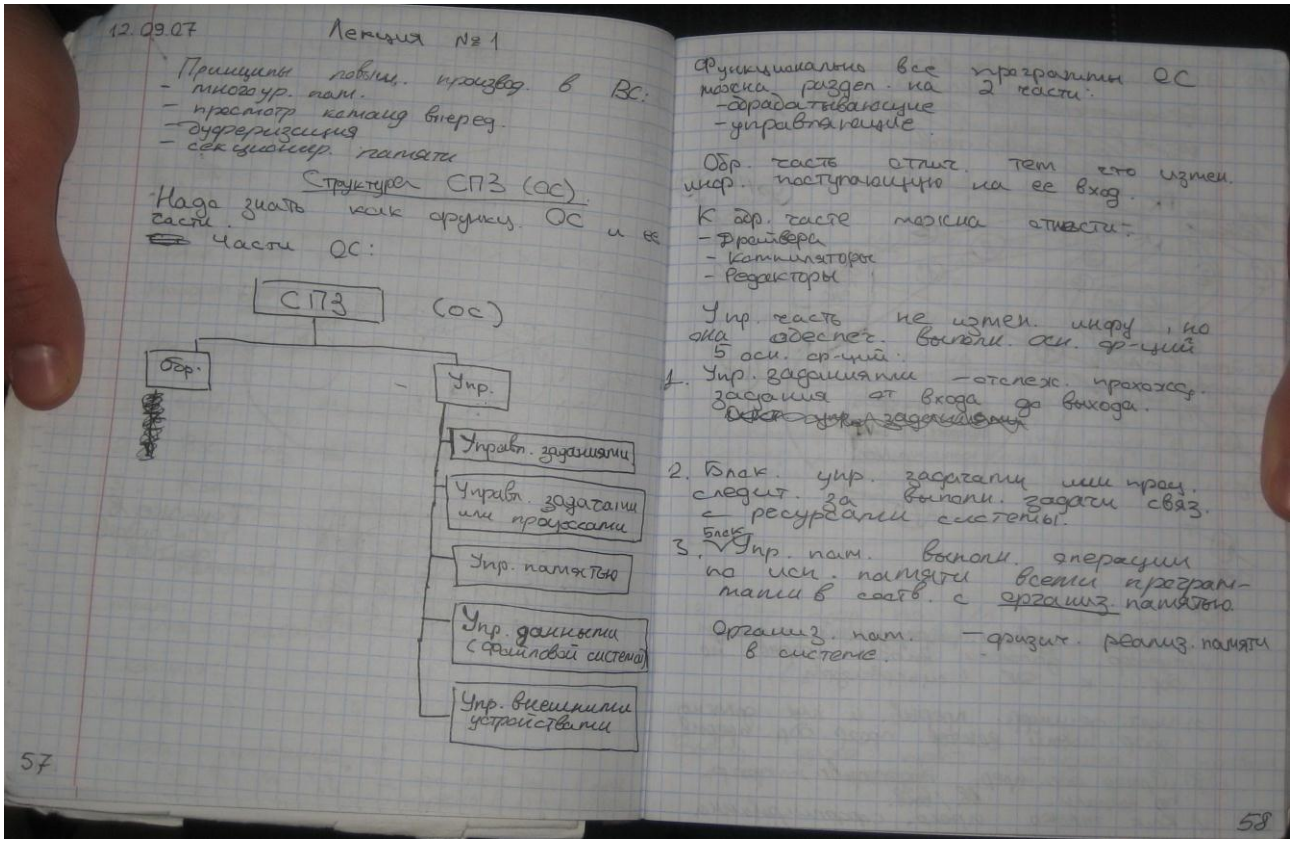

1.

2. Управление процессами, или Всѐ о PCB

Выполнение функций ОС, связанных с управлением процессами, осуществляется с помощью специальных структур данных, образующих окружение процесса, среду исполнения или образ процесса. Образ процесса состоит из двух частей: данных режима задачи и режима ядра. Образ процесса в режиме задачи состоит из сегмента кода программы, которая подчинена процессу, данных, стека, библиотек и других структур данных, к которым он может получить непосредственный доступ. Образ процесса в режиме ядра состоит из структур данных, недоступных процессу в режиме задачи, которые используются ядром для управления процессом. Оно содержит различную вспомогательную информацию, необходимую ядру во время работы процесса.

Каждому процессу в ядре операционной системы соответствует блок управления процессом (PCB – process control block). Вход в процесс (фиксация системой процесса) –это создание его блока управления (PCB), а выход из процесса – это его уничтожение, т. е. уничтожение его блока управления.

Таким образом, для каждой активизированной задачи система создает свой PCB, в котором, в сжатом виде, содержится используемая при управлении информация о процессе.

PCB – это системная структура данных, содержащая определѐнные сведения о процессе со следующими полями:

1. Идентификатор процесса (имя);
2. Идентификатор родительского процесса;
3. Текущее состояние процесса (выполнение, приостановлен, сон и т.д.);
4. Приоритет процесса;
5. Флаги, определяющие дополнительную информацию о состоянии процесса;

6. Список сигналов, ожидающих доставки;
7. Список областей памяти выделенной программе, подчиненной данному процессу;
8. Указатели на описание выделенных ему ресурсов;
9. Область сохранения регистров;
10. Права процесса (список разрешенных операций);

3.**Правила выбора размера страницы. Факторы, влияющие на выбор.**

Пусть средний размер процесса равен S байт, а страница – P байт. Кроме того, предположим, что запись для каждой страницы требует E байт. Тогда приблизительное количество страниц, необходимое для процесса, равно S/P, что займет S*E/P байт для таблицы страниц. Потеря памяти в последней странице процесса вследствие внутренней фрагментации равна P/2. Таким образом, общие накладные расжоды вследствие поддержки таблицы страниц и потери от внутренней фрагментации равны сумме этих двух составляющих:

расход = SE/P+P/2.

Первое слагаемое (размер таблицы страниц) увеличивается при уменьшении размера страниц. Второе слагаемое (внутренняя фрагментация) при увеличении размера страний возрастает. Оптимальный вариант должен находиться где-то посредине. Если взять первую производную по переменной P и приравнять ее к нулю, мы получим равенство:

- S*E/(P*P) + 1/2 = 0.

Из этого равенства мы можем получить формулу, дающую оптимальный размер страницы.В результате получим:

P = корень из (2*S*E).

При больших страницах может быть половина страницы пустой если в нее записан маленький фрагмент текста (внутренняя фрагментация). Если в оперативную память грузить такие страницы, то много ОП будет использоваться не эффективно.
С другой стороны, если программа будет слишком большой, то ей понадобится много маленьких страниц, что заберет много времени на их загрузку.

4.Таблица отображения файлов

Одним из вариантов предыдущего способа является хранение указателей не в дисковых блоках, а в индексной таблице в памяти, которая называется таблицей отображения файлов (FAT - file allocation table) (см. рис. 12.3). Этой схемы придерживаются многие ОС (MS-DOS, OS/2, MS Windows и др.)

По-прежнему существенно, что запись в директории содержит только ссылку на первый блок. Далее при помощи таблицы FAT можно локализовать блоки файла независимо от его размера. В тех строках таблицы, которые соответствуют последним блокам файлов, обычно записывается некоторое граничное значение, например EOF.

Главное достоинство данного подхода состоит в том, что по таблице отображения можно судить о физическом соседстве блоков, располагающихся на диске, и при выделении нового блока можно легко найти свободный блок диска, находящийся поблизости от других блоков данного файла. Минусом данной схемы может быть необходимость хранения в памяти этой довольно большой таблицы.

Номера блоков диска		
1		
2	10	
3	11	Начало файла F_2
4		
5	EOF	
6	2	Начало файла F_1
7	EOF	
8		
9		
10	7	
11	5	

Рис. 12.3. Метод связного списка с использованием таблицы в оперативной памяти

5. Назвать механизмы неявной реализации когерентности.

Проблема когерентности памяти для мультипроцессоров и устройств ввода/вывода имеет много аспектов. Обычно в малых мультипроцессорах используется аппаратный механизм, называемый протоколом, позволяющий решить эту проблему. Такие протоколы называются протоколами когерентности кэш-памяти. Существуют два класса таких протоколов:

Протоколы на основе справочника (directory based). Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы).

Протоколы наблюдения (snooping). Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.

Имеются две методики поддержания описанной выше когерентности. Один из методов

заключается в том, чтобы гарантировать, что процессор должен получить исключительные права доступа к элементу данных перед выполнением записи в этот элемент данных. Этот тип протоколов называется протоколом записи с аннулированием (write ivalidate protocol), поскольку при выполнении записи он аннулирует другие копии. Это наиболее часто используемый протокол как в схемах на основе справочников, так и в схемах наблюдения.

Исключительное право доступа гарантирует, что во время выполнения записи не существует никаких других копий элемента данных, в которые можно писать или из которых можно читать: все другие кэшированные копии элемента данных аннулированы. Чтобы увидеть, как такой протокол обеспечивает когерентность, рассмотрим операцию записи, вслед за которой следует операция чтения другим процессором. Поскольку запись требует исключительного права доступа, любая копия, поддерживаемая читающим процессором должна быть аннулирована (в соответствии с названием протокола). Таким образом, когда возникает операция чтения, произойдет промах кэш-памяти, который вынуждает выполнить выборку новой копии данных. Для выполнения операции записи мы можем потребовать, чтобы процессор имел достоверную (valid) копию данных в своей кэш-памяти прежде, чем выполнять в нее запись. Таким образом, если оба процессора попытаются записать в один и тот же элемент данных одновременно, один из них выиграет состязание у второго (мы вскоре увидим, как принять решение, кто из них выиграет) и вызывает аннулирование его копии. Другой процессор для завершения своей операции записи должен сначала получить новую копию данных, которая теперь уже должна содержать обновленное значение.

Альтернативой протоколу записи с аннулированием является обновление всех копий элемента данных в случае записи в этот элемент данных. Этот тип протокола называется протоколом записи с обновлением

(write update protocol) или протоколом записи с трансляцией (write broadcast protocol). Обычно в этом протоколе для снижения требований к полосе пропускания полезно отслеживать, является ли слово в кэш-памяти разделяемым объектом, или нет, а именно, содержится ли оно в других кэшах. Если нет, то нет никакой необходимости обновлять другой кэш или транслировать в него обновленные данные.

Теперь из конспекта:

Алгоритм MESI (в простонародии алгоритм с обратной записью) является одним из алгоритмов, обеспечивающих когерентность (согласованность) памяти.

Small intro:

Механизмы рализации когерентности могут быть явными и неявными для программиста. Все архитектуры делятся на 4 категории:

1. Явное размещение и явный доступ (все операции над данными явны с точки зрения размещения и доступа. Команды Send() и Receive())
2. Неявное размещение и неявный доступ (доступ к данным прозрачен для программиста). Т.е. оперирование данными происходит на уровне команд читать/писать без явного указания адресов.
3. Неявное размещение (как в страничной организации) явный доступ. Используется разделяемое множество страниц на ВУ. При запросе страницы система автоматически обеспечивает согласование.
4. Явное размещение с указанием разделяемых (? Или раздела) страниц и неявный доступ с помощью команд Load() и Store(). При этом используется технология MEMORY CHANNEL. Каждый компьютер имеет виртуальную память и разделяемые страницы, отображение этих страниц имеется во всех остальных компьютерах системы их удаление производится с использованием специальной команды. При этом используется специальная сетевая карта, обеспечивающая

взаимодействие память-память.

MESI - один из алгоритмов *неявной* реализации когерентности. Идея его такова: Память сосредоточена. Подключение к ней идет через шину. Все транзакции также реализуются только через шину, поэтому есть возможность их отслеживания. Каждая строка в *кэшах* имеет следующие состояния:

1. M - модифицирована (операция R/W разрешена только в этом модуле)
2. E - монопольно копирована (операция R/W разрешена во всех модулях)
3. S - множественно копирована (операция R/W разрешена во всех модулях, где есть копия страницы)
4. I - запрещена к использованию.

Периодически по шине отправляются циклы опроса состояния строк.