Федеральное агентство по образованию РФ
Московский государственный университет геодезии и картографии
(МИИГАиК)

А.А. Кудлаев

Управление ресурсами ЭВМ

Учебно-методическое пособие по курсу
«Операционные системы»

Москва 2008 г.

Составитель: к.т.н., доцент кафедры ВТиАОАИ Кудлаев А.А.

Управление ресурсами ЭВМ. Учебно-методическое пособие по курсу «Операционные системы». М., МИИГАиК, 2008, 32 стр.

Учебно-методическое пособие составлено в соответствии с утвержденной программой курса «Операционные системы» для студентов факультета прикладной космонавтики специальности ИС и ОТЗИ, рекомендовано кафедрой вычислительной техники и автоматизированной обработки аэрокосмической информации.

В учебно-методическом пособии рассмотрены задачи обеспечения рационального управления основными ресурсами вычислительной системы для повышения ее производительности, и действия операционных систем по выполнению этих задач. Пособие содержит примеры и практические задания для реализации в среде MS EXCEL средствами VBA.

Библиография: 11 названий.

Рецензенты:
Ректор Интернет-Университета Информационных Технологий А.В. Шкред
К.т.н., преподаватель кафедры «ВТиАОАИ» МИИГАиК Д.В. Учаев

ОГЛАВЛЕНИЕ:

Введение

Задача курса ОС - дать обзор основных функций, выполняемых операционными системами, и способов их реализации.

Задача операционных систем заключается в том, чтобы сделать аппаратуру доступной, удобной для пользователей, обеспечить рациональное управление аппаратурой для достижения высокой производительности.

Операционная система должна уметь делать следующее:

- обеспечивать загрузку пользовательских программ в оперативную память и их исполнение;
- обеспечивать управление памятью;
- обеспечивать работу с устройствами долговременной памяти;
- предоставлять стандартизованный доступ к различным периферийным устройствам;
- предоставлять пользовательский интерфейс для взаимодействия с вычислительной системой.

Существуют ОС, функции которых этим и исчерпываются. Одна из хорошо известных систем такого типа - *MS DOS*.

Более развитые ОС предоставляют также следующие возможности:

- параллельное (или псевдопараллельное, если машина имеет только один процессор) исполнение нескольких задач;
- организацию взаимодействия задач друг с другом;
- организацию межмашинного взаимодействия и разделения ресурсов;
- защиту системных ресурсов, данных и программ пользователя, исполняющихся процессов и самой себя от ошибочных и зловредных действий пользователей и их программ;
- аутентификацию, авторизацию и другие средства обеспечения безопасности.

Операционная система управляет следующими основными ресурсами:

- процессорами;
- памятью;
- устройствами ввода-вывода.

В методическом пособии рассматриваются задачи, выполняемые операционными системами по обеспечению рационального управления основными ресурсами вычислительной системы для повышения ее производительности.

Расчетно-графическая работа №1
Планирование процессов

Одним из наиболее ограниченных ресурсов вычислительной системы является процессорное время. Для его распределения между процессами в системе применяется процедура планирования процессов.

1.1. Уровни планирования

Существует два вида планирования в вычислительных системах:

- планирование заданий;
- планирование использования процессора.

Планирование заданий выступает в качестве *долгосрочного планирования процессов*. Оно отвечает за порождение новых процессов в системе, определяя количество процессов, одновременно находящихся в ней. Долгосрочное планирование осуществляется достаточно редко.

Планирование использования процессора выступает в качестве *краткосрочного планирования процессов*. Оно проводится при обращении исполняющегося процесса к устройствам ввода-вывода или просто по завершении определенного интервала времени.

В некоторых вычислительных системах бывает выгодно для повышения их производительности временно удалить выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для выполнения. Когда и какой из процессов нужно перекачать на диск и вернуть обратно, решается промежуточным уровнем *планирования процессов — среднесрочным.*

1.2. Критерии планирования и требования к алгоритмам

Для каждого уровня планирования процессов существуют различные алгоритмы. Выбор алгоритма определяется классом задач и целями. К числу таких целей можно отнести:

- справедливость использования процессора;
- эффективность загрузки;
- сокращение полного времени выполнения;
- сокращение времени ожидания;
- сокращение времени отклика.

Требования к свойствам алгоритмов:

- были предсказуемыми;
- имели минимальные накладные расходы;
- равномерно загружали ресурсы вычислительной системы;
- обладали масштабируемостью.

1.3. Параметры планирования

Для осуществления поставленных целей алгоритмы планирования должны опираться на параметры планирования. *Параметры планирования вычислительной системы* делятся на две группы:

- *статические параметры* не изменяются в ходе функционирования (ресурсы вычислительной системы);
- *динамические параметры* подвержены изменениям и описывают количество свободных ресурсов в текущий момент времени.

Параметры планирования процессов также делятся на две группы:

- *статические параметры* (характеристики, присущие заданиям на этапе загрузки - приоритет задачи, запрошенное время *CPU*, соотношение времени *CPU* к времени ввода-вывода, необходимые ресурсы вычислительной системы).

Алгоритмы долгосрочного планирования используют в своей работе статические и динамические параметры вычислительной системы и статические параметры процессов.

Алгоритмы краткосрочного и среднесрочного планирования дополнительно учитывают и динамические характеристики процессов:

- *CPU burst* - промежуток времени непрерывного использования процессора;
- *I/O burst* - промежуток времени непрерывного ожидания ввода-вывода.

1.4. Вытесняющее и невытесняющее планирование

Процесс планирования осуществляется частью ОС, называемой планировщиком. Планировщик принимает решения о выборе для исполнения нового процесса, из числа находящихся в состоянии готовность, в следующих четырех случаях:

1. процесс переводится из состояния *исполнение* в состояние *завершение*;
2. процесс переводится из состояния *исполнение* в состояние *ожидание*;
3. процесс переводится из состояния *исполнение* в состояние *готовность* (например, после прерывания от таймера);
4. процесс переводится из состояния *ожидание* в состояние *готовность*.

Для случаев 1 и 2 говорят о *невытесняющем планировании*. В противном случае говорят о *вытесняющем планировании*.

Невытесняющее планирование просто реализуемо и достаточно эффективно. Однако при невытесняющем планировании возникает проблема возможности полного захвата процессора одним процессом. Вытесняющее планирование обычно используется в системах разделения времени.

1.5. Алгоритмы планирования

1.5.1. First-Come, First-Served (FCFS)

Процессы, находящиеся в состоянии готовность (Г), организованы в очередь. Выбор нового процесса для исполнения (И) осуществляется из начала очереди. Такой алгоритм выбора процесса осуществляет *невытесняющее планирование*.

Пример. В состоянии *готовность* находятся три процесса p_1, p_2 и p_3, для которых известны их *CPU burst*:

Процесс	P_1	P_2	P_3
Продолжительность очередного *CPU burst*	13	4	1

Если процессы расположены в очереди процессов готовых к исполнению в порядке p_1, p_2, p_3, то картина их выполнения выглядит так:

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P1	И	И	И	И	И	И	И	И	И	И	И	И	И					
P2	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	
P3	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И

Среднее время ожидания	10,00
Среднее полное время выполнения	16,00

Среднее время ожидания: (0 + 13 + 17)/3 = 10 единиц времени.

Среднее полное время выполнения: (13 + 17 + 18)/3 = 16.

Если те же процессы расположены в порядке p_3, p_2, p_1, то картина их выполнения будет соответствовать содержимому таблицы:

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P3	И																	
P2	Г	И	И	И	И													
P1	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И	И	И	И	И

Среднее время ожидания	2,00
Среднее полное время выполнения	8,00

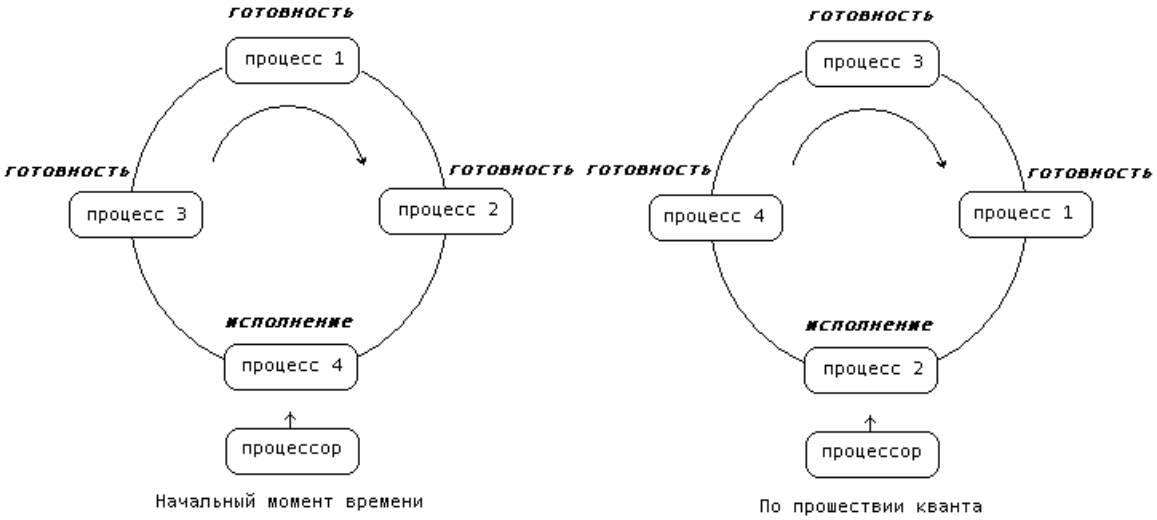
Среднее время ожидания в 5 раз меньше предыдущего случая.

Среднее полное время выполнения в 2 раза меньше первой расстановки процессов.

Среднее время ожидания и среднее полное время выполнения для этого алгоритма зависят от порядка расположения процессов в очереди: короткие процессы, перешедшие в состояние готовность после длительного процесса, будут долго ждать начала своего выполнения.

1.5.2. Round Robin (RR)

Модификацией алгоритма *FCFS* является алгоритм *Round Robin*. Это тот же самый алгоритм, только реализованный в режиме вытесняющего планирования. Все готовые процессы организованны циклически: каждый процесс находится около процессора фиксированный квант времени.



Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии готовность, в очередь. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении определенного кванта времени. При выполнении процесса возможны два варианта:

1. Время использования *CPU*, требующееся процессу, меньше или равно продолжительности кванта времени. Тогда процесс освобождает *CPU* до истечения кванта времени, на исполнение выбирается новый процесс из начала очереди и таймер начинает отсчет кванта заново.

2. Продолжительность остатка текущего *CPU burst* процесса больше, чем квант времени. Тогда по истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов.

Рассмотрим предыдущий пример с порядком процессов p_1, p_2, p_3 и величиной кванта времени равной 4. Выполнение этих процессов иллюстрируется таблицей:

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P1	И	И	И	И	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И
P2	Г	Г	Г	Г	И	И	И	И										
P3	Г	Г	Г	Г	Г	Г	Г	Г	И									

Среднее время ожидания	5,67
Среднее полное время выполнения	11,67

Первым для исполнения выбирается процесс p_1. Продолжительность его *CPU burst* больше, чем величина кванта времени, и поэтому процесс исполняется до истечения кванта. После этого он помещается в конец очереди готовых к исполнению процессов. Следующим начинает выполняться процесс p_2. Время его исполнения совпадает с величиной выделенного кванта, поэтому процесс работает до своего завершения. Процессор выделяется процессу p_3. Он завершается до истечения отпущенного ему процессорного времени, и очередные кванты отмеряются процессу p_1.

Среднее время ожидания и среднее полное время выполнения для обратного порядка процессов не отличаются от соответствующих времен для алгоритма *FCFS*:

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P3	И																	
P2	Г	И	И	И	И													
P1	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И	И	И	И	И

Среднее время ожидания	2,00
Среднее полное время выполнения	8,00

На производительность алгоритма RR сильно влияет величина кванта времени. Рассмотрим тот же самый пример с порядком процессов p_1, p_2, p_3 для величины кванта времени равной 1:

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P1	И	Г	Г	И	Г	И	Г	И	Г	И	И	И	И	И	И	И	И	И
P2	Г	И	Г	Г	И	Г	И	Г	И									
P3	Г	Г	И															

Среднее время ожидания	4,00
Среднее полное время выполнения	10,00

При очень больших величинах кванта времени, когда каждый процесс успевает завершить свой *CPU burst* до возникновения прерывания по времени, алгоритм *RR* вырождается в алгоритм *FCFS*.

При очень малых величинах создается иллюзия того, что каждый из n процессов работает на своем собственном виртуальном процессоре с производительностью ~ 1/n от производительности реального процессора.

В реальных условиях при слишком малой величине кванта времени и, соответственно, слишком частом переключении контекста, накладные расходы на переключение резко снижают производительность системы.

1.5.3. Shortest-Job-First (SJF)

Для алгоритмов *FCFS* и *RR* существенным является порядок расположения процессов в очереди процессов готовых к исполнению: если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает. Если знать время *CPU burst* для процессов, находящихся в состоянии готовность, то можно выбрать для исполнения процесс с минимальной длительностью *CPU burst*. Описанный алгоритм получил название “короткий работает первым” или *Shortest Job First* (*SJF*).

SJF алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим. При невытесняющем *SJF* планировании процессор предоставляется избранному процессу на все требующееся ему время. При вытесняющем *SJF* планировании учитывается появление новых процессов в очереди готовых к исполнению. Если *CPU burst* нового процесса меньше, чем остаток *CPU burst* у исполняющегося, то исполняющийся процесс вытесняется новым.

Пример работы невытесняющего алгоритма SJF

Пусть в состоянии готовность находятся четыре процесса p_1, p_2, p_3 и p_4, для которых известны времена их очередных *CPU burst*:

Процесс	P_1	P_2	P_3	P_4
Продолжительность очередного CPU burst	5	3	7	1

Первым для исполнения будет выбран процесс p_4, имеющий наименьшее значение очередного *CPU burst*. После его завершения для исполнения выбирается процесс p_2, затем p_1 и, наконец, p_3:

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P1	Г	Г	Г	Г	И	И	И	И	И							
P2	Г	И	И	И												
P3	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
P4	И															

Среднее время ожидания	3,50
Среднее полное время выполнения	7,50

Среднее время ожидания для алгоритма SJF: (4 + 1 + 9 + 0)/4 = 3,5 единицы времени. Для алгоритма *FCFS* при порядке процессов p_1, p_2, p_3, p_4 эта величина будет равняться (0 + 5 + 8 + 15)/4 = 7 единицам времени, т. е. будет в 2 раза больше, чем для алгоритма *SJF*.

Для заданного набора процессов алгоритм *SJF* является оптимальным по минимизации среднего времени ожидания среди невытесняющих алгоритмов.

Пример работы вытесняющего алгоритма SJF

Возьмем ряд процессов p_1, p_2, p_3 и p_4 с различными временами *CPU burst* и различными моментами их появления в очереди процессов готовых к исполнению:

Процесс	Время появления в очереди	Продолжительность очередного CPU burst
P_1	1	6
P_2	3	2
P_3	7	7
P_4	1	5

В начальный момент времени в состоянии готовность находятся процессы p_1 и p_4. Меньшее *CPU burst* у процесса p_4, поэтому он выбирается для исполнения. Через 2 единицы времени в систему поступает процесс p_2. Время его *CPU burst* меньше, чем остаток *CPU burst* у процесса p_4, который вытесняется из состояния исполнение. По прошествии еще 2-х единиц времени процесс p_2 завершается, и для исполнения выбирается процесс p_4. В момент времени 7 в очереди появляется процесс p_3, для его работы нужно 7 единиц времени, а процессу p_4 осталось трудиться 1 единицу времени, поэтому процесс p_4 остается в состоянии исполнение. После его завершения в момент времени 8 в очереди находятся процессы p_1 и p_3, из которых выбирается процесс p_1. Последним получит возможность выполняться процесс p_3.

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P1	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И							
P2			И	И																
P3							Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
P4	И	И	Г	Г	И	И	И													

Среднее время ожидания	4,00
Среднее полное время выполнения	9,00

Основную сложность при реализации алгоритма *SJF* представляет невозможность точного знания времени очередного *CPU burst* для исполняющихся процессов.

1.5.4. Гарантированное планирование

При интерактивной работе N пользователей в вычислительной системе можно применить алгоритм планирования, который гарантирует, что каждый из пользователей будет иметь в своем распоряжении ~ 1/N часть процессорного времени. Пронумеруем всех пользователей от 1 до N. Для каждого пользователя с номером i введем две величины:

T_i - длительность сеанса его общения с машиной,

τ_i - суммарное время *CPU*, выделенное всем его процессам в течение сеанса.

Справедливым для пользователя было бы получение T_i/N процессорного времени.

Если $\tau_i << T_i/N$, то *i* - й пользователь обделен процессорным временем.

Если $\tau_i >> T_i/N$ то система благоволит к пользователю с номером *i*.

Вычислим для каждого пользовательского процесса значение коэффициента справедливости $\tau_i N/T_i$ и будем предоставлять очередной квант времени процессу с наименьшей величиной этого отношения. Предложенный алгоритм называют алгоритмом гарантированного планирования.

1.5.5. Приоритетное планирование

Алгоритмы *SJF* и гарантированного планирования представляют собой частные случаи приоритетного планирования. При приоритетном планировании каждому процессу присваивается определенное числовое значение - приоритет, в соответствии с которым ему выделяется процессор. Для алгоритма *SJF* в качестве такого приоритета выступает оценка продолжительности следующего *CPU burst*. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс. Для алгоритма гарантированного планирования приоритетом служит вычисленный коэффициент справедливости. Чем он меньше, тем больше приоритет у процесса.

Принципы назначения приоритетов могут опираться как на внутренние критерии вычислительной системы, так и на внешние по отношению к ней. Внутренние используют различные количественные и качественные характеристики процесса для вычисления его приоритета: ограничения по времени использования процессора, требования к размеру памяти, число открытых файлов и используемых устройств ввода-вывода. Внешние критерии: важность процесса для достижения каких-либо целей, стоимость оплаченного процессорного времени и других политических факторов.

Планирование с использованием приоритетов может быть как вытесняющим, так и невытесняющим.

Примеры использования различных режимов приоритетного планирования

Пусть в очередь процессов, находящихся в состоянии готовность, поступают те же процессы p_1, p_2, p_3 и p_4, им присвоены приоритеты (большее значение соответствует меньшему приоритету):

Процесс	Время появления в очереди	Продолжительность очередного CPU burst	Приоритет
P_1	1	6	4
P_2	3	2	3
P_3	7	7	2
P_4	1	5	1

Невытесняющее приоритетное планирование:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P1	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И
P2			Г	Г	Г	И	И													
P3							Г	И	И	И	И	И	И	И						
P4	И	И	И	И	И															

Среднее время ожидания	4,50
Среднее полное время выполнения	9,50

Первым для выполнения, ко времени 1, выбирается процесс p_4, обладающий наивысшим приоритетом. После его завершения, к моменту времени 6, в очереди процессов, готовых к исполнению, находятся процессы p_1 и p_2. Больший приоритет из них у процесса p_2, он и начнет исполняться. В момент времени 7 в системе появится процесс p_3, он будет избран для исполнения. Последним будет исполняться процесс p_1.

Вытесняющее приоритетное планирование:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P1	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И
P2			Г	Г	Г	И	Г	Г	Г	Г	Г	Г	Г	И						
P3							И	И	И	И	И	И	И							
P4	И	И	И	И	И															

Среднее время ожидания	6,00
Среднее полное время выполнения	11,00

Первым исполняться начнет процесс p_4, а по его окончании процесс p_2. Однако, в момент времени 7 он будет вытеснен процессом p_3 и продолжит свое выполнение только в момент времени 14. Последним будет исполнен процесс p_1.

Приоритеты процессов, которые не изменяются с течением времени, называют статическими. Механизмы статической приоритетности легко реализовать. Однако статические приоритеты не реагируют на изменения ситуации в вычислительной системе. Более гибкими являются динамические приоритеты процессов, изменяющие свои значения по ходу исполнения процессов. Как правило, изменение приоритета процессов проводится согласованно с совершением каких-либо других операций: при рождении нового процесса, при разблокировке или блокировании процесса, по истечении определенного кванта времени или по завершении процесса. Примерами алгоритмов с динамическими приоритетами являются алгоритм *SJF* и алгоритм гарантированного планирования. Схемы с динамической приоритетностью гораздо сложнее в реализации и связанны с большими издержками по сравнению со статическими схемами.

Главная проблема приоритетного планирования заключается в том, что при ненадлежащем выборе механизма назначения и изменения приоритетов низкоприоритетные процессы могут быть не запущены неопределенно долгое время. Решение этой проблемы может быть достигнуто с помощью увеличения со временем значения приоритета процесса, находящегося в состоянии готовность. Каждый раз, по истечении определенного промежутка времени, значения приоритетов готовых процессов уменьшаются на 1. Процессу, побывавшему в состоянии исполнение, восстанавливается первоначальное значение приоритета. Даже такая грубая схема гарантирует, что любому процессу в разумные сроки будет предоставлено право на исполнение.

Задания:

Задание 1. Пусть в вычислительную систему поступают пять процессов различной длительности по следующей схеме:

Номер процесса	1	2	3	4	5
Момент поступления в систему	2	1	4	3	0
Время исполнения	4	3	5	2	9

Чему равно среднее время ожидания процесса при использовании невытесняющего алгоритма *SJF*? При вычислениях считать, что процессы не совершают операций ввода-вывода, временем переключения контекста пренебречь.

Задание 2. Пусть в вычислительную систему поступают пять процессов различной длительности с разными приоритетами по следующей схеме:

Номер процесса	1	2	3	4	5
Момент поступления в систему	3	6	0	2	4
Время исполнения	10	4	4	1	3
Приоритет	1	0	3	4	2

Чему равно среднее время между стартом процесса и его завершением при использовании вытесняющего приоритетного планирования? При вычислениях считать, что процессы не совершают операций ввода-вывода, временем переключения контекста пренебречь. Наивысшим приоритетом является приоритет 0.

Задание 3. Пусть в вычислительную систему поступают пять процессов различной длительности по следующей схеме:

Номер процесса	1	2	3	4	5
Момент поступления в систему	2	1	4	3	0
Время исполнения	4	3	5	2	9

Чему равно среднее время ожидания процесса при использовании вытесняющего алгоритма *SJF*? При вычислениях считать, что процессы не совершают операций ввода-вывода, временем переключения контекста пренебречь.

Задание 4. На алгоритмическом языке VBA реализовать программы невытесняющего планирования FCFS и вытесняющего планирования RR (см. приложение).

Задание 5. На алгоритмическом языке VBA самостоятельно реализовать программу планирования SJF в невытесняющем и вытесняющем режимах.

Задание 6. На алгоритмическом языке VBA самостоятельно реализовать программу приоритетного планирования в невытесняющем и вытесняющем режимах.

Расчетно-графическая работа №2
Управление виртуальной памятью

Виртуальная память. Этот механизм позволил реализовать идею неполного нахождения исполняемой программы в оперативной памяти. Основная часть программы находится на МД и необходимый для ее дальнейшего выполнения кусок может быть загружен в оперативную память, а ненужный выкачан обратно на МД.

Большинство ОС используют сегментно-страничную виртуальную память. Для обеспечения нужной производительности менеджер памяти ОС старается поддерживать в оперативной памяти актуальную информацию, пытаясь угадать, к каким логическим адресам последует обращение в недалеком будущем.

2.1 Исключительные ситуации при работе с памятью

При попытке выполнить операцию, когда нужной страницы в памяти нет, возникает исключительная ситуация *страничное нарушение*, приводящая к вызову специальной программы - *обработчика* этой исключительной ситуации.

Обращения к не присутствующей странице определяет производительность страничной системы. Оптимизация может происходить по пути:

- уменьшения частоты страничных нарушений,
- увеличения скорости их обработки.

Снижение частоты страничных нарушений является одной из ключевых задач системы управления памятью. Ее решение обычно связано с правильным выбором алгоритма замещения страниц.

2.2 Стратегии управления страничной памятью

Стратегия выборки определяет в какой момент следует переписать страницу из вторичной памяти в первичную. Алгоритм выборки вступает в действие в тот момент, когда процесс обращается к не присутствующей странице, содержимое которой в данный момент находится на диске.

Выборка бывает по запросу и с упреждением. При опережающем чтении кроме страницы, вызвавшей исключительную ситуацию, в память загружается несколько страниц, окружающих ее. Такой алгоритм оптимизирует работу с диском, поскольку загружается несколько страниц за одно обращение к диску.

Стратегия размещения определяет, в какое место первичной памяти поместить поступающую страницу: в системах со страничной организацией - в любой свободный страничный кадр.

Стратегия замещения определяет, какую страницу нужно вытолкнуть во внешнюю память, чтобы освободить место. Разумная стратегия замещения позволяет оптимизировать хранение в памяти самой необходимой информации и снизить частоту страничных нарушений.

2.3 Алгоритмы замещения страниц

Алгоритмы замещения страниц делятся на локальные и глобальные.

- *Локальные алгоритмы* распределяют фиксированное или динамически настраиваемое число страниц для каждого процесса. Когда процесс израсходует все предназначенные ему страницы, система будет удалять из физической памяти одну из его страниц, а не из страниц других процессов.

- *Глобальный алгоритм* замещения в случае возникновения исключительной ситуации освобождает любую физическую страницу, независимо от того, какому процессу она принадлежала.

Недостатки глобальных алгоритмов:

- делают одни процессы чувствительными к поведению других процессов;

- некорректно работающее приложение может подорвать работу всей системы, пытаясь захватить все больше памяти.

В многозадачной системе лучше использовать более сложные, но эффективные локальные алгоритмы.

Алгоритм оценивается на конкретной последовательности ссылок к памяти, для которой подсчитывается число страничных нарушений. Эта последовательность называется *reference string,* ее можно генерировать при помощи датчика случайных чисел.

Большинство процессоров имеют аппаратные средства, позволяющие собирать статистику обращений к памяти. Эти средства включают два специальных флага на каждый элемент таблицы страниц:

- *флаг обращения* автоматически устанавливается, когда происходит любое обращение к этой странице;

- *флаг изменения* устанавливается, если производится запись в эту страницу.

2.3.1 Выталкивание первой пришедшей страницы (FIFO)

Суть алгоритма: *для замещения выбирается старейшая страница* (каждой странице присваивается временная метка). Недостатком этой стратегии является большая вероятность замещения активно используемых страниц.

2.3.2 Оптимальный алгоритм (OPT)

Суть алгоритма: *замещать страницу, которая не будет использоваться в течение длительного периода времени*. Каждая страница помечается числом инструкций, которые будут выполнены, прежде чем на эту страницу будет сделана первая ссылка.

Этот алгоритм нереализуем и применяется для оценки качества реализуемых алгоритмов, имеет минимальную частоту страничных нарушений среди всех алгоритмов.

2.3.3 Выталкивание дольше всего не использовавшейся страницы. LRU (The Least Recently Used) Algorithm

Суть алгоритма: *замещать страницу, которая не использовалась в течение долгого времени*. Для реализации алгоритма необходимо иметь связанный список всех страниц в памяти, в начале которого будут находиться часто используемые страницы. Список должен обновляться при каждой ссылке.

LRU не страдает от аномалии *Belady*, часто используется и считается хорошим. Недостаток алгоритма - много времени тратится на поиск страниц в списке.

2.3.4 Выталкивание редко используемой страницы. NFU (Not Frequently Used) алгоритм

LRU требует специальной аппаратной поддержки, которой большинство современных процессоров не предоставляет. Алгоритм *NFU* близок к *LRU*, и не требует специальной поддержки.

Для него требуются программные счетчики, по одному на каждую страницу, которые сначала равны нулю. При каждом прерывании по времени ОС сканирует все страницы в памяти и у каждой страницы с установленным флагом обращения увеличивает на единицу значение счетчика, а флаг обращения сбрасывает.

Суть алгоритма: *кандидатом на освобождение оказывается страница с наименьшим значением счетчика.*

Главным недостатком алгоритма *NFU* является то, что он никогда ничего не забывает. Например, страница, к которой очень много обращались некоторое время, а потом обращаться перестали, не будет удалена из памяти, потому что ее счетчик содержит большую величину.

Другим недостатком алгоритма является длительность процесса сканирования таблиц страниц.

Модификация алгоритма, которая реализует "забывание": при каждом прерывании по времени содержимое каждого счетчика сдвигается вправо на 1 бит, а уже затем производится его увеличение для страниц с установленным флагом обращения.

2.3.5 Другие алгоритмы

Алгоритм *Second-Chance* - модификации *FIFO*, которая позволяет избежать потери часто используемых страниц за счет анализа бита r (*reference*) для самой старой страницы. Если r = 1, то страница, в отличие от *FIFO*, не выталкивается, а очищается бит, и страница становится в конец очереди. Если на все страницы ссылались, он превращается в *FIFO*. Данный алгоритм использовался в BSD Unix.

Аномалия Belady

Предположение, что чем больше страничных кадров имеет память, тем реже будут иметь место страничные нарушения, не всегда верно. Как было установлено, определенные последовательности обращений к страницам приводят в действительности к увеличению числа страничных нарушений при увеличении кадров, выделенных процессу. Это явление носит название аномалии *FIFO*.

Три кадра (9 faults) оказываются лучше чем 4 кадра (10 faults) для 012301401234 цепочки чередования страниц при выборе стратегии *FIFO*.

	0	***1***	***2***	***3***	***0***	***1***	***4***	***0***	***1***	***2***	***3***	***4***	
Старая страница	0	1	2	3	0	1	4	4	4	2	3	3	
		0	1	2	3	0	1	1	1	4	2	2	
Новая страница			0	1	2	3	0	0	0	1	4	4	
	p	*p*	*p*	*p*	*p*	*p*	*p*			*p*	*p*		*9 page fault*

	0	***1***	***2***	***3***	***0***	***1***	***4***	***0***	***1***	***2***	***3***	***4***	
Старая страница	0	1	2	3	3	3	4	0	1	2	3	4	
		0	1	2	2	2	3	4	0	1	2	3	
			0	1	1	1	2	3	4	0	1	2	
Новая страница				0	0	0	1	2	3	4	0	1	
	p	*p*	*p*	*p*			*p*	*p*	*p*	*p*	*p*	*p*	*9 page fault*

2.4. Thrashing. Свойство локальности

Высокая частота страничных нарушений называется *трешинг* (пробуксовка). Процесс находится в состоянии трешинга, если он больше времени занимается подкачкой страниц, нежели собственно выполнением. Результатом трешинга является снижение производительности.

Эффект трешинга, возникающий при использовании глобальных алгоритмов, может быть ограничен за счет использования локальных алгоритмов замещения. Если даже один из процессов попал в трешинг, это не сказывается на других процессах.

Для предотвращения трешинга нужно выделить процессу необходимое число кадров, которые процесс реально использует. Этот подход определяет *модель локальности* выполнения процесса.

Модель локальности состоит в том, что когда процесс выполняется, он двигается от одной локальности к другой. *Локальность - набор страниц, которые активно используются вместе*.

Задания:

Задание 1. Для некоторого процесса известна следующая строка запросов страниц памяти: *7, 1, 2, 3, 2, 4, 2, 1, 0, 3, 7, 2, 1, 2, 7, 1, 7, 2, 3*. Сколько ситуаций отказа страницы возникнет для данного процесса при использовании алгоритма замещения страниц *LRU* и 3-х страничных кадрах?

Задание 2. Для некоторого процесса известна следующая строка запросов страниц памяти: *7, 1, 2, 3, 2, 4, 2, 1, 0, 3, 7, 2, 1, 2, 7, 1, 7, 2, 3*. Сколько ситуаций отказа страницы возникнет для данного процесса при использовании алгоритма замещения страниц *OPT* и 3-х страничных кадрах?

Задание 3. Для некоторого процесса известна следующая строка запросов страниц памяти: *7, 1, 2, 3, 2, 4, 2, 1, 0, 3, 7, 2, 1, 2, 7, 1, 7, 2, 3*. Сколько ситуаций отказа страницы возникнет для данного процесса при использовании алгоритма замещения страниц *FIFO* и 3-х страничных кадрах?

Задание 4. На алгоритмическом языке VBA самостоятельно реализовать программу замещения страниц виртуальной памяти, состоящей из N страничных кадров, по алгоритмам FIFO, OPT, LRU, NFU с определением количества ситуаций отказа страницы для заданной строки запросов страниц памяти.

Расчетно-графическая работа №3
Система управления вводом-выводом

Функционирование любой вычислительной системы обычно сводится к выполнению двух видов работы: обработка информации и операции по осуществлению ее ввода-вывода. С точки зрения операционной системы "обработкой информации" являются только операции, совершаемые процессором над данными, находящимися в памяти на уровне иерархии не ниже чем оперативная память. Все остальное относится к "операциям ввода-вывода", т.е. к обмену информацией с внешними устройствами.

3.1 Планирование запросов

При использовании *неблокирующегося системного вызова* может оказаться, что нужное устройство уже занято выполнением некоторых операций. В этом случае *неблокирующийся вызов* может немедленно вернуться, не выполнив запрошенных команд.

При организации запроса на совершение операций ввода-вывода с помощью *блокирующегося* или *асинхронного вызова* занятость устройства приводит к необходимости постановки запроса в очередь к данному устройству. В результате с каждым устройством оказывается связан список неудовлетворенных запросов процессов, находящихся в состоянии ожидания, и запросов, выполняющихся в асинхронном режиме. Состояние ожидание расщепляется на набор очередей процессов, дожидающихся различных устройств ввода-вывода.

После завершения выполнения текущего запроса операционная система должна решить, какой из запросов в списке должен быть удовлетворен следующим, и инициировать его исполнение. Критерии и цели такого планирования мало отличаются от критериев и целей планирования процессов.

Задача планирования использования устройства обычно возлагается на *базовую подсистему ввода-вывода*, однако для некоторых устройств лучшие алгоритмы планирования могут быть тесно связаны с деталями их внутреннего функционирования. В таких случаях операция планирования переносится внутрь драйвера соответствующего устройства, так как эти детали скрыты от базовой подсистемы.

3.2 Алгоритмы планирования запросов к жесткому диску

Прежде чем приступить к непосредственному изложению самих алгоритмов, полезно вспомнить внутреннее устройство жесткого диска и определить, какие параметры запросов можно использовать для планирования.

3.2.1 Строение жесткого диска и параметры планирования

Жесткий магнитный диск представляет собой набор круглых пластин, находящихся на одной оси и покрытых магнитным слоем. 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. Поверхности пластин разделены на концентрические кольца, внутри которых хранится информация. Набор концентрических колец на всех пластинах для одного положения головок образует цилиндр. Каждое кольцо внутри цилиндра получило название дорожки. Все дорожки делятся на равное число секторов. Количество дорожек, цилиндров и секторов может варьироваться от одного жесткого диска к другому. Как правило, сектор является минимальным объемом информации, которая может быть прочитана с диска за один раз.

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.

При планировании использования жесткого диска естественным параметром планирования является время, которое потребуется для выполнения очередного запроса. Время, необходимое 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*);
- 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*);
- времени, необходимого 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 сложными соотношениями между номерами цилиндров и секторов предыдущего и планируемого запросов и скоростями вращения диска и перемещения головок. Разницей в задержках на вращение, как правило, пренебрегают.

3.2.2 Алгоритм 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 длительному общему времени обслуживания запросов.

Пример. Пусть на диске из 100 цилиндров (от 0 до 99) есть следующая очередь запросов: 23, 67, 55, 14, 31, 7, 84, 10 и головки в начальный момент находятся на 63-м цилиндре. Тогда положение головок будет меняться следующим образом:

$$63 \Rightarrow 23 \Rightarrow 67 \Rightarrow 55 \Rightarrow 14 \Rightarrow 31 \Rightarrow 7 \Rightarrow 84 \Rightarrow 10$$

и всего головки переместятся на 329 цилиндров. Неэффективность алгоритма хорошо иллюстрируется двумя последними перемещениями с 7 цилиндра через весь диск на 84 цилиндр и затем опять через весь диск на цилиндр 10. Простая замена порядка двух последних перемещений ($7 \Rightarrow 10 \Rightarrow 84$) позволила бы существенно сократить общее время обслуживания запросов.

3.2.3 Алгоритм Short Seek Time First (SSTF)

Разумным является первоочередное обслуживание запросов, данные для которых лежат рядом с текущей позицией головок, а уж затем далеко отстоящих. Алгоритм *Short Seek Time First (SSTF)* – короткое ищется первым по времени –исходит из этой позиции. Для очередного обслуживания выбирается запрос, данные для которого лежат наиболее близко к текущему положению магнитных головок. Естественно, что при наличии равноудаленных запросов решение о выборе между ними может приниматься исходя из различных соображений, например по алгоритму *FCFS*. Для предыдущего примера алгоритм даст такую последовательность положений головок:

$$63 \Rightarrow 67 \Rightarrow 55 \Rightarrow 31 \Rightarrow 23 \Rightarrow 14 \Rightarrow 10 \Rightarrow 7 \Rightarrow 84$$

и всего головки переместятся на 141 цилиндр.

Алгоритм похож на алгоритм *SJF* планирования процессов, если за аналог оценки времени очередного *CPU burst* процесса выбирать расстояние между текущим положением головки и положением, необходимым для удовлетворения запроса. Так же, как алгоритм *SJF*, он может приводить к длительному откладыванию выполнения какого-либо запроса (запросы в очереди могут появляться в любой момент времени): если все запросы, кроме одного, постоянно группируются в области с большими номерами цилиндров, то этот один запрос может находиться в очереди неопределенно долго.

Алгоритм *SSTF* не является оптимальным. Если перенести обслуживание запроса 67-го цилиндра в промежуток между запросами 7-го и 84-го цилиндров, то уменьшится общее время обслуживания. Это наблюдение приводит к идее целого семейства других алгоритмов – алгоритмов сканирования.

3.2.3 Алгоритмы сканирования (SCAN, C-SCAN, LOOK, C-LOOK)

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 направлении уменьшения номеров цилиндров. Тогда получим порядок обслуживания запросов:

63⇒55⇒31⇒23⇒14⇒10⇒7⇒0⇒67⇒84

и всего головки переместятся на 147 цилиндров.

Если знать, что обслужен последний попутный запрос в направлении движения головок, то можно 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*.

Допустим, что к моменту изменения направления движения головки в алгоритме *SCAN* (когда головка достигла одного из краев диска) у этого края накопилось большое количество новых запросов, на обслуживание которых будет потрачено достаточно много времени. 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 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

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

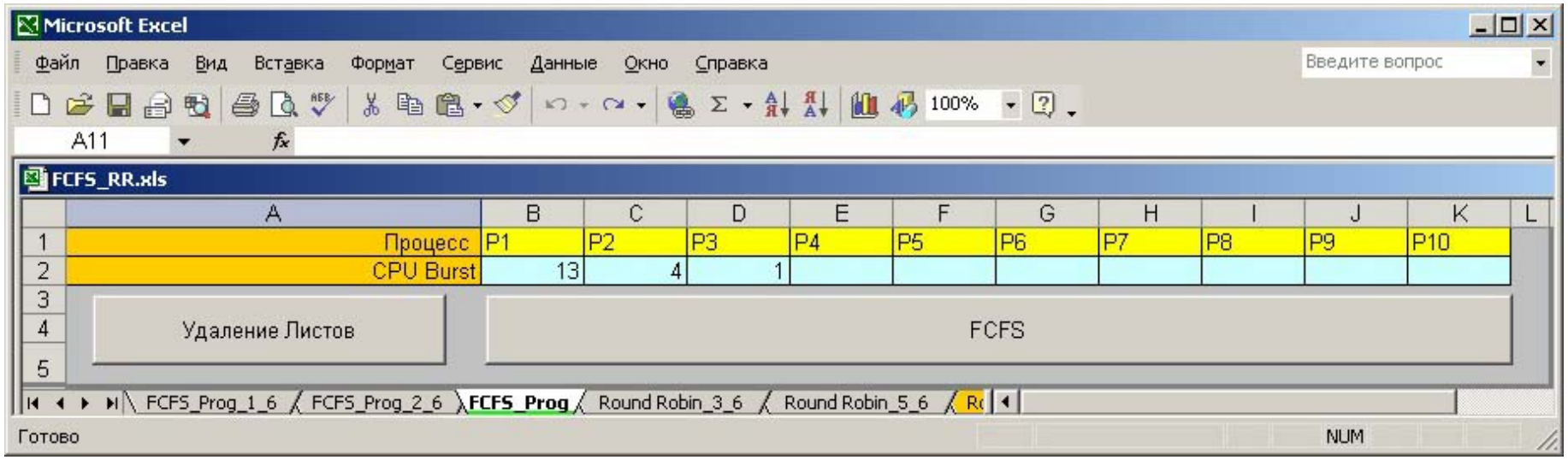
Задания:

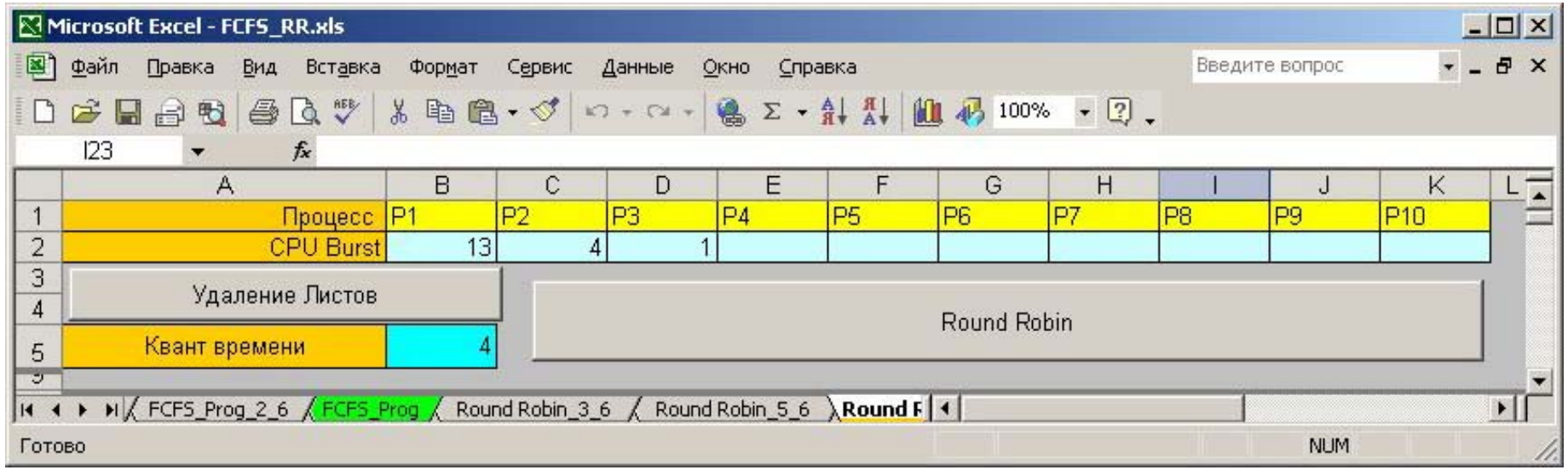
Задание 1. Пусть у нас имеется диск с 80 цилиндрами (от 0 до 79). Время перемещения головки между соседними цилиндрами составляет 1мс. Время же перевода головки с 79-го на 0-й цилиндр составляет всего 10 мс. В текущий момент времени головка находится на 45-м цилиндре и двигается в сторону увеличения номеров цилиндров. Сколько времени будет обрабатываться следующая последовательность запросов на чтение цилиндров: 10, 6, 15, 71, 1, 62, для алгоритма *SSTF* (временами чтения цилиндров и смены направления движения пренебречь)?

Задание 2. Пусть у нас имеется диск с 80 цилиндрами (от 0 до 79). Время перемещения головки между соседними цилиндрами составляет 1мс. Время же перевода головки с 79-го на 0-й цилиндр составляет всего 10 мс. В текущий момент времени головка находится на 45-ом цилиндре и двигается в сторону увеличения номеров цилиндров. Сколько времени будет обрабатываться следующая последовательность запросов на чтение цилиндров: 10, 6, 15, 71, 1, 62, для алгоритма *C-SCAN* (временами чтения цилиндров и смены направления движения пренебречь)?

Задание 3. Пусть у нас имеется диск с 80 цилиндрами (от 0 до 79). Время перемещения головки между соседними цилиндрами составляет 2 мс. В текущий момент времени головка находится на 23-м цилиндре и двигается в сторону увеличения номеров цилиндров. Сколько времени будет обрабатываться следующая последовательность запросов на чтение цилиндров: 11, 22, 10, 73, 1, 12, алгоритма *SCAN* (временами чтения цилиндров и смены направления движения головок пренебречь)?

Задание 4. На алгоритмическом языке VBA самостоятельно реализовать программу планирования использования жесткого диска с использованием алгоритмов FCFS, SSTF, SCAN, LOOK с определением числа перемещений головки для заданной строки запросов номеров цилиндров.


Microsoft Excel
Файл Правка Вид Вставка Формат Сервис Данные Окно Справка
Введите вопрос
100%
A11
FCFS_RR.xls
Процесс P1 P2 P3 P4 P5 P6 P7 P8 P9 P10
CPU Burst 13 4 1
Удаление Листов
FCFS
FCFS_Prog_1_6 FCFS_Prog_2_6 FCFS_Prog Round Robin_3_6 Round Robin_5_6
Готово
NUM


Microsoft Excel - FCFS_RR.xls
Файл Правка Вид Вставка Формат Сервис Данные Окно Справка
Введите вопрос
100%
I23
Процесс P1 P2 P3 P4 P5 P6 P7 P8 P9 P10
CPU Burst 13 4 1
Удаление Листов
Квант времени 4
Round Robin
FCFS_Prog_2_6 FCFS_Prog Round Robin_3_6 Round Robin_5_6
Готово
NUM

FCFS	Round Robin

Option Explicit
Private Sub CommandButton1_Click()
  Call Prepare("FCFS_Prog")
  Call FCFS
  Call View
  Call FCFSPar
  Call ParView
End Sub
Private Sub CommandButton2_Click()
  Call DeleteWS
End Sub
	Option Explicit
	Sub CommandButton1_Click()
	  Call Prepare("Round Robin")
	  Call RR
	  Call View
	  Call RRPAR
	  Call ParView
	End Sub
	Private Sub CommandButton2_Click()
	  Call DeleteWS
	End Sub

```
Dim TSum, PCount, i, j, PTime(0 To 10), PFTime(0 To 10), PriorityArr(0 To 10) As Integer
Dim TWSum, TFRSum, RRKvant, Time, VKvant, AppTime(1 To 10), Wait, Sum As Integer
Dim PName(1 To 10), c As String
'*************************************************************************
'TSum          Суммарное время
'PCount        Кол-во процессов
'i,j           Используемые в циклах переменные
'PTime(s)      Оставшаяся Продолжительность s-того процесса(RR)
'PFTime(s)     Полное время выполнения s-того процесса
'PName(s)      Название s-того процесса
'c             Название создаваемого листа
'TWSum         Суммарное время ожидания
'TFRSum        Суммарное время выполнения(RR)
'RRKvant       Квант времени(RR)
'Time          Текущее время(RR)
'VKvant        Выполненная часть кванта(RR)
'AppTime       Время появления процесса
'Wait          Время простоя системы
'Sum           Сумма CpuBurst процессов у кот AppTime(k)<AppTime(i)
'PriorityArr(s) Значение приоритета s-того процесса
'*************************************************************************
```

```
'Процедура создания листа результатов:
Sub Prepare(x As String)
'x- имя листа родителя
TSum = 0: PCount = 0: PFTime(0) = 0: TWSum = 0: TFRSum = 0: Time = 0
Wait = 0

For i = 1 To 10
If Worksheets(x).Cells(2, i+1)<>"" And Worksheets(x).Cells(2, i 1)<>"0" Then
      PCount = PCount + 1
      PName(PCount) = Worksheets(x).Cells(1, i + 1)
      PTime(PCount) = Worksheets(x).Cells(2, i + 1)
      TSum = TSum + Worksheets(x).Cells(2, i + 1)
End If
PFTime(PCount) = TSum
Next i

Worksheets.Add
c = x + "_" + CStr(ActiveSheet.Index) + "_" + CStr(Int(TSum / PCount))
ActiveSheet.Name = c
Worksheets(c).Cells(1, 1) = "Время"
For i = 1 To TSum
   Worksheets(c).Cells(1, i + 1) = i
Next i
For j = 1 To PCount
   Worksheets(c).Cells(j + 1, 1) = PName(j)
Next j

Worksheets(c).Range(Worksheets(c).Cells(1, 1),
Worksheets(c).Cells(PCount + 1, TSum + 1)).Select
With Selection
   .ColumnWidth = 2.29
   .HorizontalAlignment = xlCenter
   .VerticalAlignment = xlCenter
   .Borders.LineStyle = xlContinuous
   .Interior.ColorIndex = 15
End With
Worksheets(c).Range(Worksheets(c).Cells(1, 1),
Worksheets(c).Cells(1, TSum + 1)).Interior.ColorIndex = 33
Worksheets(c).Range(Worksheets(c).Cells(1, 1),
Worksheets(c).Cells(PCount + 1, 1)).Interior.ColorIndex = 33
Worksheets(c).Range(Worksheets(c).Cells(1, 1),
Worksheets(c).Cells(PCount + 1, 1)).Columns.AutoFit
End Sub
```

```
'Процедура графического представления алгоритма FCFS:
Sub FCFS()
For i = 1 To PCount
   For j = 1 To PFTime(i - 1)
      Worksheets(c).Cells(i + 1, j + 1) = "Г"
      Worksheets(c).Cells(i + 1, j + 1).Interior.ColorIndex = 3
   Next j
   For j = PFTime(i - 1) + 1 To PFTime(i)
      Worksheets(c).Cells(i + 1, j + 1) = "И"
      Worksheets(c).Cells(i + 1, j + 1).Interior.ColorIndex = 4
   Next j
Next i
End Sub
```

```
'Процедура графического представления алгоритма Round Robin:
Sub RR()
Changed = True
RRKvant = Worksheets("Round Robin").Cells(5, 2)
While Changed
Changed = False
   For i = 1 To PCount
      If PTime(i) > 0 Then
        Changed = True
        VKvant = 0
        While PTime(i) > 0 And VKvant < RRKvant
            PTime(i) = PTime(i) - 1
            VKvant = VKvant + 1
            Time = Time + 1
            Worksheets(c).Cells(i + 1, Time + 1) = "И"
            Worksheets(c).Cells(i + 1, Time + 1).Interior.ColorIndex = 4
            For j = 1 To PCount
               If PTime(j) > 0 And i <> j Then
               Worksheets(c).Cells(j + 1, Time + 1) = "Г"
               Worksheets(c).Cells(j + 1, Time + 1).Interior.ColorIndex = 3
               End If
            Next j
        Wend
      End If
   Next i
Wend
End Sub
```

```
'Подготовка вывода параметров эффективности алгоритма:
Sub View()
Worksheets(c).Range(Worksheets(c).Cells(PCount + 2, 1),
Worksheets(c).Cells(PCount + 5, TSum + 2)).Interior.ColorIndex = 48
Worksheets(c).Range(Worksheets(c).Cells(1, TSum + 2),
Worksheets(c).Cells(PCount + 5, TSum + 2)).Interior.ColorIndex = 48

Worksheets(c).Cells(PCount + 3, 1) = "Среднее время ожидания"
Worksheets(c).Range(Worksheets(c).Cells(PCount + 3, 1),
Worksheets(c).Cells(PCount + 3, 11)).Select
With Selection
   .HorizontalAlignment = xlRight
   .MergeCells = True
   .Interior.ColorIndex = 6
   .Borders.LineStyle = xlContinuous
End With

Worksheets(c).Cells(PCount + 4, 1) = "Среднее полное время выполнения"
Worksheets(c).Range(Worksheets(c).Cells(PCount + 4, 1),
Worksheets(c).Cells(PCount + 4, 11)).Select
With Selection
   .HorizontalAlignment = xlRight
   .MergeCells = True
   .Interior.ColorIndex = 6
   .Borders.LineStyle = xlContinuous
End With
End Sub

'Процедура определения параметров производительности FCFS:
'суммарного времени ожидания и суммарного времени выполнения
Sub FCFSPar()
For j = 1 To PCount - 1
   TWSum = TWSum + PFTime(j)
Next j
TFRSum = TWSum + PFTime(PCount)
End Sub

'Процедура определения параметров производительности RR:
'суммарного времени ожидания и суммарного времени выполнения
Sub RRPAR()
For i = 1 To PCount
   For j = 1 To TSum
      If Worksheets(c).Cells(i + 1, j + 1) = "Г" Then TWSum = TWSum + 1
   Next j: Next i
TFRSum = TWSum + TSum
End Sub
```

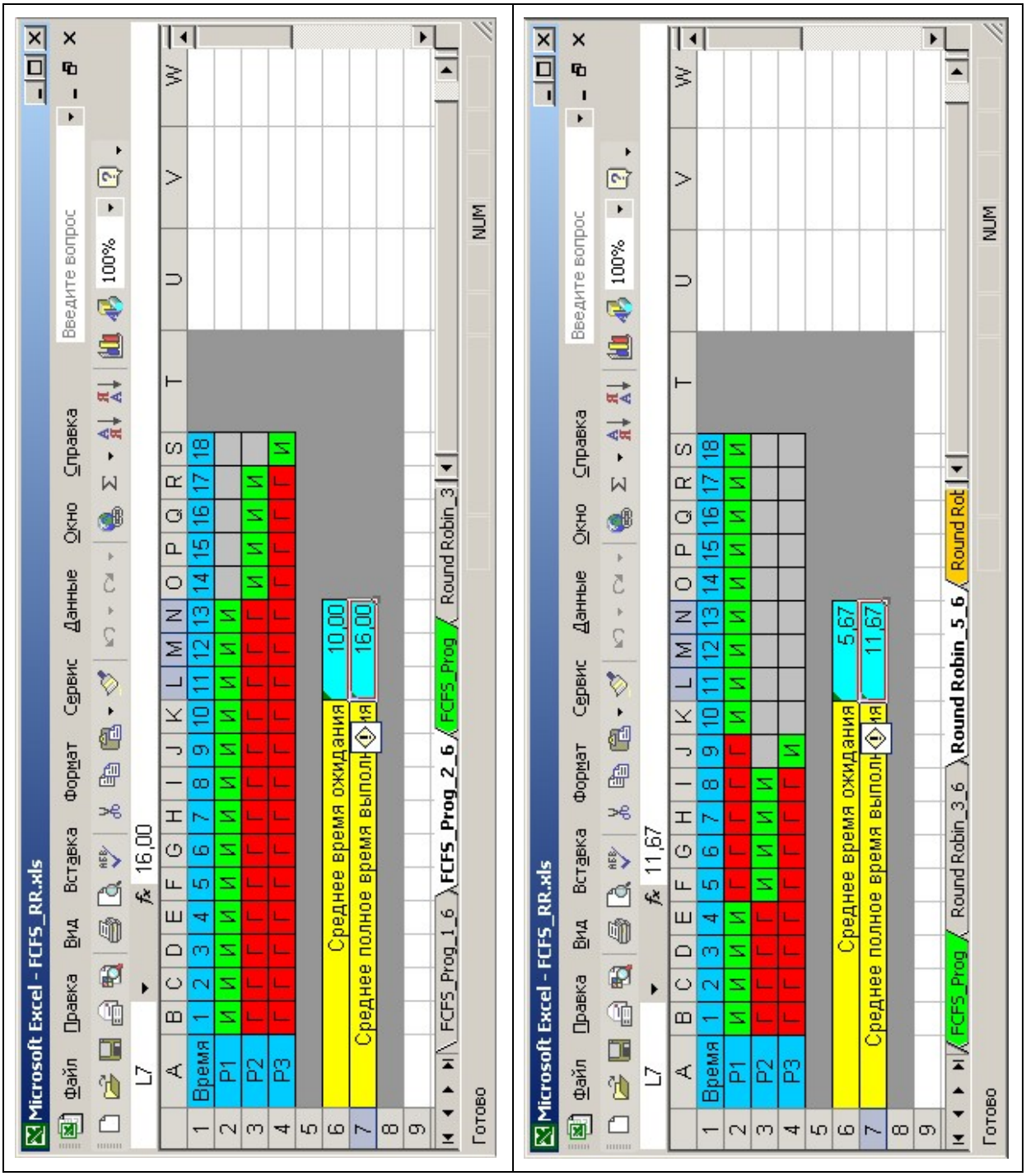
```
'Процедура вывода параметров производительности
Sub ParView()
Worksheets(c).Range(Worksheets(c).Cells(PCount + 3, 12),
Worksheets(c).Cells(PCount + 3, 14)).Select
With Selection
   .HorizontalAlignment = xlRight
   .MergeCells = True
   .Cells = Format(TWSum / PCount, "Fixed")
   .Interior.ColorIndex = 8
   .Borders.LineStyle = xlContinuous
End With

Worksheets(c).Range(Worksheets(c).Cells(PCount + 4, 12),
Worksheets(c).Cells(PCount + 4, 14)).Select
With Selection
   .HorizontalAlignment = xlRight
   .MergeCells = True
   .Cells = CStr(Format(TFRSum / PCount, "Standard"))
   .Interior.ColorIndex = 8
   .Borders.LineStyle = xlContinuous
End With
End Sub



'Процедура удаления сгенерированных листов
Sub DeleteWS()
Dim nm As Worksheet
If Worksheets.Count = 4 Then
   MsgBox "Нет сгенерированных листов для удаления!"
   Exit Sub
End If

For Each nm In Worksheets
   If nm.Name <> "FCFS_Prog" And nm.Name <> "Round Robin" Then
   nm.Delete
   End If
Next
End Sub
```


Microsoft Excel - FCFS_RR.xls
Файл Правка Вид Вставка Формат Сервис Данные Окно Справка
Введите вопрос
L7
16,00
Время 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
P1 И И И И И И И И И И И И
P2 Г Г Г Г Г Г Г Г Г Г Г Г И И И
P3 Г Г Г Г Г Г Г Г Г Г Г Г И
Среднее время ожидания 10,00
Среднее полное время выполнения 16,00
FCFS_Prog_1_6
FCFS_Prog_2_6
FCFS_Prog
Round Robin_3
Готово
NUM
Microsoft Excel - FCFS_RR.xls
Файл Правка Вид Вставка Формат Сервис Данные Окно Справка
Введите вопрос
L7
11,67
Время 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
P1 И И И Г Г Г Г И И И И И И И И
P2 Г Г Г Г И И И И
P3 Г Г Г Г Г Г Г И
Среднее время ожидания 5,67
Среднее полное время выполнения 11,67
FCFS_Prog
Round Robin_3_6
Round Robin_5_6
Round Rot
Готово
NUM

ЛИТЕРАТУРА:

а) основная:

1· Карпов В.Е., Коньков К.А. Основы операционных систем. Курс лекций. Учебное пособие.- М.: ИНТУИТ.РУ «Интернет Университет Информационных Технологий», 2005 – 536 с.

2· Коньков К.А. Устройство и функционирование ОС Windows. Курс лекций. Учебное пособие.- М.: ИНТУИТ.РУ «Интернет Университет Информационных Технологий», 2008 – 208 с.

б) дополнительная литература:

1· Гордеев А.В. Операционные системы: Учебник для вузов. – СПб.: Питер, 2004.- 416

2· Еремин Е.А. Популярные лекции об устройстве компьютера.- СПб.: БХВ-Петербург, 2003.- 272 с.

3· Иртегов Д.В. Введение в операционные системы. - СПб.: БХВ-Петербург, 2002.- 624

4· Курячий Г.В., Маслинский К.А. Операционная система LINUX. Курс лекций. Учебное пособие. .- М.: ИНТУИТ.РУ «Интернет Университет Информационных Технологий», 2005 – 392 с.

5· Нортон П. Персональный компьютер изнутри. - М.: Бином, 1995.- 448 с.

6· Таненбаум Э. Современные операционные системы СПб.: Издательский дом Питер, 2002

Средства обеспечения освоения дисциплины:

1· http://www.intuit.ru/department/os/osintro/ - Основы операционных систем.

2· http://www.intuit.ru/department/os/osmswin/ - Основы организации операционных систем Microsoft Windows

3· http://www.intuit.ru/department/os/linux/ - Операционная система Linux.