«Операционные системы»
Контрольная работа 2.

Задание 1. Страничная память.

Задание. Надо назначить каждому процессу количество страниц. Всего страниц 6 и их надо распределить между тремя процессами (колонка **Lehtede arv**). Если не имеется страницы, с которой надо производить операции, то её должна заместить другая. Rn – n чтений со страницы; Wn – n записей на страницу.

Цель. Найти наименьший cost = swap-in + swap-out. Вспомогательную таблицу можно использовать для моделирования процессов.

Решение. Итак, предположим, нам дан ряд: R1 W1 R2 W2 R3 R1 R2. Как видно, здесь имеются следующие сочетания: R1 W1 и R2 W2. Так вот, если будет идти такое сочетание Rn Wn, где n-число, то желательно при его окончании «завершить страницу» и начать новую, так как тогда получится только in на Rn, если же не заканчивать страницу, то получится вдобавок и out и cost таким образом увеличится. Для большей наглядности давайте рассмотрим этот ряд для разного числа страниц.

1 стр.	R1	W1	R2	W2	R3	R1	R2
	in	**out**	**in**	**out**	**in**	**in**	**in**

cost = 5+2 = 7

2 стр.	R1	W1					
	in						
		R2	W2	R3	R1	R2	
		in	**out**	**in**	**in**	**in**	

cost = 5+1 = 6

3 стр.	R1	W1					
	in						
		R2	W2				
		in					
			R3	R1	R2		
			in	**in**	**in**		

cost = 5

Во вспомогательной таблице надо также учитывать, чтобы на одной странице не было сразу несколько R (или W) одновременно.

А теперь приведу примеры нескольких заданий с контрольных (то что жирным шрифтом – то надо самому вписывать в пустые поля).

Пример 1.

Tellimused	Lehtede arv
R1 W1 R2 W2 R3 R1 R2	**3**
R5 R6 W6 R4 R5 R6 W6	**1**
R8 W8 R9 W9 R7 R8 R9	**2**

Тут существует несколько равносильных вариантов, я выбрал к примеру такой.

Заполняем вспомогательную таблицу:

3 стр	R1	W1					
	in						
		R2	W2				
		in					
			R3	R1	R2		
			in	**in**	**in**		
1 стр	R5	R6	W6	R4	R5	R6	W6
	in	**in**	**out**	**in**	**in**	**in**	
2 стр	R8	W8					
	in						
		R9	W9	R7	R8	R9	
		in	**out**	**in**	**in**	**in**	

Строки для in и out у препода в задании не даны, поэтому легче их надписывать ручкой на листе бумаги.

swap-in: **15** swap-out: **2** cost: **17**

Пример 2.

Tellimused	Lehtede arv
W1 R2 W2 R3 R1 R2 R3	**2**
R6 W6 R4 R5 R6 W6 W5	**2**
W8 R9 W9 R7 R8 R9 R7	**2**

Тут также существует несколько равносильных вариантов, я выбрал такой.

Заполняем вспомогательную таблицу:

2 стр	W1	R2	W2				
	in	**in**					
	R3		R1	R2	R3		
	in		**in**	**in**	**in**		
2 стр	R6	W6					
	in						
		R4	R5	R6	W6	W5	
		in	**in**	**in**	**out**	**in**	
2 стр	W8	R9	W9				
	in	**in**					
	R7		R8	R9	R7		
	in		**in**	**in**	**in**		

swap-in: **17** swap-out: **1** cost: **18**

Задание 2. Запросы к жесткому диску.

Справочная информация: **Алгоритмы планирования запросов к жесткому диску**

Прежде чем приступить к непосредственному изложению самих алгоритмов, давайте вспомним внутреннее устройство жесткого диска и определим, какие параметры запросов мы можем использовать для планирования.

Строение жесткого диска и параметры планирования

Современный жесткий магнитный диск представляет собой набор круглых пластин, находящихся на одной оси и покрытых с одной или двух сторон специальным магнитным слоем (см. рис. 1). Около каждой рабочей поверхности каждой пластины расположены магнитные головки для чтения и записи информации. Эти головки присоединены к специальному рычагу, который может перемещать весь блок головок над поверхностями пластин как единое целое. Поверхности пластин разделены на концентрические кольца, внутри которых, собственно, и может храниться информация. Набор концентрических колец на всех пластинах для одного положения головок (т. е. все кольца, равноудаленные от оси) образует цилиндр. Каждое кольцо внутри цилиндра получило название дорожки (по одной или две дорожки на каждую пластину). Все дорожки делятся на равное число секторов. Количество дорожек, цилиндров и секторов может варьироваться от одного жесткого диска к другому в достаточно широких пределах. Как правило, сектор является минимальным объемом информации, которая может быть прочитана с диска за один раз.

При работе диска набор пластин вращается вокруг своей оси с высокой скоростью, подставляя по очереди под головки соответствующих дорожек все их сектора. Номер сектора, номер дорожки и номер цилиндра однозначно определяют положение данных на жестком диске и, наряду с типом совершаемой операции – чтение или запись, полностью характеризуют часть запроса, связанную с устройством, при обмене информацией в объеме одного сектора.

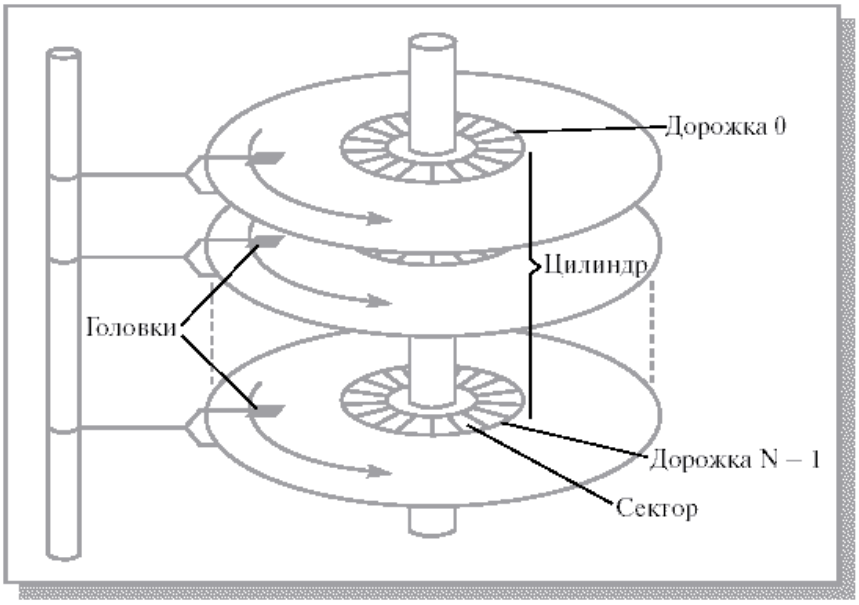

Рис. 1 Схема жесткого диска

При планировании использования жесткого диска естественным параметром планирования является время, которое потребуется для выполнения очередного запроса. Время, не-

обходимое для чтения или записи определенного сектора на определенной дорожке определенного цилиндра, можно разделить на две составляющие: время обмена информацией между магнитной головкой и компьютером, которое обычно не зависит от положения данных и определяется скоростью их передачи (transfer speed), и время, необходимое для позиционирования головки над заданным сектором, – время позиционирования (positioning time). Время позиционирования, в свою очередь, состоит из времени, необходимого для перемещения головок на нужный цилиндр, – времени поиска (seek time) и времени, которое требуется для того, чтобы нужный сектор довернулся под головку, – задержки на вращение (rotational latency). Времена поиска пропорциональны разнице между номерами цилиндров предыдущего и планируемого запросов, и их легко сравнивать. Задержка на вращение определяется довольно сложными соотношениями между номерами цилиндров и секторов предыдущего и планируемого запросов и скоростями вращения диска и перемещения головок. Без знания соотношения этих скоростей сравнение становится невозможным. Поэтому естественно, что набор параметров планирования сокращается до времени поиска различных запросов, определяемого текущим положением головки и номерами требуемых цилиндров, а разницей в задержках на вращение пренебрегают.

Целью задания является показать ряд цилиндров, которые идут в соответствии с представленным в задании алгоритмом, а также вычислить общее передвижение.
Итак, предположим, нам даны следующие цилиндры:

Algsilinder: **63**
Tellimus: **23, 67, 55, 14, 31, 7, 84, 10**
Silindrite numbrid**: 0...99**
Liikumise algsuund: **kasvav**

Теперь посмотрим, как же надо расположить цилиндры в соответствии с алгоритмами.

First Come First Served (FCFS)

Простейшим алгоритмом является алгоритм ***First Come First Served (FCFS)*** – первым пришел, первым обслужен. Все запросы организуются в очередь FIFO и обслуживаются в порядке поступления. Алгоритм прост в реализации, но может приводить к достаточно длительному общему времени обслуживания запросов. Положение головок будет меняться следующим образом:

63→23→67→55→14→31→7→84→10
и всего головки переместятся на 40+44+12+41+17+24+77+74 = **329** цилиндров.

Short Seek Time First (SSTF)

Алгоритм *Short Seek Time First (SSTF)* – короткое время поиска. Для очередного обслуживания будем выбирать запрос, данные для которого лежат наиболее близко к текущему положению магнитных головок. Алгоритм даст такую последовательность положений головок:

63→67→55→31→23→14→10→7→84
и всего головки переместятся на **141** цилиндр.

SCAN

В простейшем из алгоритмов сканирования – ***SCAN*** – головки постоянно перемещаются от одного края диска до другого, по ходу дела обслуживая все встречающиеся запросы. По достижении другого края направление движения меняется, и все повторяется снова. Пусть в предыдущем примере в начальный момент времени головки двигаются в направ-

лении уменьшения номеров цилиндров. Тогда мы и получим порядок обслуживания запросов, подсмотренный в конце предыдущего раздела. Последовательность перемещения головок выглядит следующим образом:

63→55→31→23→14→10→7→0→67→84

и всего головки переместятся на **147** цилиндров.

LOOK

Если мы знаем, что обслужили последний попутный запрос в направлении движения головок, то мы можем не доходить до края диска, а сразу изменить направление движения на обратное:

63→55→31→23→14→10→7→67→84

и всего головки переместятся на **133** цилиндра. Полученная модификация алгоритма *SCAN* получила название *LOOK*.

C-SCAN

Допустим, что к моменту изменения направления движения головки в алгоритме *SCAN*, т. е. когда головка достигла одного из краев диска, у этого края накопилось большое количество новых запросов, на обслуживание которых будет потрачено достаточно много времени (не забываем, что надо не только перемещать головку, но еще и передавать прочитанные данные!). Тогда запросы, относящиеся к другому краю диска и поступившие раньше, будут ждать обслуживания несправедливо долго. Для сокращения времени ожидания запросов применяется другая модификация алгоритма *SCAN* – циклическое сканирование. Когда головка достигает одного из краев диска, она без чтения попутных запросов (иногда существенно быстрее, чем при выполнении обычного поиска цилиндра) перемещается на другой край, откуда вновь начинает движение в прежнем направлении. Для этого алгоритма, получившего название ***C-SCAN***, последовательность перемещений будет выглядеть так:

63→55→31→23→14→10→7→0→99→84→67

C-LOOK

По аналогии с алгоритмом *LOOK* для алгоритма *SCAN* можно предложить и алгоритм *C-LOOK* для алгоритма *C-SCAN*:

63→55→31→23→14→10→7→84→67

Задание 3. Проблема тупиков.

Введение. Предположим, что несколько процессов конкурируют за обладание конечным числом *ресурсов*. Если запрашиваемый процессом *ресурс* недоступен, ОС переводит данный процесс в состояние ожидания. В случае когда требуемый *ресурс* удерживается другим ожидающим процессом, первый процесс не сможет сменить свое состояние. Такая ситуация называется ***тупиком (deadlock)***. Говорят, что в мультипрограммной системе процесс находится в состоянии *тупика*, если он ожидает события, которое никогда не произойдет. Системная *тупиковая ситуация*, или «зависание системы», является следствием того, что один или более процессов находятся в состоянии *тупика*. Иногда подобные ситуации называют ***взаимоблокировками***. В общем случае проблема *тупиков* эффективного решения не имеет.

Рассмотрим пример. Предположим, что два процесса осуществляют вывод с ленты на принтер. Один из них успел монополизировать ленту и претендует на принтер, а другой наоборот. После этого оба процесса оказываются заблокированными в ожидании второго *ресурса* (см. рис. 2).

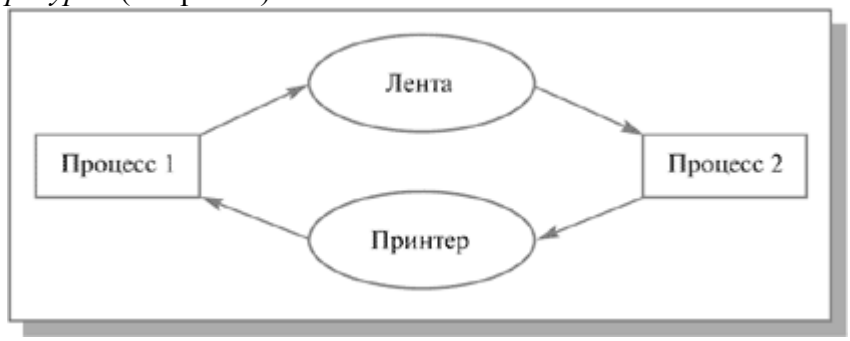

Рис. 2. Пример тупиковой ситуации

Определение. Множество процессов находится в *тупиковой ситуации*, если каждый процесс из множества ожидает события, которое может вызвать только другой процесс данного множества. Так как все процессы чего-то ожидают, то ни один из них не сможет инициировать событие, которое разбудило бы другого члена множества и, следовательно, все процессы будут спать вместе.

Выше приведен пример *взаимоблокировки*, возникающей при работе с так называемыми выделенными устройствами. *Тупики*, однако, могут иметь место и в других ситуациях. Например, в системах управления базами данных записи могут быть локализованы процессами, чтобы избежать состояния гонок (см. лекцию 5 «Алгоритмы синхронизации»). В этом случае может получиться так, что один из процессов заблокировал записи, необходимые другому процессу, и наоборот. Таким образом, *тупики* могут иметь место как на аппаратных, так и на программных *ресурсах*.

Тупики также могут быть вызваны ошибками программирования. Например, процесс может напрасно ждать открытия семафора, потому что в некорректно написанном приложении эту операцию забыли предусмотреть. Другой причиной бесконечного ожидания может быть дискриминационная политика по отношению к некоторым процессам. Однако чаще всего событие, которого ждет процесс в *тупиковой ситуации*, – освобождение *ресурса*, поэтому в дальнейшем будут рассмотрены методы борьбы с *тупиками* ресурсного типа.

Ресурсами могут быть как устройства, так и данные. Некоторые *ресурсы* допускают разделение между процессами, то есть являются **разделяемыми** *ресурсами*. Например, память, процессор, диски коллективно используются процессами. Другие не допускают разделения, то есть являются **выделенными**, например лентопротяжное устройство. К *взаимоблокировке* может привести использование как выделенных, так и разделяемых *ресурсов*. Например, чтение с разделяемого диска может одновременно осуществляться не-

сколькими процессами, тогда как запись предполагает исключительный доступ к данным на диске. Можно считать, что часть диска, куда происходит запись, выделена конкретному процессу. Поэтому в дальнейшем мы будем исходить из предположения, что *тупики* связаны с выделенными *ресурсами* , то есть *тупики* возникают, когда процессу предоставляется эксклюзивный доступ к устройствам, файлам и другим *ресурсам*.
Традиционная последовательность событий при работе с *ресурсом* состоит из запроса, использования и освобождения *ресурса*. Тип запроса зависит от природы *ресурса* и от ОС. Запрос может быть явным, например специальный вызов request, или неявным – open для открытия файла. Обычно, если *ресурс* занят и запрос отклонен, запрашивающий процесс переходит в состояние ожидания.
Далее в данной лекции будут рассматриваться вопросы обнаружения, предотвращения, обхода *тупиков* и восстановления после *тупиков*. Как правило, борьба с *тупиками* – очень дорогостоящее мероприятие. Тем не менее для ряда систем, например для систем реального времени, иного выхода нет.

Задание. По графу выяснить состояние данной системы: **deadlock (тупик), safe** или **unsafe.**

Если **deadlock**, то должен быть цикл, причем в этом цикле не должно быть пунктирных линий. Если же в нем есть пунктирные линии, то это **unsafe,** то есть существует опасность тупика (если пунктиры заменить на обычные линии, то при **unsafe** должен получиться тупик). В остальных случаях будет **safe.**