Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Оренбургский государственный университет»

Кафедра информатики

В.В. Извозчикова

ОПЕРАЦИОННЫЕ СИСТЕМЫ, СРЕДЫ И ОБОЛОЧКИ

Методические указания

Рекомендовано к изданию редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по образовательной программе высшего образования по направлению подготовки 09.03.02 Информационные системы и технологии

Оренбург
2019

УДК 004.65(076)
ББК 32.973-018.2я7
И34

Рецензент – доцент, кандидат педагогических наук М. И. Глотова

Извозчикова, В. В.
И34 Операционные системы, среды и оболочки : методические указания / В. В. Извозчикова; Оренбургский гос. ун.-т. – Оренбург : ОГУ, 2019.

Методические указания содержат общие рекомендации по выполнению и оформлению курсовой работы и расчетно-графического задания, варианты заданий.

Методические указания предназначены для выполнения курсовой работы по дисциплине «Операционные системы, среды и оболочки» и расчетно-графического задания по дисциплине «Инструментальные средства информационных систем» для обучающихся по направлению подготовки 09.03.02 Информационные системы и технологии

УДК 004.65(076)
ББК 32.973-018.2я7

Содержание

Введение 4
1 Управление процессами 5
1.1 Алгоритмы планирования процессов 5
1.2 Алгоритмы организации взаимодействия процессов 12
1.3 Способы борьбы с тупиками 17
2 Методы распределения памяти 23
2.1 Типы адресов 23
2.2 Распределение памяти без использования внешней памяти 25
2.3 Распределение памяти с использованием внешней памяти 26
3 Организация файловой системы 29
3.1 Иерархия каталогов 30
3. 2 Способы логической организации файлов 31
3.3 Физическая организация файла 34
4 Моделирование алгоритмов управления процессами 36
4.1 Выбор языка программирования 36
4. 2 Разработка алгоритма управления процессами 37
4.3 Выполнение тестовой проверки 39
5 Литература, рекомендуемая для изучения 43
Приложение А *(обязательное)* Варианты заданий 46

Введение

Предмет изучения раздела «Операционные системы, среды и оболочки» – принципы построения, организация, основные функции, режимы работы и средства операционных систем, обеспечивающих функционирование современных компьютеров, информационных и вычислительных систем различного назначения.

Целью освоения раздела «Операционные системы, среды и оболочки» является получение знаний основ построения, функционирования и использования ОС и средств поддержки процессов решения задач на персональном компьютере (ПК) и в среде некоторой прикладной автоматизированной или информационной системы.

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, устройствами), особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и другими свойствами.

Чтобы разобраться в принципах работы операционной системы, необходимо изучить различные алгоритмы и методы работы с определенными данными. Важно знать, что из себя представляют алгоритмы управления процессами (планирование процессов, организация взаимодействия между процессами, способы борьбы с тупиками), методы распределения памяти и организации файловой системы (иерархия каталогов, способы логической организации, физическая организация).

Целью курсовой работы и расчетно-графического задания является закрепление, углубление и контроль знаний, полученных в процессе изучения дисциплин «Операционные системы, среды и оболочки» и «Инструментальные средства информационных систем».

Задачи:

– исследование и анализ алгоритмов управления ресурсами ОС;

– выбор языка программирования и разработка алгоритма программы;

– разработка программы, моделирующий один из алгоритмов управления ресурсами операционной системой;

– проверка алгоритма на конкретном примере.

1 Управление процессами

Важнейшей частью операционной системы является подсистема управления процессами. Процесс – это программа пользователя в ходе ее выполнения в компьютерной системе. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

1.1 Алгоритмы планирования процессов

1.1.1 Уровни планирования.

Существует два вида планирования в вычислительных системах: планировании заданий и планировании использования процессора. Планирование заданий появилось в пакетных системах после того, как для хранения сформированных пакетов заданий начали использоваться магнитные диски. Магнитные диски, будучи устройствами прямого доступа, позволяют загружать задания в компьютер в произвольном порядке, а не только в том, в котором они были записаны на диск. Изменяя порядок загрузки заданий в вычислительную систему, можно повысить эффективность ее использования. Процедуру выбора очередного задания для загрузки в машину, т. е. для порождения соответствующего процесса, называется планированием заданий. 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», называется планированием процессов. Оба этих вида планирования можно рассматривать, как различные уровни планирования процессов [1-3].

Планирование заданий выступает в качестве долгосрочного планирования процессов. Оно отвечает за порождение новых процессов в системе, определяя ее степень мультипрограммирования, т. е. количество процессов, одновременно находящихся в ней. Если степень мультипрограммирования системы поддерживается постоянной, т. е. среднее количество процессов в компьютере не меняется, то новые процессы могут появляться только после завершения ранее загруженных. Поэтому долгосрочное планирование осуществляется достаточно редко, между появлением новых процессов могут проходить минуты и даже десятки минут. Решение о выборе для запуска того или иного процесса оказывает влияние на функционирование вычислительной системы на протяжении достаточно длительного интервала времени. В некоторых операционных системах долгосрочное планирование сведено к минимуму или совсем отсутствует. Так, например, во многих интерактивных системах разделения времени порождение процесса происходит сразу после появления соответствующего запроса. Поддержание разумной степени мультипрограммирования осуществляется за счет ограничения количества пользователей, которые могут работать в системе, и человеческой психологии. Если между нажатием на клавишу и появлением символа на экране проходит 20-30 секунд, то многие пользователи предпочтут прекратить работу и продолжить ее, когда система будет менее загружена.

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.

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 лите-

ратуре оно употребляется без перевода – свопинг. Когда и какой из процессов нужно перекачать на диск и вернуть обратно, решается дополнительным промежуточным уровнем планирования процессов– среднесрочным.

1.1.2 Параметры планирования.

Для осуществления поставленных целей разумные алгоритмы планирования должны опираться на какие-либо характеристики процессов в системе, заданий в очереди на загрузку, состояния самой вычислительной системы, иными словами, на параметры планирования. В этом разделе мы опишем ряд таких параметров, не претендуя на полноту изложения.

Все параметры планирования можно разбить на две большие группы: статические параметры и динамические параметры. Статические параметры не изменяются в ходе функционирования вычислительной системы, динамические же, напротив, подвержены постоянным изменениям.

К статическим параметрам вычислительной системы можно отнести предельные значения ее ресурсов (размер оперативной памяти, максимальное количество памяти на диске для осуществления свопинга, количество подключенных устройств ввода-вывода и т. п.). Динамические параметры системы описывают количество свободных ресурсов в текущий момент времени.

К статическим параметрам процессов относятся характеристики, как правило, присущие заданиям уже на этапе загрузки:

–каким пользователем запущен процесс или сформировано задание;

–насколько важной является поставленная задача, т. е. каков приоритет ее выполнения;

–сколько процессорного времени запрошено пользователем для решения задачи;

–каково соотношение процессорного времени и времени, необходимого для осуществления операций ввода-вывода;

–какие ресурсы вычислительной системы (оперативная память, устройства ввода-вывода, специальные библиотеки и системные программы и т. д.) и в каком количестве необходимы заданию[4,5].

Алгоритмы долгосрочного планирования используют в своей работе статические и динамические параметры вычислительной системы и статические параметры процессов (динамические параметры процессов на этапе загрузки заданий еще не известны). Алгоритмы краткосрочного и среднесрочного планирования дополнительно учитывают и динамические характеристики процессов. Для среднесрочного планирования в качестве таких характеристик может выступать следующая информация:

–сколько времени прошло со времени выгрузки процесса на диск или его загрузки в оперативную память;

–сколько оперативной памяти занимает процесс;

–сколько процессорного времени было уже предоставлено процессу.

Для краткосрочного планирования нам понадобится ввести еще два динамических параметра. 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 *CPUburst*, а промежуток времени непрерывного ожидания ввода-вывода – *I/Oburst*.

1.1.3 Вытесняющее и невытесняющее планирование.

Процесс планирования осуществляется частью операционной системы, называемой планировщиком. Планировщик может принимать решения о выборе для исполнения нового процесса, из числа находящихся в состоянии готовность, в следующих четырех случаях.

1. Когда процесс переводится из состояния исполнение в состояние завершение.

2. Когда процесс переводится из состояния исполнение в состояние ожидание.

3. Когда процесс переводится из состояния исполнение в состояние готовность (например, после прерывания от таймера).

4. Когда процесс переводится из состояния ожидание в состояние готовность (завершилась операция ввода-вывода или произошло другое событие).

5. В случаях 1 и 2 процесс, находившийся в состоянии исполнение, не может дальше исполняться, и для выполнения всегда необходимо выбрать новый процесс. В

случаях 3 и 4 планирование может не проводиться, процесс, который исполнялся до прерывания, может продолжать свое выполнение после обработки прерывания. Если планирование осуществляется только в случаях 1 и 2, говорят, что имеет место невытесняющее (nonpreemptive) планирование. В противном случае говорят о вытесняющем (preemptive) планировании. Термин “вытесняющее планирование” возник потому, что исполняющийся процесс помимо своей воли может быть вытеснен из состояния исполнение другим процессом.

1.1.4 Алгоритмы планирования.

Планирование процессов включает в себя решение следующих задач:

– определение момента времени для смены выполняемого процесса;

– выбор процесса на выполнение из очереди готовых процессов;

– переключение контекстов "старого" и "нового" процессов.

Первые две задачи решаются программными средствами, а последняя в значительной степени аппаратно.

Существует множество различных алгоритмов планирования процессов, поразному решающих вышеперечисленные задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Среди этого множества алгоритмов рассмотрим подробнее две группы наиболее часто встречающихся алгоритмов: алгоритмы, основанные на квантовании, и алгоритмы, основанные на приоритетах[6-8].

1.1.4.1 Алгоритм планирования процессов вытесняющий RR.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если: процесс завершился и покинул систему, произошла ошибка, процесс перешел в состояние «ожидание», исчерпан квант процессорного времени, отведенный данному процессу.Процесс, который исчерпал свой квант, переводится в состояние «готовность» и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый процесс из очереди готовых. Таким образом, ни один процесс не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени.

Модификацией алгоритма FCFS является алгоритм, получивший название RoundRobin (RoundRobin – это вид детской карусели в США) или сокращенно RR. Поведение алгоритма представлено на рисунке 1.1. По сути дела, это тот же самый алгоритм, только реализованный в режиме вытесняющего планирования. Можно представить себе все множество готовых процессов организованным циклически – процессы сидят на карусели. Карусель вращается так, что каждый процесс находится около процессора небольшой фиксированный квант времени, обычно 10–100 миллисекунд. Пока процесс находится рядом с процессором, он получает процессор в свое распоряжение и может исполняться. Реализуется такой алгоритм так же, как и FCFS, с помощью организации процессов, находящихся в состоянии готовность, в очередь FIFO. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении кванта времени.

Рисунок 1.1 – Процесс работы алгоритма RR

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 собственном виртуальном процессоре с производительностью ~ 1/n от производительности реального процессора. Правда, это

справедливо лишь при теоретическом анализе при условии пренебрежения временами переключения контекста процессов. В реальных условиях при слишком малой величине кванта времени и, соответственно, слишком частом переключении контекста накладные расходы на переключение резко снижают производительность системы.

1.1.4.2 Приоритетное планирование предполагает, что каждому процессу присваивается определенное числовое значение – приоритет, в соответствии с которым ему выделяется процессор. Чем меньше значение числа, тем выше приоритет процесса, тем больше у него привилегий. Процессы с одинаковыми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst (время непрерывного использования). 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 у процесса приоритет.

Алгоритмы назначения приоритетов процессов опираются на внутренние и внешние факторы по отношению к определенной системе.

К внутренним параметрам относятся различные количественные и качественные характеристики процесса такие как:

– ограничения по времени использования процессора;

– требования к памяти;

– количество открытых файлов и используемых устройств ввода–вывода;

– отношение средних продолжительностей I/O burst к CPU burst.

В качестве внешних параметров могут выступать:

– важность процесса;

– стоимость оплаченного процессорного времени.

Внутренние факторы могут использоваться для автоматического назначения приоритетов самой операционной системой, а внешние для принудительного, с помощью оператора.

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.

Главная проблема приоритетного планирования заключается в том, что при ненадлежащем выборе механизма назначения и изменения приоритетов низкоприоритетные процессы могут не запускаться неопределенно долгое время. Обычно случается одно из двух. Или они все же дожидаются своей очереди на исполнение или вычислительную систему приходится выключать, и они теряются (при остановке IBM 7094 в Массачусетском технологическом институте в 1973 году были найдены процессы, запущенные в 1967 году и ни разу с тех пор не исполнявшиеся). Решение этой проблемы может быть достигнуто с помощью увеличения со временем значения приоритета процесса, находящегося в состоянии готовность. Пусть изначально процессам присваиваются приоритеты от 128 до 255. Каждый раз по истечении определенного промежутка времени значения приоритетов готовых процессов уменьшаются на 1. Процессу, побывавшему в состоянии исполнение, присваивается первоначальное значение приоритета. Даже такая грубая схема гарантирует, что любому процессу в разумные сроки будет предоставлено право на исполнение.

1.2 Алгоритмы организации взаимодействия процессов

Сформулируем пять условий, которые должны выполняться для хорошего программного алгоритма организации взаимодействия процессов, имеющих критические участки, если они могут проходить их в произвольном порядке[5,9].

1. Задача должна быть решена чисто программным способом на обычной машине, не имеющей специальных команд взаимоисключения. При этом предполагается, что основные инструкции языка программирования (такие примитивные инструкции, как load, store, test) являются атомарными операциями.

2. Не должно существовать никаких предположений об относительных скоростях выполняющихся процессов или числе процессоров, на которых они исполняются.

3. Если процесс P_i исполняется в своем критическом участке, то не существует никаких других процессов, которые исполняются в соответствующих критических секциях. Это условие получило название условия взаимоисключения (mutualexclusion).

4. Процессы, которые находятся вне своих критических участков и не собираются входить в них, не могут препятствовать другим процессам входить в их собственные критические участки. Если нет процессов в критических секциях и имеются процессы, желающие войти в них, то только те процессы, которые не исполняются в remainder section, должны принимать решение о том, какой процесс войдет в свою критическую секцию. Такое решение не должно приниматься бесконечно долго. Это условие получило название условия прогресса (progress).

5. Не должно возникать неограниченно долгого ожидания для входа одного из процессов в свой критический участок. От того момента, когда процесс запросил разрешение на вход в критическую секцию, и до того момента, когда он это разрешение получил, другие процессы могут пройти через свои критические участки лишь ограниченное число раз. Это условие получило название условия ограниченного ожидания (boundwaiting).

1.2.1 Алгоритм булочной (Bakeryalgorithm).

Давайте рассмотрим теперь соответствующий алгоритм для n взаимодействующих процессов, который получил название алгоритм булочной, хотя применительно к нашим условиям его следовало бы скорее назвать алгоритм регистратуры в поликлинике. Основная его идея выглядит так. Каждый вновь прибывающий клиент (он же процесс) получает талончик на обслуживание с номером. Клиент с наименьшим номером на талончике обслуживается следующим. К сожалению, из-за неатомарности операции вычисления следующего номера алгоритм булочной не гарантирует, что у всех процессов будут талончики с разными номерами. В случае равенства номеров на талончиках у двух или более клиентов первым обслуживается клиент с меньшим значением имени (имена можно сравнивать в лексикографическом порядке).

1.2.2 Семафорные примитивы чрезвычайно широко используются при проектировании разнообразных вычислительных процессов. При этом некоторые задачи являются настолько «типичными», что их детальное рассмотрение уже стало классическим в соответствующих учебных пособиях. Не будем делать исключений и мы.

Задача «поставщик-потребитель».Решение задачи «поставщик-потребитель» является характерным примером использования семафорных операций. Содержательная постановка этой задачи уже была нами описана в начале этой главы. Разделяемыми переменными здесь являются счетчики свободных и занятых буферов, которые должны быть защищены со стороны обоих процессов, то есть действия по посылке и получению сообщений должны быть синхронизированы.

Другой важной и часто встречающейся задачей, решение которой также требует синхронизации, является задача «читатели-писатели». Эта задача имеет много вариантов. Наиболее характерная область ее использования — построение систем управления файлами и базами данных, информационно-справочных систем. Два класса процессов имеют доступ к некоторому ресурсу (области памяти, файлам). «Читатели» — это процессы, которые могут параллельно считывать информацию из некоторой общей области памяти, являющейся критическим ресурсом. «Писатели» — это процессы, записывающие информацию в эту область памяти, исключаядруг друга, а также процессы «читатели». Имеются различные варианты взаимодействия между писателями и читателями. Наиболее широко распространены следующие условия. Устанавливается приоритет в использование критического ресурса процессам «читатели». Это означает, что если хотя бы один читатель пользуется ресурсом, то он закрыт для всех писателей и доступен для всех читателей. Во втором варианте, наоборот, больший приоритет у процессов «писатели». При появлении запроса от писателя необходимо закрыть дальнейший доступ всем тем читателям, которые запросят критический ресурс после него. Помимо системы управления файлами другим типичным примером решения задачи «читатели-писатели» может служить система автоматизированной продажи билетов. Процессы «читатели» обеспечивают нас справочной информацией о наличии свободных билетов на тот или иной рейс. Обычно программы, решающие проблему «читатели-писатели», используют как семафо-

ры, так и мониторные схемы с взаимным исключением, то есть такие, которые блокируют доступ к критическим ресурсам для всех остальных процессов, если один из них модифицирует значения общих переменных. Взаимное исключение требует, чтобы писатель ждал завершения всех текущих операций чтения. При условии, что писатель имеет более высокий приоритет, чем читатель, такое ожидание в ряде случаев весьма нежелательно. Кроме того, реализация принципа взаимного исключения в многопроцессорных системах может вызвать определенную избыточность.

1.2.3 Для того, чтобы облегчить написание корректных программ, Чарльзом Хоаром в 1974 году было предложено высокоуровневое средство синхронизации, называемое монитором (рисунок 1.2).

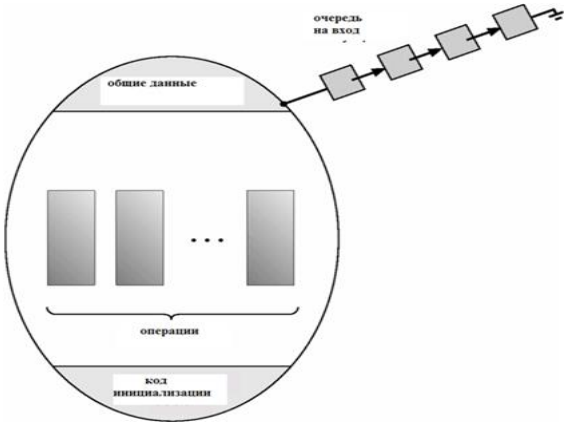

Рисунок 1.2 – Схематическое изображение монитор

Монитор представляет собой набор процедур, переменных и других структур данных, объединенных в программный модуль, который обеспечивает функциональность, эквивалентную функциональности семафора (рисунок 2). Основные характеристики монитора:

– локальные переменные монитора доступны только его процедурам и не доступны внешним процедурам;

– процесс входит в монитор путем вызова одной из его процедур;

– в мониторе может находиться только один активный процесс и любой другой процесс, вызвавший монитор, будет приостановлен в ожидании доступности монитора.

Компилятор обрабатывает вызовы процедур монитора особым образом. Обычно, когда процесс вызывает процедуру монитора, то первые несколько инструкций этой процедуры проверяют, не активен ли какой–либо другой процесс по отношению к этому монитору. Если да, то вызывающий процесс приостанавливается, пока другой процесс не освободит монитор. Таким образом, исключение входа нескольких процессов в монитор реализуется не программистом, а компилятором, что делает ошибки менее вероятными.

Однако этого было недостаточно чтобы в полном объеме реализовать решение задач, возникающих при взаимодействии процессов. Нужны еще и средства организации очередности процессов, подобно семафорам. Для этого в мониторах было введено понятие условных переменных, над которыми можно совершать две операции – wait и signal.

Если функция монитора не может выполняться дальше, пока не наступит некоторое событие, она выполняет операцию wait над какой–либо условной переменной. При этом процесс, выполнивший операцию wait, блокируется, становится неактивным, и другой процесс получает возможность войти в монитор.

Когда ожидаемое событие происходит, другой процесс внутри функции–метода совершает операцию signal над той же самой условной переменной. Это приводит к пробуждению ранее заблокированного процесса, и он становится активным. Если несколько процессов дожидались операции signal для этой переменной, то активным становится только один из них. Что нужно предпринять для того, чтобы не оказалось двух процессов, разбудившего и пробужденного, одновременно активных внутри монитора? Хоар предложил, чтобы пробужденный процесс подавлял исполнение разбудившего процесса, пока он сам не покинет монитор. Несколько позже Хансен предложил другой механизм: разбудивший процесс покидает монитор немедленно после исполнения операции signal.

В распределенных системах, состоящих из нескольких процессоров, каждый из которых имеет собственную оперативную память, семафоры и мониторы оказываются непригодными. В таких системах синхронизация может быть реализована только с помощью обмена сообщениями.

1.3 Способы борьбы с тупиками

Если средствами синхронизации пользоваться неосторожно, то могут возникнуть непредвиденные затруднения. Предположим, что несколько процессов конкурируют за обладание конечным числом ресурсов. Если запрашиваемый процессом ресурс недоступен, процесс переходит в состояние ожидания. В случае если требуемый ресурс удерживается другим ожидающим процессом, то первый процесс не сможет сменить свое состояние. Такая ситуация называется тупиком. В мультипрограммной системе процесс находится в состоянии тупика, дедлока (deadlock) или клинча, 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[4-6].

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 (рисунок 1.3).

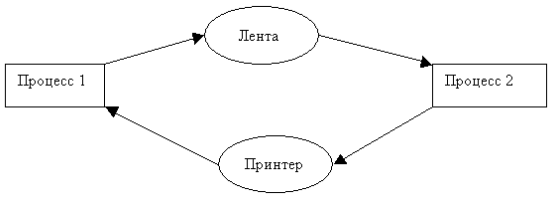

Рисунок 1.3– Пример тупиковой ситуации

Тупики также могут иметь место в ситуациях, не требующих выделенных ресурсов. Например, в системах управления базами данных процессы могут локализовывать записи, чтобы избежать гонок. В этом случае может получиться так, что один из процессов заблокировал записи, требуемые другому процессу и наоборот. Т.о. тупики могут иметь место, как на аппаратных, так и на программных ресурсах.

Другой пример возникновение тупика в системах спулинга. Режим спулинга ввод-вывод с буферизацией информации, предназначенной для печати, на диске и организации очереди на печать часто применяется для повышения производительности системы. Программа, осуществляющая вывод на печать должна полностью

сформировать свои выходные данные в промежуточном файле, после чего начинается реальная распечатка. В итоге, несколько заданий может оказаться в тупиковой ситуации, если предусмотренная емкость буфера для промежуточных файлов будет заполнена до того, как одно из заданий закончит свою работу. Возможные решения: увеличить размер буфера, или не принимать дополнительные задания, если файл спулинга близок к какому то порогу насыщения, например, заполнен на 75%.

Множество процессов находится в тупиковой ситуации, если каждый процесс из множества ожидает события, которое только другой процесс данного множества может вызвать. 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. Обычно событие, которого ждет процесс в тупиковой ситуации - освобождение ресурса.

Во многих случаях цена борьбы с тупиками, которую приходится платить высока. Тем не менее, для ряда систем, например для систем реального времени, нет иного выхода.

1.3.1 Проанализируем условия возникновения тупиков.

В 1971 г. Коффман, Элфик и Шошани сформулировали следующие четыре условия для возникновения тупиков.

1.Условие взаимоисключения (Mutualexclusion). Каждый ресурс выделен в точности одному процессу или доступен. Процессы требуют предоставления им монопольного управления ресурсами, которые им выделяются.

2.Условие ожидания ресурсов (Holdandwait). Процессы удерживают за собой ресурсы, уже выделенные им, ожидая в то же время выделения дополнительных ресурсов (которые при этом обычно удерживаются другими процессами).

3. Условие неперераспределяемости (Nopreemtion). Ресурс, данный ранее, не может быть принудительно забран у процесса. Освобождены они могут быть только процессом, который их удерживает.

4.Условие кругового ожидания (Circularwait). Существует кольцевая цепь процессов, в которой каждый процесс удерживает за собой один или более ресурсов, требующихся другим процессам цепи.

Для тупика необходимо выполнение всех четырехусловий.

Обычно тупик моделируется прямым графом, наподобие того, что изображен на рис. 7.1, состоящим из узлов двух видов: прямоугольников процессов и эллипсов ресурсов. Стрелки, направленные от ресурса к процессу, показывают, что ресурс выделен данному процессу.

1.3.2 Рассмотрим основные направления борьбы с тупиками.

В связи с проблемой тупиков были выполнено много интересных исследований в области информатики и операционных систем.

Основные направления борьбы с тупиками:

1) игнорировать данную проблему;

2) обнаружение тупиков;

3) восстановление после тупиков;

4) предотвращение тупиков за счет тщательного выделения ресурсов или нарушения одного из условий возникновения тупиков.

1.3.3 Алгоритм страуса.

Простейший подход - игнорировать проблему тупиков. Различные люди реагируют на подобную стратегию по-разному. Математики находят ее неприемлемой и утверждают, что тупики должны быть предотвращены любой ценой. Инженеры задают вопрос: как часто возникает данная проблема и как часто система виснет по другим причинам? Если тупик встречается раз в пять лет, но аварийный останов системы из-за отказов оборудования, ошибок компиляторов или ОС происходит раз в месяц, большинство инженеров не пожелают пожертвовать производительностью или удобством, чтобы ликвидировать тупик.

Алгоритм страуса (способ борьбы с тупиками) в информатике, это стратегия пренебрежения потенциальными проблемами из-за того, что вероятность их возникновения чрезвычайно мала – «прогрузить голову в песок и представить, что проблемы нет». То есть, предполагается, что эффективнее позволить проблеме реализоваться, чем попытаться ее предупредить. Такой подход может быть применен для взаимного блокирования при параллельном программировании, если мы верим в очень низкую вероятность такой блокировки, а стоимость выявления или предотвращения

высоковата. Это один из методов работы с взаимными блокировками. Другие методы: избежание (алгоритм Питерсона), предотвращение (алгоритм банкира), обнаружение и восстановление. В худшем случае (когда проблемы возникают достаточно часто) этот алгоритм приводит к крайне низкой производительности системы.

Обнаружение тупика- это установление факта, что возникла тупиковая ситуация и определение процессов и ресурсов, вовлеченных в эту ситуацию. Как правило, алгоритмы обнаружения применяются, когда выполнены первые три необходимых условия возникновения тупиковой ситуации. Затем проверяется наличие режима кругового ожидания. При этом активно используются графы распределения ресурсов.

Алгоритмы с низкой производительностью в худших случаях в основном применяют из-за того, что низкую производительность они обнаруживают на искусственных примерах, которые почти не встречаются в практике; типичные примеры-это симплекс-метод и алгоритм проверки типов для Standard ML. Такими случаями как целочисленное переполнение целых фиксированной ширины в языках программирования также часто пренебрегают из-за того, что они происходят лишь в исключительных случаях, которые не встречаются в практических входных данных.

Такой подход может быть использован при решении проблем взаимоблокировок в параллельном программировании, если они считаются очень редкими, а стоимость обнаружения или предотвращения высока. Например, если каждый ПК блокируется один раз в 10 лет, одна перезагрузка может быть менее болезненной, чем ограничения, необходимые для ее предотвращения.

Набор процессов блокируется, если каждый процесс в наборе ожидает события, которое может вызвать только другой процесс в наборе. Обычно событие является выпуском текущего ресурса, и ни один из процессов не может запускаться, освобождать ресурсы и пробуждаться.

Алгоритм страуса делает вид, что нет никаких проблем и разумно использовать, если тупики происходят очень редко, и стоимость их предотвращения будет высокой. Операционные системы UNIX и Windows используют этот подход[10,11].

Хотя использование алгоритма страуса является одним из методов борьбы с тупиками, существуют и другие эффективные методы, такие как динамическое избегание, алгоритм банкира, обнаружение и восстановление, а также предотвращение.

1.3.4 Более разумным представляется алгоритм банкира, разработанный ЭдсгеромВибеДейкстрой, который базируется на так называемых безопасных или надежных состояниях. Безопасным состоянием назовем такое состояние, перевод системы в которое не приведет к появлению тупиков. Алгоритм проверяет, ведет ли выполнение каждого запроса к небезопасному состоянию. Если да, то запрос отклоняется. Если удовлетворение запроса к ресурсу приводит к безопасному состоянию, ресурс предоставляется процессу. Модель алгоритма основана на действиях банкира, который, имея в наличии капитал, выдает кредиты.

На рисунке 1.4(а) показаны четыре клиента: A, B, C и D каждый из которых получил определенное количество единиц кредита. Банкир знает, что не всем клиентам тотчас же понадобится максимальная сумма их кредита, поэтому для обслуживания их потребностей он зарезервировал только 10 единиц, а не все 22, которые нужны клиентам. (Чтобы провести аналогию с компьютерной системой, будем считать, что клиенты – это процессы, единицы – накопители на магнитной ленте, а банкир – это операционная система.)

Клиенты занимаются своими делами, запрашивая время от времени ссуды (то есть запрашивая ресурсы). В какой–то определенный момент возникает ситуация, показанная на рисунке 1.4(б). Это состояние не представляет опасности, поскольку при оставшихся двух единицах банкир может отложить выполнение любых запросов, за исключением запроса клиента C, позволяя C завершить свои дела и высвободить все четыре своих ресурса. Имея в своем распоряжении четыре единицы ресурса, банкир может позволить получить необходимые единицы либо D, либо B и т. д.

Рассмотрим, что получится, если запрос от B одной дополнительной единицы будет удовлетворен в ситуации, показанной на рисунке 1.4(б). Мы получим небезопасную ситуацию, показанную на рисунке 1.4(в). Если все клиенты внезапно запросят свои максимальные ссуды, банкир не сможет удовлетворить никого из них, и мы получим взаимоблокировку. Небезопасное состояние не обязательно приводит к вза-

имоблокировке, поскольку клиенту может не понадобиться вся доступная максимальная сумма кредита, но банкир не может рассчитывать на это.

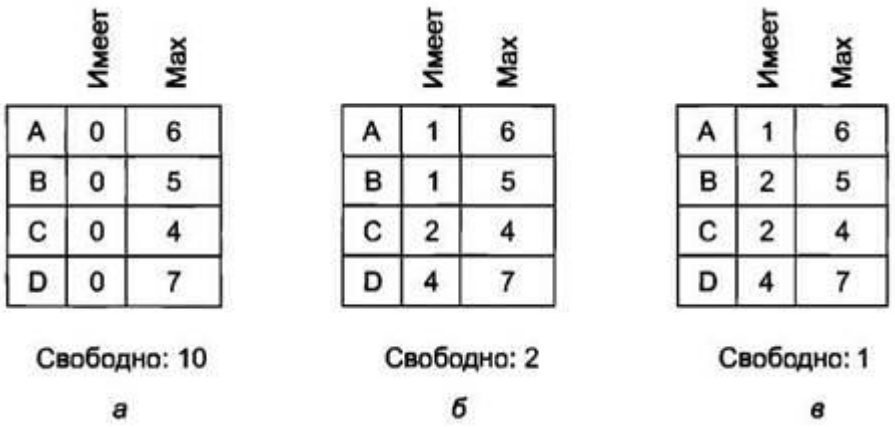

Рисунок 1.4– Состояние распределения ресурсов: а – безопасное, б – безопасное, в – небезопасное

Алгоритм банкира рассматривает каждый запрос по мере поступления и проверяет, приведет ли его удовлетворение к безопасному состоянию. Если да, то запрос удовлетворяется, в противном случае запрос откладывается до лучших времен. Чтобы понять, является ли состояние безопасным, банкир проверяет, может ли он предоставить достаточно ресурсов для удовлетворения запросов какого-нибудь клиента. Если да, то эти ссуды считаются возвращенными, после чего проверяется следующий ближайший к пределу займа клиент и т. д. Если в конечном счете все ссуды могут быть погашены, состояние является безопасным и исходный запрос можно удовлетворить.

Данный алгоритм обладает тем достоинством, что при его использовании нет необходимости в перераспределении ресурсов и откате процессов назад. Однако использование этого метода требует выполнения ряда условий:

– число пользователей и число ресурсов фиксировано,

– число работающих пользователей должно оставаться постоянным,

– алгоритм требует, чтобы клиенты гарантированно возвращали ресурсы,

Чаще всего данная информация отсутствует.

Наличие таких жестких и зачастую неприемлемых требований может склонить разработчиков к выбору других решений проблемы тупиков.

2 Методы распределения памяти

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса.

Функциями ОС по управлению памятью являются: отслеживание свободной и занятой памяти, выделение памяти процессам и освобождение памяти при завершении процессов, вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место, а также настройка адресов программы на конкретную область физической памяти.

Все методы управления памятью могут быть разделены на два класса: методы, которые используют перемещение процессов между оперативной памятью и диском, и методы, которые не делают этого (рисунок 2.1) [12-14].

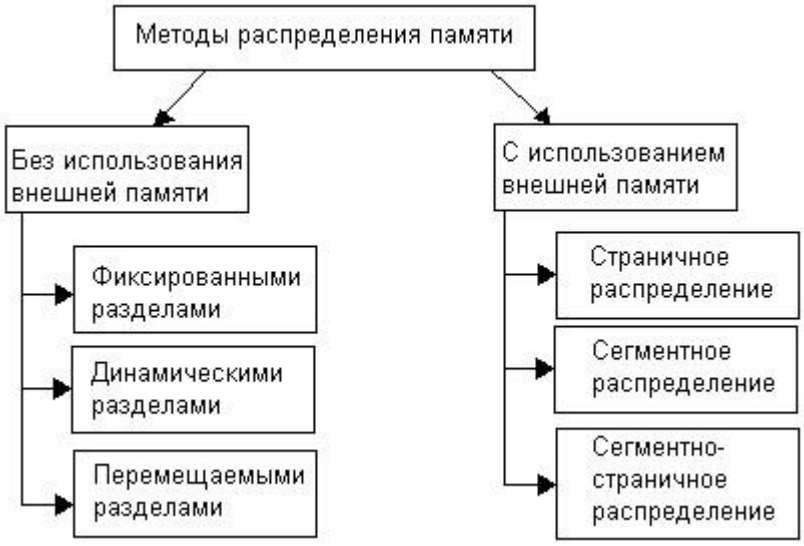

Рисунок 2.1– Классификация методов распределения памяти

2.1 Типы адресов

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса.

Символьные имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Так как во время трансляции в общем случае не известно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что программа будет размещена, начиная с нулевого адреса.

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды. Переход от виртуальных адресов к физическим может осуществляться двумя способами. В первом случае замену виртуальных адресов на физические делает специальная системная программа - перемещающий загрузчик. Перемещающий загрузчик на основании имеющихся у него исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, и информации, предоставленной транслятором об адресно-зависимых константах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими.

Второй способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, при этом операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Второй способ является более гибким, он допускает перемещение программы во время ее выполнения, в то время как перемещающий загрузчик жестко привязывает программу к первоначально выделенному ей участку памяти. Вместе с тем использование перемещающего загрузчика уменьшает накладные расходы, так как преобразование каждого виртуального адреса происходит только один раз во время загрузки, а во втором случае - каждый раз при обращении по данному адресу[15-18].

2.2 Распределение памяти без использования внешней памяти

Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее генерации. Очередная задача, поступившая на выполнение, помещается либо в общую очередь (рисунок 2.2,а), либо в очередь к некоторому разделу (рисунок 2.2,б).

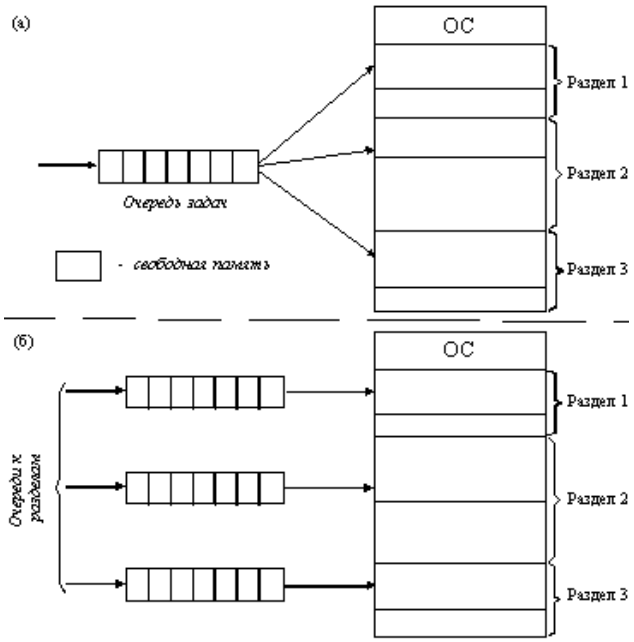

Рисунок 2.2 – Распределение памяти фиксированными разделами: а) - с общей очередью; б) - с отдельными очередями

Подсистема управления памятью выполняет следующие задачи:

– сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел;

– осуществляет загрузку программы и настройку адресов.

При очевидном преимуществе - простоте реализации данный метод имеет существенный недостаток - жесткость. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов, не зависимо от того, какой размер имеют программы. Даже если программа имеет небольшой объем, она будет занимать весь раздел, что приводит к не-

эффективному использованию памяти. С другой стороны, даже если объем оперативной памяти машины позволяет выполнить некоторую программу, разбиение памяти на разделы не позволяет сделать этого.

2.3 Распределение памяти с использованием внешней памяти

На рисунке 2.3 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью. Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками). Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т.д., это позволяет упростить механизм преобразования адресов.

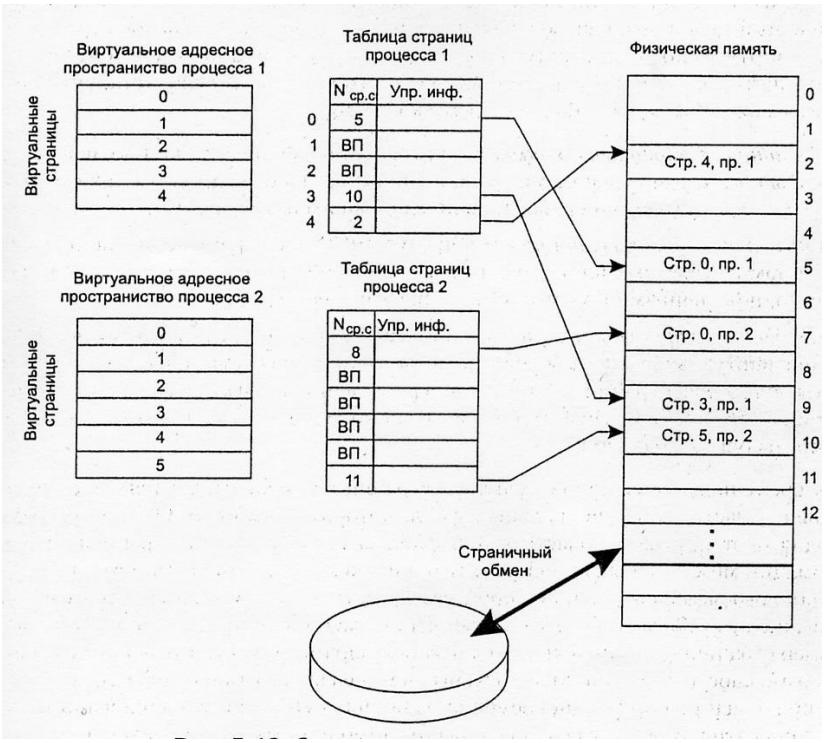

Рисунок 2.3– Страничное распределение памяти

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные – на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке операционная система создает для каждого процесса информационную структуру – таблицу страниц, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак невыгружаемости (выгрузка некоторых страниц может быть запрещена), признак обращения к странице (используется для подсчета числа обращений за определенный период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти.

При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы страниц данного процесса.

При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из оперативной памяти.

В данной ситуации может быть использовано много разных критериев выбора, наиболее популярные из них следующие:

– дольше всего не использовавшаяся страница;

– первая попавшаяся страница;

– страница, к которой в последнее время было меньше всего обращений.

В некоторых системах используется понятие рабочего множества страниц. Рабочее множество определяется для каждого процесса и представляет собой перечень наиболее часто используемых страниц, которые должны постоянно находиться в оперативной памяти и поэтому не подлежат выгрузке.

После того, как выбрана страница, которая должна покинуть оперативную память, анализируется ее признак модификации (из таблицы страниц). Если выталкиваемая страница с момента загрузки была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то она может быть просто уничтожена, то есть соответствующая физическая страница объявляется свободной.

Страничное распределение памяти может быть реализовано в упрощенном варианте, без выгрузки страниц на диск. В этом случае все виртуальные страницы всех процессов постоянно находятся в оперативной памяти. Такой вариант страничной организации хотя и не предоставляет пользователю виртуальной памяти, но почти исключает фрагментацию за счет того, что программа может загружаться в несмежные области, а также того, что при загрузке виртуальных страниц никогда не образуется остатков.

3 Организация файловой системы

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие "файловая система" включает:

- совокупность всех файлов на диске,
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы-каталоги.

Обычные файлы в свою очередь подразделяются на текстовые и двоичные. Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не используют ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все операционные системы должны уметь распознавать хотя бы один тип файлов - их собственные исполняемые файлы.

Специальные файлы - это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

3.1 Иерархия каталогов

Связующим звеном между системой управления файлами и набором файлов служит файловый каталог.

Каталог - это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны - это файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

Простейшая форма системы каталогов состоит в том, что имеется один каталог, в котором содержатся все файлы. Каталог содержит информацию о файлах, включая атрибуты, местоположение, принадлежность. Пользователи обращаются к файлам по символьным именам. Однако способности человеческой памяти ограничивают количество имен объектов, к которым пользователь может обращаться по именам. Иерархическая организация пространства имен позволяет значительно расширить эти границы. Именно поэтому каталоговые системы имеют иерархическую структуру. Граф, описывающий иерархию каталогов, может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть, если файл может входить в несколько каталогов.

Например, в Ms–Dos и Windows каталоги образуют древовидную структуру, а в UNIX – сетевую[10,11.19-21]. Примеры древовидной и сетевой структуры приведены на рисунках 3.1 и 3.2 соответственно.

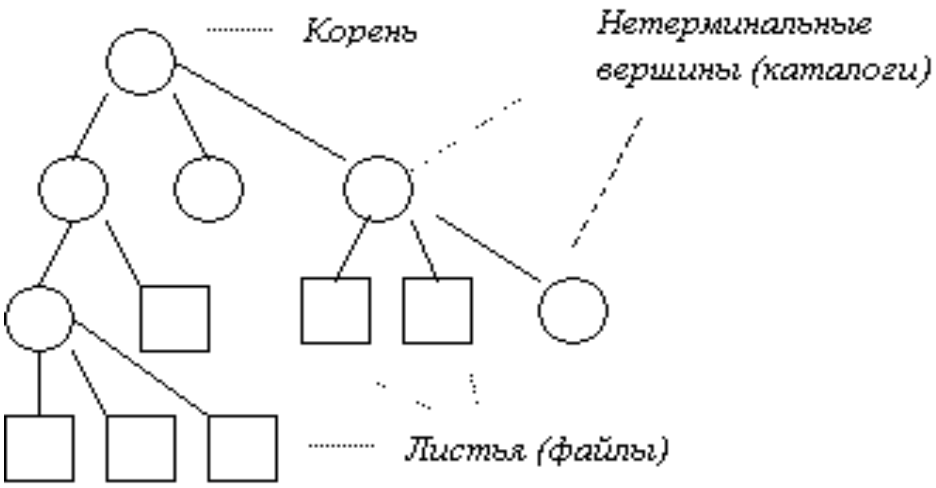

Рисунок 3.1 – Иерархическая (древовидная) структура каталогов

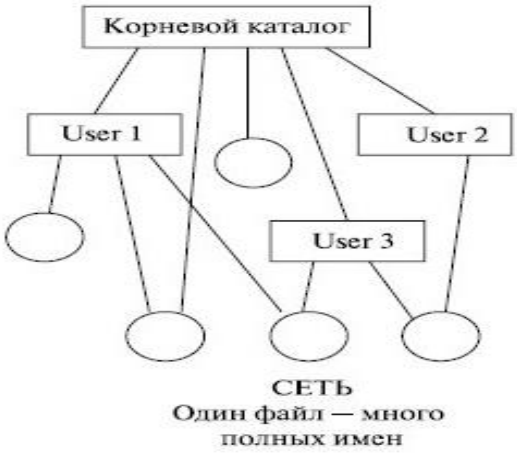

Рисунок 3.2– Сетевая структура каталогов

3. 2 Способы логической организации файлов

В общем случае данные, содержащиеся в файле, имеют некоторую логическую структуру. Эта структура (организация) файла является базой при разработке программы, предназначенной для обработки этих данных. Поддержание структуры данных может быть целиком возложено на приложение либо в той или иной степени эту работу может взять на себя файловая система.

В первом случае, когда все действия, связанные со структуризацией и интерпретацией содержимого файла, целиком относятся к ведению приложения, файл

представляется файловой системе неструктурированной последовательностью данных. Приложение формирует запросы к файловой системе на ввод–вывод, используя общие для всех приложений системные средства, например, указывая смещение от начала файла и количество байт, которые необходимо считать или записать. Поступивший к приложению поток байт интерпретируется в соответствии с заложенной в программе логикой. Следует подчеркнуть, что интерпретация данных никак не связана с действительным способом их хранения в файловой системе.

Модель файла, в соответствии с которой содержимое файла представляется неструктурированной последовательностью байт, стала популярной вместе с ОС UNIX, и теперь широко используется в современных ОС. Неструктурированная модель файла позволяет легко организовать разделение файла между несколькими приложениями, поскольку разные приложения могут по–своему структурировать и интерпретировать данные, содержащиеся в файле.

Другая модель файла – структурированный файл. В этом случае поддержание структуры файла поручается файловой системе. Файловая система видит файл как упорядоченную последовательность логических записей. ФС предоставляет приложению доступ к записи, а вся дальнейшая обработка данных, содержащихся в этой записи, выполняется приложением!

Известно пять фундаментальных способов организации файлов:

– смешанный файл;

– последовательный файл;

– индексно–последовательный файл;

– индексируемый файл;

– файл прямого доступа.

При выборе способа организации файла нужно учитывать несколько критериев:

– быстрота доступа;

– легкость обновления;

– экономность хранения;

– простота обслуживания;

– надежность.

Одним из методов преодоления недостатков последовательного файла является индексно–последовательная организация файла. В этом случае файл состоит из трех частей (файлов): главный файл, содержащий записи с последовательно идущими ключами, индексный файл, содержащий индексное поле, и указатель в главный с ключами, файл переполнения (рисунок 3.3).

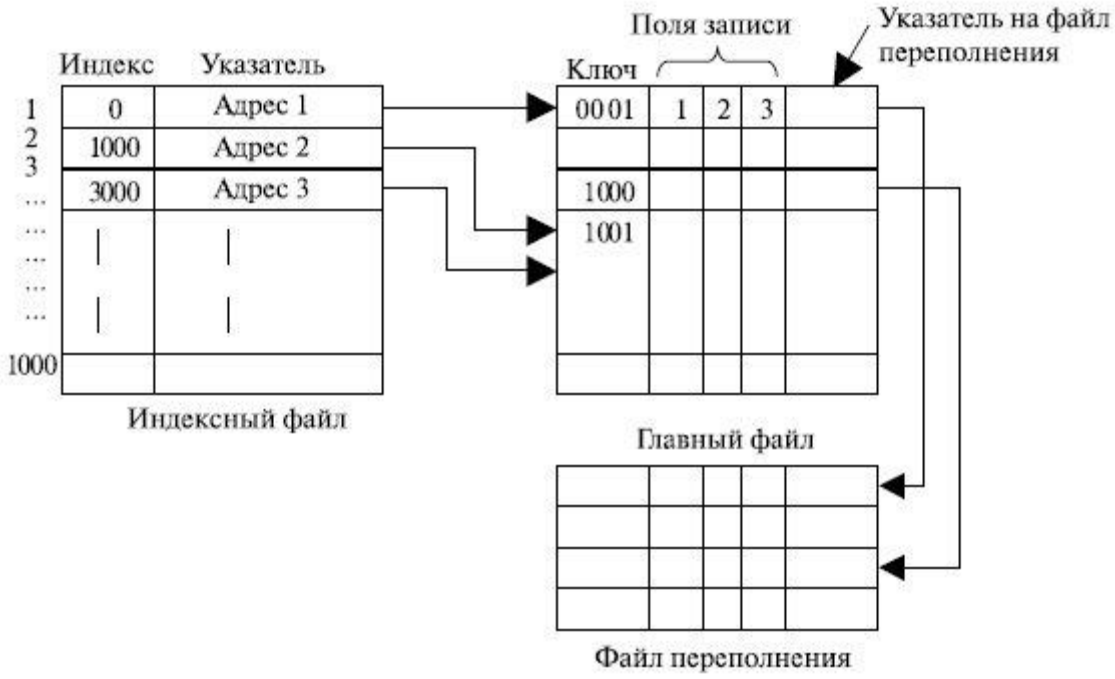

Рисунок 3.3 – Индексная логическая организация файлов

Для поиска нужной записи по ее ключу сначала выполняется поиск в индексном файле. После того как в нем найдено наибольшее значение ключа, которое не превышает искомое, продолжается поиск в главном файле. Например, пусть последовательный файл(главный) содержит 1 млн записей. Для поиска определенного ключевого значения необходимо в среднем 0,5 млн операций доступа к записям. Если создать индексный файл, содержащий 1000 элементов, то потребуется в среднем 500 операций доступа к индексному файлу, после чего еще нужно в среднем 500 операций доступа к главному файлу. В результате средняя длина поиска уменьшилась с 0,5 млн до 1000. Еще лучшего результата можно достичь, используя многоуровневую индексацию. При этом нижний уровень индексного файла рассматривается как последовательный файл, для которого создается индексный файл верхнего уровня.

Дополнения к файлу обрабатываются следующим образом. В каждой записи главного файла содержится дополнительное поле, невидимое для приложения и являющееся указателем на файл переполнения. Если в файле производится вставка новой записи, она добавляется в файл переполнения. Запись в главном файле, непосредственно предшествующая новой записи в логической последовательности, обновляется и указывает на новую запись в файле переполнения. Время от времени выполняется слияние индексно–последовательного файла с файлом переполнения.

3.3 Физическая организация файла

Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей – блоков. Блок – наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью. Существует несколько вариантов физической организации файлов (рисунок 3.4):

– непрерывное размещение – простейший вариант физической организации, при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти;

– размещение в виде связанного списка блоков дисковой памяти, когда в начале каждого блока содержится указатель на следующий блок;

– связанный список индексов, когда с каждым блоком связывается некоторый элемент - индекс.

– простое перечисление номеров блоков, занимаемых этим файлом.

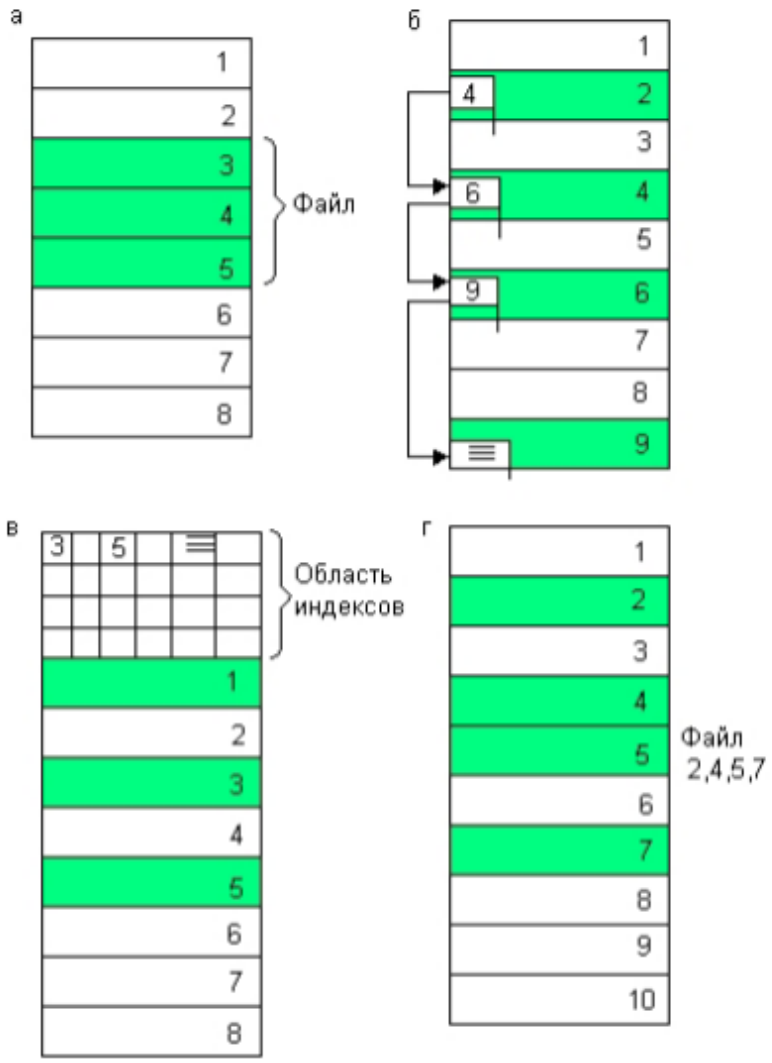

Рисунок 3.4 – Физическая организация файла: непрерывное размещение (а); связанный список кластеров (б); связанный список индексов (в); перечень номеров кластеров (г)

4 Моделирование алгоритмов управления процессами

4.1 Выбор языка программирования

C# (произносится как "си шарп") – простой, современный объектно-ориентированный и типобезопасный язык программирования.C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript. Здесь представлен обзор основных компонентов языка. Если вы хотите изучить язык с помощью интерактивных примеров, рекомендуем поработать с нашими вводными руководствами по C# .

C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов — это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации. C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# отлично подходит для создания и применения программных компонентов.

Вот лишь несколько функций языка C#, обеспечивающих надежность и устойчивость приложений: сборка мусора автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами; обработка исключений предоставляет структурированный и расширяемый способ выявлять и обрабатывать ошибки; строгая типизация языка не позволяет обращаться к неинициализированным переменным, выходить за пределы индексируемых массивов или выполнять неконтролируемое приведение типов.

В C# существует единая система типов. Все типы C#, включая типы-примитивы, такие как int и double, наследуют от одного корневого типа object. Таким образом, все типы используют общий набор операций, и значения любого типа можно хранить, передавать и обрабатывать схожим образом. Кроме того, C# поддержи-

вает пользовательские ссылочные типы и типы значений, позволяя как динамически выделять память для объектов, так и хранить упрощенные структуры в стеке.

Чтобы обеспечить совместимость программ и библиотек C# при дальнейшем развитии, при разработке C# много внимания было уделено управлению версиями. Многие языки программирования обходят вниманием этот вопрос, и в результате программы на этих языках ломаются чаще, чем хотелось бы, при выходе новых версий зависимых библиотек. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как раздельные модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

4. 2 Разработка алгоритма управления процессами

В качестве примера приведен простейший алгоритм планирования *FCFS*– FirstCome, FirstServed (первым пришел, первым обслужен).

Преимуществом алгоритма FCFS является легкость его реализации, в то же время он имеет и много недостатков. Рассмотрим следующий пример. Пусть в состоянии ***готовность*** находятся три процесса p_0, p_1 и p_2, для которых известны времена их очередных CPU burst. Эти времена приведены в таблице 4.1. в некоторых условных единицах. Для простоты будем полагать, что вся деятельность процессов ограничивается использованием только одного промежутка CPU burst, что процессы не совершают операций ввода-вывода, и что время переключения контекста пренебрежимо мало.

Таблица 4.1 – Очередь процессов

Процесс	p_0	p_1	p_2
Продолжительность очередного CPU burst	13	4	1

Если процессы расположены в очереди процессов готовых к исполнению в порядке p_0, p_1, p_2, то картина их выполнения выглядит так, как показано на рисунке 4.1. Первым для выполнения выбирается процесс p_0, который получает процессор на все

время своего CPU burst, т. е. на 13 единиц времени. После его окончания в состояние ***исполнение*** переводится процесс p_1, занимая процессор на 4 единицы времени. И, наконец, возможность работать получает процесс p_2. Время ожидания для процесса p_0 составляет 0 единиц времени, для процесса p_1 – 13 единиц, для процесса p_2 – 13 + 4 = 17 единиц. Таким образом, среднее время ожидания в этом случае – (0 + 13 + 17)/3 = 10 единиц времени. Полное время выполнения для процесса p_0 составляет 13 единиц времени, для процесса p_1 – 13 + 4 = 17 единиц, для процесса p_2 – 13 + 4 + 1 = 18 единиц. Среднее полное время выполнения оказывается равным (13 + 17 + 18)/3 = 16 единицам времени.

Рисунок 4.1– Выполнение процессов при порядке p_0,p_1,p_2

Если те же самые процессы расположены в порядке p_2, p_1, p_0, то картина их выполнения будет соответствовать рисунку 4.2. Время ожидания для процесса p_0 равняется 5 единицам времени, для процесса p_1 – 1 единице, для процесса p_2 – 0 единиц. Среднее время ожидания составит (5 + 1 + 0)/3 = 2 единицы времени. Это в 5 (!) раз меньше, чем в предыдущем случае. Полное время выполнения для процесса p_0 получается равным 18 единицам времени, для процесса p_1 – 5 единицам, для процесса p_2 – 1 единице. Среднее полное время выполнения составляет (18 + 5 + 1)/3 =8 единиц времени, что почти в 2,7 раза меньше чем при первой расстановке процессов.

Рисунок 4.2 – Выполнение процессов при порядке p_2,p_1,p_0

Как видим, среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если у нас есть процесс с длительным CPU burst, то короткие процессы, перешедшие в состояние ***готовность*** после длительного процесса, будут очень долго ждать начала своего выполнения. Поэтому алгоритм FCFS практически неприменим для систем разделения времени. Слишком большим получается среднее время отклика в интерактивных процессах.

4.3 Выполнение тестовой проверки

На рисунках 4.3 – 4.7 показаны примеры, иллюстрирующие расчет параметров и работу алгоритма FCFS.

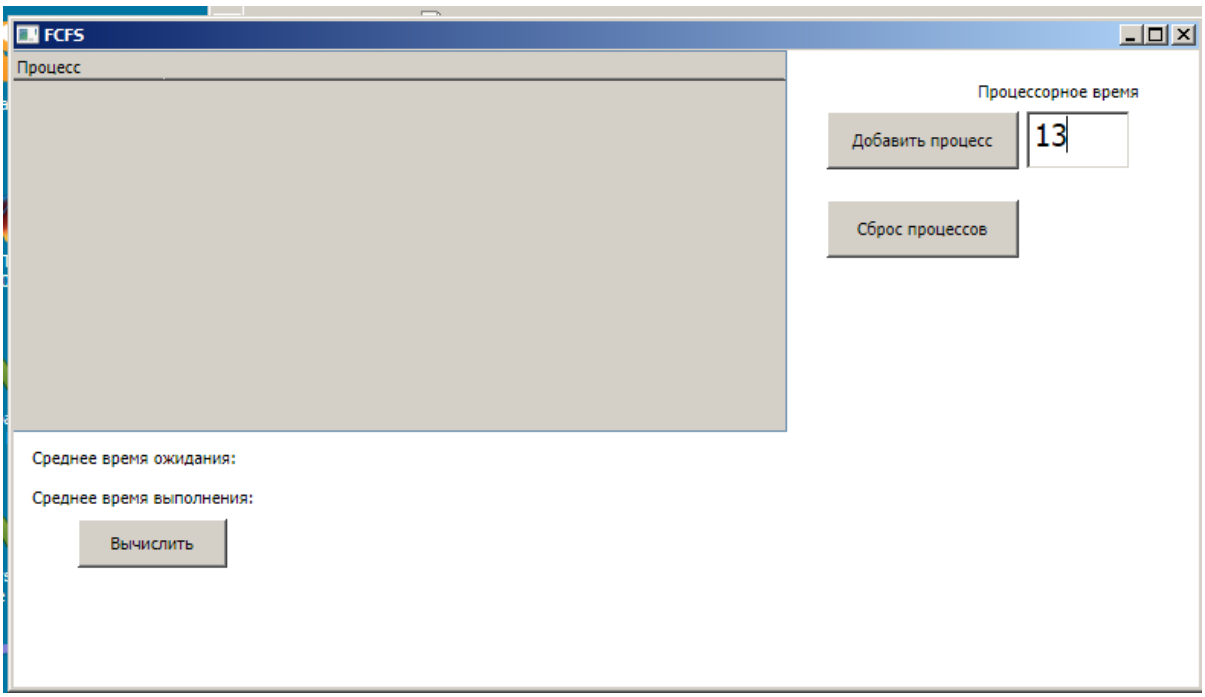

Рисунок 4.3 – Интерфейс программы

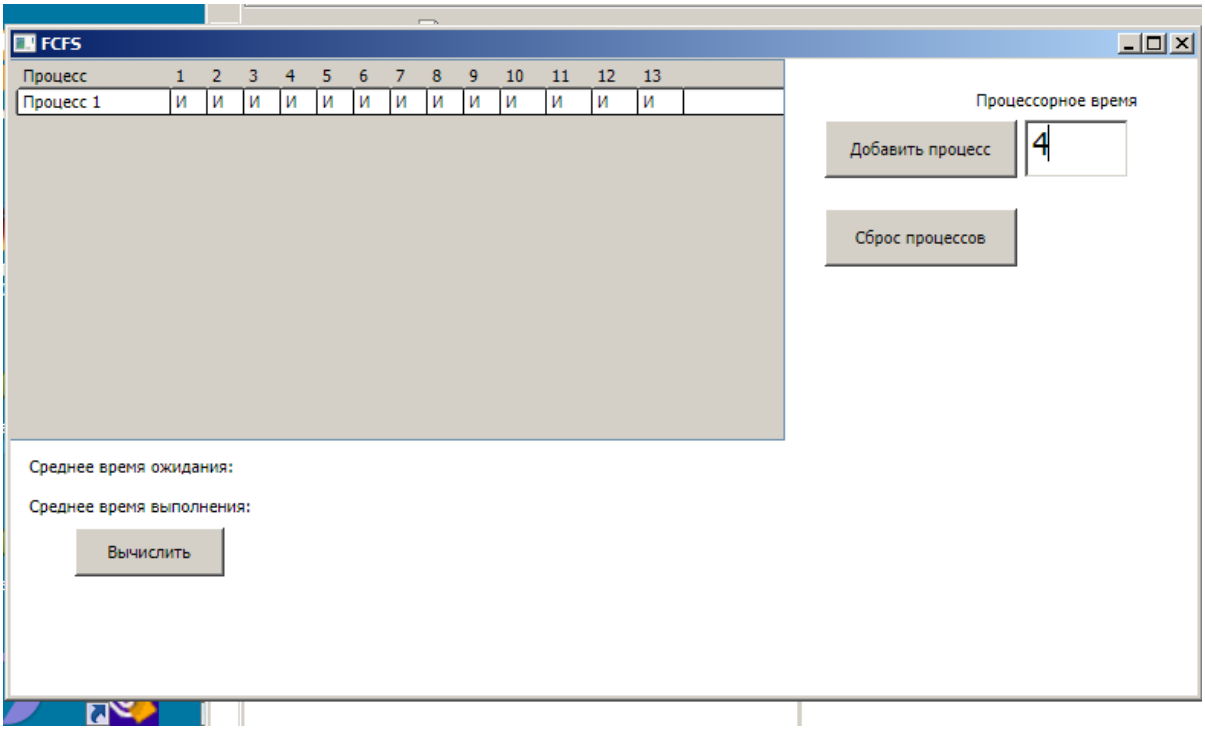

Рисунок 4.4 – Ввод параметров процессов

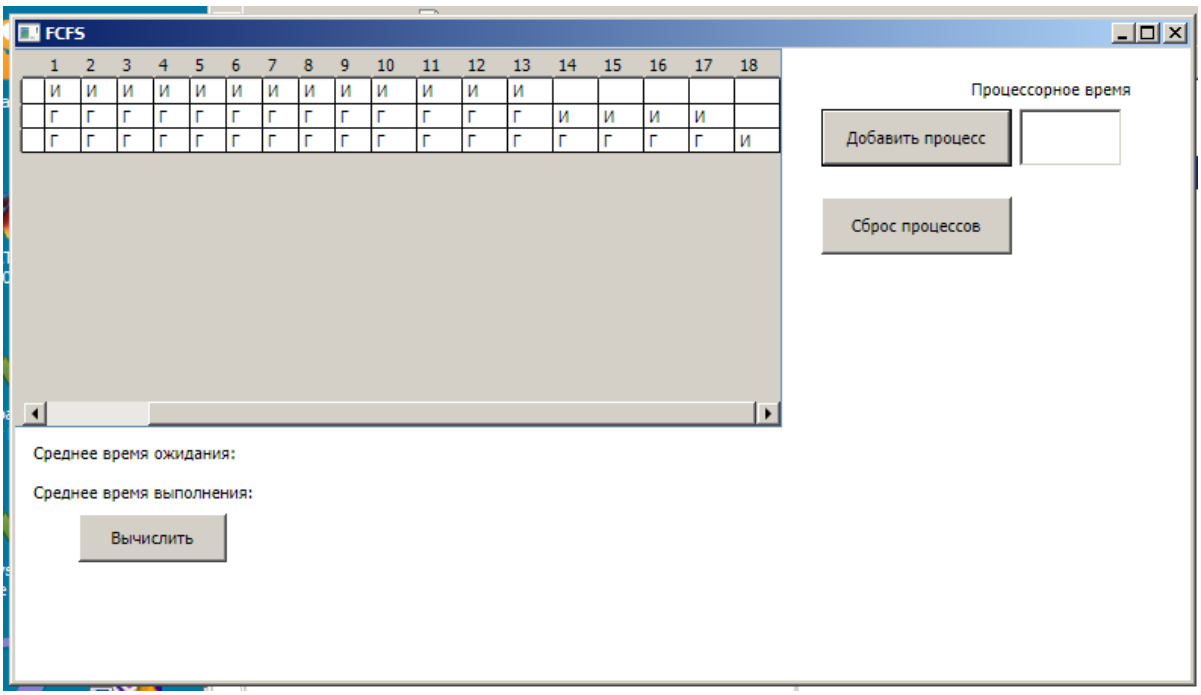

Рисунок 4.5 – Вид диаграммы поведения процессов

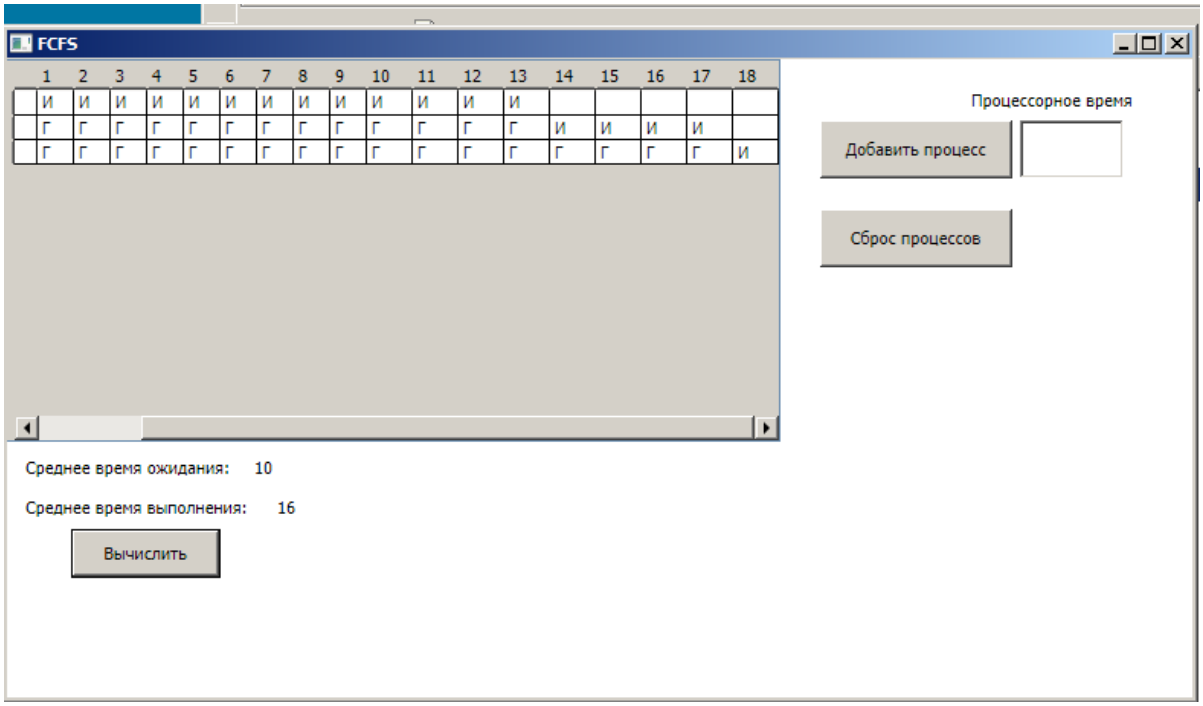

Рисунок 4.6 – Расчет параметров

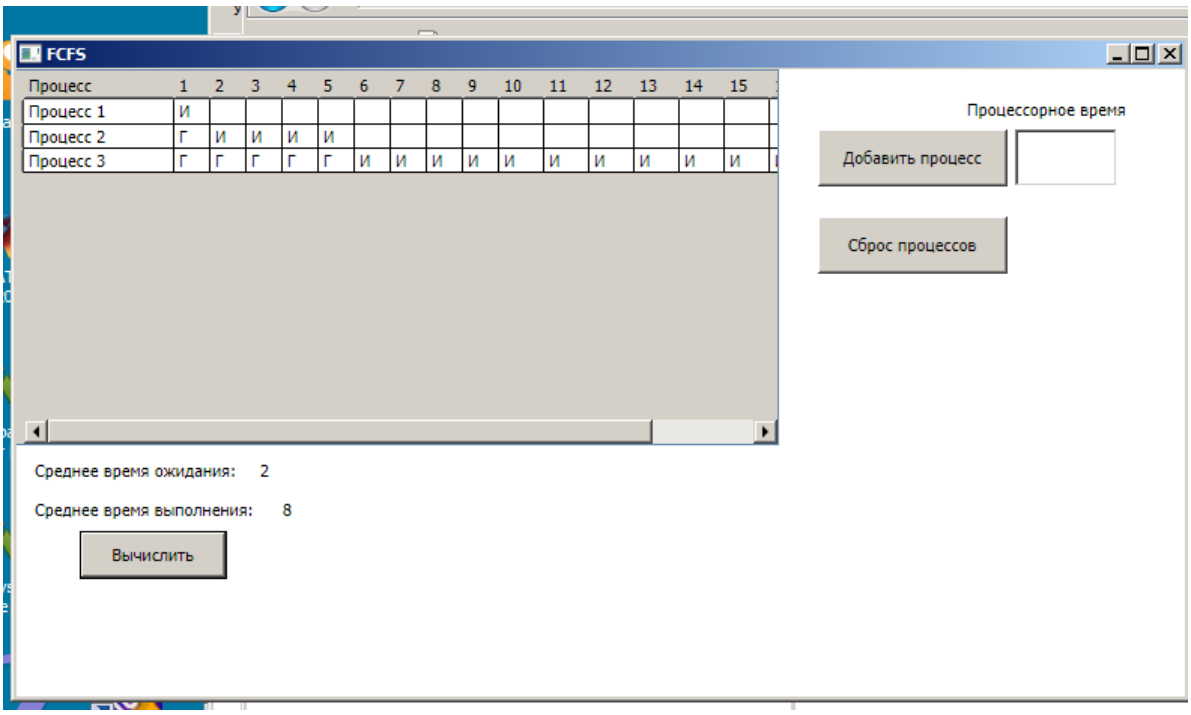

Рисунок 4.7 – Расчет параметров

Разработанная программа позволяет исследовать работу алгоритма FCFS при различных вариантах поступления процессов в очередь.

Тестовый пример подтвердил сделанные выше теоретические рассуждения и ручные расчеты.

5 Литература, рекомендуемая для изучения

1.Гриценко, Ю.Б. Операционные среды, системы и оболочки: учебное пособие / Ю.Б. Гриценко ; Томский межвузовский центр дистанционного образования (ТУСУР). – Томск : Томский государственный университет систем управления и радиоэлектроники, 2005. – 281 с. : табл., схем. ; то же [Электронный ресурс]. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=208656

2.Гриценко, Ю.Б. Операционные среды, системы и оболочки : учебное пособие / Ю.Б. Гриценко; Томский межвузовский центр дистанционного образования (ТУСУР). – Томск : Томский государственный университет систем управления и радиоэлектроники, 2005. – 281 с. : табл., схем. ; то же [Электронный ресурс]. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=208656

3.Гордеев, А. В. Операционные системы : учебник / А. В. Гордеев .– 2-е изд. – Санкт Петербург : Питер, 2007. – 416 с. – (Учебник для вузов).– Библиогр.: с. 406-408. – Алф. указ.: с. 409-415. – ISBN 978-5-94723-632-3.

4.Бэкон, Д. Операционные системы. Параллельные и распределенные системы = OperatingSystems. Concurrent and Distributed Software Design [Текст] /Д. Бэкон, Т. Харрис . – СПб. [и др.] : Питер, 2004.– 800 с. : ил.– Парал. тит. л. на англ. яз. – Алф. указ.: с. 785. – ISBN 5-94723-969-8.

5.Таненбаум,Э. Современные операционные системы =Modernoperatingsystems / Э. Таненбаум.– 2-е изд. – СПб. : Питер, 2007. – 1038 с.

6.Олифер, В. Г. Сетевые операционные системы [Текст] : учеб.для вузов / В. Г. Олифер, Н. А. Олифер. – СПб. : Питер, 2002. - 544 с. : ил - ISBN 5-272-00120-6.

7.Партыка, Т. Л. Операционные системы, среды и оболочки: учеб. пособие / Т. Л. Партыка, И. И. Попов .– М. : ФОРУМ - ИНФРА-М, 2004.– 400 с.

8.Сафонов, В.О. Основы современных операционных систем: учебное пособие / В.О. Сафонов.– М.: Интернет-Университет Информационных Технологий, 2011.– 584 с. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=233210

9.Назаров, С.В. Современные операционные системы: учебное пособие / С.В. Назаров, А.И. Широков. – М.: Интернет-Университет Информационных Технологий, 2011. – 280 с. –Режим доступа: http://biblioclub.ru/index.php?page=book&id=233197

10.Булатов, В. Н. Операционные системы на платформе IBM PC: учеб. пособие / В. Н. Булатов .– Оренбург : ОГУ, 2005. – 324 с. – Библиогр. : с. 293. – ISBN 5-7410-0639-6.

11.Вавренюк, А. Б. Операционные системы. Основы UNIX [Электронный ресурс] / Вавренюк А. Б. – НИЦ ИНФРА-М, 2015. – Режим доступа: http://znanium.com/catalog/product/504874

12.Козлов, О. А. Операционные системы / Ю. Ф. Михайлов, С. А. Зайцева, О. А. Козлов.– Шуя : ФГБОУ ВПО "ШГПУ", 2013. – Режим доступа:http://rucont.ru/efd/206356

13.Наточая, Е. Н. Операционные системы и оболочки [Электронный ресурс] : электронный курс лекций / Е. Н. Наточая; М-во науки и высш. образования Рос. Федерации, Федер. гос. бюджет.образоват. учреждение высш. образования "Оренбург. гос. ун-т". - Электрон.текстовые дан. (1 файл: 44.2 Мб). – Оренбург : ОГУ, 2019. – 5 с. - Загл. с тит. экрана. – Архиватор 7-Zip

14.Таненбаум, Э. Надежные и защищенные операционные системы? / Э. Таненбаум, Д. Хердер, Х. Бос // Открытые системы. СУБД,2006. – N 6. – с. 38-46. – Библиогр.: с. 46 (10 назв.).

15.Кручинин, А.Ю. Операционные системы [Электронный ресурс]: учебное пособие/ Кручинин А.Ю.– Электрон.текстовые данные.– Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2009.– 132 с.– Режим доступа: http://www.iprbookshop.ru/30115.html. – ЭБС «IPRbooks»

16.Гриценко, Ю. Б.Операционные системы. Учебное пособие. В 2-х частях. Часть 2 [Электронный ресурс] / Гриценко Ю. Б. – Томский государственный университет систем управления и радиоэлектроники, 2009. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=208655

17.Кондратьев, В.К. Операционные системы и оболочки : учебно-практическое пособие / В.К. Кондратьев, О.С. Головина ; Международный консорциум «Электрон-

ный университет», Московский государственный университет экономики, статистики и информатики, Евразийский открытый институт.– Москва : Московский государственный университет экономики, статистики и информатики, 2007.– 172 с. – ISBN 5-374-00009-8 ;то же [Электронный ресурс]. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=90663(06.06.2019)

18.Назаров, С.В. Современные операционные системы : учебное пособие / С.В. Назаров, А.И. Широков. – Москва : Интернет-Университет Информационных Технологий, 2011. – 280 с. : ил., табл., схем. – (Основы информационных технологий). – ISBN 978-5-9963-0416-5 ; то же [Электронный ресурс]. – Режим доступа: http://biblioclub.ru/index.php?page=book&id=233197

19.Курячий, Г. В. Операционная система UNIX [Текст] : курс лекций: учеб.пособие для вузов / Г. В. Курячий. – М. : Интернет-Ун-т Информ. Технологий, 2004.– 288 с. – (Основы информационных технологий) – ISBN 5-9556-0019-1.

20.Курячий, К.А. Операционная система Linux [Текст] : курс лекций: учеб. пособие / Г. В. Курячий, К. А. Маслинский .– М. : Интернет-Ун-т Информ. Технологий, 2005. - 392 с. – (Основы информационных технологий) – ISBN 5-9556-0029-9. 9

21.Волосатова, Т.М. Основные концепции операционной системы UNIX : учеб. пособие / С.В. Грошев, С.В. Родионов, Т.М. Волосатова .– М. : Изд-во МГТУ им. Н.Э. Баумана, 2010.– Режим доступа: http://rucont.ru/efd/287647

ПриложениеА

(обязательное)

Варианты заданий

Таблица А1-Варианты заданий

Варианты	1	2	3	4	5	6
Управление процессами						
а) алгоритм планирования процессов	Невытесняющий FCFS	Вытесняющий RR	Невытесняющий SJF	Вытесняющий SJF	Гарантированное планирование	Невытесняющий приорит. планир.
б) алгоритм организации взаимодействия процессов;	Алгоритм Петерсона	Алгоритм булочной	Использование семафоров	Буфер сообщений	Использование мониторов	Алгоритм Петерсона
в) способ борьбы с тупиками	Алгоритм страуса	Обнаружение тупиков	Предотвращение тупиков, алгоритм банкира	Алгоритм страуса	Обнаружение тупиков	Предотвращение тупиков, алгоритм банкира
Методы распределения памяти						
	Фиксированными разделами (без использования внешней памяти)	Динамическими разделами (без использования внешней памяти)	Перемещаемыми разделами (без использования внешней памяти)	Страничное распределение	Сегментное распределение	Сегментно-страничное распределение
Организация файловой системы						
а) иерархия каталогов;	Дерево	Сеть	По усмотрению	По усмотрению	Дерево	Сеть
б) способы логической организации;	Последовательность логических записей фиксированной длины	Последовательность логических записей переменной длины	Индексная логическая организация	Последовательность однобайтовых записей	Последовательность логических записей переменной длины	Индексная логическая организация
в) физическая организация	Непрерывное размещение	Связанный список блоков	Связанный список индексов	Перечень номеров блоков	Перечень номеров блоков	Непрерывное размещение

Продолжение таблицы А1

Варианты	7	8	9	10	11	12
Управление процессами						
а) алгоритм планирования процессов	Невытесняющий FCFS	Вытесняющий RR	Невытесняющий SJF	Вытесняющий SJF	Гарантированное планирование	Невытесняющий приорит. планир.
б) алгоритм организации взаимодействия процессов	Алгоритм булочной	Алгоритм Петерсона	Буфер сообщений	Использование семафоров	Алгоритм Петерсона	Использование мониторов
в) способ борьбы с тупиками	Алгоритм страуса	Обнаружение тупиков	Предотвращение тупиков, алгоритм банкира	Алгоритм страуса	Обнаружение тупиков	Предотвращение тупиков, алгоритм банкира
Методы распределения памяти						
	Динамическими Разделами (без использования внешней памяти)	Фиксированными разделами (без использования внешней памяти)	Сегментное распределение	Перемещаемыми разделами (без использования внешней памяти)	Сегментно-страничное распределение	Страничное распределение
Организация файловой системы						
а) иерархия каталогов;	Дерево	Сеть	По усмотрению	По усмотрению	Дерево	Сеть
б) способы логической организации	Последовательность логических записей фиксированной длины	Последовательность логических записей переменной длины	Индексная логическая организация	Последовательность однобайтовых записей	Последовательность логических записей переменной длины	Индексная логическая организация
в) физическая организация	Непрерывное размещение	Связанный список индексов	Связанный список блоков	Перечень номеров блоков	Непрерывное размещение	Перечень номеров блоков

Продолжение таблицы А1

Варианты	13	14	15	16	17	18
Управление процессами						
а) алгоритм планирования процессов	Вытесняющий приорит. планир.	Вытесняющий приорит. планир.	Невытесняющий SJF	Вытесняющий SJF	Гарантированное планирование	Невытесняющий приорит. планир.
б) алгоритм организации взаимодействия процессов	Алгоритм булочной	По усмотрению	Буфер сообщений	Использование семафоров	Алгоритм Петерсона	Использование мониторов
в) способ борьбы с тупиками	Алгоритм страуса	Алгоритм страуса	Предотвращение тупиков, алгоритм банкира	Алгоритм страуса	Обнаружение тупиков	Предотвращение тупиков, алгоритм банкира
Методы распределения памяти						
	Фиксированными разделами (без использования внешней памяти)	Сегментно-страничное распределение	Сегментное распределение	Перемещаемыми разделами (без использования внешней памяти)	Сегментно-страничное распределение	Страничное распределение
Организация файловой системы						
а) иерархия каталогов	По усмотрению	По усмотрению	По усмотрению	По усмотрению	Дерево	Сеть
б) способы логической организации;	Последовательность логических записей фиксированной длины	Последовательность однобайтовых записей	Индексная логическая организация	Последовательность однобайтовых записей	Последовательность логических записей переменной длины	Индексная логическая организация
в) физическая организация	Связанный список блоков	Связанный список блоков	Связанный список блоков	Перечень номеров блоков	Непрерывное размещение	Перечень номеров блоков