраницами

- Чем больше кнр, тем больше страниц (от. большего кол-ва)
- Но! Минимум процесс не обращается одновременно ко всем своим страницам.

⇒ IBM 370: ^{однопроцессорная машина} схема с межстраницами:

лентяя адр. адрес

головка межстраницы



- Головка межстраницы формируетadr-е вadr. об. ве ядра
- в памяти различают пок-еи текущую страницу страниц
- + п.б. адрес. кнр (страница + межстраница)

Минусы: пам. расходов как в фрейм-еи боятся стать высокие

Замещение страниц (page replacement)

① вытеснение случайной страницы

Рис. память имеет фикс. размер. Появляется ситуация, когда все опр. страницы (стрейм) распределены, но процесс, нужен страница и кнр-я, требует страницу ⇒ надо вытеснить некоторую стр. ("вытеснить") из диска (замещение - replacement) и на ее место поместить ("затеснить") другую страницу

① Затеснение случайной

- восстан.-е первое попавшееся стр.
- + характеризуется малыми маш. расходами
- + ?
- и.о. восстановления часто не-стабильно из-за замены стр.

② FIFO

- Дубли вспомогательных страниц, каждая в свою очередь имеет
- и.о. реализован: с помощью стека страниц, пустой указ.
- + не и.о. восп. только что загр. стр.
- и.о. восстановления часто не-стабильны стр.
- ← значение FIFO; увеличение объема пам. ^{максимум} и.о. приводит к уменьшению страниц

Рассмотрим пример (аноним)

	1	2	3	4	5	6	7	8	9	10	11	12
	4	3	2	1	4	3	5	4	3	2	1	5
M=3	4+3+2+	1+4+4+	3+5+5	5+5+2+	4+1+1							
частота сб. нумер F	+ + +	+ + +	+ + +	+ + +	+ +							

$$f = \frac{9}{12} = 45\%$$

увеличение числа физ. страниц на 45%.

	1	2	3	4	5	6	7	8	9	10	11	12
	4	3	2	1	4	3	5	4	3	2	1	5
M=3	4+3+2+	1+4+4+	3+5+5	5+5+2+	4+1+1							
F	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +

$$f = \frac{10}{12} \approx 83\% - \text{увеличение памяти}$$

③ Least Recently Used

- боязнь сбрасывать, а ког. значение бывш. не будет оставаться

сбрасыват
множок

- сбрасывает только
6 хвостов

когда загружал

бранил
множок

- бывш. адреса
запоминаются

запоминаются

запоминаются!

Пример

	1	2	3	4	5	6	7	8	9	10	11	12
	4	3	2	1	4	3	5	4	3	2	1	5
M=3	4+3+2+	1+4+4+	3+5+5	5+5+2+	4+1+1							
F	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +

$$f = \frac{10}{12} = 75\%$$

увелич. размер M

	1	2	3	4	5	6	7	8	9	10	11	12
	4	3	2	1	4	3	5	4	3	2	1	5
M=4	4+3+2+	1+4+4+	3+5+5	5+5+2+	4+1+1							
F	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +	+ + +

$$f = \frac{8}{12} = \frac{2}{3} \approx 67\% - \text{уменьшение}$$

Число наклонений:

[Если обрамление было к странице, первое бывш. адреса, изменяющее обрамление будет к этой же странице (загрузка)]

* как часто это обрамление к страницам? Уже хранится в пам., но не в. пам.

④ Least Frequency Used

- контролирует частоту обрамления к странице
- только что загр. страницы могут не считаться "небывш." частоту -
 - они загружают на нов. загрузку

⑤ Not Used Recently

- каждой стр. ассоциируется число обращений.

Он времена, от времени сопровождал и при обновлении
число в 1-ую (и при зачтывке).
В результате стр. еще называемые среди них, у кот. фр.бр.=0

	0	4	9
1	1		
2	5		
3	7	1	
4	2	0	

указатель
установлен

Пр. Там же зачтывки 5^{го} стр. в 1^м обращении
число установило значение 0 и не засчитано

- Также важно для модификаторов:

	обращение	модификаторы
1	0	0
2	0	?
3	1	0
4	1	0

• наиболее популярный
не модиф.-нуль, т.к. можно
запись конфликтов
(на деле несет смысл
адреса)

Запомните управление бит. поможет избежать ошибок!

В современных машинах - 4 бит. габарит строчин.

Давно это? [еще упомянутое многозадачность, т.е.
если в памяти много страниц, то можно
использовать их одновременно]

Виртуальная память убирает.

Теория рабочего множества (working set)

ПЗУ исследовано:

производительность vs ограничение памяти

?

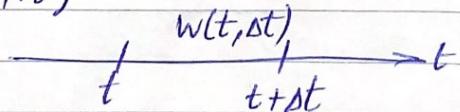
Максимальное время, сколько времени отводят на выполнение программы

⇒ рабочая память

- максим. количество страниц, которые программа использует одновременно

1968г. демонстрирует в крат. док. метод приведения страниц, и кот. отображается страницами за время от

$w(t, \Delta t)$



Размер рабочего множества - макс. от 0 до Δt

Если $\Delta t = T$ размер рабочего множества стремится к макс. размеру, но при этом это страницы, между которыми нет зон сопр.

Демонстрирует - если процесс не отдаёт память
(т.е. или программа требует анализа памяти ...)

Практический вопрос:

Программа должна иметь возможность загружать в память
рабочее место страниц - также либо страницу, к которой

Если программы не имеют такой возможности: thrashing

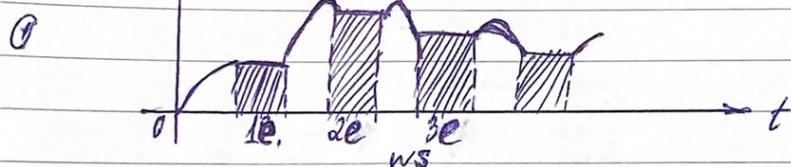
01.12.23. Лекция 14.

Примеч - подкачивание данных в память все страницы.

Багаж логист. элементов - восполнение пропусков .

⇒ Объем пам. ОС изменяется.

как это
отлично

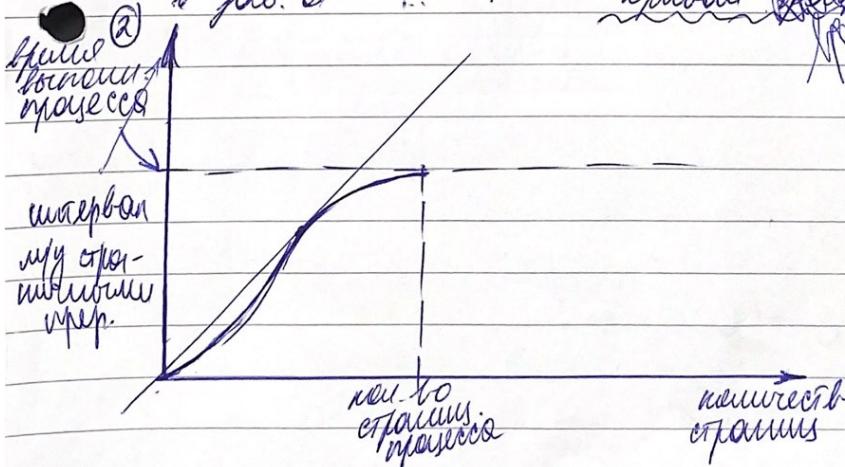


Риск подтверждается появление "переполнения процесса")

Все работает, подг. памяти раб. места, - появляется.

Процесс демонстрирует перенос между пропущенными

и раб. местами? - кривая ~~затраты~~ памяти.





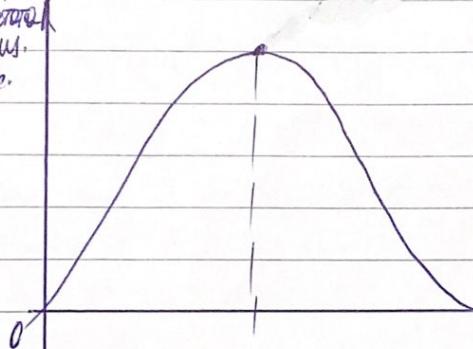
(3)

результат спадания

- с постепенным снижением ст. числа new-распр., т.к. на спадающие штормы не хватает новых компаний (или меньше ст., т.к. меньше при-т) - максимум зат-в!
- затухание - падает, т.к. результат спадающихся к результату прошлых штормов \Rightarrow меньше превращений.

(4) расчет
стаби.
из ст.

$$\text{результат ст.} = 1024 \text{ млн.} / 4096 \text{ б.}$$



дополнительная компания из ст. зат. не передает упр. из
другую ст.

Приведут ли это к конц. уп., а не к им. спаду?

(макс. кол-во)

1. концентрация приводит к нарушению норм. форм-ам изв.: кон-бо спадания

2. контроль за числом ст. прпп.

- линейный рост \rightarrow система работает некорректно \Rightarrow квадратичное \Rightarrow при формулировке уравнения, чтобы привести это ограничить все раз. штормы.

• "бупорок" - в к-то момент в пакетах нов. стартуют ст. прпп.

форм. по принципу LRU (ампераж)

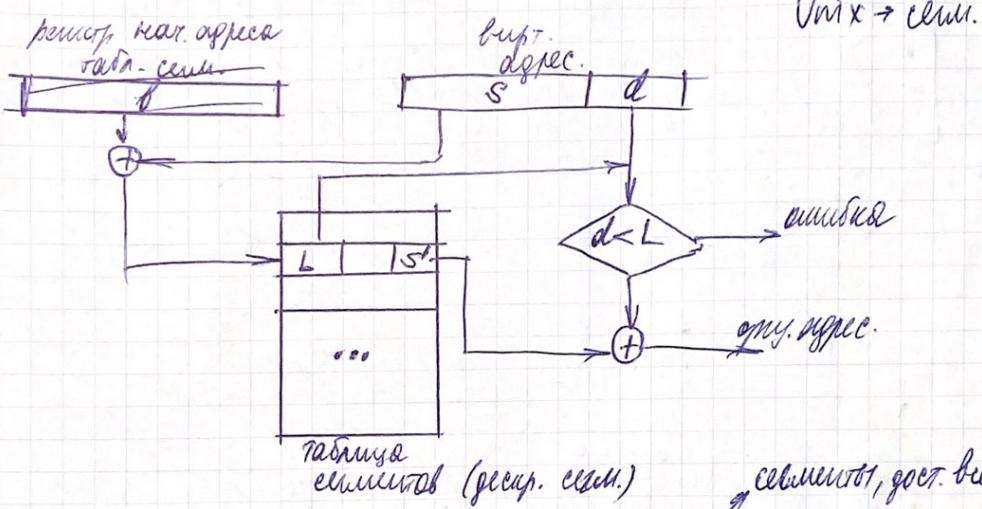
Проблемы кашевого и си. страниц

- Воринг и тех же стр. физ. писк. неч. множеств.
- один физ. физич. ф. исх. под-ко стр., другие - в их группах
- ⇒ контроль страниц ? аудио

При упр. памяти стр. по физич. физич. не стр., но проблем нет

② Сегментация по запросу

Размер сегмента определяется разрывами проф. кода (лок. дефекты)



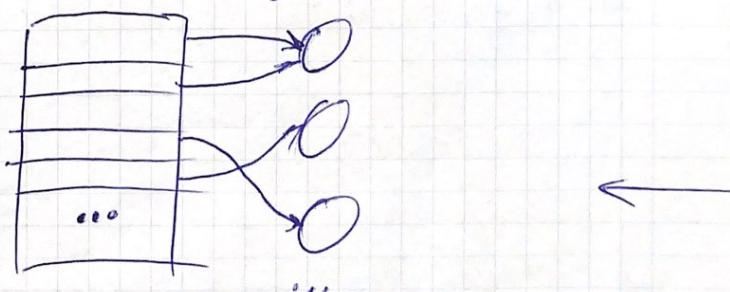
Intel → сегм. - проф. не бывает мон. табл. физич. (лок. сегм. АЛ проф.)
→ стр.

long - стр.
запом. 200. (сегм.)

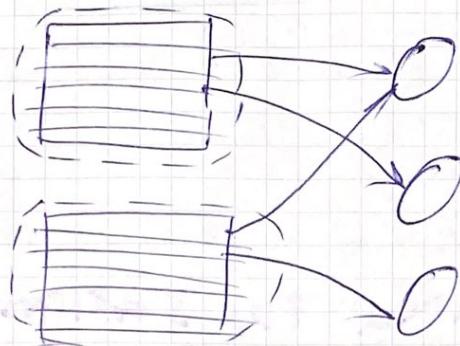
Бесшовая организация таблиц сегментов.

① Единичные разрывы

- единственный или сегментов - изолированные



2) локальные габариты

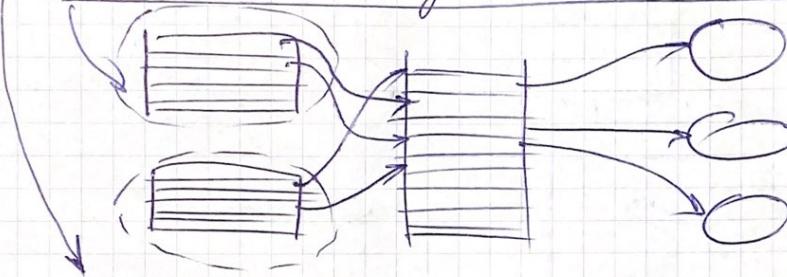


- консольные пр. имеют лок. габариты
- оруж. сенсоры имеют лок. габариты (как лок. габариты)

3) локальные габариты + масштабы

~~Лок. габариты~~

- лг. сенсор сенсора имеет его лок. б. габаритов
- лок. б. сенсора соп. сенсора имеет лок. габаритов

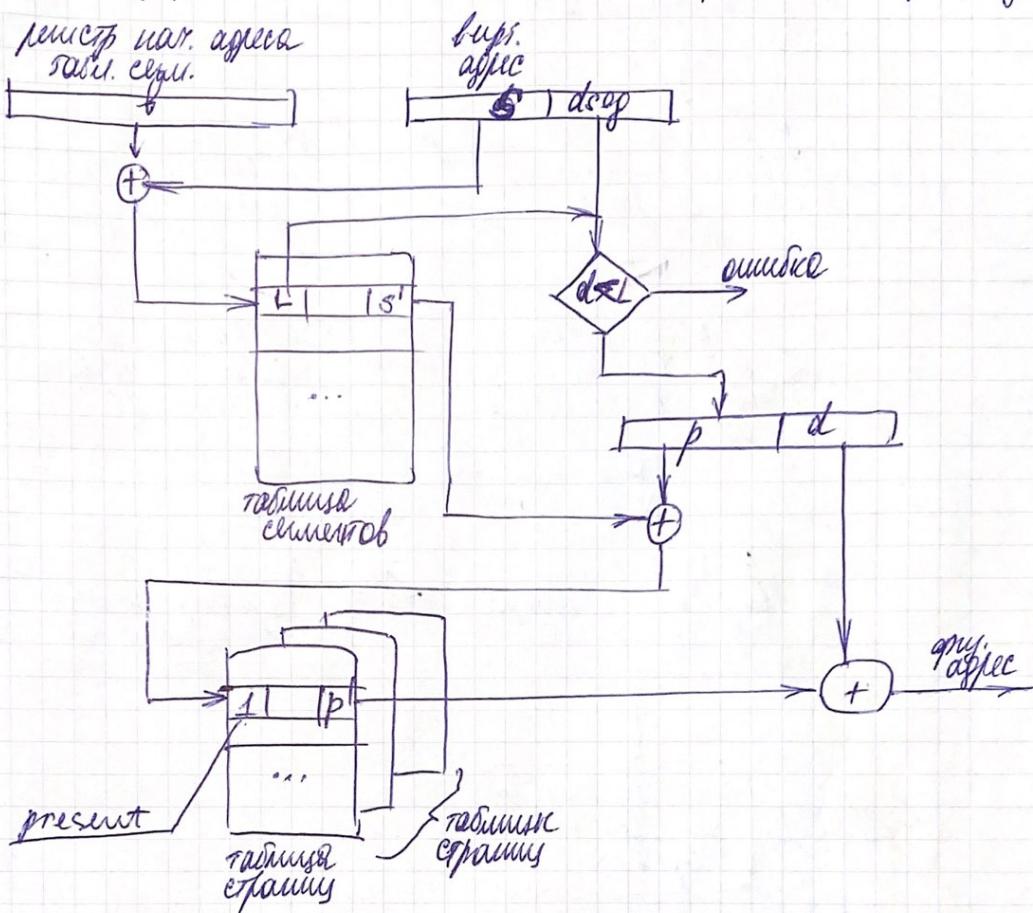


- мас. габ. сенс. сенс. ору. шинки
- лок. габ. — АП от. небу.
— лок. б. соп. сенсора не для мас. габаритов

Проблемы управления движением сенс. по заморозке.

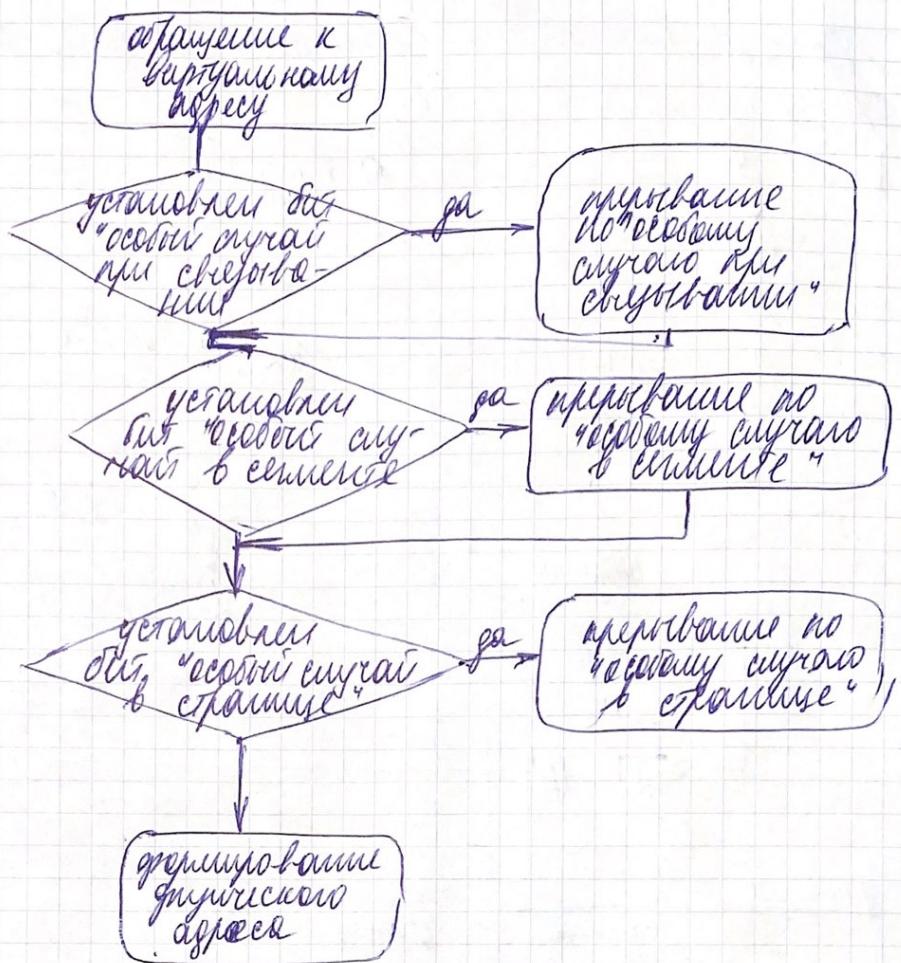
- ложные разрывы сенсоров \Rightarrow ложные разрывы движений? ...
- \circ обратные, вторые разрывы $\xrightarrow{\text{секунд. засечка}} \xrightarrow{\text{секунд. засечка}} \xrightarrow{\text{второй разр.}}$
- swapping (перестановка сенс. \Rightarrow перестановка сенс.)
- оппозиционные (ср. — скорость органы.)

Схема упр. памяти симметричн., ноз. на огранич., но фикс.



- MULTICS
- исключное управление
-

динамическое управление



1) "особый случай при свертывании"

- если к сем. не относится
или это не юр. лицо-декларатор
- если юр-декларатор первое
декл.

2) "особый случай при в семействе"

- Э декл. семейства, но не Э страны

3) "особый случай в стране"

- если нет юр. лицом, неодн. или
иер. страны

возможен в обр. порядке, обратка-верху-вниз
Возможно отр. является заранее - если страну

Чис.
ко гаджетам

Аналогичное представление

↓
управление макетами

Платформа гаджетов - ... : относительно простых.

Основные части кого - упр. внешними устройствами
Модули - драйверы.

|| Платформа управляет всеми б-ми с гаджетами. ||

Как платформа работает с внешними устройствами?

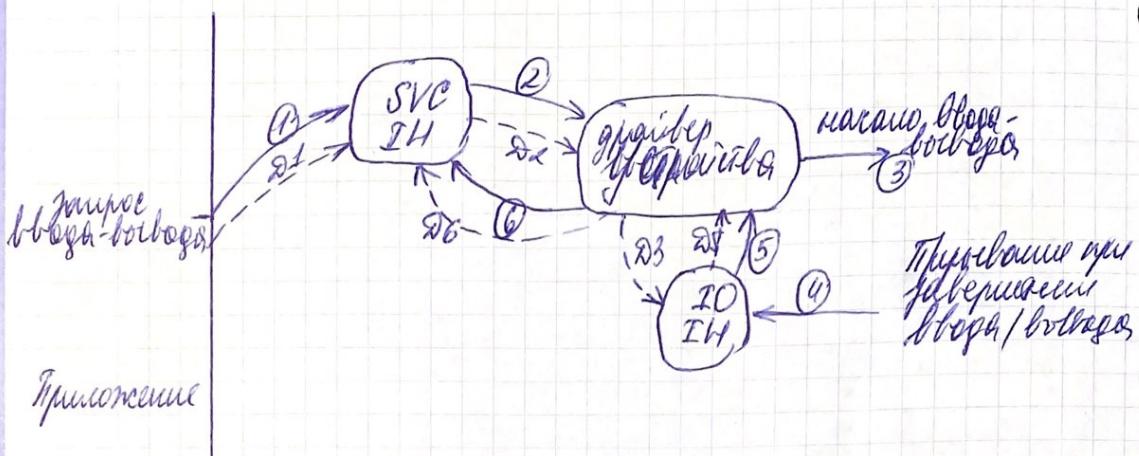
[Чтоу I - функциональная "запись представления
кого - вебера".]

• содержит все данные гаджета

Supervisor - PC в execusion mode

IM - interrupt controller

"DN" - hardware
---> hardware



В итоге нет логиче "последовательного бара-бара",
многоголосое бара-бара

Различительный бара-бара - много уровней, обычно
это миниатюрные устройства.

struct usb_driver -
└ функция входа драйвера

В результате работы драйвера драйвер команды контролируются бара-бара

Многоголосый бара-бара, контроллер устройства

→ Появляется бара-бара контроллер устройства, который управляет всеми, управляющими ядром бара-бара, который подает на контроллер ядра.

(ЭХ-шильда спасительства)

Основная задача ядра ядро - ядро - основное назначение ядра ядра ядра в будущем ядро.

"загрузка" этих данных - основное назначение.

Callback функции - функции драйвера, задача которых - передать данные приложению.

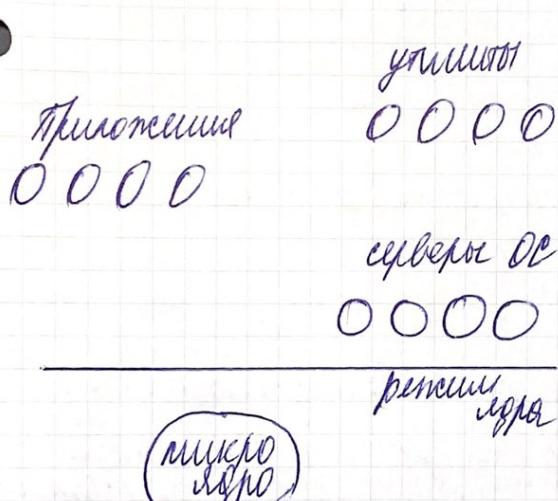
1) Многопоточное ядро - набор модулей.
Уменьшение многопоточного ядра \rightarrow перекомпилируем.

~~Модули~~

- Несколько модули ядра } первичное назначение
- Многозадачное ядро (Linux) } ядро не его назначение
(Windows)

2) Микроядро.

- первое ОР - Mach (K-M. University)
- идея микроядр. система - модуль клиент-сервер.

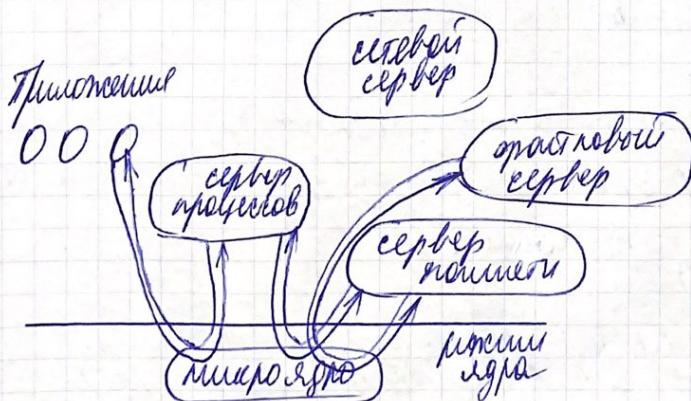


По принципу первых
машинок:

верхние уровни упр-я
программы, подпрограммы,
функции и т.д. вспом-
гательные ядра и вспом-
гательные программы

Основное ядро
(ОС - ядро)

ядра ОС:
исполнение общей
координирующей и ко-
нфигурирующей функций
и сервера-
ми ОС и для единого
серверами

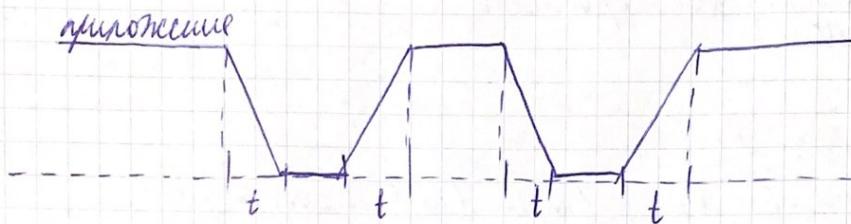


В результате ест. багажа
занимается ядро
серверы падают.

это приводит к тому
что есть проблемы с
координацией

- В результате минимума будут возникать зас. блок. запросов
при передаче сообщений. в то время между собой связанных
также присутствуют запросы на передачу контента.

- В монолитном ядре: 2 перекр. контекста приводят к ярку.
- В микроядре: 4 перекр. контекста (как и в монолитном):



[Соловьев-Русиновъ с. 25? §^o упражнение]

"Windows - 2000 не авт."

: сервисы или потоки, передачи файлов, ... - "микро",
API и т.д. - в новой реальности

(такие ОС никогда не отдают под контекст задачи без переключения ядра)

▲ ОС реального времени

Основные архитектуры ОС реального времени

- 1) монолит (одн. ядро, в ког. процес. более близко...)
- 2) ядро - единица декомпозиции (авт. менедж. ресурсов).
- 3) объекты памяти - структура данных, кот.
- 4) каналы - связь в соединении

5) соединение

* каналов - данные авт. организует

занимают ≥ 1 строку (\Rightarrow авт. физ. память тоже)

искусств. управл. фирм. памятью

(стр. авт.)

ядро \downarrow отличает о серверу памяти в пол.

регистре (в отл. от монолитного ядра)

способность авт. управл. список подключений, соединяется в ядре; процесс может иметь первых многих типов; процесс может иметь первоступенчатую ядро. Офиц. софт. в один из принципиальных портов (сварчиваемый)

22. 12. 23. December 17

Система обихода науки - источник, репрезентирующий интересы
круг задач.

Установление пределов эксплуатации и обработка земель.

¶ РЕ Інженер обирає структурні схеми під мікропр. опт-ле.

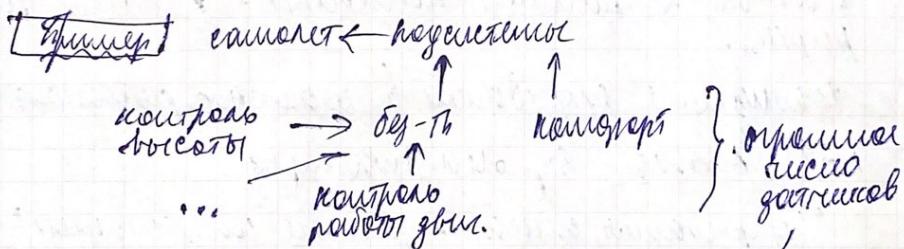
[Общественное право бывшего]

- Микроархитектурные арх-ре - это самое реальное
время.

Приоритетное значение упр-е вспомогательных процессов
но эти к системе (автоматическая, управление отладкой и т.д.,
"умный дом"); имеется ввиду управляемого упр-я и т.п.

Оп. [posit 1] Решающее время в ОС - способность ОС определять предстоящие задачи срока в опр. моментах времени.

ан. стимулирующие DC, занятые за первое время)
стимулирующие DC, занятые за первое время)



секунд. или. временн
период замка на вход
вок. системе (Боргхейд);

Былое
былое первое
былое первое

Большое число ~~имеют~~ ^{имеют} оружия

- многочлен, система уравнений, задача оптимизации
 - одна переменная?

~~Tipper:~~ QNX Neutrino RTOS v.6.0

неч: микроязык, формено в отв. процессов языковых
грац. иск., отв. языка в сенс - язык-система

Все процессы основы происх. потому что они сочтены.

QNX предоставляет расписание трех типов:

1) алгоритм FIFO

2) алг. RR

3) sporadic scheduling ("иррегулярное" - неупр. "вспыхивающее склонение")
↑
spor.

Допустимо в QNX: ^{богаты} на основе sporadic scheduling
использовать свой способ мониторинга.

- гарантированный бюджет
- период выполнения
- бюджет

Бюджет (бюд.) - время, за которое может
выполниться задача. приоритетный принцип, что он выше
помощи.

Период выполнения - период времени, в течение которого
может пройти задача.

Ограничено макс. число одновременных назначений.

Управление вытеснением \Rightarrow

\Rightarrow когда каждое FD (исполнительное) процессор может
использовать память одновременно - для них назначается
предприятие.

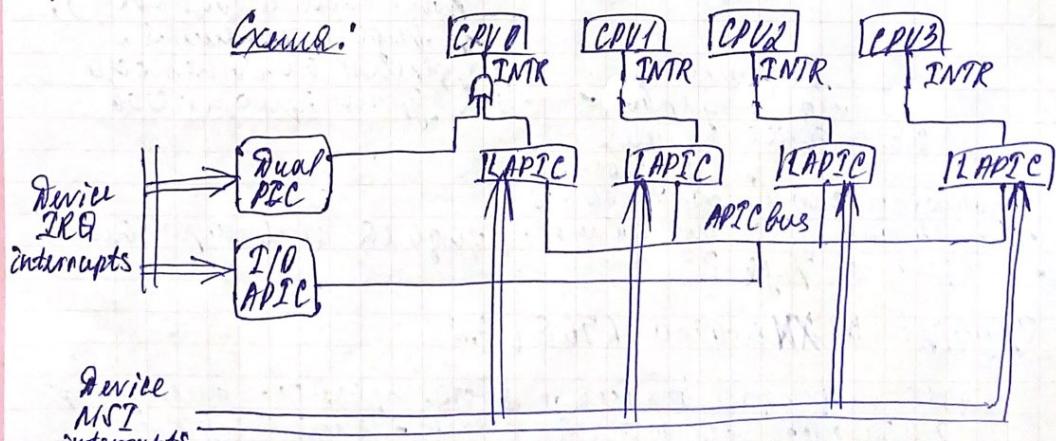
Рекомендации по управлению в системах реального времени.

MSI - Message Signaled Interrupts.

"прерывания, синхронизирующие соединения"

- отменяют некоторые прерывания (такие как прерывания
ввода/вывода)

Example:



Device
MSI
interrupts

APIC - Advanced PIC
LAPIC - Local APIC

? PIC - programmed interrupt
controller ?

Как видно из схемы, организовать "старое" прер-и
x8086 можно если процессор CPU.
может

ISA - норма памяти, определяет
PCI - ? более быстрое, для нее лучше организовать
в себе

MSI: Позволяет не-и вид. памяти прер-и устройство напрямую
запрашивать прер-и, заменяющие прер-и. создаётся по спр.
акесс в памяти.

Для записи такого сооб. используется механизм
так называемый (bus mastering). Он находит свою
пам. но это-то и DMA (direct memory access)
для записи устройства может идти до RAM напрям.

В синих PCI-х и PCIe драйверы не исп. прер-и.
исп. прер-и, исп-ся только MSI.

Схема - абстракция

Т.к. MSI не-и в другие записи в память, то не-и,
хотя под. все условия прохождения!

1. Retry
2. Master_Abort
3. Target_Abort

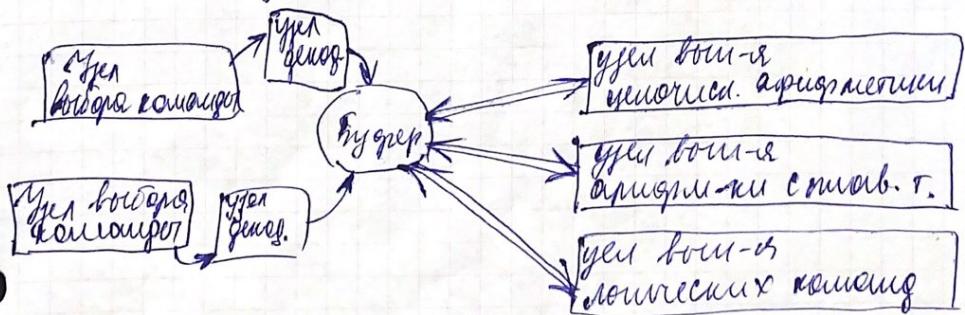
+ любых поддержка в IPQ.

MSI и.о. замаскированы.
или все под. в группе устройств.

Собр. организует эти. универсализации
одной команды проходит из 3 стадии ~~установки~~:

1. подготовка
2. демодификация
3. готов к выполнению

Готов - суперск. процессор:



Очевидно что должна заработать, что регулирует более ранние
события в том, что было для них после. более позд.

Минимум

1, 1, 3, 5, 8 - born.

1, 6, 4, 9, 10 - еще нет

PC - адрес 9, 10 или 11
свободна

- 1) Старт команду укaz. на команду, где нет free
команды помимо команды
- 2) на одна команда, на кор. укaz. PC, не born.
- 3) кор. команды, на кор. укaz. PC, известно

Чтобы увидеть, связ. с данной командами, где
отмечено где обращение пред.

... ожид