БИЛЕТ № 13

1) Основные структурные единицы ОС и их характеристика.

Если программа меняет информацию, с которой она работает (помимо своей собственной), то это *обрабатывающая* программа (компиляторы, загрузчики, редакторы связей), если не меняет, – то *управляющая*.

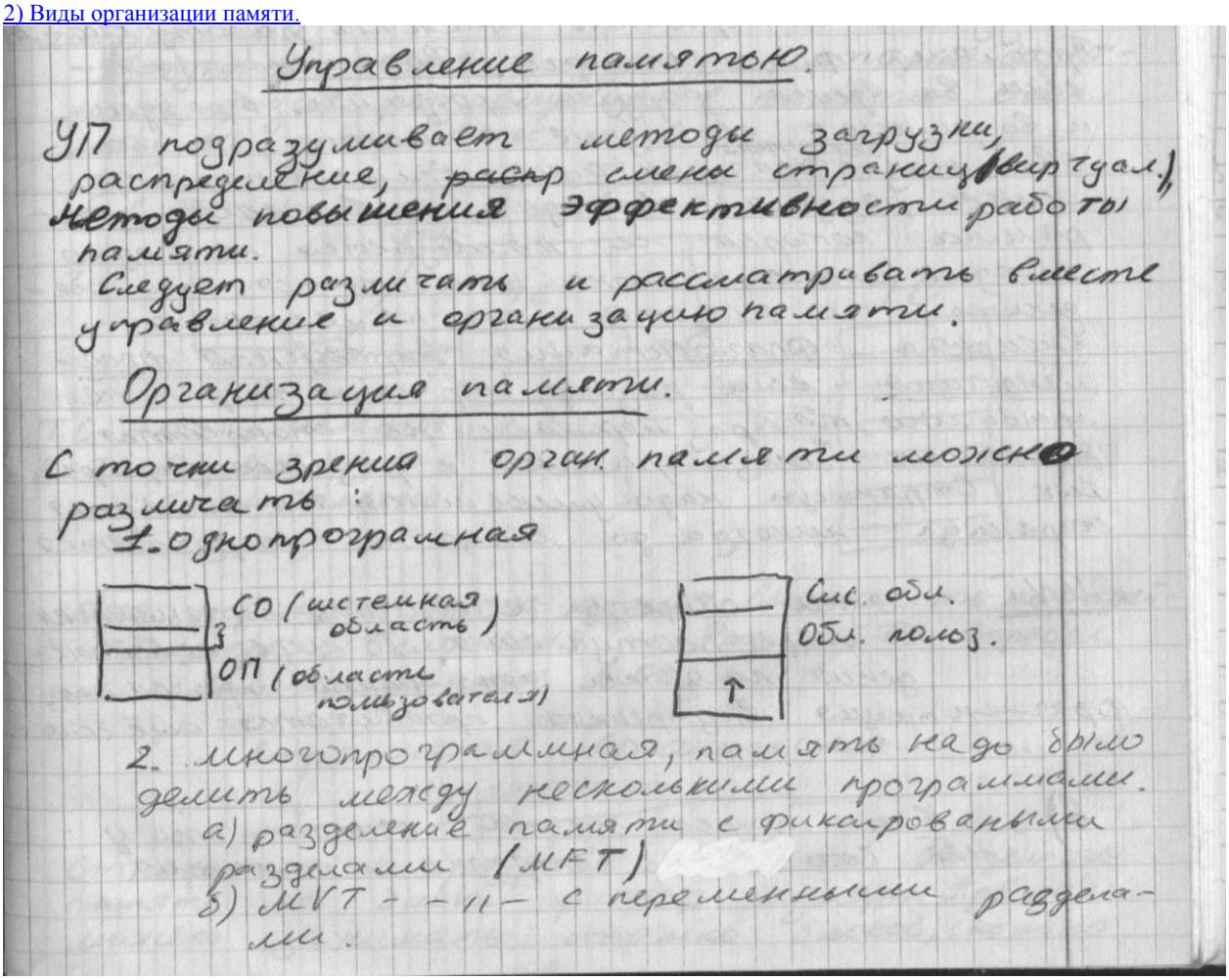
Управляющие программы выполняют следующие действия:

1. **Управление** – на уровне ядра, в режиме супервизора осуществляется:
   - отслеживание входа и выхода задания в системе;
   - отслеживание выполнения собственной задачи и управление ее выполнением;
   - обеспечение работы систем управления файлами, памятью, внешними устройствами.
2. **Управление задачами.** Следит за выполнением задачи на ресурсах системы
3. **Система управления файлами.**
4. **Управление памятью.**
   - программное (обеспечивает эффективное функционирование памяти в соответствии с ее организацией; программа решает свои задачи на уровне процесса, то есть память выделяется процессу);
   - микропрограммное (программа запрашивает у админа, какие разделы необходимо выполнять; у каждого раздела – своя очередь);
5. **Управление внешними устройствами.**
6. Управление заданиями отслеживает прохождение задания от входа до выхода

Современные ОС имеют сложную структуру, каждый элемент которой выполняет определенные функции по управлению компьютером.

- Управление файловой системой. Процесс работы компьютера сводится к обмену файлами между устройствами. В ОС имеются программные модули, управляющие файловой системой.
- Командный процессор. В состав ОС входит специальная программа – командный процессор, – которая запрашивает у пользователя команды (запуск программы, копирование, удаление и т.д.) и выполняет их.
- Драйверы устройств. В состав ОС входят драйверы устройств, специальные программы, которые обеспечивают управление работой устройств и согласование информационного обмена с другими устройствами, позволяют производить настройку параметров устройств. Каждому устройству соответствует свой драйвер. Программы (драйверы) для поддержки наиболее распространенных устройств входят в Windows, а для остальных устройств поставляются вместе с этими устройствами или контроллерами. Можно самому установить или переустановить драйвер.
- Графический интерфейс. Для упрощения работы пользователя в состав ОС входят программные модули, создающие графический пользовательский интерфейс. Пользователь может вводить команды с помощью мыши, а не с клавиатуры.
- Сервисные программы. (утилиты). Позволяют обслуживать диски (проверять, сжимать, дефрагментировать), выполнять операции с файлами (архивировать и т.д.), работать в сетях и т.д.
- Справочная система. Позволяет оперативно получить необходимую информацию о функционировании ОС.

2) Виды организации памяти.



Типичные форматы СУС-м с управлением памятью

Страничная

№стр	Смещ

Управл. блок	№ кадра

Сегментная

№ сегм	Смещение

Упр блок	Длина	адрес табл. страниц / адрес сегмента

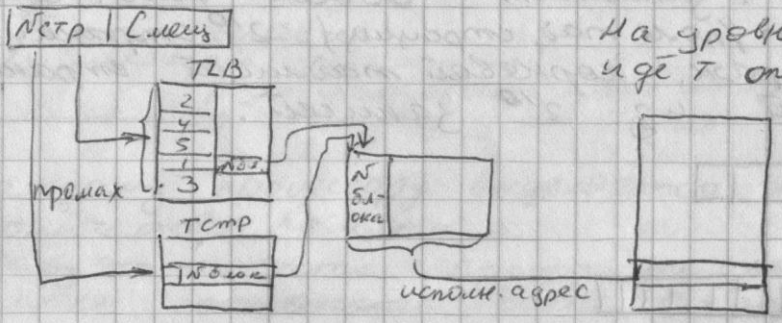
Сегм - стр

№сегм	№стр	смещен.

Упр. блок	Длина	Адр. сегмента

Упр. блок	№ кадра

В машинах преобразов. адресса идёт на аппаратном уровне. В машинах исп. двух уровневые кэши. Один из кэшей явл. ассоциативная память в которой реализ. TLB-буфер (буфер быстрого преобр. адреса). Ассоциативная память ведёт обращение не по адресу, а по содержимому.



На уровне регистров идёт опер. сложения.

Если идёт промах, тогда ищется нужн. страница в таблице страниц.
Если нету в Табл. страниц и TLB, то идет страничное прерывание и подкач в ОП измен. содержимое TLB и Табл. страниц. Если нету в Табл. страниц, то образ из TLB.

в) -11- с перемещаемыми разделами
г) страничная организация (чистая; по запросу;)
д) сегментная организация.
е) виртуальная организация памяти.

Виртуальная подразумивает страничную, но не на оборот,

MFT вся память во время инсталяции делится на разделы, фиксированого размера. Для каждого раздела формировалась своя очередь и каждая имела приоритет класса (15 разделов - 15 классов). Проблемы:
- неудачное планирование снижала работу сис-мы
- Проблема фрагментации. Фрагментация - если во время загрузки программ в раздел и возникают свободные области, то это явл. наз. внутренняя фрагментация. Если в памяти имые разделы свободны, а в очереди программа которая не помещается ни в одно свободное место, но она поместится если объединить их вместе - она поместится - внешняя фрагментация. Страничная фрагментация - сам размер страницы выбран не правильно, то размер таблицы становится большим сопоставимой с размером программы. Страницы надо уменьшать, т.к. последняя страница никогда до конца не загружается.

MVT одна очередь и память выделяется по мере поступления. По мере освобождения памяти исполненой программы, фрагментация внутренняя проявляется ещё больше.

в) Как только сис-ма считает, что у неё что-то не в порядке с памятью.

Возникает проблема с настройкой адресных констант. Возможна применение схемы либо с текущей либо с глобальной перадресацией. Локальная, глобальная настройка констант. Возможно применение схемы текущей или глобальной перадресации.

Страничная организация памяти — это такой способ управления памятью, при котором пространство адресов памяти разбивается на блоки фиксированной длины. При страничной организации все про¬странство оперативной памяти физически делится на отрезки рав¬ной длины, называемые физическими страницами. Программы и данные делятся на

называемые страницами содержательные части того же размера, что и физические страницы оперативной памяти . Таким образом, одну страницу информации можно загрузить в одну физическую страницу. Для каждой текущей задачи создается таблица страниц.
Диспетчер памяти для каждой страницы формирует соответствующий дескриптор. Дескриптор содержит так называемый бит присутствия. Если он = 1, это означает, что данная страница сейчас размещена в ОП. Если он = 0, то страница расположена во внешней памяти. Защита страничной памяти основана на контроле уровня доступа к каждой странице.
для обеспечения целостности, секретности и взаимной изоляции выполняющихся программ должны быть предусмотрены различные режимы доступа к страницам, которые реализуются с помощью специальных индикаторов доступа в элементах таблиц страниц. Следствием такого использования является значительный рост таблиц страниц каждого пользователя. Одно из решений проблемы сокращения длины таблиц основано на введении многоуровневой организации таблиц. Частным случаем многоуровневой организации таблиц является сегментация при страничной организации памяти.
Основное достоинство страничного способа распределения памяти - минимально возможная фрагментация (эффективное распределение памяти). Недостатки: 1) потери памяти на размещение таблиц страниц
2) потери процессорного времени на обработку таблиц страниц (диспетчер памяти).
3) Программы разбиваются на страницы случайно, без учета логических взаимосвязей, имеющихся в коде Þ межстраничные переходы осуществляются чаще, чем межсегментные + трудности в организации разделения программных модулей между выполняющими процесами

Сегментная и сегментно-страничная организация памяти
Существуют две другие схемы организации управления памятью: **сегментная и сегментно-страничная**. Сегменты, в отличие от страниц, могут иметь переменный размер. Идея сегментации изложена во введении. При сегментной организации виртуальный адрес является двумерным как для программиста, так и для операционной системы, и состоит из двух полей – номера сегмента и смещения внутри сегмента. Подчеркнем, что в отличие от страничной организации, где линейный адрес преобразован в двумерный операционной системой для удобства отображения, здесь двумерность адреса является следствием представления пользователя о процессе не в виде линейного массива байтов, а как набор сегментов переменного размера (данные, код, стек...).
Программисты, пишущие на языках низкого уровня, должны иметь представление о сегментной организации, явным образом меняя значения сегментных регистров (это хорошо видно по текстам программ, написанных на Ассемблере).
Логическое адресное пространство – набор сегментов. Каждый сегмент имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия). В отличие от страничной схемы, где пользователь задает только один адрес, который разбивается на номер страницы и смещение прозрачным для программиста образом, в сегментной схеме пользователь специфицирует каждый адрес двумя величинами: именем сегмента и смещением.
Каждый сегмент – линейная последовательность адресов, начинающаяся с 0. Максимальный размер сегмента определяется разрядностью процессора (при 32-разрядной адресации это 232 байт или 4 Гбайт). Размер сегмента может меняться динамически (например, сегмент стека). В элементе таблицы сегментов помимо физического адреса начала сегмента обычно содержится и длина сегмента. Если размер смещения в виртуальном адресе выходит за пределы размера сегмента, возникает исключительная ситуация.
Логический адрес – упорядоченная пара v=(s,d), номер сегмента и смещение внутри сегмента.
В системах, где сегменты поддерживаются аппаратно, эти параметры обычно хранятся в таблице дескрипторов сегментов, а программа обращается к этим дескрипторам по номерам-селекторам. При этом в контекст каждого процесса входит набор сегментных регистров, содержащих селекторы текущих сегментов кода, стека, данных и т. д. и определяющих, какие сегменты будут использоваться при разных видах обращений к памяти. Это позволяет процессору уже на аппаратном уровне определять допустимость обращений к памяти, упрощая реализацию защиты информации от повреждения и несанкционированного доступа.
Аппаратная поддержка сегментов распространена мало (главным образом на процессорах Intel). В большинстве ОС сегментация реализуется на уровне, не зависящем от аппаратуры.
Хранить в памяти сегменты большого размера целиком так же неудобно, как и хранить процесс непрерывным блоком. Напрашивается идея разбиения сегментов на страницы. При сегментно-страничной организации памяти происходит двухуровневая трансляция виртуального адреса в физический. В этом случае логический адрес состоит из трех полей: номера сегмента логической памяти, номера страницы внутри сегмента и смещения внутри страницы. Соответственно, используются две таблицы отображения – таблица сегментов, связывающая номер сегмента с таблицей страниц, и отдельная таблица страниц для каждого сегмента.
Сегментно-страничная и страничная организация памяти позволяет легко организовать совместное использование одних и тех же данных и программного кода разными задачами. Для этого различные логические блоки памяти разных процессов отображают в один и тот же блок физической памяти, где размещается разделяемый фрагмент кода или данных.

3) Методы повышения эффективности носителей данных.

Доп.инфа..
Производительность файловых систем:
Во многих файловых системах применяют оптимизацию, для увеличения производительности:

1. **Кэширование.** Для минимизации количества обращений к диску применяют блочный или буферный кэш. Кэш – набор блоков, логически принадлежащих диску, но хранящиеся в оперативке. Происходит захват всех запросов чтения к диску и проверке требуемой информации в КЭШе. Если блок присутствует – запрос чтения происходит без обращения к диску. Иначе блок считывается с диска в кэш, оттуда копируется по нужному адресу памяти.
2. **Опережающее чтение блока.** Метод заключается в попытке получить блоки блоки диска в кэш прежде чем они потребуются. Применимо только для последовательного считывания файлов.
3. **Снижение времени перемещения блока головок.** Блоки с высокой вероятностью доступа помещаются близко друг к другу, желательно на один цилиндр.
4. **Перемещение i-узлов в середину диска.** Среднее расстояние перемещения блока головок уменьшается вдвое.
5. **Разбиение диска на группы цилиндров, каждая со своими i-узлами, блоками и списками свободных блоков.** При создании нового файла может быть выбран любой i-узел, но предпринимается попытка найти блок в той же группе цилиндров, что и i-узел.

Оптимизация доступа к диску

Методы оптимизации:

1. **FCFS** (первый пришел – первый обслужился);
2. **SSTF** (поиск с найменшим временем**);**
3. **SCAN** (сканирующий алгоритм, принцип действия заключается в том, что головка двигается внутри и по дороге обслуживает все запросы);
4. **C-SCAN** (запросы обрабатываются от внешней дорожки к внутренней до конца, потом идет перескок на внешнюю дорожку, и так до конца запроса);
5. **N-STEP-SCAN** (точно также как SCAN, но после начала движения запросы ставятся в отдельную очередь);
6. **SLTF** (с наименьшим временем ожидания первого; обрабатываются запросы под головкой в рамках всего цилиндра);

Схема Эшенбаха пытается обработать только одну дорожку.

Приемы повышения эффективности работы с внешними носителями

Существует 2 подхода к повышению эффективности работы с внешними носителями:

1.Оптимизация физического обращения к диску (перемещения головок,…)

2.Оптимизация логического обращения (RAID)

В первом случае алгоритмы делятся на 2 вида.

Когда текущая дорожка неизвестна используются следующие алгоритмы:

1.Random access

2.FIFO – самый простой – операционная система ловит запросы к диску

3.LIFO

Если текущая дорожка известна то используются:

1.SSTF(стратегия наименьшего времени обслуживания) , еще до ввода-вывода определяется текущее перемещение головки. Из очереди заявок выбирается заявка с ближайшим перемещением.

Недостаток: при частом обращении некоторые заявки долго ожидают.

2.SCAN – головка перемещается в одном направлении, по пути обслуживая заявки. Все вновь поступившие заявки включаются в обслуживание. Т.е происходит обратный проход.

3.C-SCAN – SCAN без обратного прохода.

4.NstepSCAN. Используются 2 очереди. В начале сканируется очередь, она фиксируется и новые заявки поступают в дополнительную очередь. При скачке назад очереди перезаписываются.

5. По приоритету.

При втором подходе(RAID).RAID-массив – избыточный массив независимых дисков ,рассматриваемых ОС как один диск. Обеспечивается либо параллельный , либо независимый доступ.

RAID0 – чисто параллельный доступ .Избыточность не используется.

Disk1	Disk2	Disk3	Disk4
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

RAID1 – «зеркало».

Disk1	Disk2	Disk3	Disk4
0	1	0	1
2	3	2	3
…	…	…	…

RAID2 – второй диск хранит код Хемминга(исправляет 1 ошибку и обнаруживает 2) первого диска.

Disk1	Disk2	Disk3	Disk4
0	F(0)		

RAID3 - Используется четность с чередующимися битами.

RAID4 – Используется технология независимых дисков.

RAID5 – Защита чередующимися блоками.

Disk1	Disk2	Disk3	Disk4	Disk5
0	1	2	3	F(0-3)
4	5	6	F(4-7)	7
8	9	F(8-11)	10	11
12	13	14	F(12-15)	15

RAID6

Disk1	Disk2	Disk3	Disk4	Disk5	Disk6
0	1	2	3	F(0-2)	F(0-3)
4	5	6	F(4-6)	F(4-7)	7

4) Модель непротиворечивости - слабая непротиворечивости

Репликация данных

Важным вопросом для распределенных систем является репликация данных. Данные обычно реплицируются для повышения надежности и увеличения производительности. Одна из основных проблем при этом — сохранение непротиворечивости реплик. Говоря менее формально, это означает, что если в одну из копий вносятся изменения, то нам необходимо обеспечить, чтобы эти изменения были внесены и в другие копии, иначе реплики больше не будут одинаковыми.

Вопросы, связанные с репликацией

1. как должны расходиться обновления по копиям
2. поддержка непротиворечивости
3. проблема кеширования

Основные проблемы репликации

Первая проблема - обеспечение взаимного исключения при доступе к объекту. Существует два решения:

- объект защищает себя сам (к примеру, доступ к объекту совершается при помощи синхронизированных методов в Java - каждому клиенту будет создано по потоку и виртуальная Java-машина не даст воспользоваться методом доступа к ресурсу обоим потокам в один момент времени)
- система защищает объект (ОС сервера создает некоторый адаптер, который делает доступным объект, только одному клиенту в один момент времени)

Вторая проблема - поддержка непротиворечивости реплик. Здесь снова выделяют два пути решения:

- решение основанное на осведомленности объекта о том, что он был реплицирован. По сути, обязанность поддерживать непротиворечивость реплик перекладывается на сам объект. То есть в системе нет централизованного механизма поддержки непротиворечивости репликаций. Преимущество в том, что объект может реализовывать некоторые специфичные для него методы поддержки непротиворечивости.
- обязанность поддержки непротиворечивости накладывается на систему управления распределенной системой. Это упрощает создание реплицируемых объектов, но если для поддержки непротиворечивости нужны некоторые специфичные для объекта методы, это создает трудности.

Модели непротиворечивости, ориентированные на данные

По традиции непротиворечивость всегда обсуждается в контексте операций чтения и записи над совместно используемыми данными, доступными в распределенной памяти (разделяемой) или в файловой системе (распределенной).

Модель непротиворечивости (consistency model), по существу, представляет собой контракт между процессами и хранилищем данных. Он гласит, что если процессы согласны соблюдать некоторые правила, хранилище соглашается работать правильно. То есть, если процесс соблюдает некоторые правила, он может быть уверен, что данные которые он читает являются актуальными. Чем сложнее правила - тем сложнее их соблюдать процессу, но тем с большей вероятностью прочитанные данные действительно являются актуальными.

- **слабая**

Рассмотрим случай, когда процесс внутри критической области заносит записи в реплицируемую базу данных. Хотя другие процессы даже не предполагают работать с новыми записями до выхода этого процесса из критической области, система управления базой данных не имеет никаких средств, чтобы узнать, находится процесс в критической области или нет, и может распространить изменения на все копии базы данных.

Наилучшим решением в этом случае было бы позволить процессу покинуть критическую область и затем убедиться, что окончательные результаты разосланы туда, куда нужно, и не обращать внимания на то, разосланы всем копиям промежуточные результаты или нет. Это можно сделать, введя так называемую переменную синхронизации (synchronization variable). Переменная синхронизации S имеет только одну ассоциированную с ней операцию, synchronize(S), которая синхронизирует все локальные копии хранилища данных. Напомним, что процесс P осуществляет операции только с локальной копией хранилища. В ходе синхронизации хранилища данных все локальные операции записи процесса P распространяются на остальные копии, а операции записи других процессов — на копию данных P.

Используя переменные синхронизации для частичного определения непротиворечивости, мы приходим к так называемой слабой непротиворечивости (weak consistency).

Основная идея: Если процесс активно работает с репликой, ему, скорее всего, не нужно что бы промежуточные результаты его работы попадали в общий доступ. Поэтому он сначала проводит несколько операций с репликой (блок операций), а после этого вызывает процедуру синхронизации, которая делает актуальной все реплики.

5) Контекст процесса, состав и назначение.

Контекст процесса включает в себя содержимое адресного пространства задачи, выделенного процессу, а также содержимое относящихся к процессу аппаратных регистров и структур данных ядра. С формальной точки зрения, контекст процесса объединяет в себе пользовательский контекст, регистровый контекст и системный контекст (*). Пользовательский контекст состоит из команд и данных процесса, стека задачи и содержимого совместно используемого пространства памяти в виртуальных адресах процесса. Те части виртуального адресного пространства процесса, которые периодически отсутствуют в оперативной памяти вследствие выгрузки или замещения страниц, также включаются в пользовательский контекст.

Регистровый контекст состоит из следующих компонент:

- Счетчика команд, указывающего адрес следующей команды, которую будет выполнять центральный процессор; этот адрес является виртуальным адресом внутри пространства ядра или пространства задачи.
- Регистра состояния процессора (PS), который указывает аппаратный статус машины по отношению к процессу. Регистр PS, например, обычно содержит подполя, которые указывают, является ли результат последних вычислений нулевым, положительным или отрицательным, переполнен ли регистр с установкой бита переноса и т.д. Операции, влияющие на установку регистра PS, выполняются для отдельного процесса, потому-то в регистре PS и содержится аппаратный статус машины по отношению к процессу. В других имеющих важное значение подполях регистра PS указывается текущий уровень прерывания процессора, а также текущий и предыдущий режимы выполнения процесса (режим ядра/задачи). По значению подполя текущего режима выполнения процесса устанавливается, может ли процесс выполнять привилегированные команды и обращаться к адресному пространству ядра.

- Указателя вершины стека, в котором содержится адрес следующего элемента стека ядра или стека задачи, в соответствии с режимом выполнения процесса. В зависимости от архитектуры машины указатель вершины стека показывает на следующий свободный элемент стека или на последний используемый элемент. От архитектуры машины также зависит направление увеличения стека (к старшим или младшим адресам), но для нас сейчас эти вопросы несущественны.
- Регистров общего назначения, в которых содержится информация, сгенерированная процессом во время его выполнения. Чтобы облегчить последующие объяснения, выделим среди них два регистра - регистр 0 и регистр 1 - для дополнительного использования при передаче информации между процессами и ядром.

Системный контекст процесса имеет "статическую часть" (первые три элемента в нижеследующем списке) и "динамическую часть" (последние два элемента). На протяжении всего времени выполнения процесс постоянно располагает одной статической частью системного контекста, но может иметь переменное число динамических частей. Динамическую часть системного контекста можно представить в виде стека, элементами которого являются контекстные уровни, которые помещаются в стек ядром или выталкиваются из стека при наступлении различных событий. Системный контекст включает в себя следующие компоненты:

- Запись в таблице процессов, описывающая состояние процесса (раздел 6.1) и содержащая различную управляющую информацию, к которой ядро всегда может обратиться.
- Часть адресного пространства задачи, выделенная процессу, где хранится управляющая информация о процессе, доступная только в контексте процесса. Общие управляющие параметры, такие как приоритет процесса, хранятся в таблице процессов, поскольку обращение к ним должно производиться за пределами контекста процесса.
- Записи частной таблицы областей процесса, общие таблицы областей и таблицы страниц, необходимые для преобразования виртуальных адресов в физические, в связи с чем в них описываются области команд, данных, стека и другие области, принадлежащие процессу. Если несколько процессов совместно используют общие области, эти области входят составной частью в контекст каждого процесса, поскольку каждый процесс работает с этими областями независимо от других процессов. В задачи управления памятью входит идентификация участков виртуального адресного пространства процесса, не являющихся резидентными в памяти.
- Стек ядра, в котором хранятся записи процедур ядра, если процесс выполняется в режиме ядра. Несмотря на то, что все процессы пользуются одними и теми же программами ядра, каждый из них имеет свою собственную копию стека ядра для хранения индивидуальных обращений к функциям ядра. Пусть, например, один процесс вызывает функцию creat и приостанавливается в ожидании назначения нового индекса, а другой процесс вызывает функцию read и приостанавливается в ожидании завершения передачи данных с диска в память. Оба процесса обращаются к функциям ядра и у каждого из них имеется в наличии отдельный стек, в котором хранится последовательность выполненных обращений. Ядро должно иметь возможность восстанавливать содержимое стека ядра и положение указателя вершины стека для того, чтобы возобновлять выполнение процесса в режиме ядра. В различных системах стек ядра часто располагается в пространстве процесса, однако этот стек является логически-независимым и, таким образом, может помещаться в самостоятельной области памяти. Когда процесс выполняется в режиме задачи, соответствующий ему стек ядра пуст.
- Динамическая часть системного контекста процесса, состоящая из нескольких уровней и имеющая вид стека, который освобождается от элементов в порядке, обратном порядку их поступления. На каждом уровне системного контекста содержится информация, необходимая для восстановления предыдущего уровня и включающая в себя регистровый контекст предыдущего уровня.

(*) Используемые в данном разделе термины "пользовательский контекст" (user-level context), "регистровый контекст" (register context), "системный контекст" (system-level context) и "контекстные уровни" (context layers) введены автором.

Ядро помещает контекстный уровень в стек при возникновении прерывания, при обращении к системной функции или при переключении контекста процесса. Контекстный уровень выталкивается из стека после завершения обработки прерывания, при возврате процесса в режим задачи после выполнения системной функции, или при переключении контекста. Таким образом, переключение контекста влечет за собой как помещение контекстного уровня в стек, так и извлечение уровня из стека: ядро помещает в стек контекстный уровень старого процесса, а извлекает из стека контекстный уровень нового процесса. Информация, необходимая для восстановления текущего контекстного уровня, хранится в записи таблицы процессов.

На Рисунке 6.8 изображены компоненты контекста процесса. Слева на рисунке изображена статическая часть контекста. В нее входят: пользовательский контекст, состоящий из программ процесса (машинных инструкций), данных, стека и разделяемой памяти (если она имеется), а также статическая часть системного контекста, состоящая из записи таблицы процессов, пространства процесса и записей частной таблицы областей (информации, необходимой для трансляции виртуальных адресов пользовательского контекста). Справа на рисунке изображена динамическая часть контекста. Она имеет вид стека и включает в себя несколько элементов, хранящих регистровый контекст предыдущего уровня и стек ядра для текущего уровня. Нулевой контекстный уровень представляет собой пустой уровень, относящийся к пользовательскому контексту; увеличение стека здесь идет в адресном пространстве задачи, стек ядра недействителен. Стрелка, соединяющая между собой статическую часть системного контекста и верхний уровень динамической части контекста, означает то, что в таблице процессов хранится информация, позволяющая ядру восстанавливать текущий контекстный уровень процесса.

Процесс выполняется в рамках своего контекста или, если говорить более точно, в рамках своего текущего контекстного уровня. Количество контекстных уровней ограничивается числом поддерживаемых в машине уровней прерывания. Например, если в машине поддерживаются разные уровни прерываний для программ, терминалов, дисков, всех остальных периферийных устройств и таймера, то есть 5 уровней прерывания, то, следовательно, у процесса может быть не более 7 контекстных уровней: по одному на каждый уровень прерывания, 1 для системных функций и 1 для пользовательского контекста. 7 уровней будет достаточно, даже если прерывания будут поступать в "наихудшем" из возможных порядков, поскольку прерывание данного уровня блокируется (то есть его обработка откладывается центральным процессором) до тех пор, пока ядро не обработает все прерывания этого и более высоких уровней.

Несмотря на то, что ядро всегда исполняет контекст какого-нибудь процесса, логическая функция, которую ядро реализует в каждый момент, не всегда имеет отношение к данному процессу. Например, если возвращая данные, дисковое запоминающее устройство посылает прерывание, то прерывается выполнение текущего процесса и ядро обрабатывает прерывание на новом контекстном уровне этого процесса, даже если данные относятся к другому процессу. Программы

обработки прерываний обычно не обращаются к статическим составляющим контекста процесса и не видоизменяют их, так как эти части не связаны с прерываниями.

Большинство процессоров построены таким образом, что сигнал прерывания проверяется ими только после завершения выполнения текущей команды выполняющегося процесса. Однако, в системе могут существовать прерывания, для которых невозможно ожидание завершения очередной команды, например, если возникает ошибка в самой команде. Поэтому в системе могут использоваться одновременно несколько из перечисленных способов для обработки различных прерываний.

Обнаружив сигнал прерывания, процессор должен запомнить состояние прерванного процесса для того, чтобы продолжить его выполнение после обработки прерывания. После этого в процессор загружается новый процесс, выполняющий обработку прерывания. Эта процедура получила название **переключения контекста** или **переключения состояния**. Она является одной из важнейших в организации работы операционной системы, причем ее реализация требует поддержки и выполнения некоторых функций на аппаратном уровне.

---- --- -- ---- ---- Рассмотрим механизм передачи управления **программе обработки прерывания (IH)**. Как было сказано выше, ОС запоминает состояние прерванного процесса и передает управление IH. Эта операция называется **переключением контекста**. При реализации переключения используются **слова состояния программы (PSW)**, с помощью которых осуществляется управление порядком выполнения команд. В **PSW** содержится информация относительно состояния процесса, обеспечивающая продолжение прерванной программы на момент прерывания.