МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Восточно-Сибирский государственный университет технологий и управления»

Факультет компьютерных наук и технологий

Кафедра «Программная инженерия и искусственный интеллект»

Курсовая работа

по дисциплине «Операционные системы»

на тему

«Моделирование работы подсистемы ОС»

Вариант 5.3.2

|  |  |
| --- | --- |
| Выполнил: | студент гр. Б761-2 |
| \_\_\_\_\_\_\_\_\_\_ | Немеров А.П. |
| Руководитель: | ст. преп. кафедры СИ |
| \_\_\_\_\_\_\_\_\_\_ | Глушкова И.И. |
| Оценка: | \_\_\_\_\_\_\_\_\_\_\_ |
| Дата защиты: | \_\_\_\_\_\_\_\_\_\_\_ |

Улан-Удэ

2022

ВОСТОЧНО-СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ

Кафедра программная инженерия и искусственный интеллект

ЗАДАНИЕ

на курсовую работу

|  |
| --- |
| Дисциплина: Операционные системы |
| Тема: «Моделирование подсистемы ОС. Вариант 5.3.2» |
| Исполнитель(и): Немеров А.П. |
| Руководитель: Глушкова И.И. |
| Краткое содержание работы: Данная курсовая работа посвящена разбора |
| многоуровневой очереди (2 уровня: высокий приоритет – Round robin, низкий |
| приоритет – FCFS), разработка алгоритмов распределения памяти и |
| размещение файлов. |
| 1. Теоретический раздел: |
| 1.1 Словесная постановка задачи |
| 1.2 Операционные системы и их функции |
| 1.3 Подсистема управления процессами |
| 1.4 Подсистема управления памятью |
| 1.5 Подсистема управления файловой системы |
| 2 Практический раздел: |
| 2.1 Алгоритм страничного распределения памяти |
| 2.2 Непрерывное размещение файлов |
| 3 Экспериментальный раздел |
| Сроки выполнения проекта по графику: |
| 1. Теоретический раздел – 50% до 29.10.2022 |
| 1. Практический раздел – 90% до 8.12.2022 |
| 3. Защита – 100% 15.12.2022 |
| Требования к оформлению: |
| 1. Расчетно-пояснительная записка курсовой работы должна быть |
| представлена в электронной и твердой копиях. |
| 2. Объем РПЗ должен быть не менее 20 машинописных страниц без |
| учета приложений |
| 3. РПЗ оформляется по ГОСТу 7.32-2017. |

|  |  |
| --- | --- |
| Руководитель | Глушкова И.И. |
| Исполнитель | Немеров А.П. |
| Дата выдачи | "\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_2022 г. |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc90478882)

[1 Теоретический раздел 6](#_Toc90478883)

[1.1 Словесная постановка задачи 6](#_Toc90478884)

[1.2 Операционные системы и их функции, процессы 6](#_Toc90478885)

[1.3 Подсистема управления процессами 8](#_Toc90478886)

[1.3.1 SJF с предсказанием 9](#_Toc90478888)

[1.4 Подсистема управления памятью 12](#_Toc90478889)

[1.4.1 Страничное распределение памяти 13](#_Toc90478890)

[1.5 Подсистема управления файловой системы 18](#_Toc90478891)

[1.5.1 Непрерывное размещение файлов 19](#_Toc90478892)

[2 Практический раздел 22](#_Toc90478893)

[2.1 Алгоритм страничного распределения памяти 22](#_Toc90478894)

[2.2 Алгоритм планирования SJF 27](#_Toc90478896)

[3 Экспериментальный раздел 30](#_Toc90478897)

[3.1 Страничное распределение памяти 30](#_Toc90478898)

[3.2 Алгоритм планирования SJF 32](#_Toc90478900)

[Заключение 34](#_Toc90478901)

[Список использованных источников 35](#_Toc90478902)

[Приложение А 36](#_Toc90478903)

**ВВЕДЕНИЕ**

Операционные системы (операционные системы) окружают нас повсюду – системное программное обеспечение персональных компьютеров, серверов, мобильных устройств, сетевых устройств (маршрутизаторов, коммутаторов) и даже современных автомобилей (бортовых компьютеров), телевизоров и прочего. Любой компьютер представляет собой связанный кластер: процессор, память и устройства ввода-вывода.

Основная функция ОС персонального компьютера:

* управление ресурсами и процессами, а также совместное использование вычислительных ресурсов группой приложений (многозадачность) — центральная функция ОС, которая является базой для разных системных архитектур.

Операционная система является одним из таких уровней и представляет собой интерфейс («прослойку») между пользователем ресурсов компьютера и самими ресурсами, управляющий взаимодействиями как между пользователь-ресурс, так и пользователь-пользователь, устройство-устройство.

Целью курсовой работы является углубление теоретических знаний работы подсистемы управления процессами и ее моделирование.

Задачи курсовой работы:

* углублённое изучение подсистем ОС.
* изучить принцип работы подсистемы управления процессами, памяти и файловой подсистемы, и их организации.
* разработать алгоритмов страничного распределения памяти и непрерывного размещения файлов.

Расчетно-пояснительная записка состоит из двух частей:

* теоретической, содержащей словесную постановку задачи, краткое описание алгоритмов планирования, распределения памяти и размещения файлов;
* практической, содержащей формальную постановку задачи, проектирование.

# **1 Теоретический раздел**

## **Словесная постановка задачи**

Изучить подсистему файловой системы и работу управления памятью. Проанализировать модель работы многоуровневой очереди (2 уровня: высокий приоритет – Round robin, низкий приоритет – FCFS). Для компьютерной реализации моделирования подсистем управления процессами, памятью и файлами, был выбран язык программирования С++. В данном курсовой работе реализованы алгоритмы страничного распределения памяти (страница – 1024 байт) непрерывного размещения (кластер – 512 байт) и подсистемы управления процессами.

## **Операционные системы и их функции, процессы**

ОС управляет всеми устройствами компьютерной системы

(процессорами, оперативной памятью, дисками, клавиатурой, монитором, принтерами, сетевыми устройствами и др.) и обеспечивает пользователя удобным интерфейсом для работы с аппаратурой Рисунок 1.

Обычно ОС определяется через ее функции - под ОС понимают комплекс управляющих и обрабатывающих программ, который, с одной стороны, выступает как интерфейс между аппаратурой компьютера и пользователем с его задачами, а с другой - предназначен для наиболее эффективного использования ресурсов вычислительной системы и организации надежных вычислений.

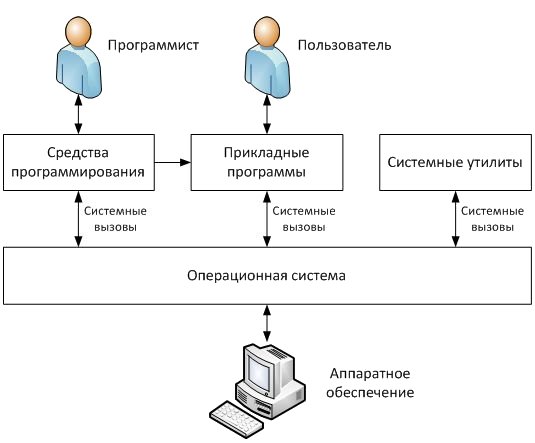


Рисунок 1 – Наглядное понимание роли ОС

Функции операционной системы:

* обеспечение выполнения программ – загрузка программ в память, предоставление программам процессорного времени, обработка системных вызовов;
* управление оперативной памятью – эффективное выделение памяти программам, учет свободной и занятой памяти;
* управление внешней памятью – поддержка различных файловых систем;
* управление вводом-выводом – обеспечение работы с различными периферийными устройствами;
* предоставление пользовательского интерфейса;
* обеспечение безопасности – защита информации и других ресурсов системы от несанкционированного использования;
* организация сетевого взаимодействия.

Все функции реализованы в виде подсистем, которые в свою очередь являются структурными компонентами ОС.

## **Подсистема управления процессами**

Планирование процессов — это деятельность диспетчера процессов, которая обрабатывает удаление запущенного процесса из процессора и выбор другого процесса на основе конкретной стратегии.

Планирование процессов является неотъемлемой частью многопрограммных операционных систем. Эти операционные системы позволяют одновременно загружать несколько процессов в исполняемую память, а установленный процесс разделяет процессор с использованием временного мультиплексирования.

Планировщики — это специализированное системное программное обеспечение, которое обрабатывает планирование процессов несколькими способами. Их основная задача - выбрать задания, которые будут отправлены в систему, и решить, какой процесс начать. Планировщики бывают трех типов:

* долгосрочный планировщик;
* краткосрочный планировщик;
* среднесрочный планировщик.

В данной работе будет рассматриваться краткосрочный планировщик. Он также называется планировщиком ЦП. Его основная цель — повысить производительность системы в соответствии с выбранным набором критериев. Это изменение состояния готовности в рабочее состояние процесса. Планировщик ЦП выбирает процесс среди процессов, готовых к выполнению, и выделяет ЦП одному из них.

Многоуровневая очередь (Multilevel Queue) — это алгоритм, сочетающий несколько очередей со своими алгоритмами планирования (FCFS, RR, SJF, приоритетное планирование, и т.д.), каждая такая очередь имеет фиксированный приоритет. И, например, если в очереди высокого приоритета будет хоть одна не выполненная задача, то процессорное время не будет предоставлено очереди более низкого приоритета, пока не выполнятся все задачи высокого уровня.

Существует два основных типа методов многоуровневой очереди:

* многоуровневая очередь без обратной связи;
* многоуровневая очередь с обратной связью.

Многоуровневая очередь с обратной связью имеет тот же алгоритм что и многоуровневая очередь без обратной связи, за исключением того, что процессы могут мигрировать в другие очереди.

### **1.3.1 Многоуровневая очередь без обратной связи**

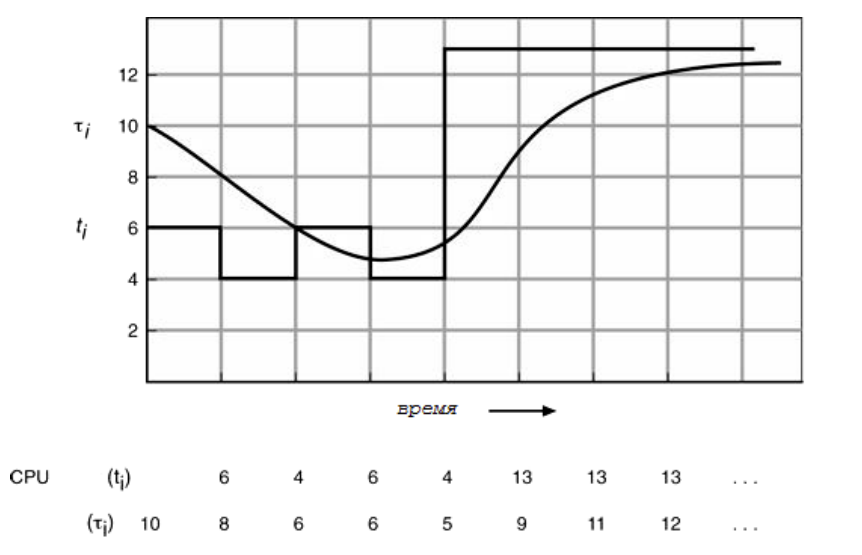
Хотя алгоритм SJF и оптимален, в общем случае его невозможно реализовать на уровне планирования, так как нет возможности узнать длину следующего интервала счета. Один из подходов состоит в аппроксимации планирования SJF - планировщик не может знать длину следующего интервала счета, но может предсказать ее.

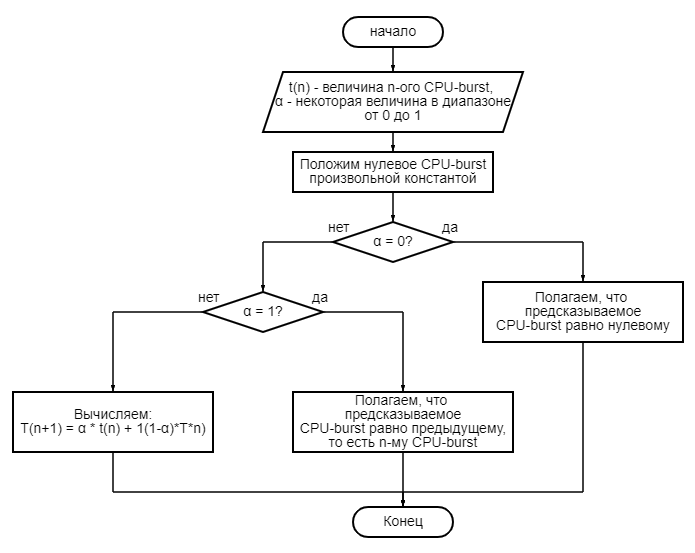
Предсказать следующий интервал нам позволяет SJF с предсказанием (Preemptive SJF Scheduling).

Предположим, что каждый следующий цикл счета будет похож на предыдущие. Таким образом, вычисляя приблизительную длину следующего цикла счета, можно выбирать процесс с самым коротким предсказанным циклом счета.

Для предсказания длины следующего интервала счета обычно используется экспоненциальное среднее измеренных длин предыдущих циклов счета. Обозначим длину n-го цикла счета tn , а предсказанное значение следующего цикла счета τn+1 . Тогда, для некоторого

Эта формула определяет экспоненциальное среднее. Значение tn соответствует самым последним данным, а τn — истории предыдущих измерений. Параметр α управляет относительным весом последнего измерения и истории измерений. Если α = 0, то , и последняя информация не играет никакой роли (текущие условия предполагаются неустойчивыми); если же α = 1, тогда τn+1 = tn, и только последний цикл счета играет роль (история измерений предполагается неправильной и случайной). Обычно α = 1/2, так что последнее измерения и прошлые измерения имеют одинаковый вес. Начальное значение 0 τ может быть задано константой или средним значением для всей системы. На рисунке 2 показано экспоненциальное среднее при α = 1/2 и τ0 =10.На рисунке 3 показана блок-схема алгоритма.

  
Рисунок 2 - Прогнозирование длины цикла работы

  
Рисунок 3 – Блок-схема алгоритма SJF с предсказанием

Алгоритм SJF может быть как вытесняющим, так и невытесняющим. Выбор появляется в тот момент, когда новый процесс попадает в очередь, а предыдущий процесс все еще исполняется. Новый процесс может иметь более короткий следующий цикл счета, чем оставшийся цикл выполняющегося процесса. Вытесняющий алгоритм SJF прервет текущий процесс, а вариант без вытеснения позволит текущему процессу закончить свой цикл. Планирование SJF с вытеснением иногда называется алгоритмом «кратчайший оставшийся участок — первым» (shortest-remaining-time-first).

Для примера рассмотрим следующие четыре процесса на Таблице 1.

Таблица 1 – Процессы для рассмотрения

|  |  |  |
| --- | --- | --- |
| Процесс | Время поступления | Участок счета |
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

Алгоритм SJF с вытеснением будет работать, как показано на рисунке 4.

  
Рисунок 4 – Работа алгоритма SJF с вытеснением

В момент времени 0 начинает исполняться процесс P1, поскольку он является единственным процессом в очереди. Процесс P2 поступает в момент времени 1. Оставшееся время процесса P1 (7 миллисекунд) больше, чем время, требующееся процессу P2 (4 миллисекунды), таким образом, процесс P1 приостанавливается и начинается выполнение процесс P2. Среднее время ожидания для этого примера 6,5 миллисекунд. Алгоритм SJF без вытеснения даст среднее время ожидания 7,75 миллисекунд.

## **Подсистема управления памятью**

Оперативная память (ОП) — один из важнейших ресурсов ОС, без которого невозможна работа на персональном компьютере. Таким образом, изучение механизмов использования памяти в отношении различных задач и самих операционных систем связано с многопоточными механизмами, событиями, прерываниями и т.п. С появлением первых поколений компьютеров им было введено понятие ОЗУ, часто называемое физической памятью. ОЗУ представляет собой упорядоченный набор ячеек, и все они пронумерованы, что означает, что к каждой из них можно получить доступ, введя порядковый номер (адрес). Количество ячеек физической памяти ограничено и стабильно. Центральный процессор извлекает команды и данные из оп и помещает результаты вычислений в оп. В то же время, с точки зрения программиста, ОП — это область, в которой можно работать с некоторыми логическими именами (переменными), которые могут быть символическими или числовыми. Следует отметить, что ряд переменных не сортируется, хотя отдельные элементы могут быть размещены в определенном порядке, например, в элементах массива. Имена переменных и точки входа программных модулей создают пространство имен.

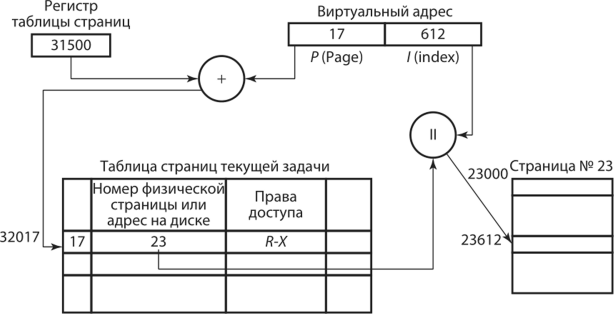
Операционная система должна привязать каждое имя, указанное пользователем, к месту физической памяти, то есть сопоставить пространство имен с физической памятью компьютера. Как правило, это сопоставление выполняется в два этапа: сначала системой программирования, а затем операционной системой, управляющей соответствующими аппаратными средствами. В современных ОС переход с первого этапа на второй осуществляется через промежуточный виртуальный или логический адрес. Однако можно сказать, что набор всех разрешенных для некоторых программ значений виртуального адреса определяет виртуальное адресное пространство или виртуальную память. Виртуальное адресное пространство в первую очередь зависит от архитектуры процессора и системы программирования и практически не зависит от объема фактической физической памяти, установленной на персональном компьютере.

### **1.4.1 Страничное распределение памяти**

При страничном распределении все фрагменты программы, на которые она разбивается (за исключением последней ее части), получаются одинаковыми. Одинаковыми полагаются и единицы памяти, которые предоставляются для размещения фрагментов программы. Эти одинаковые части называют страницами и говорят, что память разбивается на физические страницы, а программа — на виртуальные страницы. Часть виртуальных страниц задачи размещается в оперативной памяти, а часть — во внешней. Обычно место во внешней памяти, в качестве которой в абсолютном большинстве случаев выступают накопители на магнитных дисках (поскольку они относятся к быстродействующим устройствам с прямым доступом), называют файлом подкачки или страничным файлом (paging file). Иногда этот файл называют swap-файлом, тем самым подчеркивая, что записи этого файла — страницы — замещают друг друга в оперативной памяти. В некоторых ОС выгруженные страницы располагаются не в файле, а в специальном разделе дискового пространства. В UNIX-системах для этих целей выделяется специальный раздел, но кроме него могут быть использованы и файлы, выполняющие те же функции, если объема раздела недостаточно.

Разбиение всей оперативной памяти на страницы одинаковой величины, причем величина каждой страницы выбирается кратной степени двойки, приводит к тому, что вместо одномерного адресного пространства памяти можно говорить о двумерном. Первая координата адресного пространства — это номер страницы, а вторая координата — номер ячейки внутри выбранной страницы (его называют индексом). Таким образом, физический адрес определяется парой (Рр, /), а виртуальный адрес — парой (Pv, /), где Pv — это номер виртуальной страницы; Рр — это номер физической страницы и i — это индекс ячейки внутри страницы. Количество битов, отводимое под индекс, определяет размер страницы, а количество битов, отводимое под номер виртуальной страницы, — объем возможной виртуальной памяти, которой может пользоваться программа. Отображение, осуществляемое системой во время исполнения, сводится к отображению Pv в Рр и приписывании к полученному значению битов адреса, задаваемых величиной i. При этом нет необходимости ограничивать число виртуальных страниц числом физических, т.е. непоместившиеся страницы можно размещать во внешней памяти, которая в данном случае служит расширением оперативной.

Для отображения виртуального адресного пространства задачи на физическую память, как и в случае с сегментным способом организации, для каждой задачи необходимо иметь таблицу страниц для трансляции адресных пространств. Для описания каждой страницы диспетчер памяти ОС заводит соответствующий дескриптор, который отличается от дескриптора сегмента прежде всего тем, что в нем нет необходимости иметь поле длины, — ведь все страницы имеют одинаковый размер. По номеру виртуальной страницы в таблице дескрипторов страниц текущей задачи находится соответствующий элемент (дескриптор). Если бит присутствия имеет единичное значение, значит, данная страница сейчас размещена в оперативной, а не во внешней памяти и мы в дескрипторе имеем номер физической страницы, отведенной под данную виртуальную. Если же бит присутствия равен нулю, то в дескрипторе мы будем иметь адрес виртуальной страницы, расположенной сейчас во внешней памяти. Таким образом и осуществляется трансляция виртуального адресного пространства на физическую память. Этот механизм трансляции проиллюстрирован на рисунке 5.

  
Рисунок 5 – Страничный способ организации памяти

Защита памяти страницы основана на контроле уровня доступа к каждой странице. Обычно возможны следующие уровни доступа: только для чтения; чтение/запись; только выполнение. В таком случае каждой странице предоставляется соответствующий код уровня доступа. При преобразовании логического адреса в физический сравнивается значение кода разрешенного уровня доступа с фактически требуемым. При их несовпадении работа программы прерывается.

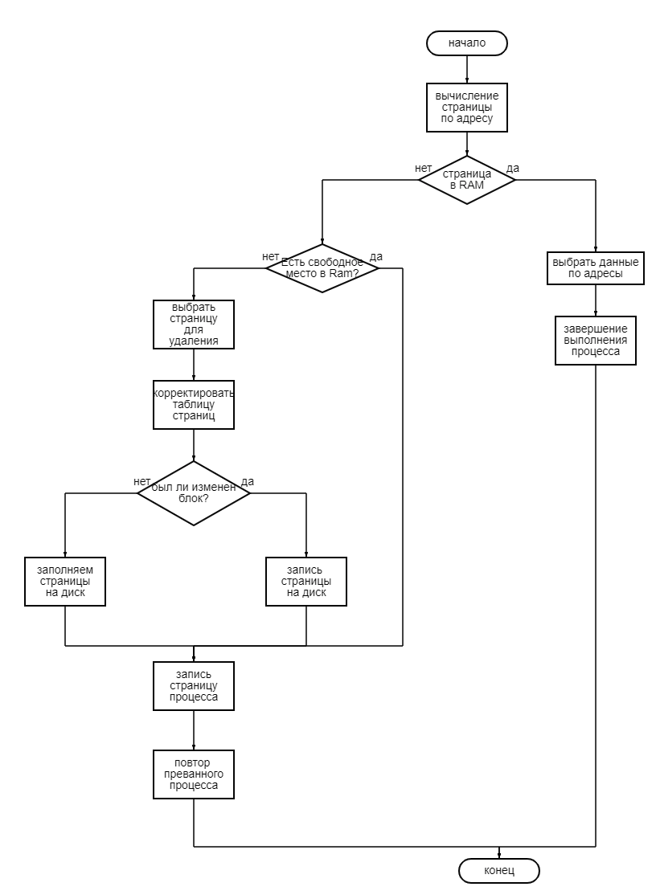
При обращении к виртуальной странице, не оказавшейся в данный момент в оперативной памяти, возникает прерывание и управление передается диспетчеру памяти, который должен найти свободное место. Обычно предоставляется первая же свободная страница. Если свободной физической страницы нет, то диспетчер памяти по одной из вышеупомянутых дисциплин замещения (LRU, LFU, FIFO, random) определит страницу, подлежащую расформированию или сохранению во внешней памяти. На ее место он разместит ту новую виртуальную страницу, к которой было обращение из задачи, но ее не оказалось в оперативной памяти.

Для использования дисциплин LRU и LFU в процессоре должны быть соответствующие аппаратные средства. В дескрипторе страницы размещается бит обращения, и этот бит становится единичным при обращении к дескриптору.

Если объем физической памяти невелик и даже часто нужные страницы не могут быть помещены в оперативную память, происходит так называемый сдвиг. Другими словами, пробуксовка — это ситуация, при которой загрузка нужной нам страницы вызывает перемещение во внешнюю память той страницы, с которой мы тоже активно работаем. Очевидно, что это очень плохое явление. Чтобы этого избежать, рекомендуется увеличить объем оперативной памяти (сейчас это стало самым простым решением), уменьшить количество задач, выполняемых параллельно, или попытаться использовать более эффективные дисциплины замещения.

Основным преимуществом страничного способа распределения памяти является минимально возможная фрагментация. Поскольку для каждой задачи может быть незаполненная страница, очевидно, что память может использоваться достаточно эффективно.

На Рисунке 6 показана блок-схема алгоритма страничного распределения.

  
Рисунок 6 – Блок-схема алгоритма страничного распределения

## **Подсистема управления файловой системы**

Файловая система позволяет программам выполнять ряд операций, достаточно простых для выполнения действий над некоторыми абстрактными объектами, представляющими файл.

В современных компьютерах часто в качестве основного устройства долговременной памяти используется жесткий диск. Физически данные на диске обрабатываются так же, как ячейки в массиве – по количеству блоков данных. Современный диск содержит миллиарды блоков, поэтому человеку практически невозможно работать с необработанными данными на диске. Фактически, файловая система предоставляет доступ только к группе блоков дисков, представляющих логически связанные данные, под значимым символическим именем.

Чтобы выполнить свою функцию, файловая система должна, как минимум, вести учет пустых и занятых блоков на диске, следить за тем, принадлежат ли блоки диска определенным файлам и порядок блоков в файле.

Самый простой вариант физической организации – непрерывное размещение в наборе соседних кластеров. Основным преимуществом этого метода является высокая скорость доступа, так как затраты на поиск и считывание кластеров файла минимальны. На рисунке 7 показана схема непрерывного размещения.

  
Рисунок 7 – Непрерывное размещение файлов

### **1.5.1 Непрерывное размещение файлов**

Физическая организация выделяет способ непрерывного размещения файлов на диске и учет соответствия блоков диска файлам.

У непрерывного распределения дискового пространства есть два существенных преимущества.

Во-первых, его просто реализовать, поскольку отслеживание местонахождения принадлежащих файлу блоков сводится всего лишь к запоминанию двух чисел: дискового адреса первого блока и количества блоков в файле. При наличии номера первого блока номер любого другого блока может быть вычислен путем простого сложения.

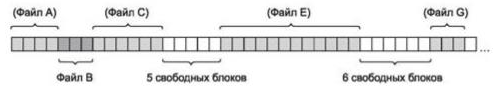
Во-вторых, он обладает превосходной производительностью чтения, так как весь файл можно читать с диска за один процесс. Для нее потребуется только одна операция позиционирования (на первый блок). После этого никаких позиционирований или ожиданий подхода нужного сектора диска уже не потребуется, поэтому данные поступают на скорости, равной максимальной пропускной способности диска.

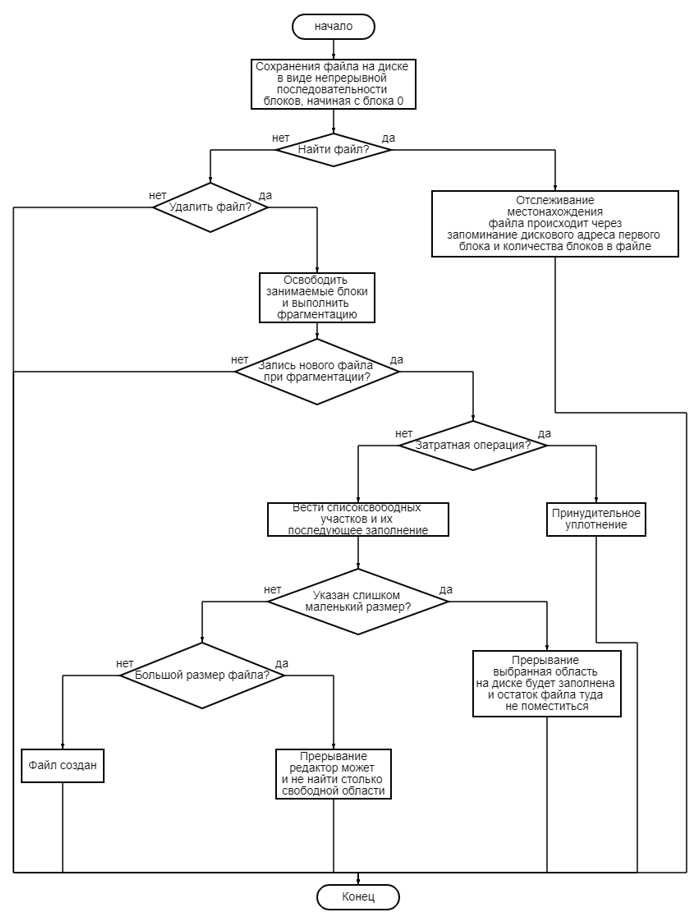
Таким образом, он характеризуется постоянным размещением, простотой применения и высокой производительностью.

Непрерывное размещение также имеет очень серьезный недостаток: со временем диск фрагментируется. Естественно, при удалении файла его блоки освобождаются и на диске остается последовательность свободных блоков. Немедленное уплотнение файлов на диске для устранения такой последовательности свободных блоков («дыры») не осуществляется, поскольку для этого потребуется скопировать все блоки, — а их могут быть миллионы, — следующие за ней, что при использовании больших дисков займет несколько часов или даже дней. В результате диск содержит вперемешку файлы и последовательности свободных блоков.

Сначала фрагментация не является проблемой, так как каждый новый файл можно записать в конец диска, следуя предыдущему файлу. Но со временем диск будет заполнен и его придется либо декомпрессировать, что является очень дорогостоящим процессом, либо придется повторно использовать пустые массивы блоков между файлами, для этого нужно будет вести список таких свободных мест, что вполне возможно. Однако при создании нового файла необходимо знать его окончательный размер, чтобы выбрать подходящее место для размещения.

Он не сможет продолжать работать, не ответив на этот вопрос. Если в итоге был определен очень маленький размер, то программа вынуждена преждевременно перестать работать, так как выбранное пространство на диске заполнено, а остальная часть файла там не поместится. Если пользователь попытается обойти эту проблему, намеренно указав большой конечный объем, предполагая, что он составляет 1 Гбайт, редактор может не найти такого большого свободного места и сообщить, что файл будет создан рисунок 8. Блок-схема алгоритма показана на Рисунке 9.

  
Рисунок 8 – Состояние диска после   
удаления файлов и приведение дискового пространства к фрагментации

Рисунок 9 – Блок-схема алгоритма непрерывного размещения файлов

# **2 Практический раздел**

## **Алгоритм страничного распределения памяти**

Входные данные:

* количество процессов;
* размер оперативной памяти;
* размер виртуальной памяти;
* размер процесса.

Выходные данные:

* количество занятых страниц;
* информация о количестве страниц оперативной и виртуальной памяти до и после распределения.

Программа начинается с объявления переменных.

rp- размер памяти

rc -размер страницы

voc- размер памяти

vvp- размер виртуальной памяти

cv- свободное место в последнем блоке

kc- количество страниц

vockc-количество страниц оперативной памяти

vvpkc-количество страниц виртуальной памяти  
page = 256 байт(страница);  
n < 10000.

Пользователь вводит количество процессов (n), размер оперативной памяти (ram) и виртуальной (virtualMemory). Далее, происходит вычисление, сколько страниц может поместить оперативная и виртуальная память. Для оперативной памяти используются формула – . Для виртуальной –

С помощью цикла для каждого процесса происходит вычисление следующих позиций:

* свободное место в последней странице);
* остаток страниц в виртуальной и оперативной памяти после размещения файлов;
* количество страниц, занимаемых процессом(kc), при условии, что размер процесса кратен размеру страницы оно будет равно делению разницы размера процесса на остаток от деления размера процесса на размер страницы . В противном случае оно будет равно делению разницы размера процесса на остаток от деления размера процесса на размер страницы плюс

Если количество страниц оперативной и виртуальной памяти не будет хватать, то программа выведет на экран «ошибка».

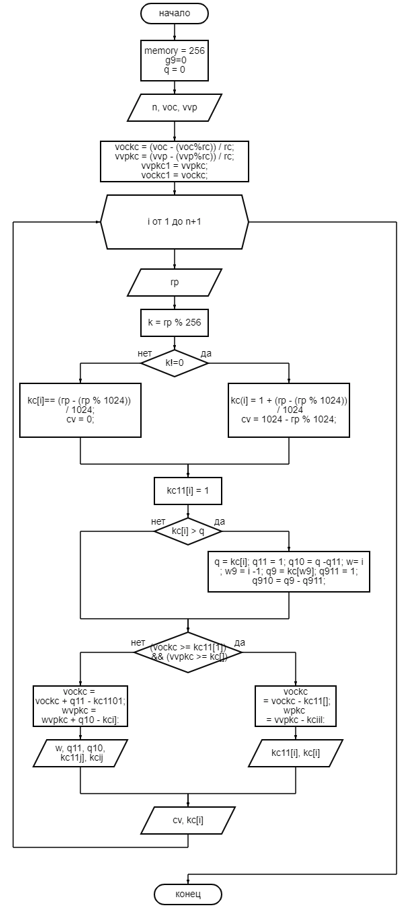
Программа выводит на экран для каждого файла следующую информацию:

* размер страницы (page =256);
* количество страниц занято (occupied);
* свободно в последнем кластере (freeKlaster).

После того как цикл закончился программа выводит на экран:

* страницы в виртуальной памяти после размещения файлов(pagesInVirtualMemory);
* страницы в оперативной памяти после размещения файлов (pagesInRAM);
* остаток страниц в виртуальной памяти после размещения файлов (remainingPagesInVirtualMemory);
* остаток страниц в оперативной памяти после размещения файлов (remainingPagesInRAM).

Блок-схема алгоритма показана на рисунке 10.

 **Рисунок 10 – Блок-схема страничного распределения памяти**

## **Алгоритм планирования SJF**

При изучении работы алгоритма SJF для его функционирования принимаем:

Информация о процессах в формате {id, burst, time}, где:

* id – идентификатор процесса;
* burst – время исполнения процесса;
* time – время прихода процесса.

При упреждающем планировании SJF, когда приходит процесс, операционная система помещает его в очередь готовности. Процесс с наименьшим временем выполнения пакета запускает его выполнение. Теперь, если прибудет какой-то новый процесс с наименьшим кратчайшим временем, в этом случае текущий процесс будет снят с выполнения, и более короткий, чем самый короткий процесс, выиграет цикл процессора.

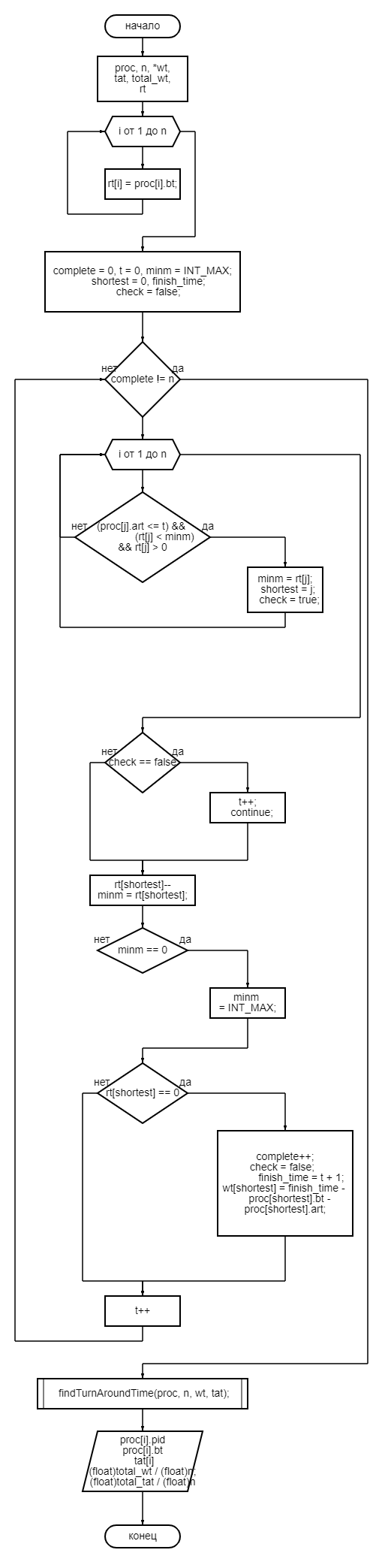
Реализация моего алгоритма:

* находится процесс с минимальным оставшимся временем в каждый отдельный временной круг;
* сокращаю время найденного процесса на 1;
* проверяю становится ли оставшееся время равным 0;
* увеличиваю счетчик завершения процесса;
* происходит расчёт время ожиданий для каждого процесса.

void findWaitingTime(Process proc[], int n, int wt[]) – функция для расчета среднего времени процесса. Ищется процесс с минимальным оставшимся временем среди процессов, которые поступают в текущее время. У найденного процесса сокращается оставшееся время на один.

void findTurnAroundTime(Process proc[], int n, int wt[], int tat[]) – функция для расчета общего времени выполнения всех процессов.

Блок-схема алгоритма изображена на рисунке 12.

  
Рисунок 12 – Блок-схема алгоритма SJF с предсказанием

# **3 Экспериментальный раздел**

## **3.1 Страничное распределение памяти**

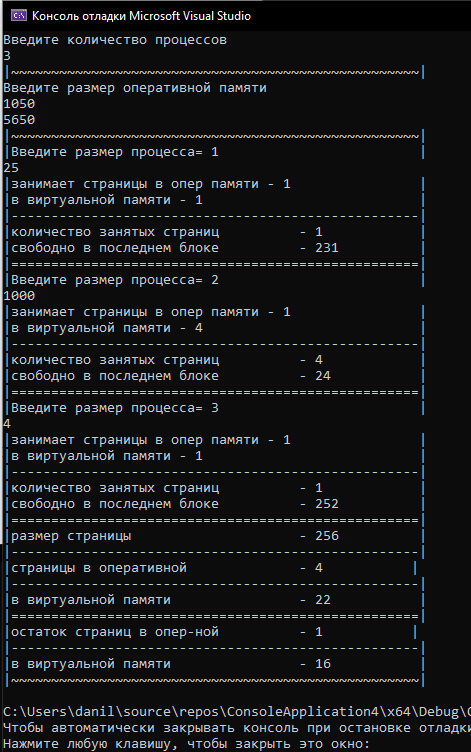
Входные данные:

* Количество процессов-3;
* размер оперативной памяти 1050;
* размер виртуальной памяти 5650;
* размер процесса 1-25;
* размер процесса 2-1000;
* размер процесса 3-4.

Выходные данные:

* Количество занятых страниц;
* информация об удаленных страницах;

Информация о количестве страниц оперативной и виртуальной памяти до и после распределения на рисунке 13.

  
Рисунок 13 – Результат работы страничного распределения памяти

## **3.****2 Алгоритм планирования SJF**

Входные данные:

* процессы.

Выходные данные:

* процессы вместе со всеми деталями;
* среднее время ожидания;
* среднее время оборота.

Информация о процессах и времени ожидания на рисунке 15.

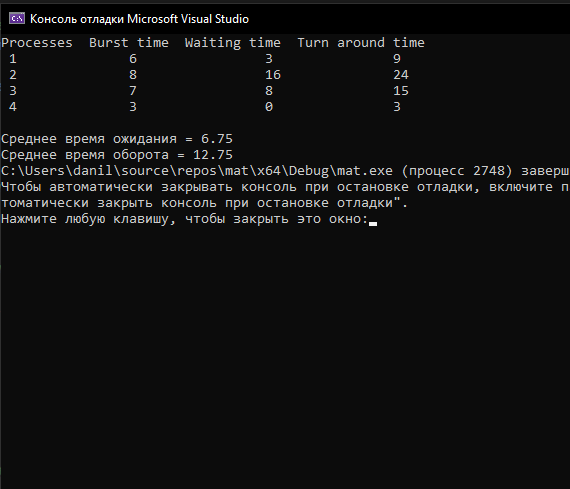


Рисунок 15 – Результат работы алгоритма планирования

# **ЗАКЛЮЧЕНИЕ**

В данной курсовой работе была проведена глубокая работа над понимаем подсистем операционной системы. По изученным моделям были разработаны алгоритмы управления процессами, памятью и файлами на языке программирования С. С помощью построенных моделей были подтверждены изученные теоретические знания. Все задачи решены, все цели были достигнуты.

# **Список использованных источников**

1. Тихомиров, В.П. Операционная система ДЕМОС: инструментальные средства программирования / [Текст] В.П. Тихомиров, М.И. Давидов. - М.: Финансы и статистика, 1988. - 206 c.
2. Грибанов, В.П. Операционные системы / [Текст] В.П. Грибанов, С.В. Дробин, В.Д. Медведев. - М.: Финансы и статистика, 1990. - 239 c.
3. Данилочкин Справочник системного программиста по операционной системе ОС ЕС / Данилочкин, В.П. и. - М.: Финансы и статистика, 1982. - 288 c.
4. Операционная система // FoxFord URL [Электронный ресурс]: https://foxford.ru/wiki/informatika/operatsionnaya-sistema (дата обращения: 15.10.2021).
5. Самая короткая работа сначала (SJF) // coderlessons URL [Электронный ресурс]: https://coderlessons.com/tutorials/akademicheskii/osnovy-operatsionnykh-sistem/27-samaia-korotkaia-rabota-snachala-sjf (дата обращения: 17.10.2021).
6. Operating Systems: Three Easy Pieces. Part 4 // habr URL [Электронный ресурс]: https://habr.com/ru/post/449026/ (дата обращения: 19.10.2021).

# **Приложение А**

Алгоритм планирования SJF с предсказаниями

#include <iostream>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

// Функция для расчета среднего времени

void findWaitingTime(Process proc[], int n,

int wt[])

{

int \*rt = new int[n];

for (int i = 0; i < n; i++)

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

/\*Процесс до тех пор, пока все процессы не получат

завершенный\*/

while (complete != n) {

/\*Ищется процесс с минимальным

оставшимся временем среди

процессов, которые поступают до

текущего времени\*/

for (int j = 0; j < n; j++) {

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Сокращается оставшееся время на один

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)

minm = INT\_MAX;

if (rt[shortest] == 0) {

complete++;

check = false;

finish\_time = t + 1;

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

t++;

}

}

// Функция для расчета времени разворота

void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

for (int i = 0; i < n; i++)

tat[i] = proc[i].bt + wt[i];

}

// Функция для расчета среднего времени

void findavgTime(Process proc[], int n)

{

int \*wt = new int[n], \*tat = new int[n]

, total\_wt = 0,

total\_tat = 0;

// Функция для определения времени ожидания всех

// процессы

findWaitingTime(proc, n, wt);

// Функция для определения времени выполнения

// всех процессов

findTurnAroundTime(proc, n, wt, tat);

cout << "Processes "

<< " Burst time "

<< " Waiting time "

<< " Turn around time\n";

// Рассчитать общее время ожидания и

// общее время выполнения работ

for (int i = 0; i < n; i++) {

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t\t " << wt[i]

<< "\t\t " << tat[i] << endl;

}

cout << "\nСреднее время ожидания = "

<< (float)total\_wt / (float)n;

cout << "\nСреднее время оборота = "

<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

Process proc[] = { { 1, 6, 1 }, { 2, 8, 1 },

{ 3, 7, 2 }, { 4, 3, 3 } };

int n = sizeof(proc) / sizeof(proc[0]);

findavgTime(proc, n);

return 0;

}

Страничное распределение памяти

void pageDistribution() {

int qwer, vvp, memory, cv, k, rp, ccc[10000]

cout << "Введите количество процессов \n";

cin >> n;

printf("|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n");

printf("Введите размер оперативной памяти \n");

scanf("%u %u", &voc, &vvp);

int w9, q, q11, q10, w, voc, vockc

int vvpkc, vockc1,i, n, kcq1[10000], kc10[10000];

memory = 256;

q9 = 0;

q = 0;

vockc = (voc - (voc % memory)) / memory;

vvpkc = (vvp - (vvp % memory)) / memory;

vvpkc1 = vvpkc;

vockc1 = vockc;

printf("|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n");

for (i = 1; i < n + 1; i++)

{

printf("|Введите размер процесса= %-25u |\n", i);

scanf("%u", &rp);

k = rp % 256;

if (k != 0)

{

kc[i] = 1 + (rp - (rp % 256)) / 256;

cv = 256 - rp % 256;

}

else

{

kc[i] = (rp - (rp % 256)) / 256;

cv = 0;

}

kc11[i] = 1;

if (kc[i] > q)

{

q = kc[i];

q11 = 1;

q10 = q - q11;

w = i;

w9 = i - 1;

q9 = kc[w9];

q911 = 1;

q910 = q9 - q911;

}

if ((vockc >= kc11[i]) && (vvpkc >= kc[i]))

{

vockc = vockc - kc11[i];

vvpkc = vvpkc - kc[i];

printf("|занимает страницы в опер памяти – %-17u|\n", kc11[i]);

printf("|в виртуальной памяти – %-28u|\n", kc[i]);

}

else

{

vockc = vockc + q11 - kc11[i];

vvpkc = vvpkc + q10 - kc[i];

printf("|кол удаленых стр в опер памяти %-3u пр-сса– %-8u|\n", w, q11);

printf("|занимает страницы в опер памяти – %-8u|\n", kc11[i]);

printf("| в виртуальной памяти | %-8u|\n", kc[i]);

}

printf("|---------------------------------------------------|\n");

printf("|количество занятых страниц – %-13u|\n", kc[i]);

printf("|свободно в последнем блоке – %-13u|\n", cv);

printf("|===================================================|\n");

}

printf("|размер страницы – %-13u|\n", memory);

printf("|---------------------------------------------------|\n");

printf("|страницы в оперативной – %-12u|\n", vockc1);

printf("|---------------------------------------------------|\n");

printf("|в виртуальной памяти – %-13u|\n", vvpkc1);

printf("|===================================================|\n");

printf("|остаток страниц в опер-ной – %-12u|\n", vockc);

printf("|---------------------------------------------------|\n");

printf("|в виртуальной памяти – %-13u|\n", vvpkc);

printf("|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|\n");

}

