МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ "САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"

Алгоритмы планирования процессов

Методические указания к лабораторным работам

по курсу «Системное программирование»

2011

Автор: КУПРИЯНОВ Александр Викторович

Методические указания к лабораторной работе предназначены для бакалавров направления 010400.68 “Прикладная математика и информатика”.

1. ПРОЦЕССЫ В ОПЕРАЦИОННЫХ СИСТЕМАХ

Понятие процесса характеризует некоторую совокупность набора исполняющихся команд, ассоциированных с ним ресурсов (выделенная для исполнения память или адресное пространство, стеки, используемые файлы и устройства ввода-вывода и т. д.) и текущего момента его выполнения (значения регистров, программного счетчика, состояние стека и значения переменных), находящуюся под управлением операционной системы. Не существует взаимно однозначного соответствия между процессами и программами, обрабатываемыми вычислительными системами. Как будет показано в дальнейшем, в некоторых операционных системах для работы определенных программ может организовываться более одного процесса или один и тот же процесс может исполнять последовательно несколько различных программ. Более того, даже в случае обработки только одной программы в рамках одного процесса, нельзя считать, что процесс представляет собой просто динамическое описание кода исполняемого файла, данных и выделенных для них ресурсов. Процесс находится под управлением операционной системы и поэтому в нем может выполняться часть кода ее ядра (не находящегося в исполняемом файле!), как в случаях, специально запланированных авторами программы (например, при использовании системных вызовов), так и в непредусмотренных ими ситуациях (например, при обработке внешних прерываний)

При некоторых допущениях можно считать, что все, что выполняется в вычислительных системах (не только программы пользователей, но и, возможно, определенные части операционных систем), организовано как набор процессов. Понятно, что реально на однопроцессорной компьютерной системе в каждый момент времени может исполняться только один процесс. Для мультипрограммных вычислительных систем псевдопараллельная обработка нескольких процессов достигается с помощью переключения процессора с одного процесса на другой. Пока один процесс выполняется, остальные ждут своей очереди на получение процессора.

Процесс не может сам перейти из одного состояния в другое. Изменением состояния процессов занимается операционная система, совершая операции над ними. Количество таких операций в нашей модели пока совпадает с количеством стрелок на диаграмме состояний (рис. 1).

- создание процесса
- завершение процесса;
- приостановка процесса (перевод из состояния исполнение в состояние готовность)
- запуск процесса (перевод из состояния готовность в состояние исполнение);
- блокирование процесса (перевод из состояния исполнение в состояние ожидание)
- разблокирование процесса (перевод из состояния ожидание в состояние готовность);

Рис. 1 Диаграмма состояний процесса.

Операции создания и завершения процесса являются одноразовыми, так как применяются к процессу не более одного раза (некоторые системные процессы никогда не завершаются при работе вычислительной системы). Все остальные операции, связанные с изменением состояния процессов, будь то запуск или блокировка, как правило, являются многоразовыми.

2. ПЛАНИРОВАНИЕ ПРОЦЕССОВ

Планирование использования процессора впервые возникает в мультипрограммных вычислительных системах, где в состоянии готовность могут одновременно находиться несколько процессов. Именно для процедуры выбора из них одного процесса, который получит процессор в свое распоряжение, т.е. будет переведен в состояние исполнение, используется словосочетание планирование процессов.

Планирование использования процессора выступает в качестве краткосрочного планирования процессов. Оно проводится, к примеру, при обращении исполняющегося процесса к устройствам ввода-вывода или просто по завершении определенного интервала времени. Поэтому краткосрочное планирование осуществляется весьма часто, как правило, не реже одного раза в 100 миллисекунд. Выбор нового процесса для исполнения оказывает влияние на функционирование системы до наступления очередного аналогичного события, т. е. в течение короткого промежутка времени, что и обусловило название этого уровня планирования - краткосрочное.

В некоторых вычислительных системах бывает выгодно для повышения их производительности временно удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для дальнейшего выполнения. Такая процедура в англоязычной литературе получила название swapping, что можно перевести на русский язык как перекачка, хотя в профессиональной литературе оно употребляется без перевода - свопинг.

2.1 Критерии планирования

Выбор конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, которых мы хотим достичь, используя планирование. К числу таких целей можно отнести:

- Справедливость: гарантировать каждому заданию или процессу определенную часть времени использования процессора в компьютерной системе, стараясь не допустить возникновения ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя фактически не приступал к выполнению.
- Эффективность: постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов готовых к исполнению. В реальных вычислительных системах загрузка процессора колеблется от 40 до 90 процентов.
- Сокращение полного времени выполнения (turnaround time): обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением.
- Сокращение времени ожидания (waiting time): минимизировать время, которое проводят процессы в состоянии готовность и задания в очереди для загрузки.
- Сокращение времени отклика (response time): минимизировать время, которое требуется процессу в интерактивных системах для ответа на запрос пользователя.

Независимо от поставленных целей планирования желательно также, чтобы алгоритмы обладали следующими свойствами:

- Были предсказуемыми. Одно и то же задание должно выполняться приблизительно за одно и то же время. Применение алгоритма планирования не должно приводить, к примеру, к извлечению корня квадратного из 4 за сотые доли секунды при одном запуске и за несколько суток при втором запуске.
- Имели минимальные накладные расходы, связанные с их работой. Если на каждые 100 миллисекунд, выделенных процессу для использования процессора, будет приходиться 200 миллисекунд на определение того, какой именно процесс получит процессор в свое распоряжение, и на переключение контекста, то такой алгоритм, очевидно, использовать не стоит.
- Равномерно загружали ресурсы вычислительной системы, отдавая предпочтение тем процессам, которые будут занимать малоиспользуемые ресурсы.
- Обладали масштабируемостью, т. е. не сразу теряли работоспособность при увеличении нагрузки. Например, рост количества процессов в системе в два раза не должен приводить к увеличению полного времени выполнения процессов на порядок.

Многие из приведенных выше целей и свойств являются противоречивыми. Улучшая работу алгоритма с точки зрения одного критерия, мы ухудшаем ее с точки зрения другого. Приспосабливая алгоритм под один класс задач, мы тем самым дискриминируем задачи другого класса.

2.2 Параметры планирования

Необходимость алгоритма планирования зависит от задач, для которых будет использоваться операционная система.

Задачи алгоритмов планирования:

- Справедливость - каждому процессу справедливую долю процессорного времени
- Контроль над выполнением принятой политики
- Баланс - поддержка занятости всех частей системы (например: чтобы были заняты процессор и устройства ввода/вывода)
- Пропускная способность - количество задач в час
- Оборотное время - минимизация времени на ожидание обслуживания и обработку задач.
- Использование процесса - чтобы процессор всегда был занят.
- Время отклика - быстрая реакция на запросы
- Соразмерность - выполнение ожиданий пользователя (например: пользователь не готов к долгой загрузке системы)
- Окончание работы к сроку - предотвращение потери данных
- Предсказуемость - предотвращение деградации качества в мультимедийных системах (например: потерь качества звука должно быть меньше чем видео)

Для краткосрочного планирования вводятся динамические параметры. Деятельность любого процесса можно представить как последовательность циклов использования процессора и ожидания завершения операций ввода-вывода. Промежуток времени непрерывного использования процессора носит на английском языке название CPU burst, а промежуток времени непрерывного ожидания ввода-вывода - I/O burst. Для краткости изложения мы будем использовать термины CPU burst и I/O burst без перевода. Значения продолжительности последних и очередных CPU burst и I/O burst являются важными динамическими параметрами процесса.

Для оценки эффективности функционирования алгоритма планирования могут быть применены количественные показатели. Обозначим через t – процессорное время, необходимое процессу для выполнения (будем его называть длительностью процесса).

Обозначим через T – общее время пребывания процесса в системе. Эту величину – интервал между моментом ввода процесса в систему и моментом получения результатов – также называют иногда временем реакции процесса. Наряду со временем реакции могут быть полезны также и другие показатели.

Потерянное время M = T - t; определяет время, в течение которого процесс находился в системе, но не выполнялся.

Отношение реактивности R = t / T; показывает долю процессорного времени (времени выполнения) или долю потерянного времени в общем времени реакции.

Штрафное отношение P = T / t; показывает, во сколько раз общее время выполнения процесса превышает необходимое процессорное время.

Средние значения величин T, M, R и P могут служить количественными показателями эффективности.

3. АЛГОРИТМЫ ПЛАНИРОВАНИЯ ПРОЦЕССОВ

3.2 Алгоритм FCFS

FCFS (first come – first serve; первым пришел – первым обслуживается) – простейший алгоритм, работа, которой понятна из ее названия. Это алгоритм без вытеснения, то есть процесс, выбранный для выполнения на ЦП, не прерывается, пока не завершится (или не перейдет в состояние ожидания по собственной инициативе). FCFS обеспечивает минимум накладных расходов. Среднее потерянное время при применении этого алгоритма не зависит от длительности процесса, но штрафное отношение при равном потерянном времени будет большим для коротких процессов. Поэтому алгоритм FCFS считается лучшим для длинных процессов. Существенным достоинством этого алгоритма наряду с его простотой является то обстоятельство, что FCFS гарантирует отсутствие бесконечного откладывания процессов: любой поступивший в систему процесс будет, в конце концов, выполнен независимо от степени загрузки системы.

3.3 Алгоритм RR

RR (round robin – карусель) – простейший алгоритм с вытеснением. Процесс получает в свое распоряжение ЦП на некоторый квант времени Q (в простейшем случае размер кванта фиксирован). Если за время Q процесс не завершился, он вытесняется из ЦП и направляется в конец очереди готовых процессов, где ждет выделения ему следующего кванта, и т.д. Показатели эффективности RR существенно зависят от выбора величины кванта Q. RR обеспечивает наилучшие показатели, если длительность большинства процессов приближается к размеру кванта, но не превосходит его. Тогда большинство процессов укладываются в один квант и не становятся в очередь повторно. При величине кванта, стремящейся к бесконечности, RR вырождается в FCFS. При Q, стремящемся к 0, накладные расходы на переключение процессов возрастают настолько, что поглощают весь ресурс ЦП. RR обеспечивает наилучшие показатели справедливости: штрафное отношение P на большом участке длительностей процессов t остается практически постоянным. Только на участке t<Q штрафное отношение начинает изменяться и при уменьшении t от Q до 0 возрастает экспоненциально. Потерянное же время M существенно растет с увеличением длительности процесса.

3.4 Алгоритм SJF

SJF (shortest job first – самая короткая работа – первой) – невытесняющий алгоритм, в котором наивысший приоритет имеет самый короткий процесс. Для того чтобы применять этот алгоритм, должна быть известна длительность процесса – задаваться пользователем или вычисляться методом экстраполяции. Для коротких процессов SJN обеспечивает лучшие показатели, чем RR, как по потерянному времени, так и по штрафному отношению. SJN обеспечивает максимальную пропускную способность системы – выполнение максимального числа процессов в единицу времени, но показатели для длинных процессов значительно худшие, а при высокой степени загрузки системы активизация длинных процессов может откладываться до бесконечности. Штрафное отношение слабо изменяется на основном интервале значений t, но значительно возрастает для самых коротких процессов: такой процесс при поступлении в систему имеет самый высокий приоритет, но вынужден ждать, пока закончится текущий активный процесс.

3.5 Алгоритм PSJF

PSJF (preemptive SJN – SJN с вытеснением) – текущий активный процесс прерывается, если его оставшееся время выполнения больше, чем у новоприбывшего процесса. Алгоритм обеспечивает еще большее предпочтение коротким процессам перед длинными. В частности, в ней устраняется то возрастание штрафного отношения для самых коротких процессов, которое имеет место в SJN.

3.6 Алгоритм RR SJF

Модификация алгоритма RR с переупорядочиванием процессов в очереди в соответствии с оставшимся временем выполнения.

3.7 Приоритетное планирование

Алгоритм SJF представляет собой частный случай приоритетного планирования. При приоритетном планировании каждому процессу присваивается определенное числовое значение - приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс.

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. Это могут быть, например, определенные ограничения по времени использования процессора, требования к размеру памяти, число открытых файлов и используемых устройств ввода-вывода, отношение средних продолжительностей I/O burst к CPU burst и т. д. Внешние критерии исходят из таких параметров, как 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. При вытесняющем планировании процесс с более высоким приоритетом, появившийся в очереди готовых процессов, вытесняет исполняющийся процесс с более низким приоритетом. В случае невытесняющего планирования он просто становится в начало очереди готовых процессов. Рассмотрим примеры использования различных режимов приоритетного планирования.

3.8 Алгоритм HPRN

HPRN (highest penalty ratio next – с наибольшим штрафным отношением – следующий) – алгоритм без вытеснения, обеспечивающий наилучшие показатели справедливости. Это достигается за счет динамического переопределения приоритетов. Всякий раз при освобождении ЦП для всех готовых процессов вычисляется текущее штрафное отношение

$$p[i]=(w[i]+t[i])/t[i]$$

где i – номер процесса; w[i] – время, затраченное процессом на ожидание; t[i] – длительность процесса, предзаданная или прогнозируемая. Для только что поступившего процесса p[i]=1. ЦП отдается процессу, имеющему наибольшее значение p[i]. Для коротких процессов HPRN обеспечивает примерно те же показатели справедливости, что и SJN, для длинных – более близкие к FCFS. На большом диапазоне средних длительностей процессов показатели, обеспечиваемые HPRN, представляют среднее между SJN и

FCFS и слабо зависят от длительности. Еще одно достоинство HPRN в том, что во времени ожидания может учитываться (с некоторыми весовыми коэффициентами) и ожидание в других очередях и, таким образом, выполняется более полный учет загрузки системы. Существенным недостатком метода является необходимость перевычисления штрафного отношения для всех процессов при каждом переключении, что плохо согласуется с общей политикой минимизации накладных расходов в дисциплинах без вытеснения.

3.9 Алгоритм SRR

SRR (selfish RR – эгоистичный RR) – метод с вытеснением, дающий дополнительные преимущества выполняемым процессам, что позволяет повысить пропускную способность. Все процессы разделяются на две категории: новые и выбранные. Новыми считаются те процессы, которые не получили еще ни одного кванта времени ЦП, все остальные процессы – выбранные. При поступлении в систему каждому процессу дается некоторый приоритет P0, одинаковый для всех процессов, который в дальнейшем возрастает. В конце каждого кванта времени пересчитываются приоритеты всех процессов, причем приоритеты новых процессов возрастают на величину dA, а выбранных – на величину dB. ЦП отдается процессу с наивысшим приоритетом, а при равенстве приоритетов – тому, который раньше поставлен в очередь. Показатели алгоритма существенно зависят от выбранного соотношения между dA и dB. При dB/dA=0 алгоритм вырождается в обыкновенный RR, при dB/dA>=1 – в FCFS. Собственно алгоритм SRR обеспечивается в диапазоне значений 0<dB/dA<1.

3.10 Лотерейное планирование

Процессам раздаются "лотерейные билеты" на доступ к ресурсам. Планировщик может выбрать любой билет, случайным образом. Чем больше билетов у процесса, тем больше у него приоритет. Распределение квантов времени обычно осуществляют по алгоритму RR с приоритетами.

3.11 Алгоритм HLRR

HLRR (“half-life round-robin”). Алгоритм полураспада является модификацией алгоритма RR. С каждым i-м процессом связано некоторое приоритетное число P[i]. Чем оно меньше, тем выше приоритет процесса.

Каждый новый процесс получает некоторое исходное значение приоритетного числа P0, одинаковое для всех процессов. Кроме того, с каждым процессом связан счетчик процессорного времени U[i] с исходным значением 0. Процесс с наименьшим значением P[i] получает квант времени Q (при равенстве приоритетных чисел ЦП отдается процессу, ожидающему дольше). За время кванта интервальный таймер выдает несколько сигналов-прерываний с интервалом dT. По каждому такому прерыванию счетчик U[i] активного (только активного!) процесса увеличивается на 1.

Использование ЦП процессом заканчивается при истечении кванта или при переходе процесса в ожидание. При этом модифицируются счетчики процессорного времени всех (в том числе и неактивных) процессов:

U[i] = U[i] / 2

и для всех процессов перевычисляются приоритетные числа:

P[i] = P0 + U[i] / 2.

и модифицируется очередь выполнения процессов.

3.12 Многоуровневые очереди (Multilevel Queue)

Для систем, в которых процессы могут быть легко рассортированы на разные группы, был разработан другой класс алгоритмов планирования. Для каждой группы процессов создается своя очередь процессов, находящихся в состоянии готовность. Этим очередям приписываются фиксированные приоритеты. Например, приоритет очереди системных процессов устанавливается больше, чем приоритет очередей пользовательских процессов. А приоритет очереди процессов, запущенных студентами, - ниже, чем для очереди процессов, запущенных преподавателями. Это значит, что ни один пользовательский процесс не будет выбран для исполнения, пока есть хоть один готовый системный процесс, и ни один студенческий процесс не получит в свое распоряжение процессор, если есть процессы преподавателей, готовые к исполнению. Внутри этих очередей для планирования могут применяться самые разные алгоритмы. Так, например, для больших счетных процессов, не требующих взаимодействия с пользователем (фоновых процессов), может использоваться алгоритм FCFS, а для интерактивных процессов - алгоритм RR. Подобный подход, получивший название многоуровневых очередей, повышает гибкость планирования: для процессов с различными характеристиками применяется наиболее подходящий им алгоритм.

3.13 Алгоритм FB

FB (foreground-background – передний-задний планы) – очередь готовых процессов расщепляется на две подочереди – очередь переднего плана и очередь заднего плана. Очереди обслуживаются по алгоритмам RR, но очередь переднего плана имеет абсолютный приоритет: пока в ней есть процессы, очередь заднего плана не обслуживается. Новый процесс направляется в очередь переднего плана. Если процесс использовал установленное число N квантов в очереди переднего плана, но не завершился, он переводится в очередь заднего плана.

3.14 Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

Дальнейшим развитием алгоритма многоуровневых очередей является добавление к нему механизма обратной связи. Здесь процесс не постоянно приписан к определенной очереди, а может мигрировать из очереди в очередь, в зависимости от своего поведения.

Для простоты рассмотрим ситуацию, когда процессы в состоянии готовность организованы в 4 очереди. Планирование процессов между очередями осуществляется на основе вытесняющего приоритетного механизма. Чем выше располагается очередь, тем выше ее приоритет. Процессы в очереди 1 не могут исполняться, если в очереди 0 есть хотя бы один процесс. Процессы в очереди 2 не будут выбраны для выполнения, пока есть хоть один процесс в очередях 0 и 1. И, наконец, процесс в очереди 3 может получить процессор в свое распоряжение только тогда, когда очереди 0, 1 и 2 пусты. Если при работе процесса появляется другой процесс в какой-либо более приоритетной очереди, исполняющийся процесс вытесняется появившимся. Планирование процессов внутри очередей 0-2 осуществляется с использованием алгоритма RR, планирование процессов в очереди 3 основывается на алгоритме FCFS.

Родившийся процесс поступает в очередь 0. При выборе на исполнение он получает в свое распоряжение квант времени размером Q единиц. Если продолжительность его CPU burst меньше этого кванта времени, процесс остается в очереди 0. В противном случае, он переходит в очередь 1. Для процессов из очереди 1 квант времени имеет вели-

чину 2Q. Если процесс не укладывается в это время, он переходит в очередь 2. Если укладывается - остается в очереди 1. В очереди 2 величина кванта времени составляет 4Q единицы. Если и этого мало для непрерывной работы процесса, процесс поступает в очередь 3, для которой квантование времени не применяется, и, при отсутствии готовых процессов в других очередях, он может исполняться до окончания своего CPU burst. Чем больше значение продолжительности CPU burst, тем в менее приоритетную очередь попадает процесс, но тем на большее процессорное время он может рассчитывать для своего выполнения. Таким образом, через некоторое время все процессы, требующие малого времени работы процессора окажутся размещенными в высокоприоритетных очередях, а все процессы, требующие большого счета и с низкими запросами к времени отклика, - в низкоприоритетных.

Организация перемещения процессов из очередей с низкими приоритетами в очереди с большими приоритетами позволяет более полно учитывать изменение поведения процессов с течением времени.

4. ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

4.1. Исходные данные

- Вариант задания (предоставляется преподавателем);

4.2. Общий план выполнения работы

1 Реализовать предложенный алгоритм планирования.

2 Все параметры алгоритма планирования задаются пользователем.

3 Реализовать имитацию появления процессов в системе случайным образом. Для каждого процесса генерируется время его появление в системе, количество квантов необходимые для работы процесса и если необходимо приоритет.

4 Пользователь задает параметры генератора процессов: количество работающих процессов, ограничение на максимальное время выполнения каждого процесса и если необходимо количество допустимых приоритетов.

5 Для полученного графика появления процессов в системе применить алгоритм планирования.

6 Для каждого процесса со временем выполнения t вычислять:

- T – общее время пребывания процесса в системе.
- Потерянное время M = T - t;
- Отношение реактивности R = t / T;
- Штрафное отношение P = T / t;

7 Вычислить средние значения приведенных выше параметров при использовании алгоритма планирования процессов.

8 Для сравнения реализовать также простой алгоритм планирования указанный в задании. Для простого алгоритма планирования также вычислять средние значения параметров.

9 Осуществить отображение на экране результатов планирования и выполнения процессов виде схемы, графика или таблицы.

10 Для реализации приложения допускается использовать любой язык программирования.

11 Провести тестирование работы программы необходимо для трех различных наборов входных параметров и трех различных графиков появления процессов в системе.

12 Написать отчет о проделанной работе. В отчете привести **подробное описание и визуальную схему алгоритма планирования**, а также **подробное сравнение реализованного алгоритма планирования с простым алгоритмом**.

13 Сделать выводы и написать заключение.

4.3. Содержание отчета

Отчет по работе должен содержать:

- Вариант задания.
- Описание и схема алгоритма планирования.
- Текст программы с комментариями.
- Исходные данные и листинг работы программы.
- Сравнение и анализ результатов планирования.
- Выводы и заключение о проделанной работе.

5. ВАРИАНТЫ ЗАДАНИЙ НА ЛАБОРАТОРНУЮ РАБОТУ

№	Алгоритм планирования	Простой алгоритм планирования для сравнения
1.	**SJF с приор.**	FCFS
2.	**RR с приор.**	PSJF
3.	**RR с приор.**	FCFS
4.	**HPRN**	RR
5.	**HPRN**	SJF
6.	**HPRN**	FCFS
7.	**SRR**	FCFS
8.	**SRR**	SJF
9.	**SRR**	RR
10.	**Лотерейное план.**	FCFS
11.	**Лотерейное план.**	SJF
12.	**Лотерейное план.**	RR
13.	**HLRR**	FCFS
14.	**HLRR**	SJF
15.	**HLRR**	RR
16.	**FB**	RR
17.	**FB**	SJF
18.	**MFQ**	FCFS
19.	**MFQ**	SJF
20.	**MFQ**	**FB**

6. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. А. Ю. Молчанов Системное программное обеспечение: /. - СПб.; М.; Нижний Новгород: Питер: Питер принт, 2003. - 395 с.
2. Э. Таненбаум Современные операционные системы. 2-е изд. - СПб.: Питер, 2007. - 1038 с.
3. Х. М. Дейтел, П. Д. Дейтел, Д. Р. Чофнес Операционные системы: [в 2 т.] : пер. с англ./; под ред. С. М. Молявко. - М.: Бином-Пресс, 2006 - 1023 с.

СОДЕРЖАНИЕ