МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

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

А л г о р и т м ы у п р а в л е н и я п а м я т ь ю

Методические указания к лабораторным работам

по курсу «Системное программирование»

2011

Автор: КУПРИЯНОВ Александр Викторович

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

Цель работы: изучить основы страничного распределения памяти, получить представление о преимуществах и недостатках, присущих различным методам замещения страниц.

1. Краткие теоретические сведения

На рисунке 1 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами (virtual pages). В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

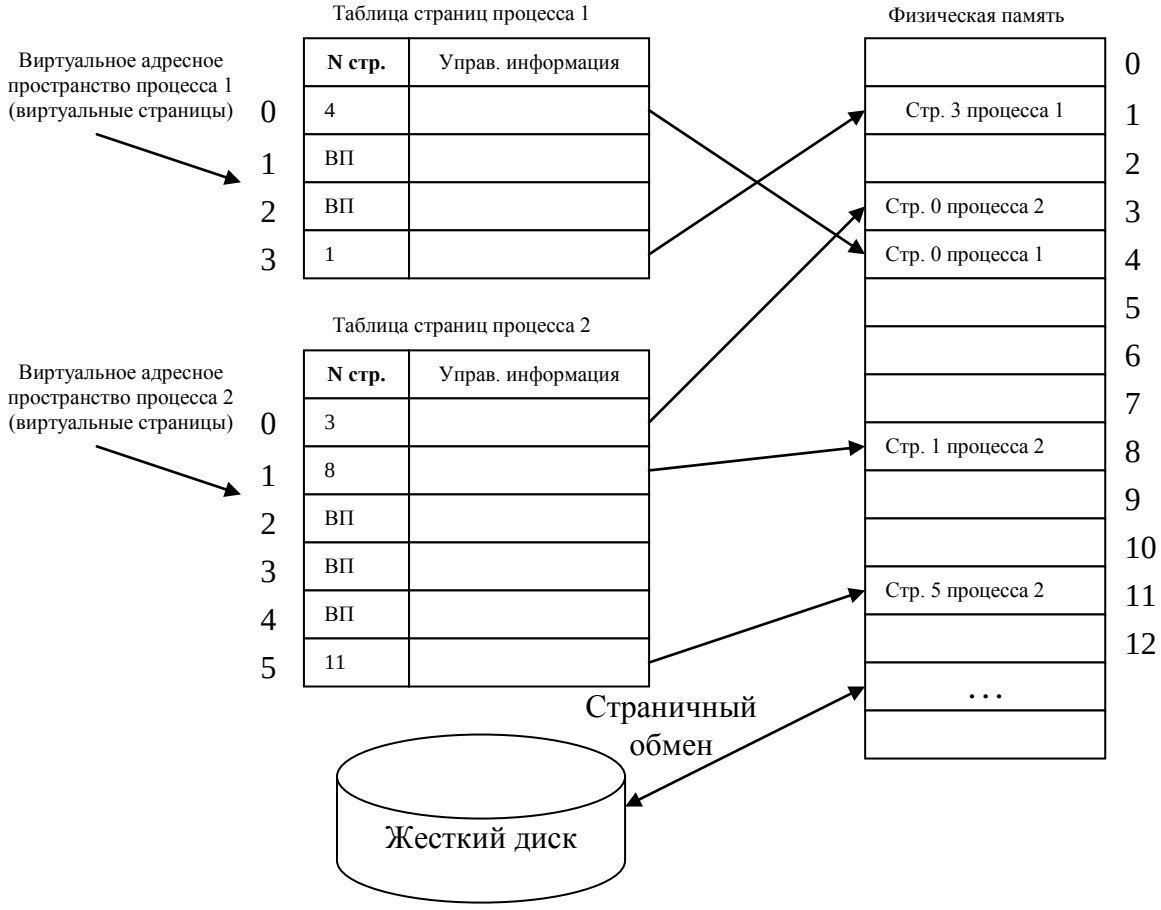

Рис. 1 – Страничное распределение памяти

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками, или кадрами).

Размер страницы выбирается равным степени двойки: 512, 1024, 4096 байт и т. д. Это позволяет упростить механизм преобразования адресов.

Операционная система при создании процесса загружает в оперативную память несколько его виртуальных страниц (начальные

страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. Для каждого процесса операционная система создает таблицу страниц – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

Запись таблицы, называемая дескриптором страницы, включает следующую информацию:

- номер физической страницы, в которую загружена данная виртуальная страница;
- признак присутствия, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти;
- признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;
- признак обращения к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Информация из таблиц страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический.

При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц, и из него извлекается описывающая страницу информация. Далее анализируется признак присутствия, и, если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический, то есть виртуальный адрес заменяется указанным в записи таблицы физическим адресом.

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 процессов, находящихся в состоянии готовности. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу (для этого операционная система должна помнить положение вытесненной страницы в страничном файле диска) и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

При страничной организации памяти есть 2 проблемы: 1) таблица страниц может быть слишком большой; 2) отображение страниц должно быть быстрым.

Для решения первой проблемы используют многоуровневые таблицы памяти, при использовании которых в памяти находятся только части таблицы страниц.

Для решения второй компьютер снабжается небольшим аппаратным устройством, служащим для отображения виртуальных адресов в физические без прохода по таблице страниц. Это устройство называется буфером быстрого преобразования адреса (TLB – Translation Lookaside Buffer) или ассоциативной памятью.

Большинство программ склонно делать огромное количество обращений к небольшому количеству страниц, а не наоборот. Таким образом, в таблице страниц только малая доля записей читается интенсивно, остальная часть едва ли вообще используется. Поэтому эта малая доля записей копируется в TLB, который работает гораздо быстрее стандартного обращения к таблице страниц.

2. Исключительные ситуации при работе с памятью

Отображение виртуального адреса в физический осуществляется при помощи таблицы страниц. Для каждой виртуальной страницы запись в таблице страниц содержит номер соответствующего страничного кадра в оперативной памяти, а также атрибуты страницы для контроля обращений к памяти.

Что же происходит, когда нужной страницы в памяти нет или операция обращения к памяти недопустима? Естественно, что операционная система должна быть как-то оповещена о происшедшем. Обычно для этого используется механизм исключительных ситуаций (exceptions). При попытке выполнить подобное обращение к виртуальной странице возникает исключительная ситуация ***"страничное нарушение"*** (*page fault*), приводящая к вызову специальной последовательности команд для обработки конкретного вида *страничного нарушения*.

Страничное нарушение может происходить в самых разных случаях: при отсутствии страницы в оперативной памяти, при попытке записи в страницу с атрибутом "только чтение" или при попытке чтения или записи страницы с атрибутом "только выполнение". В любом из этих случаев вызывается обработчик *страничного нарушения*, являющийся частью операционной системы. Ему обычно передается причина возникновения исключительной ситуации и виртуальный адрес, обращение к которому вызвало нарушение.

Нас будет интересовать конкретный вариант *страничного нарушения* - обращение к отсутствующей странице, поскольку именно его обработка во многом определяет производительность страничной системы. Когда программа обращается к виртуальной странице, отсутствующей в основной

памяти, операционная система должна выделить страницу основной памяти, переместить в нее копию виртуальной страницы из внешней памяти и модифицировать соответствующий элемент таблицы страниц.

Повышение производительности вычислительной системы может быть достигнуто за счет уменьшения частоты *страничных нарушений*, а также за счет увеличения скорости их обработки. Время эффективного доступа к отсутствующей в оперативной памяти странице складывается из:

- обслуживания исключительной ситуации (*page fault*);
- чтения (подкачки) страницы из вторичной памяти (иногда, при недостатке места в основной памяти, необходимо вытолкнуть одну из страниц из основной памяти во вторичную, то есть осуществить замещение страницы);
- возобновления выполнения процесса, вызвавшего данный *page fault*.

Для решения первой и третьей задач ОС выполняет до нескольких сот машинных инструкций в течение нескольких десятков микросекунд. Время подкачки страницы близко к нескольким десяткам миллисекунд. Проведенные исследования показывают, что вероятности *page fault* $5x10^{-7}$ оказывается достаточно, чтобы снизить производительность страничной схемы управления памятью на 10%. Таким образом, уменьшение частоты *page faults* является одной из ключевых задач системы управления памятью. Ее решение обычно связано с правильным выбором алгоритма замещения страниц.

3. Алгоритмы замещения страниц

Итак, наиболее ответственным действием менеджера памяти является выделение кадра оперативной памяти для размещения в ней виртуальной страницы, находящейся во внешней памяти. Напомним, что мы рассматриваем ситуацию, когда размер виртуальной памяти для каждого процесса может существенно превосходить размер основной памяти. Это означает, что при выделении страницы основной памяти с большой вероятностью не удастся найти свободный страничный кадр. В этом случае операционная система в соответствии с заложенными в нее критериями должна:

- найти некоторую занятую страницу основной памяти;
- переместить в случае надобности ее содержимое во внешнюю память;
- переписать в этот страничный кадр содержимое нужной виртуальной страницы из внешней памяти;
- должным образом модифицировать необходимый элемент соответствующей таблицы страниц;

- продолжить выполнение процесса, которому эта виртуальная страница понадобилась.

Заметим, что при замещении приходится дважды передавать страницу между основной и вторичной памятью. Процесс замещения может быть оптимизирован за счет использования бита модификации (один из атрибутов страницы в таблице страниц). Бит модификации устанавливается компьютером, если хотя бы один байт был записан на страницу. При выборе кандидата на замещение проверяется бит модификации. Если бит не установлен, нет необходимости переписывать данную страницу на диск, ее копия на диске уже имеется. Подобный метод также применяется к read-only-страницам, они никогда не модифицируются. Эта схема уменьшает время обработки *page fault*.

Существует большое количество разнообразных алгоритмов замещения страниц. Все они делятся на локальные и глобальные. Локальные алгоритмы, в отличие от глобальных, распределяют фиксированное или динамически настраиваемое число страниц для каждого процесса. Когда процесс израсходует все предназначенные ему страницы, система будет удалять из физической памяти одну из его страниц, а не из страниц других процессов. Глобальный же алгоритм замещения в случае возникновения исключительной ситуации удовлетворится освобождением любой физической страницы, независимо от того, какому процессу она принадлежала.

Глобальные алгоритмы имеют ряд недостатков. Во-первых, они делают одни процессы чувствительными к поведению других процессов. Например, если один процесс в системе одновременно использует большое количество страниц памяти, то все остальные приложения будут в результате ощущать сильное замедление из-за недостатка кадров памяти для своей работы. Во-вторых, некорректно работающее приложение может подорвать работу всей системы (если, конечно, в системе не предусмотрено ограничение на размер памяти, выделяемой процессу), пытаясь захватить больше памяти. Поэтому в многозадачной системе иногда приходится использовать более сложные локальные алгоритмы. Применение локальных алгоритмов требует хранения в операционной системе списка физических кадров, выделенных каждому процессу. Этот список страниц иногда называют **резидентным множеством** процесса. В одном из следующих разделов рассмотрен вариант алгоритма подкачки, основанный на приведении резидентного множества в соответствие так называемому **рабочему набору** процесса.

Эффективность алгоритма обычно оценивается на конкретной последовательности ссылок к памяти, для которой подсчитывается число возникающих *page faults*. Эта последовательность называется **строкой обращений** (reference string). Мы можем генерировать строку обращений искусственным образом при помощи датчика случайных чисел или трассируя конкретную систему.

Рассмотрим основные алгоритмы управления страницами:

1. Оптимальный алгоритм

В тот момент, когда происходит страничное прерывание, в памяти находится некоторый набор страниц. К одной из этих страниц будет обращаться следующая команда процессора (к странице, содержащей требуемую команду). На другие страницы, возможно, не будет ссылок в течение следующих 10, 100 или даже 1000 команд. Каждая страница может быть помечена количеством команд, которые будут выполняться перед первым обращением к этой странице. Оптимальный страничный алгоритм просто сообщает, что должна быть выгружена страница с наибольшей меткой.

С этим алгоритмом связана только одна проблема: он невыполним. В момент страничного прерывания операционная система не имеет возможности узнать, когда произойдет следующее обращение к каждой странице.

2. Алгоритм NRU – не использовавшаяся в последнее время страница

Чтобы дать возможность операционной системе собирать полезные статистические данные о том, какие страницы используются, а какие – нет, большинство компьютеров с виртуальной памятью поддерживают два статусных бита, связанных с каждой страницей. Бит ***R*** (Referenced – обращения) устанавливается всякий раз, когда происходит обращение к странице (чтение или запись). Бит ***M*** (Modified – изменение) устанавливается, когда страница записывается (то есть изменяется). Биты содержатся в каждом элементе таблицы страниц. Если аппаратное обеспечение не поддерживает эти биты, их можно смоделировать.

Биты ***R*** и ***M*** могут использоваться для построения простого алгоритма замещения страниц, описанного ниже. Когда процесс запускается, оба страничных бита для всех его страниц операционной системой установлены на 0. Периодически (например, при каждом прерывании по таймеру) бит ***R*** очищается, чтобы отличить страницы, к которым давно не происходило обращения от тех, на которые были ссылки. Когда возникает страничное прерывание, операционная система проверяет все страницы и делит их на четыре категории на основании текущих значений битов ***R*** и ***M***:

- класс 0: не было обращений и изменений;
- класс 1: не было обращений, страница изменена;
- класс 2: было обращение, страница не изменена;
- класс 3: произошло и обращение, и изменение.

Хотя класс 1 на первый взгляд кажется невозможным, такое случается, когда у страницы из класса 3 бит ***R*** сбрасывается во время прерывания по таймеру. Прерывания по таймеру не стирают бит ***M***, потому что эта информация необходима для того, чтобы знать, нужно ли переписывать страницу на диске или нет. Поэтому если бит ***R*** устанавливается на ноль, а ***M*** остается нетронутым, страница попадает в класс 1.

Алгоритм NRU (Not Recently Used) удаляет страницу с помощью случайного поиска в непустом классе с наименьшим номером. Привлекательность алгоритма NRU заключается в том, что он легок для понимания, умеренно сложен в реализации и дает производительность, которая может вполне оказаться достаточной.

3. Алгоритм FIFO – первым прибыл – первым обслужен

Операционная система поддерживает список всех страниц, находящихся в данный момент в памяти, в котором первая страница является старейшей, а страницы в хвосте списка попали в него совсем недавно. Когда происходит страничное прерывание, выгружается из памяти страница в голове списка, а новая страница добавляется в его конец. Данный алгоритм не используется, так как он может удалить наиболее часто вызываемую страницу.

4. Алгоритм «вторая попытка»

Модификация предыдущего алгоритма. Когда происходит страничное прерывание, то у самой «старейшей» страницы проверяется бит ***R***. Если он равен 0, т.е. страница не только дольше всех в памяти, но ещё и не используется, то страница заменяется новой. Если же бит равен 1, то странице даётся вторая попытка – бит изменяется в 0, а сама страница перемещается в конец очереди, т.е. становится самой «молодой».

5. Алгоритм «часы»

Предыдущий алгоритм является слишком неэффективным, потому что постоянно передвигает страницы по списку. Поэтому лучше хранить все страничные блоки в кольцевом списке в форме часов, при этом стрелка часов указывает на старейшую страницу.

Когда происходит страничное прерывание, проверяется та страница, на которую направлена стрелка. Если ее бит ***R*** равен 0, страница выгружается, на ее место в часовой круг встает новая страница, а стрелка сдвигается вперед на одну позицию. Если бит ***R*** равен 1, то он сбрасывается, стрелка перемещается к следующей странице. Этот процесс повторяется до тех пор, пока не находится та страница, у которой бит ***R*** = 0.

6. Алгоритм LRU – страница, не использовавшаяся дольше всего

Страницы, к которым происходит многократное обращение в нескольких последних командах, вероятно, также будут часто использоваться в следующих инструкциях. И наоборот, страницы, к которым ранее не возникало обращений, не будут употребляться в течение долгого времени. Эта идея привела к следующему реализуемому алгоритму: когда происходит страничное прерывание, выгружается из памяти страница, которая не использовалась дольше всего. Такая стратегия замещения страниц называется LRU (Least Recently Used – «менее недавно»).

Для полного осуществления алгоритма LRU необходимо поддерживать связный список всех содержащихся в памяти страниц, где последняя использовавшаяся страница находится в начале списка, а та, к которой дольше всего не было обращений – в конце. Сложность заключается в том, что список должен обновляться при каждом обращении к памяти. Поиск страницы, ее удаление, а затем вставка в начало списка – это операции, поглощающие очень много времени, даже если они выполняются аппаратно (если предположить, что необходимое оборудование можно сконструировать).

7. Алгоритм «старение»

Одна из разновидностей схемы LRU называется алгоритмом NFU (Not Frequently Used – редко использовавшаяся страница). Для него необходим программный счетчик, связанный с каждой страницей в памяти, изначально равный нулю. Во время каждого прерывания по таймеру операционная система исследует все страницы в памяти. Бит ***R*** каждой страницы (он равен 0 или 1) прибавляется к счетчику. В сущности, счетчики пытаются отследить, как часто происходило обращение к каждой странице. При страничном прерывании для замещения выбирается страница с наименьшим значением счетчика.

Основная проблема, возникающая при работе с алгоритмом NFU, заключается в том, что он никогда ничего не забывает. Например, в многоходовом компиляторе страницы, которые часто использовались во время первого прохода, могут все еще иметь высокое значение счетчика при более поздних проходах. Небольшие изменения позволяют решить эту проблему и достаточно хорошо моделировать алгоритм LRU:

- каждый счетчик сдвигается вправо на один разряд перед прибавлением бита ***R***;
- бит ***R*** добавляется в крайний слева, а не в крайний справа бит счетчика.

Когда происходит страничное прерывание, удаляется та страница, чей счетчик имеет наименьшую величину. Ясно, что счетчик страницы, к которой не было обращений, скажем, за четыре тика, будет начинаться с четырех нулей и, таким образом, иметь более низкое значение, чем счетчик страницы, на которую не ссылались в течение только трех тиков часов.

8. Алгоритм «рабочий набор»

В простейшей схеме страничной подкачки в момент запуска процессов нужные им страницы отсутствуют в памяти. Как только центральный процессор пытается выбрать первую команду, он получает страничное прерывание, побуждающее операционную систему перенести в память страницу, содержащую первую инструкцию. Обычно следом быстро происходят страничные прерывания для глобальных переменных и стека. Через некоторое время в памяти скапливается большинство необходимых процессу страниц, и он приступает к работе с относительно небольшим количеством ошибок из-за отсутствия страниц. Этот метод называется

замещением страниц по запросу (demand paging), потому что страницы загружаются в память по требованию, а не заранее.

Большинство процессов характеризуется тем, что во время выполнения любой фазы обращается к сравнительно небольшой части своих страниц.

Рабочий набор – множество страниц, которое процесс использует в данный момент.

Базовая идея алгоритма замещения страниц заключается в том, чтобы найти страницу, не включенную в рабочий набор, и выгрузить ее. Каждая запись информации о странице содержит (по крайней мере) два элемента информации: приближенное время, в которое страница использовалась в последний раз, и бит ***R*** (обращения).

Алгоритм работает следующим образом. Предполагается, что аппаратное обеспечение устанавливает биты ***R*** и ***M***, как в алгоритме NRU. Предполагается также, что периодическое прерывание по таймеру вызывает запуск программы, очищающей бит ***R*** при каждом тике часов. При каждом страничном прерывании исследуется таблица страниц и ищется страница, подходящая для удаления из памяти. Эта страница должна соответствовать следующим параметрам: бит ***R*** равен 0 и время последнего использования больше некоторой заранее заданной величины ***T***. Однако сканирование таблицы продолжается, обновляя остальные записи. Если проверена вся таблица, а кандидат на удаление не найден, это означает, что все страницы входят в рабочий набор. В этом случае, если были найдены одна или больше страниц с битом ***R*** = 0, удаляется та из них, которая имеет наибольший возраст.

Данный алгоритм очень громоздок, так как при каждом страничном прерывании следует проверять таблицу страниц до тех пор, пока не определится местоположение подходящего кандидата.

9. Алгоритм WSClock

Этот алгоритм является модификацией предыдущего. Для его использования необходима структура данных в виде кольцевого списка (Рис. 3). В исходном положении этот список пустой. Когда загружается первая страница, она добавляется в список. По мере прихода страниц они поступают в список, формируя кольцо. Каждая запись, кроме бита ***R*** и бита ***M***, содержит поле «время последнего использования» из базового алгоритма «рабочий набор».

Как и в случае алгоритма «часы», при каждом страничном прерывании первой проверяется та страница, на которую указывает стрелка. Если бит ***R*** равен 1, это значит, что страница использовалась в течение последнего такта часов, поэтому она не является идеальным кандидатом на удаление. Тогда бит ***R*** устанавливается на 0, стрелка передвигается на следующую страницу и для нее повторяется алгоритм.

Если в момент проверки бит ***R*** равен 0 и время последнего использования больше некоторой заранее заданной величины ***T***, то проверяется бит ***M*** – были ли изменения. Если нет, то страница удаляется.

Если изменения были – страница помечается как необходимая для копирования, а стрелка «часов» сдвигается.

Если стрелка часов обходит круг и возвращается обратно, то возможно два варианта: 1) запланирована операция переноса страницы на диск; 2) ничего не запланировано. В первом случае выбирается первая попавшаяся страница без изменений с битом ***R*** равным 0. Во втором случае предъявляются права на любую страницу.

Двумя наилучшими алгоритмами являются «старение» и WSCIock. Оба обеспечивают хорошую постраничную подкачку и могут быть реализованы за разумную цену.

Недостатки страничного распределения памяти – размеры страниц и частота страничных прерываний сильно влияют на производительность, все данные находятся перемешенными друг с другом.

4. Порядок выполнения работы

4.1. Исходные данные

- Вариант задания (предоставляется преподавателем);

4.2. Общий план выполнения работы

1 Реализовать предложенный алгоритм управления страницами.
2 Все параметры алгоритма задаются пользователем.
3 Реализовать имитацию обращения к страницам в системе случайным образом. Для каждой страницы генерируется время её появление в системе, общее время её существования в систему (определяется процессом) и промежутки времени в которые в систему происходит обращение к странице.
4 Пользователь задает параметры генератора страниц и обращения к ним, а также размер стека кадров.
5 Для полученного графика обращения к страницам в системе применить алгоритм планирования.
6 Вычислить количество страничных нарушений.
7 Для сравнения реализовать также оптимальный алгоритм ,и подсчитать количество страничных нарушений в при работе оптимального алгоритма.
8 Осуществить отображение на экране результатов планирования и выполнения обращения к страницам в виде схемы, графика или таблицы.
9 Для реализации приложения допускается использовать любой язык программирования.
10 Провести тестирование работы программы необходимо для трех различных наборов входных параметров и трех различных графиков появления и обращения к страницам в системе.
11 Написать отчет о проделанной работе. В отчете привести **подробное описание и визуальную схему алгоритма управления страницами**.
12 Сделать выводы и написать заключение.

4.3. Содержание отчета

Отчет по работе должен содержать:

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

5. Список рекомендуемой литературы

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

СОДЕРЖАНИЕ