**В.П. Кутепов**

**Структурная организация и распределение памяти в современных цифровых вычислительных машинах и системах**

**(учебное пособие)**

Оглавление

[1. ВВЕДЕНИЕ 3](#_Toc354479097)

[2. СТРУКТУРНАЯ ОРГАНИЗАЦИЯ И СПОСОБЫ РАСПРЕДЕЛЕНИЯ ПАМЯТИ В ЦВМ И ВС 4](#_Toc354479098)

[2.1. Технические характеристики различных устройств памяти и ее структурная организация 4](#_Toc354479099)

[2.2. Распределение памяти 6](#_Toc354479100)

[3. АНАЛИЗ ЭФФЕКТИВНОСТИ СТРАТЕГИЙ ДИНАМИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ПАМЯТИ 9](#_Toc354479101)

[3.1. Общая схема обслуживания программ в многопрограммном режиме 9](#_Toc354479102)

[3.2. Критерии эффективности организации и распределения памяти 11](#_Toc354479103)

[3.3. Размещение информации на устройствах памяти второго уровня и организация обслуживания запросов 13](#_Toc354479104)

[3.4. Характер поведения программы 15](#_Toc354479105)

[3.5. Алгоритмы определения активной информации программ 19](#_Toc354479106)

[3.5.1. Алгоритмы замещения 20](#_Toc354479107)

[3.5.2. АМА со свойствами адаптации к особенностям поведения программы при ее выполнении 23](#_Toc354479108)

[3.6. Особенности управления памятью в случае сегментного способа ее распределения 32](#_Toc354479109)

[3.7. Контроль многопрограммности, дисциплины обслуживания программ и распределение памяти 39](#_Toc354479110)

[3.7.1. Анализ влияния на производительность многопрограммной работы 40](#_Toc354479111)

[систем с ограниченными ресурсами алгоритмов определения МА 40](#_Toc354479112)

[и числа программ в активном состоянии 40](#_Toc354479113)

[3.7.2. Дисциплины обслуживания программ и распределение памяти 44](#_Toc354479114)

[Заключение 46](#_Toc354479115)

[Упражнения 47](#_Toc354479116)

[ЛИТЕРАТУРА 48](#_Toc354479117)

# 1. ВВЕДЕНИЕ

Опыт проектирования и эксплуатации цифровых вычислительных машин (ЦВМ) уже первых поколений показал, что оперативная память (ОП) является одним из основных ресурсов, сдерживающих увеличение производительности, огра­ничивающих сферу применения ЦВМ.

Переход к производству современных многопрограммных вычислительных систем (ВС), предназначенных для одновре­менного обслуживания многих пользователей, хранения и об­работки больших объемов информации, сделал еще более ак­туальной проблему увеличения емкости и быстродействия ОП, снижение ее стоимости.

Причины технического и стоимостного характера пока не позволяют использовать при создании таких ВС только уст­ройства оперативной памяти. Наряду с ОП неотъемлемой частью памяти ВС стали более дешевые и большей емкости устройства памяти на магнитных дисках, барабанах, лентах и др., на которых, как правило, размещается большая часть информации системы.

Вместе с тем, применение этих устройств памяти, имеющих на несколько порядков меньшее быстродействие, чем быстро­действие ОП, выдвинуло чрезвычайно сложную задачу макси­мального приближения быстродействия памяти ВС к быстро­действию ОП за счет соответствующей организации всей си­стемы памяти.

Одним ставшим теперь традиционным решением этой задачи явилась многоуровневая система памяти, которая под­разумевает упорядочение по уровням различных устройств па­мяти в соответствии с их быстродействием и использование каналов и других средств для реализации обмена информацией между уровнями памяти.

Однако эффективность использования памяти с такой схе­мой во многом зависит от алгоритмов управления ею. Эти ал­горитмы должны не только обеспечивать автоматический конт­роль и перемещение информации между уровнями памяти, но также обладать адаптивными свойствами к изменению активности использования различных данных в процессе функцио­нирования системы. Качество «предсказания» активно­сти использования тех или иных данных, осуществляемое ал­горитмами управления памятью, позволяет правильно распре­делять информацию по уровням памяти, сокращает непроизводительное время обменов информацией между уровнями па­мяти.

Материал, включенный в настоящее пособие, представляет собой часть курса лекций по вычислительным системам, читае­мого автором в течение нескольких лет студентам факультета Автоматики и вычислительной техники МЭИ (специальность 0647 и 0608).

Проблема организации и использования памяти в совре­менных цифровых вычислительных машинах и системах чрез­вычайно актуальна и далека от своего исчерпывающего реше­ния. Пока в этой области хозяином положения остается прак­тик, полагающийся в своих решениях на интуицию, опыт и разрозненный экспериментальный материал, собранный в про­цессе анализа работы машин и систем либо путем моделирования. Теоретик вынужден довольствоваться тем, что «латает» многочисленные дыры нерешенных частных проблем органи­зации и распределения памяти.

Основная цель настоящего пособия — рассмотреть с еди­ных позиций вопросы структурной организации и анализа эф­фективности различных стратегий распределения памяти. При этом главный акцент сделан не на изложение технических ре­шений, а на проблемах анализа целесообразности и эффектив­ности различных алгоритмов распределения памяти.

Автору хотелось, чтобы эта особенность действительно от­личала настоящее пособие от большинства учебников и книг по вычислительным системам, в которых основное внимание сосредоточено на задачах технического обеспечения управле­ния памятью.

# 2. СТРУКТУРНАЯ ОРГАНИЗАЦИЯ И СПОСОБЫ РАСПРЕДЕЛЕНИЯ ПАМЯТИ В ЦВМ И ВС

## 2.1. Технические характеристики различных устройств памяти и ее структурная организация

Как было указано во введении, память современных ВС яв­ляется многоуровневой и состоит из различных по емкости и быстродействию носителей информации. Самый верхний уро­вень памяти включает наиболее быстрые устройства, так на­зываемую сверхоперативную память, которая непосредственно участвует в обработке информации процессорами системы. Емкость этой памяти — от нескольких десятков до нескольких тысяч байтов, цикл обращения — от нескольких сотен до не­скольких десятков нс. Последние достижения в области техно­логии изготовления больших интегральных схем сделали воз­можным на промышленной основе производство накопителей сверхоперативной памяти емкостью до 32-64 К байтов (К = 1024) с циклом обращения порядка нескольких десят­ков нс.

Следующий уровень памяти носит название оперативной памяти и обычно организуется из накопителей на ферритовых сердечниках. Емкость этой памяти достигает в различных си­стемах нескольких миллионов байтов, цикл обращения — от нескольких сотен нс до нескольких мкс. Многие модели ЦВМ третьего поколения не имеют сверхоперативной памяти, по­этому в них процессор при выполнении программы работает непосредственно с ОП.

Устройства памяти прямого доступа (диски, барабаны) со­ставляют следующий за ОП по быстродействию уровень памя­ти. На этих устройствах сосредотачивается наибольшая часть обрабатываемой информации в системе. Емкость накопителей на дисках и барабанах достигает нескольких миллионов, а для отдельных типов дисков — нескольких десятков миллионов байтов.

Время, необходимое для обращения к такого рода устрой­ствам, складывается из времени перемещения механизма доступа (только для дисков), времени выбора головки считывания — записи, времени ожидания, связанного с вращением но­сителя информации, и, наконец, времени считывания либо за­писи данных.

Так, среднее время перемещения механизма доступа для накопителей на дисках машин единой серии ЭВМ [1] состав­ляет приблизительно 75-100 мс. Время выбора головки запи­си-считывания определяется временем переключения элект­ронных схем и пренебрежимо мало по сравнению с общим вре­менем обращения. Среднее время ожидания равно половине времени полного оборота носителя и составляет от 10 до 70 мс для различных устройств; скорость передачи данных — 100-150 К байтов/с для дисков и 1250 К байтов/с для барабанов [1].

Таким образом, среднее время обращения *Т*дк накопите­лю на дисках определяется формулой *Т*д= *tоб/2* + *t*д *+ t*п *,* где

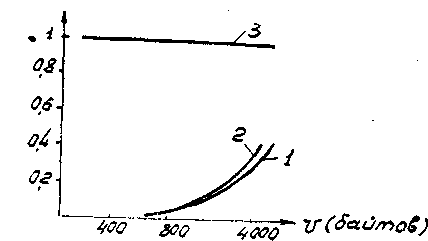


Рис. 1. Эффективность передачи данных

*tоб* — время полного оборота носителя; *t*д — время перемеще­ния механизма доступа; *t*п— время записи или считывания данных.

Аналогичной формулой определяется среднее время *Тб* об­ращения к барабану, если положить *t*д *=* 0.

Пусть *v* — количество записываемых или считываемых данных, *с* — скорость передачи данных (*t*п = *v/с).*

На рис. 1 приведены кривые зависимости от *v* отношений *t*п/*T*д для дисков (кривая 1) и *t*п/*T*бдля барабанов (кривая *2)* для *tоб* = 20 мс, *t*д = 100 мс; скорость передачи *с* для дисков взята равной 150 К байтов/с, для барабанов —1200 К байтов/с.

Эти кривые характеризуют эффективность передачи дан­ных в зависимости от объема считываемых или записываемых данных. Кривая 3 иллюстрирует аналогичную зависимость для ОП.

Приведенные данные показывают, что существует большое различие в быстродействии ОП и устройств памяти прямого доступа. Например, при цикле обращения *tо*п к ОП, равном 1 мкс, *T*б/*t*оп *≈*104, *T*д/*t*оп *≈*105.

Именно, это различие в быстродействии ОП и устройств памяти прямого доступа так остро ставит проблему разработ­ки эффективных алгоритмов управления памятью, максималь­но приближающих среднее время обращения к памяти к цик­лу обращения самых быстродействующих устройств, входя­щих в ее состав.

В общем случае, помимо перечисленных устройств памяти, отнесенных к различным уровням, в соответствии с их быстро­действием, в ВС широко используются также накопители на магнитных лентах [1] и аналогичные им устройства, образую­щие еще один уровень самой медленной памяти, предназначенной для хра­нения наименее активно используемой информа­ции системы.

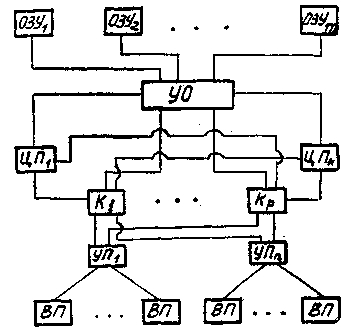


Рис. 2. Структура ВС и ее памяти

На рис. 2 приведен пример архитектуры со­временной ВС, иллюстри­рующий место и особен­ности структурной орга­низации памяти. На этом рисунке использованы следующие сокращения: ОЗУ — оперативное запоминающее устройство; УО — устройство обслуживания запросов к ОП; К — канал; ЦП — центральный процессор; ВП — внешняя память (диски, барабаны и пр.).

Следующие общие принципы структурной организации па­мяти, в большой степени определяющие ее пропускную спо­собность в целом, следуют из вышеизложенного и анализа схе­мы на рис. 2:

* иерархический многоуровневый характер построения памяти, основанный на разделении по уровням устройств па­мяти с резко различающимся быстродействием;
* модульный принцип построений памяти каждого уровня;
* увеличение числа каналов, связей между различными устройствами, применение структурных решений, обеспечиваю­щих наибольшие возможности одновременной работы различ­ных устройств памяти, повышающих ее надежность.

## 2.2. Распределение памяти

Начиная с этого параграфа, безнарушения общности рас­сматриваемых задач, предполагается, что память ЦВМ или ВС состоит из двух уровней: ОП, с которой непосредственно работают процессоры системы, и ВП — внешней памяти, объ­единяющей устройства памяти прямого доступа.

Задача распределения памяти возникает только при огра­ниченной (конечной) емкости ОП.

Ее суть заключается в определении эффективных страте­гий управления памятью, в частности, стратегий перераспреде­ления и размещения информации по уровням памяти в процес­се функционирования машины или системы, которые улучша­ют системные показатели (производительность, среднее время обслуживания программ и пр.), освобождают программиста от программирования обменов данными между уровнями памяти, если размер его программы превышает емкость ОП.

Различают *статический* и *динамический* способы распре­деления памяти.

Первый из них предполагает в общем случае участие про­граммиста в планировании распределения информации его программ (команд и данных) между уровнями памяти.

При статическом способе распределения памяти програм­ма обычно составляется с использованием адресного простран­ства, эквивалентного множеству адресов ОП. Вследствие это­го, если размер программы превышает емкость ОП, ее необхо­димо «разрезать» на части и использовать запланированный обмен между уровнями памяти в процессе выполнения.

В системах со статическим способом распределения памя­ти программе перед выполнением выделяется место в ОП, до­статочное для ее размещения, и программа-загрузчик загру­жает ее на это место, выполняя преобразование адресов, не­обходимое для правильного выполнения программы.

Например, составленная в условных адресах программа

0. СЛ 11

1. УМ 15

2. УП 40

…

К ВЫ 29

после загрузки в ОП, начиная с адреса 300, преобразуется в программу:

300 СЛ 311

301 УМ 315

302 УП 340

…

300 + К ВЫ 329

Здесь СЛ — сложение, УМ — умножение, УП — условный переход, ВЫ — вычитание.

Новая загрузка программы в ОП, начиная с другого адре­са, потребует нового преобразования ее адресов.

Таким образом, при статическом распределении памяти не­обходимо участие программиста в распределении памяти, пре­образование адресов при загрузке программы на новое место в ОП. Статический способ распределения памяти также огра­ничивает возможности эффективного использования памяти, так как требуется обычно всю программу загружать в ОП, хо­тя на отдельных этапах ее выполнения используется только часть ее информации, которую и было бы целесообразно хра­нить в ОП.

Эти недостатки статического способа распределения памя­ти особенно очевидны при переходе к многопрограммной рабо­те, коллективному использованию ЦВМ и ВС одновременно многими пользователями, когда происходит постоянная смена множества обслуживаемых программ, когда сам режим функционирования требует постоянного перемещения инфор­мации между уровнями памяти.

Есть и другие, внешние по отношению к системе причины, которые заставляют отказаться от использования статического способа распределения в пользу динамического: сложность ор­ганизации использования общей информации для многих программ, модульный принцип построения программ, возмож­ность динамически изменяемых структур данных и др.

Простейшие схемы динамического распределения памяти отличаются от статического только добавлением специальных регистров, которые осуществляют динамически «перетрансля­цию» адресов программы по месту ее произвольного размеще­ния в ОП [2].

Более сложные схемы динамического распределения памя­ти, устраняющие вышеперечисленные недостатки статического способа, основаны на концепции *математической* или, как ее сначала назвали, *виртуальной памяти*.

*Математическая память* представляет собой аналогично ОП упорядоченное множество математических адресов, которые используются при составлении программы на ма­шинном языке. Это множество, как правило, во много раз больше по мощности (числу адресов), чем множество адре­сов ОП.

Однако реализация виртуальной памяти на практике тре­бует специальных схем, необходимых для эффективного пре­образования математических адресов в физические в процессе выполнения программ.

Наиболее известны и широко применяются на практике *страничный, сегментный* и *сегментно-страничный* способы ди­намического распределения памяти или варианты реализации виртуальной памяти.

Если первый из них устраняет недостатки внутрисистемно­го характера, присущие статическому способу распределения памяти, то последние два варианта продиктованы новым под­ходом к разработке сложных программ из относительно неза­висимых модулей, или сегментов, необходимостью эффектив­ной работы с динамически изменяющимися структурами дан­ных и необходимостью построения простых и надежных схем защиты и использования общей для многих программ инфор­мации.

В системах со страничной и сегментно-страничной органи­зацией памяти единицей перемещаемой между ОП и ВП ин­формации является математическая страница, представляю­щая некоторый блок команд и (или) данных программы, взятых в ее естественной записи.

При этом, если сегмент является логической единицей ин­формации, которой оперирует программист, составляя свою программу, то понятие страницы является, скорее, внутрисистемным и, следовательно, не требуется при написании про­граммы.

В системах с сегментным способом распределения памяти перемещаемой между ОП и ВП, а следовательно, размещае­мой в ОП единицей информации при выполнении программы является сегмент.

В отличие от страничного и сегментно-страничного спосо­бов распределения памяти, где не имеет значения, какая из свободных в ОП страниц выбирается для записи загружаемой математической страницы, в системах с сегментной организа­цией памяти возникает проблема выбора свободного участка при загрузке нового сегмента в ОП.

Что касается адресации информации в системах с рассмот­ренными способами распределения памяти, то она основана на известных схемах [2] преобразования математических ад­ресов в физические адреса (адреса ОП), осуществляемого через таблицы страниц и (или) сегментов.

На рис. 3 показана схема преобразования математических адресов в физические в машине Атлас [2], в которой впервые был применен страничный способ распределения памяти.

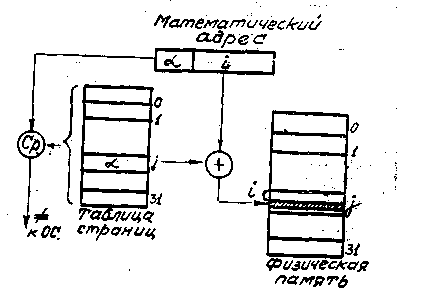


Рис. 3. Схема адресации машины Атлас

Математический адрес этой машины имеет 20 разрядов; 11старших разрядов адреса служат для представления номе­ра адресуемой математической страницы, 9 младших разрядов указывают номер слова в странице.

Таким образом, емкость математической памяти равна 220 адресов и «разделена» на 211 страниц по 512 слов каждая.

В свою очередь, ОП, имеющая емкость 214 слов, разделена на 32 страницы (на рис. 3 они нумеруются, начиная с 0) по 512 слов каждая, т. е. физический адрес имеет 14 разрядов.

Для адресации используется таблица страниц, содержащая 32 строки, каждая из которых соответствует одной из физиче­ских страниц с тем же номером и в которую заносится номер математической страницы, размещенной в соответствующей этой строке физической странице.

Для преобразования математического адреса в физический сначала осуществляется сравнение номера математической страницы, указанного в математическом адресе, с содержи­мым всех строк таблицы страниц. Если обнаружится, что в j-йстроке (0≤ j≤ 31) таблицы страниц хранится одинаковый со сравниваемым номер математической страницы, физический адрес определяется как j∙29 + i, где i— номер слова в стра­нице.

Если при сравнении окажется, что нужной страницы нет в оперативной памяти (сигнал прерывания к операционной системе(ОС)), то ОС переписывает ее из ВП на место некоторой страницы ОП.

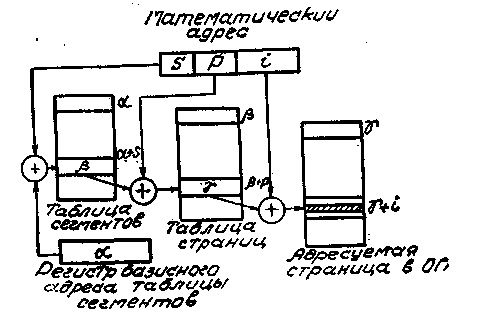


Рис. 4. Схема сегментно-страничной адресации

При этом может потребоваться предваритель­ная перепись из ОП на ВП некоторой страницы для освобож­дения места загружаемой странице. После записи в ОП новой страницы αна место j-й физической страницы в строку табли­цы страниц с номером j необходимо занести значение α.

На рис. 4 приведена схема преобразования математиче­ского адреса в физический для случая сегментно-страничной организации памяти.

На этом рисунке α,β и γ — базисные (начальные) адреса таблицы сегментов, таблицы страниц и самой страницы в ОП соответственно; *s, p, i* — номер адресуемого сегмента програм­мы, номер математической страницы в сегменте, номер слова в странице соответственно.

От этой схемы просто перейти к схеме, реализующей пре­образование адресов в случае сегментного способа распреде­ления памяти, если исключить ступень обращения к таблице страниц и считать, что часть *(р, i)* математического адреса за­дает номер слова в сегменте.

Исключив на рис. 4 ступень обращения к таблице сегмен­тов (исключив часть *s* из математического адреса), можно по­лучить другой вариант преобразования математического адре­са в физический для страничного способа распределения па­мяти, который отличается от схемы рис. 3.

В каждой строке таблицы сегментов (страниц), помимо указанной информации, может содержаться информация о программах, которым разрешено обращение к данному сегменту (странице), информация о разрешенном способе исполь­зования сегмента (страницы) (только для чтения, только для записи, для записи и чтения), указатель, находится ли сегмент (страница) в ОП или ВП; размер сегмента (страницы), ин­формация, касающаяся активности использования в процессе выполнения программы сегмента (страницы) и пр.

Перечисленные данные используются системой для защи­ты информации, организации использования общей для мно­гих программ информации и т. д. [2].

# 3. АНАЛИЗ ЭФФЕКТИВНОСТИ СТРАТЕГИЙ ДИНАМИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ПАМЯТИ

## 3.1. Общая схема обслуживания программ в многопрограммном режиме

В этом разделе рассматривается проблема эффективной организации и использования страничных, сегментных и сегментно-страничных систем памяти.

Основанные на концепции математической памяти эти спо­собы распределения памяти, как указано в разд. 2, освобожда­ют программиста от необходимости программирования обме­нов информацией между различными уровнями памяти при со­ставлении программ, размер которых превышает емкость ОП.

В них преимущество модульного принципа организации программ из относительно самостоятельных частей-модулей или сегментов хорошо сочетается с возможностью построения эффективных схем использования общей информации для раз­личных программ и схем ее защиты. Наконец, при использо­вании названных способов распределения памяти открывает­ся путь к повышению производительности многопрограммной работы за счет хранения в ОП на отдельных этапах выпол­нения программы только ее самой активной информации.

В то же время страничный, сегментный и сегментно-страничный способы организации и распределения памяти отлича­ются усложненной логикой, предъявляют более жесткие требования к быстродействию каналов, обслуживающих обмен между уровнями памяти, организации памяти каждого уровня, выбору стратегии размещения и поиска информации на дисках, барабанах.

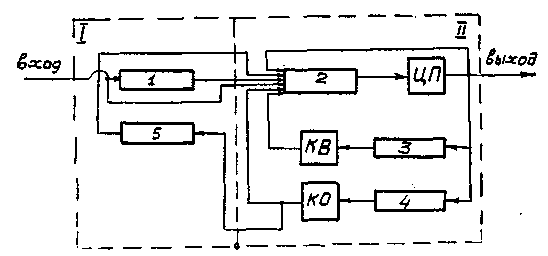


Рис. 5. Общая схема многопрограммной работы

В режиме многопрограммной работы при ис­пользовании названных способов распределения памяти чрез­вычайно актуальной является задача управления памятью, от решения которой зависит производительность и время обслу­живания программ в этом режиме.

Рассмотрим общую схему однопроцессорного обслужива­ния программ в многопрограммном режиме работы на рис. 5 применительно к страничному, сегментному или сегментно-страничному способам распределения памяти [3].

На этой схеме поступающие на вход машины или системы программы в зависимости от ограничений на время их обслу­живания либо размещаются на носителях памяти второго уровня (ВП), образуя очередь программ 1, либо загружаются сразу в ОП, образуя очередь программ 2 на обслуживание ЦП.

Предполагается, что в общем случае только часть самой активной информации программы — множество активных страниц для случая страничного и сегментно-страничного спо­собов распределения памяти или сегментов для случая сегментного способа распределения памяти загружается и нахо­дится в ОП при выполнении программы на ЦП.

*Множество активных страниц или сегментов программы (МА)* определяется на различных этапах ее выполнения с по­мощью специальных управляющих алгоритмов, называемых *алгоритмами определения МА (АМА)*.

Говорят, что программа находится в *активном состоянии*, если она имеет свои страницы или сегменты в оперативной па­мяти; в противном случае считается, что программа находится в *пассивном состоянии* (состояния I и II на рис. 5).

Согласно принятой дисциплине обслуживания программ в многопрограммном режиме работы в определенные моменты времени программы в пассивном состоянии из очереди 1 загру­жаются для выполнения в ОП (переводятся в очередь 2 про­грамм в активном состоянии). В действительности при загруз­ке программы в ОП пересылаются только страницы или сег­менты, принадлежащие ее МА.

При выполнении на ЦП программа может быть «принуди­тельно» прервана и возвращена в очередь 2, например, в слу­чае циклического обслуживания программ на ЦП.

Программа может завершить свое выполнение на ЦП и оставить машину или систему (стрелка «выход» на рис. 5); программа переходит в очередь 3 к каналу ввода — вывода (КВ), если очередной ее командой является команда ввода — вывода. При обращении к странице или сегменту[[1]](#footnote-2), отсутствую­щим в оперативной памяти, программа ставится в очередь 4 к каналу обмена (КО). После завершения обслуживания в КО или КВ программа возвращается в очередь 2 на обслуживание к ЦП. Из очереди 4 программа также может быть переведена в пассивное состояние (очередь 5), откуда, повторив этап на­чальной загрузки в оперативную память, снова перемещена в очередь 2. Таким образом, в очередях 1 и 5 находятся про­граммы, не имеющие своих страниц или сегментов в оператив­ной памяти (пассивное состояние 1).

Пусть N (t) — множество программ, выполняемых в мо­мент времени t в многопрограммном режиме работы. Пусть N1(t) и N2(t) — множество программ, находящихся в актив­ном и пассивном состояниях соответственно, N(t)= N1(t)UN2(t).

Пусть Si(t) — множество активных страниц или сегмен­тов i-и программы в момент времени i.

Предположим, что при переводе программы в активное со­стояние страницы или сегменты, принадлежащие ее МА, за­гружаются в ОП. Очевидно, что б этом случае для произвольного момента времени t должно выполняться условие:

,

где *v* — емкость ОП; Si(t) — размер Si(t), определя­емый, например, числом страниц в Si(t) (в этом случае *v* измеряется также в страницах) или количеством слов (или байтов), необходимых для размещения в ОП страниц или сегментов Si(t).

Механизм определения множества активных программ N1(t) назовем *контролем многопрограммности*, а алгоритм, который формирует N1(t) из множества программ N(t), на­зовем *алгоритмом контроля многопрограммности*.

Число программ в N1(t) будем называть *уровнем много­программности* (УМ) в момент времени t.

В режиме пакетной обработки УМ определяется размером пакета и может считаться независимым от времени; можно привести другие примеры систем, в которых УМ является по­стоянной величиной, определенной разбиением ОП на заранее фиксированное число фрагментов (или разделов), в каждом из которых допускается выполнение только одной программы.

В режимах разделения времени и реального времени, для которых поток поступающих на обслуживание программ является случайным, УМ, как правило, зависит от времени.

Как показывают теоретические исследования, а также практика работающих в многопрограммном режиме систем, алгоритмами определения МА и контроля многопрограммно­сти в большей степени определяется эффективность не только распределения памяти, но и эффективность многопрограмм­ной работы вообще.

## 3.2. Критерии эффективности организации и распределения памяти

Следующие показатели могут использоваться в качестве критериев эффективности организации и распределения па­мяти:

* производительность многопрограммной работы, измеряемая средним числом программ, выполняемых в единицу вре­мени;
* время обслуживания программ;
* аппаратные и программные «затраты», связанные с реа­лизацией того или иного способа распределения памяти и ал­горитмов управления памятью.

Пусть на достаточно большом интервале времени функцио­нирование машины или системы определены величины:

µ1— часть времени, в течение которого ЦП был занят непосредственно выполнением рабочих программ;

µ2 — часть времени, которое было затрачено ЦП на выпол­нение системных функций: обслуживание команд ввода — вы­вода, прерываний, возникающих в связи с динамическим рас­пределением памяти и пр.;

µ3— часть времени, в течение которого ЦП простаивал.

Справедливо соотношение µ1 + µ2+ µ3= 1. Очевидно, ве­личина µ1 может характеризовать производительность много­программной работы и по ее приращению можно судить об эффективности различных способов организации и распреде­ления памяти.

Увеличение µ1 равносильно уменьшению суммарного значе­ния величин µ2 и µ3. Представим µ2 как сумму µ2 = µ2’ + µ2’’ , где µ2’ определяет часть системного времени, используемого на обслуживание обменов страницами или сегментами между ОП и ВП.

Увеличение µ1 достигается за счет более совершенных ор­ганизации и алгоритмов распределения памяти и, как прави­ло, приводит к увеличению µ2’. Усложнение АМА и алгорит­мов контроля УМ увеличивает системное время, необходимое для их выполнения; часто требуются аппаратные решения, чтобы увеличением µ2’ не было перечеркнуто ожидаемое от применения новых алгоритмов распределения памяти прира­щение величины µ1.

Величина µ3, определяющая время простоя ЦП, в свою очередь, может служить для оценки производительности мно­гопрограммной работы.

Если исключить из рассмотрения необходимость обменов страницами или сегментами между уровнями памяти при вы­полнении программ (см. рис. 5), то несложно показать, что время простоя ЦП (величина µ3) уменьшается с увеличением УМ.

Действительно, пусть множество активных программ N1(t) не изменяется во времени и содержит *n* программ; пусть *P*вв — вероятность того, что любая из *n* программ при выполнении ее на ЦП переводится в очередь 3 (рис. 5), т. е. иниции­рует команду ввода — вывода, считается одинаковой для всех программ.

При этих предположениях вероятность простоя ЦП *Р*пр может быть определена как вероятность того, что все *n* про­грамм находятся в очереди 3 или на обслуживании в КВ, и равна .

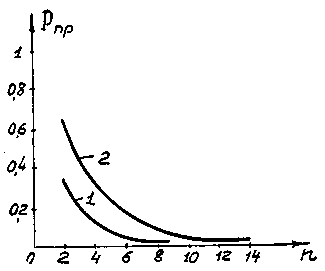
На рис. 6 приведены кривые 1 и 2 простоя ЦП (*Р*пр) в за­висимости от *n* для двух значений *P*вв, равных 0,6 и 0,8 соот­ветственно, причем, как по­казывают данные реальных систем, последнее значение ближе к среднестатистиче­ским данным реальных про­грамм.

Рис. 6. Зависимость *Р*пр от *n*

Для ОП заданной емко­сти страничный, сегментный и сегментно-страничный спо­собы распределения памяти позволяют увеличить число одновременно выполняемых программ, т. е. увеличить УМ за счет уменьшения емкости ОП, отводимой каждой программе при ее выполнении. Однако это, в свою очередь, приводит к увеличению вероятности того, что при вы­полнении программы на ЦП она будет переведена в очередь 4 (рис. 5) из-за необходимости обмена информацией с ВП. Как следствие, увеличивается время простоя ЦП, т. е. µ3. Более того, как показывает опыт эксплуатации систем, увеличение УМ без изменения емкости ОП приводит к такому отрица­тельному явлению как «болтанка» [4], суть которого состоит в том, что вместо выполнения полезной работы в системе идет интенсивный обмен информацией между уровнями памяти.

Приведенные соображения еще раз подчеркивают особое значение алгоритмов определения МА и контроля УМ в реше­нии задачи повышения производительности многопрограммной работы в системах со страничной, сегментной и сегмёнтно-страничной организацией памяти.

Рассмотрим, из каких составляющих складывается время обслуживания *t*обс программы при многопрограммной работе:

*t*обс = *t*цп + *t*обм + *t*вв + *t*с + *t*ож

Здесь *t*цп — суммарное время выполнения программы на ЦП;

*t*обм — суммарное время обмена между уровнями па­мяти при выполнении программы;

*t*вв — суммарное время выполнения команд ввода—вывода;

*t*с — системное время, в котором учитывается время на постановку в очередь или на обслуживание программы;

*t*ож — суммарное время ожидания программы в очере­дях.

Время *t*обм может составлять существенную часть времени обслуживания программы *t*обс в системах со страничной, сег­ментной и сегментно-страничной организацией памяти и так же, как и производительность в большой степени определяется алгоритмами определения МА и контроля УМ. Это время про­порционально суммарному числу обменов информацией между уровнями памяти в процессе выполнения программы, а каж­дый обмен занимает от 10 до 100 мс в зависимости от того, ба­рабаны или диски используются в качестве устройств внешней памяти.

При исследовании памяти, как изолированной системы, в качестве критерия эффективности ее организации и распреде­ления обычно используется вероятность обращения к ВП, из­меряемая отношением числа обращений к ВП к общему чис­лу обращений к памяти на большом интервале вычислений.

Более информативным критерием является среднее время обращения к памяти, которое учитывает не только быстродей­ствие различных устройств памяти, но и параллелизм их рабо­ты при обменах.

Следующие факторы оказывают существенное влияние на перечисленные критерии:

* структурная организация и технические характеристики используемых устройств памяти;
* размещение информации на устройствах внешней памя­ти и организация обслуживания к ним запросов;
* характер поведения выполняемых программ;
* алгоритмы определения активной информации программ;
* стратегия размещения информации в ОП в случае сег­ментного способа ее распределения;
* алгоритмы контроля УМ и дисциплины обслуживания программ.

В разделе 2 уже обсуждался аспект структурной органи­зации и технических характеристик устройств памяти и их влияния на эффективность распределения памяти. Ниже ана­лизируется влияние остальных факторов на эффективность распределения двухуровневой памяти применительно к стра­ничному, сегментному и сегментно-страничному способам ее распределения.

## 3.3. Размещение информации на устройствах памяти второго уровня и организация обслуживания запросов

Кривые 1, 2 и 3 на рис. 1 показывают зависимость отноше­ния времени записи или считывания к времени обращения (до­ступа) для дисков, барабанов и памяти на сердечниках от количества байтов перемещаемой информации.

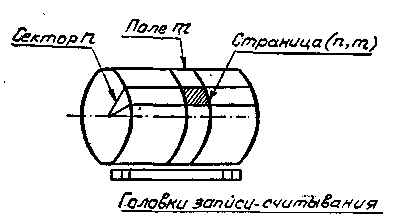
Только при объеме передаваемой информации 2000—4000 байтов ( в таких пределах на практике изменяется размер страницы) удается получить это отношение для устройств памяти прямого доступа (дисков или барабанов) в пределах 0,2 - 0,4.

Это обстоятельство, а также большое (особенно для ди­сков) время доступа ограничивает возможности применения устройств памяти на дисках и барабанах в системах со стра­ничной, сегментной и сегментно-страничной организацией па­мяти.

Так, диски с перемещающимися головками редко исполь­зуются в системах с такой организацией памяти, а эффектив­ность использования барабанов достигается только при соот­ветствующих стратегиях размещения информации и органи­зации обслуживания запросов в очереди (см. очередь запро­сов 4 к КО на рис. 5).

Далее на примере страничного способа распределения па­мяти [4] показано, каким образом достигается на практике по­вышение эффективности использования барабанов в качестве устройств ВП.

Рис. 7 иллюстрирует применяемые для этой цели разме­щение и поиск информации. На этом рисунке поверхность барабана разделена на поля, на каждом из которых может быть записано последовательно определенное число страниц. Каждая страница информации, записанной на поверхности барабана, идентифицируется по номеру поля *m*, на котором, она размещена, и по номеру сектора *n* этого поля. Таким образом, адресом страницы, по которому осуществляется ее поиск и размещение в накопителе, является упорядоченная пара целых чисел (*n*, *m*), где *m* — номер поля, а *n* — номер сектора поверхности барабана. Головки записи и считывания расположены вдоль поверхности барабана (рис. 7) и автоматически коммутируются в соответствии с требуемым полем.

Рис. 7. Размещение и поиск информации на барабане

Рассмотрим следующие варианты организации обслужи­вания очереди запросов к барабану:

* одноочередная организация обслуживания запросов (очередь 4 на рис. 5);
* многоочередная организация обслуживания запросов.

Согласно первой организации все запросы к барабану на запись или считывание страницы образуют одну очередь и обслуживаются в порядке поступления.

Во втором случае организуется очередь запросов к каж­дому сектору барабана. Запросы, поступающие на обмен, по номеру сектора, указанному в их адресе, ставятся в очередь запросов к этому сектору. Эти очереди обслуживаются в по­рядке поступления в них запросов, причем, как только голов­ки считывания — записи окажутся перед началом сектора *n*, из очереди запросов к этому сектору берется на обслужива­ние очередной запрос и по номеру поля *m*, указанному в его адресе, осуществляется коммутация головок записи — считы­вания для требуемого поля.

В соответствии с характером запроса в дальнейшем про­исходит запись или считывание требуемой страницы инфор­мации.

Сравним время обслуживания запросов для случая одноочередной и описанной многоочередной системы обслужива­ния запросов.

Пусть для простоты очередь запросов к накопителю по­стоянна, равна *l* и, как только очередной запрос начинает свое обслуживание, в очередь добавляется новое требование. Кроме того, предположим, что поступающие запросы с оди­наковой вероятностью могут поступить в очередь к любому из *k, k* ≥ 1 секторов накопителя, т. е. средняя длина очереди запросов к любому сектору равна *l’ = l/k*.

При сделанных предположениях среднее время обслужи­вания запросов для многоочередной организации обслужи­вания *t(*м*)* определяется как

*t(*м*)* = *t*об/2 + *l’∙ t*об + *t*об / *k ,*

где первое слагаемое определяет время ожидания до начала обслуживания очереди, куда поступил новый запрос; второе слагаемое задает время обслуживания *l’* запросов, предшест­вующих поступившему в очередь запросу; последнее слагае­мое есть собственно время передачи данных требуемой стра­ницы.

В случае одноочередной системы обслуживания запросов к накопителю на барабане среднее время обслуживания за­проса *t(*o*)*, поступившего в конец очереди запросов длины *l*, очевидно, равно (*t*об/2 + *t*об / *k)∙l*.

Сравнивая *t(*м*)* и *t(*o*)*, легко видеть, что для *l*>0, всегда *t(*o*)* ≥ *t(*м*)*, и это различие тем больше, чем больше значение *l*.

У многоочередной системы более высокой оказывается и интенсивность обслуживания запросов, измеряемая числом запросов, обслуживаемых в единицу времени.

Действительно, средний интервал времени между после­довательными моментами окончания обслуживания запросов для одноочередной системы равен (*t*об/2 + *t*об / *k)*, так как обслу­живание очередного запроса в очереди не может начаться раньше, чем будет обслужен предыдущий запрос.

Для многоочередной системы обслуживания запросов есть возможность за один оборот барабана обслужить не­сколько запросов (в лучшем случае *k*), если они относятся к различным секторам.

Как было указано в предыдущем параграфе, при перево­де программы в активное состояние страницы, принадлежа­щие ее МА, необходимо загрузить в ОП. Если эти страницы размещены на поверхности барабана таким образом, что имеют один и тот же номер сектора в их адресе (но различ­ные поля), при записи или чтении страниц МА потребуется столько оборотов барабана, сколько страниц в МА програм­мы. Если страницы программы размещаются на различных секторах, например, в порядке нумерации страницы разме­щаются на следующих друг за другом секторах одного по­ля, то возможно за один оборот барабана считать или запи­сать все страницы МА программы.

Таким образом, стратегия размещения информации на устройствах памяти прямого доступа и организация обслужи­вания к ним запросов оказывают существенное влияние на время обслуживания запроса и интенсивность обслуживания запросов в очереди.

## 3.4. Характер поведения программы

В этом параграфе рассматривается поведение программ в системах со страничной, сегментной и сегментно-страничной организацией памяти.

Без нарушения общности, предполагается, что программа организована таким образом, что множество ее страниц или сегментов N состоит из подмножества страниц или сегментов команд Nк и подмножества страниц или сегментов данных Nд, N= Nк U Nд.

Для идентификации страниц или сегментов, принадлежа­щих множествам N, Nк и Nд, считается, что для них введена нумерация в соответствующем множестве, и при нумерации элементов Nк и Nд они снабжаются индексами «к» и «д» со­ответственно. Например, 2, 7, 2к и 7д обозначают 2-й и 7-й элементы N, 2-й элемент Nк и 7-й элемент Nд, причем необя­зательно, что 2 и 2к, 7 и 7д обозначают одну и ту же страни­цу (сегмент).

Для задания возможных переходов между страницами или сегментами команд при выполнении программы и исполь­зуемых ими данных введем две булевские матрицы: квадрат­ную матрицу Мк размера │Nк│×│Nк│ и прямоугольную матрицу Мд размера │Nк│×│Nд│, где │Q│ обозначает чис­ло элементов в множестве Q.

Элемент Мк(*i*к, *j*к) матрицы Мк, стоящий на пересечении *i*к строки и *j*к столбца равен 1 или 0 в зависимости от того, возможен или нет непосредственный переход от страницы (сегмента) команд *i*к к странице (сегменту) команд *j*к. Оче­видно, содержательно указанный переход имеет место, если существует хотя бы одна команда передачи управления из страницы (сегмента) команд с номером *i*к в страницу (сег­мент) команд с номером *j*к либо в естественной записи про­граммы страница (сегмент) команд *j*к следует за страницей (сегментом) команд *i*к.

Аналогично, элемент матрицы Мд(*i*к, *j*д) равен 1 или 0 в зависимости от того, использует или нет страница (сегмент) команд *i*к данные, принадлежащие странице (сегменту) дан­ных *j*д.

Для наглядности в примерах мы будем также пользовать­ся эквивалентным графовым представлением программы, указывая связи между ее страницами (сегментами) команд с по­мощью направленных дуг, а связи между страницами (сег­ментами) команд и используемыми ими данными с помощью прерывистых ребер.

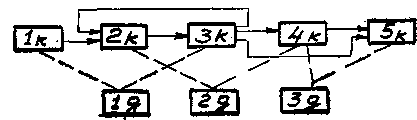


Рис. 8. Граф-схема программы

Это представление будем называть *граф-схемой программы*. На рис. 8 приведен пример граф-схе­мы программы.

С точки зрения описываемого ниже представления процес­са выполнения программы в страничных, сегментных и сегментно-страничных системах распределения памяти будем на­зывать страницу (сегмент) программы, с которой начинается ее выполнение, начальной, а страницу (сегмент), которой за­канчивается выполнение программы, — конечной.

По матричному или графовому представлению програм­мы можно определить множество путей (последовательностей страниц или сегментов) *L,* соединяющих начальную и конеч­ную страницы (сегменты) программы.

Для программы на рис. 8 это множество путей, соединяю­щих страницы (сегменты) 1к и 5к, состоит, из путей вида:

1к, (2к, 3к)n, 4к, 5к и 1к, (2к, 3к)n, 5к , n ≥ 1, где (2к, 3к)n = 2к, 3к, …2к, 3к.

*n*o раз

С целью исследования характера поведения программы и описания процесса ее выполнения в системах со страничной, сегментно-страничной и сегментной организацией памяти введем понятие *страничной (сегментной) реализации программы* как последовательности ссылок (обращений) к страницам (сегментам) программы, которая получена при ее вы­полнении и которую будем обозначать

*R = r(t1), r(t2), …, r(ti), …*

В этой последовательности *tj, j > 0* задает моменты пере­хода от страницы (сегмента) *r(tj-1)* к странице (сегменту) *r(tj)* при выполнении программы; причем для всякого *j > 0*

*r(tj) ≠ r(tj-1).*

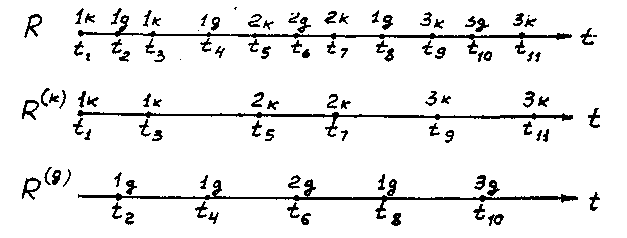
В последующем изложении часто используются также от­дельно последовательности ссылок к страницам (сегментам) команд и данных программы, полученные из *R*.

Рис. 9. Диаграмма последовательности ссылок

Эти последовательности обозначаются

*R(к) = r(ti1), r(ti2), …, r(tim), …*

*R(*д*) = r(tj1), r(tj2), …, r(tjn), …*

где *til (tjq) l > 0, q>0*, и задают на временной оси моменты перехода от одной страницы (сегмента) команд (данных) к другой при выполнении программы в соответствии с *R*.

На рис. 9 приведены примеры указанных последователь­ностей для одной из реализаций программы на рис. 8, причем над временной осью перечислены номера соответствующих страниц (сегментов). Очевидно, разность tj – tj-1 определяет интервал между ссылками *r(tj-1)* и *r(tj)* в последовательно­сти *R*, т. е. интервал использования страницы (сегмента) *r(tj-1)* после ссылки к ней в момент времени tj-1.

По последовательности *R* несложно восстанавливается множество страниц (сегментов) данных, которое использует­ся при всякой ссылке к некоторой странице (сегменту) команд. Например, при ссылке к странице (сегменту) команд 2к в момент времени t5 (рис. 9) используется всего одна страница (сегмент) данных 2д.

Очевидно, в зависимости от исходных данных в процессе выполнения программы будут получаться различные ее реа­лизации, т. е. различные последовательности *R(к)* и *R(*д*)*. Более того, может оказаться, что некоторые реализации не резуль­тативны, т. е. последовательность *R* для них бесконечна.

Для произвольной результативной реализации програм­мы, для которой последовательность страничных (сегмент­ных) ссылок *R(к)* начинается с начальной и оканчивается ко­нечной страницей (сегментом) программы, можно показать, что ( принадлежит *L*), где последовательность отличается от *R(к)* тем, что в ней опущены моменты перехода от одной страницы (сегмента) команд к другой; *L* — множество путей, соединяющих начальную и конечную страни­цы (сегменты) программы на ее граф-схеме.

Обратное утверждение о том, что каждому пути, принадлежащему *L*, соответствует последовательность некоторой результативной реализации программы, в общем случае, не является верным.

Например, если для любой реализации программы на рис. 8 циклический участок (2к, 3к) повторяется не более *m* раз, то пути с большим числом повторений в них цикличе­ского участка (2к, 3к) не имеют соответствующих им реали­заций.

Одной из важных для практических применений особенно­стей поведения программы, определяемой по ее реализации, является *локализация ссылок* к определенному подмножест­ву страниц (сегментов) на различных этапах выполнения программы. Кроме того, процессу выполнения программы свойственно «забывать» информацию, используемую на опре­деленных этапах вычислений, и больше к ней не возвращать­ся. Первая особенность связана с циклическим характером структуры программы, повторяемостью используемых при выполнении одних и тех же информационных единиц (участ­ков команд и данных). Вторая особенность характеризует на­правленный, последовательный характер использования раз­личных информационных единиц при выполнении программы.

Именно с учетом этих особенностей, присущих процессу выполнения программы, и строится большинство алгоритмов определения МА программы, рассматриваемых в следующем параграфе.

В [4] была введена абстрактная *мера локализации ссылок* к страницам (сегментам) программы, определяемая как под­множество множества страниц (сегментов) *S(t,T)* програм­мы, на которые происходили ссылки на интервале [*t-T, t*] ее выполнения.

Среди различных факторов, помимо параметра *Т*, которые определяют характер изменения *S(t,T)*, наиболее важными являются следующие факторы:

* характер задачи и применяемый метод ее решения;
* язык программирования и организация программы;
* размер страницы.

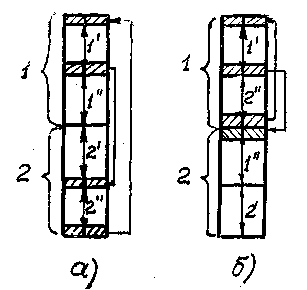
Названные факторы находят отражение в таких структур­ных и поведенческих характеристиках программ, как циклич­ность, процентное содержание в программе и ее реализации команд перехода, использование данных на различных интервалах выполнения про­граммы и пр.

Рис. 10. Варианты организа­ции программы

Большое влияние на по­ведение программы в про­цессе ее выполнения оказы­вает стиль программирова­ния, организация програм­мы, в частности, упорядоче­ние в линейном массиве, каким является машинное представление программы, различных групп команд, массивов данных.

Как показывают экспе­риментальные данные ис­следований реальных программ [4], организация программы может в несколько раз изменять число страниц (сегментов) в *S(t,T)* при выпол­нении программы.

На рис. 10, *а, б* приведены два способа упорядочения в про­грамме участков команд равной длины 1', 1", 2', 2", причем последняя команда участка 1' является командой условного перехода к первой команде участка 2", а последняя команда участка 2" передает управление первой команде участка 1'.

Пусть указанные участки оказались отнесенными к стра­ницам 1 и 2, как это показано на рис. 10*, а, б* в соответствии с двумя случаями упорядочения участков и программе.

Для последовательности 1', 2", 1', 2", 1', 2", 1', 1", 2' вы­полнения указанных участков программы первый способ ор­ганизации программы приводит к последовательности ссылок 1, 2, 1, 2, 1, 2, 1, 2 к ее страницам, а второй — к последова­тельности 1, 2.

При организации программы по второму способу на всем интервале поочередного выполнения участков 1' и 2" исполь­зуется только страница 1, в то время как в первом случае поочередно выполняются страницы 1 и 2.

Страничный, сегментно-страничный и сегментный способы распределения памяти предполагают, что единицей информа­ции, загружаемой в оперативную память, является страница или сегмент. Однако, как следует из предыдущего примера, только часть команд или данных вызванной в оперативную память страницы (сегмента) используется при ее выполне­нии. Более того, для некоторых реализаций может использо­ваться только часть страниц (сегментов) программы.

Как показывают экспериментальные данные, с увеличени­ем размера страницы или сегмента увеличивается доля неис­пользуемой в них информации.

В работе [4] введен *коэффициент* сжатия, позволяющий оценить долю неиспользуемой (но занимаемой) программой при выполнении емкости ОП в зависимости от размера страницы *v*стр.

Коэффициент сжатия *с*(*v*стр) —это отношение числа стра­ниц программы, к которым произошли ссылки при ее выпол­нении, к общему числу страниц для заданного размера стра­ницы *v*стр.

Согласно данным, приведенным в [4], *с*(*v*стр) можно аппроксимировать следующим выражением для 27 ≤ *v*стр  ≤ 213 (*v*стр в байтах):

*с*(*v*стр) = a + b ∙ log2*v*стр, a≥0, b≥0.

Из экспериментальных исследований следует, что коэф­фициент сжатия обладает следующими особенностями:

а) уменьшение размера страницы в 2 раза уменьшает значение *с*(*v*стр) на 10 - 15%, причем 0,1≤ b ≤ 0,15;

б) для малых *v*стр, 1≤ *v*стр ≤ 27 выражение a + b ∙ log2*v*стр — нижняя граница *с*(*v*стр). При этом 0,1≤ a ≤ 0,4;

в) для *v*стр ≥ 211 значение *с*(*v*стр) ≥ 0,8.

Таким образом, уменьшение размера страницы может при­вести к значительному улучшению использования памяти при соответствующей стратегии ее распределения.

Однако уменьшение размера страницы ведет к увеличе­нию емкости памяти, занимаемой таблицами страниц; резко ухудшает эффективность использования накопителей на бара­банах в качестве устройств памяти второго уровня (см. пара­граф 3.3). Учет этих противоречивых факторов привел к тому, что применяемый на практике размер страницы в байтах за­ключен в следующих границах: 512≤ *v*стр ≤ 4096.

## 3.5. Алгоритмы определения активной информации программ

Как следует из предыдущего параграфа, на различных этапах выполнения программы используется только часть ее информации — множество активных страниц (сегментов). Эффективность страничного, сегментно-страничного и сег­ментного способов динамического распределения памяти в большой степени зависит от алгоритмов определения МА. Мощностью МА (количеством страниц (сегментов) в МА) программы определяется требуемая для нее емкость ОП.

Известные стратегии, или алгоритмы определения МА, применяемые на практике или привлекшие интерес исследова­телей по теоретическим соображениям, можно разделить на два класса в соответствии с тем, определяется МА в момент ссылки к очередной странице (сегменту) в процессе выполне­ния программы или это делается заранее (алгоритмы опреде­ления МА с опережением).

Алгоритмы первого класса получили название в техниче­ской литературе алгоритмов определения МА по требованию, поскольку вызов новой страницы или сегмента в ОП происхо­дит только в момент обращения к ним.

В качестве примера алгоритма второго класса можно при­вести алгоритм, согласно которому при переходе к выполне­нию некоторого циклического участка программы в ОП загру­жаются сразу все страницы (сегменты) команд этого уча­стка.

Известен следующий теоретический результат: если время на пересылку одной страницы равно *t1*, а на пересылку *k* страниц требуется время *t*k и при этом *t*k ≥ *t1∙k*, то оптимальный алгоритм, минимизирующий суммарное количество обме­нов страницами между уровнями памяти в процессе выпол­нения программы, находится в классе АМА по требованию.

Соотношение *t*k ≥ *t1∙k* соблюдается, когда в качестве уст­ройств ВП используется ОП. Для устройств памяти прямого доступа это соотношение в общем случае не является верным.

Другим важным признаком классификации АМА являет­ся характер используемой ими информации о программе и ее поведении в процессе выполнения для определения МА.

Применяемые на практике АМА в системах универсально­го назначения обычно используют для определения МА ин­формацию о поведении программы, полученную до рассмат­риваемого момента ее выполнения (информацию о поведении программы в прошлом).

Только в системах специального назначения с заранее фиксированным множеством выполняемых программ либо для многократно выполняемых программ (программ, напри­мер, операционной системы) оказывается возможным получе­ние заранее информации о структуре и поведении программы, используемой для построения эффективных АМА.

### 3.5.1. Алгоритмы замещения

Особое место среди алгоритмов определения АМА по тре­бованию занимают так называемые алгоритмы замещения.

Алгоритмы замещения основаны на определенных прави­лах замещения страниц (сегментов) в ОП, если для очеред­ной выполняемой страницы (сегмента) программы нет сво­бодного места в ОП.

На примере страничного и сегментно-страничного способов распределения памяти принцип работы алгоритмов замеще­ния страниц в ОП может быть описан следующим образом.

Рассмотрим последовательность ссылок к страницам команд и данных программы при ее выполнении:

*R = r(t1), r(t2), …, r(tm), …*

Обозначим *S(ti)* МА программы в момент времени *ti* , i = 1, 2, ..., определенный в соответствии с последователь­ностью *R*.

Напомним, что страницы *S(ti)* должны находиться в ОП при выполнении программы.

Положим *S(0)*= Ø, где Ø — пустое множество и пусть *r(ti)* — очередная ссылка к странице программы при ее вы­полнении.

Следующие правила определяют действия алгоритмов за­мещения:

1) если *r(ti)S(ti-1)*, то *S(ti)= S(ti-1)*;

2) если *r(ti)S(ti-1)* ( означает здесь и далее, что *r(ti)* не принадлежит множеству) и в ОП есть свободная страница, то *S(ti)*= *S(ti-1)*U *r(ti)*;

3) если *r(ti)S(ti-1)* и в ОП нет свободных страниц, то *S(ti)*= *S’(ti-1)*U *r(ti)*, где *S’(ti-1)*= *S(ti-1)/x*, где *х S(ti-1)*, «/» — операция теоретико-множественного вычи­тания.

Через *х* в последнем правиле обозначена исключаемая из *S(ti-1)* страница (страница, которая замещается в ОП), на место которой должна быть помещена страница *r(ti)*.

Стратегия, в соответствии с которой осуществляется выбор замещаемой страницы ОП, и определяет суть того или иного алгоритма замещения. Приведем примеры известных алго­ритмов замещения.

*Алгоритм замещения FIFO.*

Страницы, принадлежащие МА программы и находящиеся в ОП, упорядочиваются по времени из занесения в ОП в цик­лический список. Специальный указатель содержит номер страницы, занесенной в ОП раньше всех остальных. Когда требуется освободить место для вновь затребованной страни­цы, для удаления выбирается страница, номер которой нахо­дится в указателе, а в него помещается номер страницы, сле­дующей в списке за удаленной. Описанный способ замеще­ния страниц основан на предположении, что обращение к странице, занесенной в ОП первой, наименее вероятно.

*Частотный алгоритм замещения.*

Каждой странице МА программы в ОП соответствует счетчик, который увеличивается на единицу при всяком обра­щении к странице. Для удаления выбирается страница, кото­рой соответствует счетчик с наименьшим содержимым. Счёт­чики периодически устанавливаются в ноль. Алгоритм осно­ван на допущении, что для определенных интервалов выпол­нения программы существует набор страниц, обращение к ко­торым происходит наиболее часто.

Следует обратить внимание на то, что за установкой счет­чиков в ноль следует переходной процесс, в течение которого алгоритм эквивалентен алгоритму со случайным выбором замещаемой страницы.

*Алгоритм замещения LRU.*

Согласно этому алгоритму для, каждой страницы МА программы в ОП сопоставляется регистр, в который заносит­ся реальное время последнего обращения к странице. Для замещения выбирается страница с минимальным временем по­следнего к ней обращения.

Алгоритм основан на предположении, что обращение к странице, к которой ссылались ранее других, наименее ве­роятно.

Из теоретических исследований алгоритмов замещения следует, что не для всякой программы (последовательности ссылок) вероятность замещения монотонно убывает с увели­чением числа страниц *v*, отводимых в ОП программе. Так, на­пример, для последовательности ссылок к страницам 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4 при использовании алгоритма замещения *FIFO* для емкости ОП, равной 3 страницам, количество заме­щений меньше, чем для *v* =4. Этот результат, мало согласую­щийся с выводами интуиции, получил название аномалии ал­горитмов замещения.

Однако известен класс алгоритмов замещения (к нему относятся алгоритмы *LRU*, частотный, описываемый ниже ал­горитм ОРТ), у которых указанная аномалия не имеет места.

В технической литературе алгоритмы этого класса также называют стековыми алгоритмами.

Можно привести примеры программ, для которых исполь­зование перечисленных алгоритмов замещения дает наихуд­шие результаты — максимальное число обменов между уров­нями памяти для заданной емкости ОП.

Рассмотрим следующую последовательность ссылок при выполнении циклического участка программы:

R(K)=1K,2K,3K,4K,…, 1K,2K,3K,4K,…

Если емкость ОП равна 3 страницам, то при каждой но­вой ссылке к странице использование алгоритмов *FIFO* и *LRU* приводит к обмену между уровнями памяти.

Не лучшим образом в этом случае работает также частот­ный алгоритм замещения, если при выборе замещаемой стра­ницы оказывается несколько страниц — кандидатов на удале­ние, как это имеет место для рассматриваемого примера, и среди них замещаемая страница выбирается случайно (как и поступают на практике).

Алгоритмы замещения страниц наиболее тщательно иссле­дованы теоретически и в практическом применении.

Известен *принцип оптимальности* при замещении страниц: для произвольной последовательности ссылок к страницам программы число замещений страниц при заданной емкости ОП минимально, если для удаления выбирается в ОП страница, которая имеет максимальное время до следующего к ней обращения.

Считается, что страница, к которой не будет больше обра­щений, имеет бесконечное время следующего к ней обраще­ния; если несколько страниц в ОП при замещении являются кандидатами на удаление, то удаляется любая из таких страниц.

Применяя принцип оптимальности, теперь можно пока­зать, что для предыдущего примера алгоритмы замещения *FIFO* и *LRU* действительно выбирают замещаемую страницу по правилу, прямо противоположному этому принципу.

Сформулированный принцип оптимальности позволяет по­строить *оптимальный алгоритм замещения* страниц (назовем его ОРТ), который хотя и не пригоден для практического ис­пользования, однако может служить в качестве «эталона» при сравнении эффективности различных алгоритмов замещения.

Рассмотрим пример выполнения программы

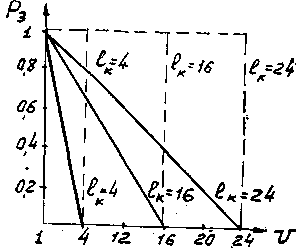
R(K)=1K, 2K,…, *l*K, 1K, 2K,…, *l*K, 1K, 2K,…, *l*K,…

в котором в циклической последовательности вызываются страницы команд 1К, 2К, ..., *l*K, *l*K >1.

Можно показать, что вероятность замещения страниц Р3 по алгоритму ОРТ для заданной емкости ОП *v*, *v* ≥ 1(пусть *v* измеряется в страницах), определяемая как отношение числа замещений к общему числу ссылок к страницам для большого числа повторений цикличе­ского участка программы, выражается формулой

Р3=(*lK-v*)/( *lK-1*).

На рис. 11 сплошные линии показывают зависи­мость Р3 для алгоритма ОРТ от емкости ОП *v* для раз­личных значений *l*K, а на рис. 12 приведены зависи­мости Р3 от *l*K для различных значений *v*.

**Рис.11. Зависимость Р3 от *v*

На рис. 11 прерывистые кривые также показывают зави­симость Р3 от *v* для алгоритмов *LRU* и *FIFO*, причем веро­ятность замещения для этих алгоритмов равна 1, если *l*K >*v*, и равна 0, если *l*K ≤*v*.

Анализ зависимостей рис. 11 и 12 показывает, что вероят­ность замещений даже для оптимального алгоритма может быть очень большой, если велики разность между числом страниц выполняемого циклического участка программы и емкостью ОП.

Следовательно, алгоритмы замещения могут быть крайне неэффективными, что особенно проявляется при применении их в режиме многопрограммной работы.

Как следует из анализа многопрограммной работы, данном в параграфе 3.1, наиболее существенными характеристиками эффективности АМА являются среднее значение размера МА и вероятность обменов информацией между уровнями памяти при выполнении программ. Первой характеристикой опреде­ляется среднее значение емкости ОП, используемой при вы­полнении программы; второй — интенсивность замещений страниц в ОП.

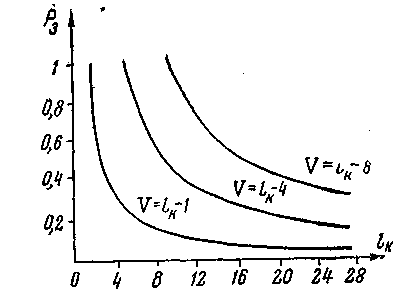


Рис. 12. Зависимость *Р3 (1К)*

Пусть *n* — число ссылок в последовательности ссылок *R*, соответствующей некоторой реализации программы, - число замещений при заданной емкости ОП *х* и применении алгоритма замещения *A*; — среднее время выполнения программы на ЦП, измеряемое между двумя по­следовательными ссылками к различным страницам програм­мы в последовательности ссылок *R.*

Интенсивность замещений , отнесенная к процес­сорному времени, т. е. времени выполнения программы на ЦП без учета обмена, ввода—вывода и ожидания может быть определена приближенно из выражения

Это приближение тем точнее, чем больше ссылок *n* в по­следовательности *R* . Отметим, что число обменов страницами между уровнями памяти и число замещений для заданных ал­горитма замещения *А* и числа страниц *х*, выделенных про­грамме в ОП при ее выполнении, в общем случае, различны. Это объясняется тем, что при замещении сначала из ОП должна быть удалена страница (как правило, страницы дан­ных изменяются при выполнении и их необходимо при заме­щении «копировать» в ВП), и только после этого на ее место загружается вызываемая страница.

На основе анализа экспериментальных данных реальных программ на рис. 13 показан общий характер зависимости для различных алгоритмов замещения интенсивности замеще­ний от числа страниц ОП *х,* выделяемых выполняемой про­грамме.

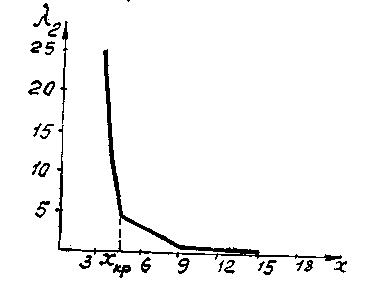


Рис. 13. Зависимость от *х*

Как следует из анализа этой зависимости, для каждой программы существует некоторое критическое значение ем­кости *хкр*, такое, что для *х*≥ *хкр* интенсивность замещений страницами мало изменяется при изменении *х*; однако для *х*< *хкр* интенсивность замещений весьма критична даже к не­большим изменениям *х*.

Основанные на различных правилах удаления страниц, алгоритмы замещения не учитывают особенностей поведения программы на отдельных этапах ее выполнения. Поэтому емкость ОП, необходимая для эффективного выполнения программы, т. е. близкая к *хкр*, для этих алгоритмов форми­руется «слепо», например, путем выделения каждой программе постоянного на все время ее выполнения числа страниц в ОП.

Рассмотрим пример выполнения программы в соответст­вии с последовательностью ссылок к ее страницам *R* = 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, .... 1, 2, 3, 4, 5, 6, 7, 8, .... 6, 7, 8, в кото­рой повторяется по 100 раз выполнение в циклической после­довательности страниц 1, 2, 3, 4, 5 и 6, 7, 8.

Выделив программе в ОП 4 страницы и используя алго­ритмы замещения *LRU* или *FIFO*, получим, что Р3≈3/8.

Однако, если программе при выполнении первого цикличе­ского участка выделить 5 страниц, а второго — 3 страни­цы, то вероятность замещений страниц Р3 становится близкой к 0. При этом программа использует то же самое среднее число страниц ОП на ссылку, равное 4, как и в случае алго­ритмов *LRU* и *FIFO*.

Естественно, что вторая стратегия распределения памяти гораздо эффективнее алгоритмов замещения страниц, однако для ее реализации на практике необходимы адаптивные АМА, обладающие свойствами распознавания особенностей поведения программы в различные моменты времени ее вы­полнения.

### 3.5.2. АМА со свойствами адаптации к особенностям поведения программы при ее выполнении

*Алгоритм Деннига* [4]*.*

Алгоритм Деннига основан на введенном в параграфе 3.4 понятии локализации ссылок при выполнении программы. Согласно этому алгоритму множество активных страниц (сег­ментов) *S(t,T)* в момент времени *t* выполнения программы включает те ее страницы (сегменты) команд и данных, на ко­торые были ссылки на интервале [*t-T, t*]. Таким образом, в отличие от алгоритмов замещения размер *S(t,T)* не остается постоянным в процессе выполнения программы.

Пусть получена реализация *R*= 1, 2, 3, 4, 1, 2, 3, 4, ..., 1, 2, 3, 4, причем время между двумя ссылками постоянно и рав­но *∆t*, *T*= 2 ∙ *∆t*. Тогда *S(0,T)* = {1}, *S(∆t,T)* = {1, 2}, *S(*2 ∙ *∆t,T)*={1,2,3}, *S(*3 ∙ *∆t,T)*={2,3,4}, *S(*4 ∙ *∆t,T)*={2,3,4}, *S(*5 ∙ *∆t,T)* = {1,2,4}, *S(*6 ∙ *∆t,T)* = {1,2,3}, *S(*7 ∙ *∆t,T)*= {2,3,4} и т.д. Для Т=4∙*∆t*, начиная с *t*=3∙*∆t*, *S(t,T)* = {1,2,3,4}.

Одним из самых уязвимых мест алгоритма Деннига яв­ляется проблема правильного выбора параметра *Т.* При больших значениях *Т* увеличивается время нахождения в ОП страниц (сегментов) программы, которые не используются.

При малых значениях *Т*, как показывает предыдущий при­мер, алгоритм Деннига при выполнении циклических участ­ков программы действует аналогично алгоритму *LRU*, т. е. противоположно принципу оптимальности.

При практической реализации алгоритма Деннига каж­дой странице (сегменту) МА программы в ОП соответствует регистр сдвига, состоящий из *k* разрядов. При первом обра­щении к странице (сегменту) 1-й разряд ее (его) регистра устанавливается в единицу. Периодически, через равные фиксированные промежутки времени τ содержимое регистров сдвигается на один разряд от младших разрядов к старшим. Страницы (сегменты), к которым не было обращения в тече­ние времени, равном или большем фиксированного промежут­ка *Т* (Т= τ∙ *k*), исключаются из МА.

*Алгоритм определения МА, основанный на анализе меж­страничных (межсегментных) связей программы* [3].

Этот алгоритм так же, как и предыдущий, относится к классу АМА по требованию и наряду с последовательностью ссылок к страницам (сегментам) программы для определения МА использует информацию о межстраничных (межсегмент­ных) связях программы (матрицы Мк и Мд, введенные в па­раграфе 3.4).

Пусть *R(t)* — последовательность ссылок к страницам (сегментам) программы в момент времени *t* ее выполнения; *R(*к*)(t)* и *R(*д*)(t)* —- соответствующие *R(t)* последовательности ссылок к страницам (сегментам) команд и данных програм­мы; {*R(t)}*, {*R(*к*)(t)}* и {*R(*д*)(t)}* — множество элементов этих последовательностей.

Пусть Nк и Nд — множество страниц (сегментов) команд и данных программы соответственно.

В описываемом алгоритме контроль, а следовательно, из­менение МА происходит в моменты ссылки к очередной стра­нице (сегменту) при выполнении программы. В произволь­ный момент времени *t* МА, обозначаемое *S(t)*, определяется как *S(t)=S*к*(t)U S*д*(t),* где *S*к*(t)* и *S*д*(t)*— множество актив­ных страниц (сегментов) команд и данных программы соот­ветственно.

Положим *S*к*(0)*= *S*д*(0)*=Ø, где Ø — пустое множество.

Пусть *r(ti)* =*q* ссылка к очередной странице (сегменту) команд *q* при выполнении программы; *α(r(ti))* — подмно­жество множества Nк, объединяющее вместе с *r(ti)* страни­цы (сегменты) команд, к которым существует хотя бы один путь на граф-схеме программы, исходящий от *q* (см пара­граф 3.4).

*Определение*. При всякой ссылке к странице (сегменту) команд *r(ti)* положим

*S*к*(ti)* = *α(r(ti)){ R(*к*)(ti)}.*

Таким образом, множество *S*к*(ti)* включает подмножество страниц (сегментов) команд программы, к которым уже ссы­лались на интервале [0, *ti*] и к которым возможны еще ссылки в будущем, так как только они связаны путями со страницей (сегментом) *r(ti)*.

Например, для последовательности ссылок *R* = 1к, 2к, 3к, 2к, 3к, 4к, 5к, соответствующей одной из реализаций програм­мы, изображенной на рис. 8, имеем *S*к*(t1)*  = {1к}, *S*к*(t2)*= {2к}, *S*к*(t3)*= *S*к*(t4)*= *S*к*(t5)* = {2к, 3к}, *S*к*(t6)*= {4к}, *S*к*(t7)=*{5к}.

Обозначим β(*α(r(ti))*) — подмножество страниц (сегмен­тов) данных программы, которые имеют связи со страницами (сегментами) команд множества *α(r(ti))*. Связи между стра­ницами (сегментами) команд и данных определяются по матрице Мд, введенной в параграфе 3.4.

Пусть *r(ti)* очередная ссылка в *R(ti)* к странице (сегмен­ту) данных программы при ее выполнении; *r(tj)*, *tj* < *ti* — ссылка к странице (сегменту) команд, непосредственно пред­шествующая в *R(ti)* ссылке *r(ti)*.

*Определение.* Положим *S*д*(ti)*= β(*α(r(tj))*) { *R(*д*)(ti)* }

Очевидно, *S*д*(ti)* объединяет страницы (сегменты) данных программы, к которым были ссылки на интервале [0, *ti*] ее выполнения и к которым еще возможны ссылки, так как только эти страницы (сегменты) данных имеют связи со стра­ницами (сегментами) команд множества *α(r(tj))*.

В соответствии с определением *S*д*(ti)* для последователь­ности ссылок *R* = 1к, 1д, 1к, 2д, 2к, 1д, 3к, 1д, 4к, 3д, 4к, 2д, 5к (см. рис. 8) имеем *S*д*(t2)*= {1д}, *S*д*(t4)*= {1д, 2д}, *S*д*(t6)*= *S*д*(t8)*= {1д, 2д}, *S*д*(t10)*= *S*д*(t12)*= {2д, 3д}.

Пусть *r(ti)=q* — повторная ссылка в *R(ti)* к некоторой странице (сегменту) команд *q* и *r(ti-v),v* > 0, ссылка к стра­нице (сегменту) команд *r(ti-v)*, непосредственно предшест­вующая в *R(*к*)(ti)* ссылке *r(ti)*. Таким образом, *r(ti){ R(*к*)(ti-v)}*.

*Утверждение.* Если *r(ti)=q* — повторная ссылка к стра­нице (сегменту) команд *q* в *R(ti)*, то *S*к*(ti)*= *S*к*(ti-v)*.

*Доказательство.* Во-первых, заметим, если от *a* Nк существует путь на граф-схеме программы к *b* Nк, то α(*a*) α(*b*). Поэтому, если *а* и *b* связаны путями в обе сто­роны, т. е. существует путь от *а* к *b* и от *b* к *а*, α(*a*)= α(*b*).

Если *r(ti){ R(*к*)(ti-v)}*, то на граф-схеме должны существовать, по крайней мере, два пути, соединяющие стра­ницы (сегменты) команд *r(ti)* и *r(ti-v)* в обе стороны, либо ссылка *r(ti)* непосредственно следует за ссылкой *r(ti-v)* в последовательности *R(*к*)(ti)*.

В обоих случаях α(*r(ti)*)= α(*r(ti-v)*) и, поскольку

.

Пусть справедливы обозначения, предшествующие дока­занному утверждению, и *r(ti)=q* — первая в последователь­ности *R(ti)* ссылка к странице (сегменту) команд *q*, т. е.

*r(ti) { R(к)(ti-v)}.*

*Утверждение.* Если *r(ti-v)* α(*r(ti)*), то *S*к*(ti)* = *S*к*(ti-v)*U*r(ti)*; в противном случае *S*к*(ti)* = *r(ti)*.

*Доказательство.* Предположим, что *r(ti-v)* α(*r(ti)*). Очевидно, в этом случае *r(ti)* и *r(ti-v)* связаны путями в обе стороны и, следовательно, α(*r(ti)*)= α(*r(ti-v)*).

Отсюда .

Пусть *r(ti)* α(*r(ti)*).Следовательно, для любого мо­мента времени, соответствующего ссылке к странице (сегмен­ту) команд *r(ti)*, *ti≤ tj*,

*r(ti)* α(*r(ti)*) и =Ф

(Ф — пустое множество), из чего следует

.

На рис. 14 приведена блок-схема алгоритма контроля *S*к*(ti)*, вытекающая из доказанных утверждений.

Покажем, что для описанного алгоритма в процессе вы­полнения программы ни одна из ее страниц (сегментов) команд не может быть сначала исключена, а затем снова включена в множество активных страниц (сегментов).

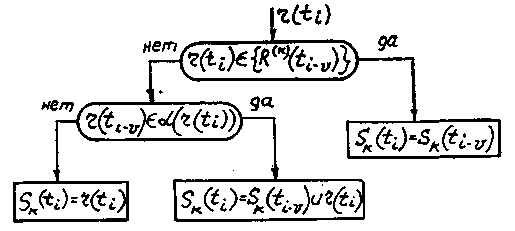


Рис. 14. Блок-схема алгоритма

Пусть *ti* и *ti-m*, *m*>0, — моменты последовательного обра­щения к одной и той же странице (сегменту) команд в по­следовательности ссылок *R*, т. е. *r(ti)*= *r(ti-m)*=*q*.

*Утверждение.* Для всякой ссылки *r(ti)*=*q* к странице (сегменту) команд, такой, что *ti-m≤ tj≤ ti*, *q S*к*(tj)*.

*Доказательство.* Пусть, напротив, существует неко­торый момент времени *ti-m≤ tl≤ ti*, такой, что *q S*к*(tl)*. Так как *S*к*(tl)*= , то *q*. Но из этого следует, что повторная ссылка к *q* после *tl*, невоз­можна, а это противоречит условию утверждения.

Пусть в произвольный момент времени *t* выполнения про­граммы ее страницы (сегменты), принадлежащие *S(t)*, нахо­дятся в ОП, причем первоначально вся программа размеще­на на устройствах ВП.

Обозначим *n*oбм — число ссылок в последовательности *r(tm)*, соответствующей некоторой реализации программы, таких, что , 1*≤ i≤ m*. Иначе говоря, *n*oбм — ко­личество загрузок в ОП страниц (сегментов) программы при ее выполнении и контроле *S(t)* в соответствии с вышеописан­ным алгоритмом.

Положим — среднее значение *S(t)*, т. е. емкости ОП, занимаемой программой на интервале [0, *tm*] ее выполнения.

*Следствие.* Для произвольной последовательности ссы­лок *R*: а) *n*oбм и б) *v*ср < *v*п, где —число страниц (сегментов) программы, *v*п — ее размер.

Из пункта *б* следствия вытекает, что страничный и сег­ментный способы распределения памяти обеспечивают лучшее использование памяти по сравнению со статическим ее распределением, когда программа полностью в процессе ее вы­полнения размещена в ОП.

Кроме того, из пункта *а* следует, что для предложенного алгоритма определения *S(t)* каждая страница (сегмент) за­гружается в ОП не более одного раза при выполнении программы.

Пусть *ti*  и *tj*, *tj* > *ti* — некоторые моменты ссылок в после­довательности *R* к страницам (сегментам) команд програм­мы; | *S*к*(t)*| — число страниц (сегментов) в *S*к*(t).*

*Определение.* Величины

и ,

где / — операция теоретико-множественного вычитания, назо­вем приращением и обновлением *S*к*(t)* соответственно на ин­тервале [*ti*, *tj*] выполнения программы.

Пусть *r(ti-v)* и *r(ti)*, *v*>0, непосредственно следующие в *R* ссылки к страницам (сегментам) команд программы.

*Определение.* Назовем состоянием множества активных страниц (сегментов) команд программы в момент времени *ti* ее выполнения пару <>.

В приведенной ниже таблице перечислены все возможные состояния МА команд программы в зависимости только от знаков величин .

|  |  |  |  |
| --- | --- | --- | --- |
|  | >0 | = 0 | <0 |
| =0 | — | 1 | — |
| > 0 | 2 | 3 | 4 |

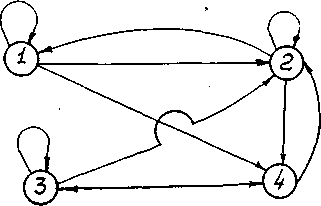
На рис. 15 приведена диаграмма смены состояний МА команд программы, осуществляемая при каждой новой ссыл­ке к странице (сегменту) команд программы при ее выполне­нии в соответствии с некоторой последовательностью *R*.

Рис. 15. Диаграмма состоянии МА

Следующие выводы можно сделать, анализи­руя изменения состояний на диаграмме рис. 15:

1) переход в состоя­ние 1 характеризуется стабилизацией МА ко­манд программы, иначе говоря, в этом состоянии в программе повторяется выполнение некоторого циклического участка (или участков), все страницы (сегменты) которого уже вклю­чены в МА команд программы;

2) при переходе в состояние 2 происходит увеличение ко­личества страниц (сегментов) в МА команд программы, при­чем в отличие от предыдущего случая, переход в это состояние может рассматриваться как некоторый переходный про­цесс начального накопления страниц (сегментов) команд программы, имеющих циклическую структуру межстранич­ных (межсегментных) связей;

3) переход в состояние 3 возможен при условии, что вы­полняемый участок программы имеет линейную структуру межстраничных (межсегментных) связей и, более того, страницы (сегменты) команд программы, используемые на преды­дущих шагах вычислений, не потребуются, поскольку к ним отсутствуют пути, ведущие от страницы (сегмента) команд, вызванной в момент перехода в состояние 3;

4) переход в состояние 4 может рассматриваться как пе­реход от участка выполнения программы, имеющего цикличе­скую структуру межстраничных (межсегментных) связей, к участку с линейной структурой этих связей. Аналогично пре­дыдущему случаю, все страницы (сегменты) команд, вызван­ные до начала этого перехода, больше не будут использовать­ся и занимаемая ими ОП может считаться свободной.

Таким образом, контроль состояний МА команд програм­мы позволяет определять характер структуры межстраничных (межсегментных) связей выполняемого в рассматриваемый момент времени участка программы.

*АМА, основанные на анализе динамических циклов.*

Основной недостаток предыдущего АМА заключается в том, что в МА часто включаются страницы (сегменты) про­граммы, которые после определенного периода активности не используются.

Как развитие этого алгоритма, автором данной работы были предложены АМА, основанные на понятии динамиче­ского цикла, которые отличает более высокая степень адапта­ции к особенностям поведения программы на различных эта­пах ее выполнения.

Эти алгоритмы были подробно исследованы и была пока­зана их высокая эффективность по сравнению со многими широко применяемыми на практике АМА[[2]](#footnote-3).

Перечислим исходные посылки, определившие подход к построению рассматриваемых АМА:

1. интервалы активности определенных страниц (сегмен­тов) при выполнении программы являются следствием ее циклической структуры;
2. велика вероятность того, что после перехода к выпол­нению некоторого циклического участка или системы цикли­ческих участков программы, «принадлежащие» им страницы (сегменты) команд будут выполняться многократно;
3. множество страниц (сегментов) данных программы, активизированных при выполнении некоторого циклического участка, относительно устойчиво к изменениям на всем ин­тервале повторений циклического участка.

Для определения АМА, основанных на понятии динамиче­ского цикла, ниже в качестве реализации программы исполь­зуется последовательность чередующихся ссылок к различным ее страницам (сегментам) команд и данных *R=r(t1), r(t2)…, r(tm),…*

Рассмотрим в этой последовательности произвольную ссылку, *r(ti)*, такую, что для нее существует наибольшее *tj* < *ti* такое, что, *r(ti)= r(tj).*

*Определение.* Назовем динамическим циклом (ДЦ) *b(ti, tj)*, соответствующим интервалу [*ti, tj*] выполнения про­граммы, множество ее страниц (сегментов), принадлежа­щих отрезку *r(tj), r(tj+1),…, r(ti)*  последовательности *R*.

*Определение.* ДЦ *bm(tjm, tim)* следует за ДЦ[[3]](#footnote-4) *bn(tjn, tin)* (*bn - < bm*), если *tim* > *tin*; ДЦ *bm* непосредственно следует за ДЦ *bn* (*bn - < bm*), если *bn - < bm*, и не существует другого ДЦ *bq*, такого, что *bn - < bq* и *bq - < bm*.

Введенное отношение непосредственного следования меж­ду ДЦ упорядочивает их согласно порядка появления в *R* и позволяет от *R* перейти к иному представлению процесса вы­полнения программы в виде последовательности *R*(ц) =а1, а2, .... ак, ..., где аi, — страница (сегмент) программы в момент *ti* первой ссылки к ней или ДЦ, обнаруженный в мо­мент времени *ti*.

Следующий пример иллюстрирует детали указанного пе­рехода от *R* к *R*(ц):

*R* = 1, 2, 1, 2, 3, 4, 5, 4, 5, 4, 3, 4, 3, 6

*R*(ц) = 1, 2, *b3*, *b4*, 3, 4, 5, *b8*, *b9*, *b10*, *b11*, *b12*, *b13*, 6

Здесь ={1,2}, = {1,2}, = = ={4, 5}, = {3,4,5}, = ={3,4}.

*Определение.* ДЦ *bm(tjm, tim)* и *bn(tjn, tin)*  называются независимыми, если *tjn* > *tim* или *tjm* > *tin*.

*Определение.* ДЦ *bm(tjm, tim)* называется вложенным в ДЦ *bn(tjn, tin)*, если *tjm* > *tin*, и *tim* > *tin*.

Таким образом, множество страниц (сегментов) яв­ляется собственным подмножеством множества .

*Определение.* ДЦ имеет ранг вложенности 0, если он не имеет вложенных ДЦ; ДЦ имеет ранг вложенности *i*, если максимальный ранг вложенных в него ДЦ равен *i*—1.

Следовательно, если некоторый ДЦ *b(tj, ti)* имеет ранг вложенности *i* >0, то можно указать, по крайней мере, одну последовательность из *i* —1 вложенных друг в друга ДЦ, каждый из которых, в свою очередь, вложен в *b(tj, ti)*.

*Определение. bm(tjm, tim)* назовем зацепленным, или пересекающимся с ДЦ *bn(tjn, tin)*, если выполняется одна из следующих пар условий:

или .

Очевидно, для Зацепленных ДЦ *bm* и *bn* справедливо , т. е. зацепленные ДЦ имеют хотя бы одну об­щую страницу (сегмент).

В предыдущем примере ДЦ *b3*  зацеплен с *b4*; *b8* — с *b9*; *b8* и *b9* — с *b10*; *b12* — с *b11*; *b13* — с *b12* и *b11*; ДЦ *b8*, *b9* и *b10* вложены в *b11*; ДЦ *b3* и *b8* независимы; попарно независи­мы также ДЦ *b8* и *b12*, *b8* и *b13*, *b9* и *b12*, *b9* и *b13*, *b10* и *b13*.

Докажем некоторые важные утверждения, касающиеся свойств и отношений ДЦ.

*Утверждение.* Если *bm(tjm, tim)* *bn(tjn, tin)*, то *bm* и *bn* независимы в том и только в том случае, если .

*Доказательство.* Пусть ДЦ *bm* и *bn* независимы. Со­гласно определению независимых ДЦ, *tjn* > *tim.*

Покажем, что .

Предполагая противное, окажется, что множества должны иметь хотя бы одну общую страницу (сегмент), к которой были ссылки в *R* на интервалах [*tjm, tim*] и [*tjn, tin*]. Но из этого следует, что *bm* и *bn* не могут быть непосредствен­но следующими ДЦ.

Покажем, что если для непосредственно следующих ДЦ *bm* и *bn* выполняется условие , ДЦ *bm* и *bn* яв­ляются независимыми, т. е. *tjn* > *tim*.

Полагая противное, т. е. *tjn*  *tim*, приходим в противо­речие с условием .

Из доказанного утверждения следует, что независимость между непосредственно следующими ДЦ *bm* и *bn* может контролироваться как путем проверки условия *tjn* > *tim*, так и проверкой отношения.

*Утверждение.* Для непосредственно следующих зацеплен­ных ДЦ *bm(tjm, tin)* и *bn(tjn, tin)* выполнима одна из следую­щих логических формул:

1) ;

2) ;

3) ;

4) ˥()˥(, где ˥ — операции конъюнкции и отрицания.

*Доказательство.* Далее в качестве доказательства утверждения приведены примеры непосредственно следующих ДЦ, для которых истинной является одна из перечисленных логических формул:

1. *R* = 1,2, 1,2; *R*(ц) = 1, 2, *b3, b4*; = ={1,2};
2. *R* = 1, 2, 1, 3, 2; *R(*ц) = 1, 2, *b3,* 3, *b5*; ;
3. *R* =1, 2, 3, 1, 3; *R*(ц) = 1, 2, 3, *b4, b5*; ;
4. *R* =1,2,1,3,1; *R*(ц) =1, 2, *b3,* 3*, b5*;

˥()˥(.

Таким образом, как для зацепленных, так и вложенных непосредственно следующих ДЦ *bm* *bn* может выпол­няться условие .

*Следствие.* Если для непосредственно следующих ДЦ вы­полняется одна из перечисленных в сформулированном выше утверждении логических формул, то такие ДЦ являются вло­женными или зацепленными.

Следовательно, если *bm* *bn* и выполняется формула , то ДЦ *bm* и *bn* являются за­цепленными; в противном случае необходим дополнительный анализ для того, чтобы установить: зацепленными или вло­женными являются рассматриваемые ДЦ.

Рассмотрим наиболее интересные по практическим сооб­ражениям АМА, основанные на анализе ДЦ в последователь­ности *R*(ц) .

Пусть задана некоторая реализация программы *R* и со­ответствующая этой реализации последовательность *R*(ц).

Обозначим *R*(ц) (*i*) начальный отрезок последовательности *R*(ц) , соответствующий начальному отрезку последовательно­сти *R* включительно до *i*-й ссылки; {*R*(ц) (*i*)} — множество ДЦ, принадлежащих *R*(ц) (*i*).

При всякой *i* -й ссылке в *R* описываемые ниже АМА отно­сят к МА *S(i)* множество страниц (сегментов) программы, которые принадлежат определенному числу самых последних в последовательности *R*(ц) (*i*) ДЦ, удовлетворяющих заданно­му отношению между ДЦ.

Алгоритмы *Аm(1)*: в МА включаются страницы (сегмен­ты), принадлежащие *m* самым последним ДЦ в последова­тельности *R*(ц) (*i*).

Алгоритмы *Аm(2)*: в МА включаются страницы (сегменты) программы, принадлежащие *m* различным в теоретико-множественном смысле самым последним ДЦ в *R*(ц) (*i*).

Иначе говоря, два ДЦ в {*R*(ц) (*i*)} не различаются, если равны множества принадлежащих им страниц (сегментов).

Алгоритмы *Аm(3)*: в МА включаются страницы (сегмен­ты), принадлежащие *m* самым последним в *R*(ц) (*i*) ДЦ, из ко­торых каждые непосредственно следующие ДЦ являются вложенными или зацепленными.

Пусть *r(i)* — *i*-я ссылка в последовательности *R*, — множество страниц (сегментов) программы, принадлежащих *m* самым последним в *R*(ц) (*i*) ДЦ, которые удовлетворяют в зависимости от значения признака *Р*, равного 1, 2 или 3, условиям, наложенным на отношения между ДЦ в алгорит­мах *Аm(1)*, *Аm(2)* и *Аm(3)* соответственно. При этом предпо­лагается, что если число ДЦ в *R*(ц) (*i*), удовлетворяющих тре­буемым условиям, меньше *m*, то к необходимо от­нести максимальное количество таких ДЦ.

Пусть *S(i)* — МА программы в момент времени ссылки к ее странице (сегменту) *r(i)*.

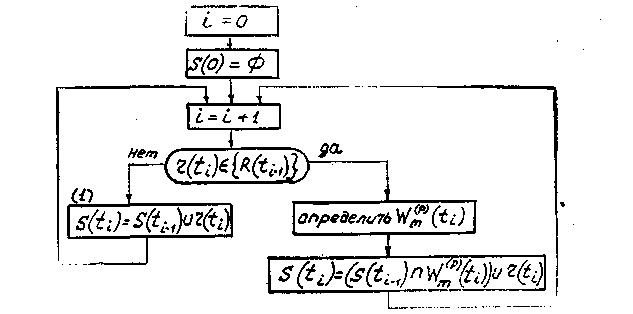


Рис. 16. Блок-схема алгоритмов *А (Р)*

На рис. 16 изображена блок-схема действий при контро­ле *S(i)* в соответствии с одним из вышеперечисленных АМА.

Возможен и другой вариант контроля *S(i)*, если блок 1 заменить следующим: *S(i)* = *r(i),* (*S(ti)* = *S(i)* и *r(ti)* = *r(i))*.

Первый вариант следует принципу: пока не обнаружен очередной ДЦ, страницы (сегменты) программы, к которым происходят обращения впервые, включаются в МА. Во вто­ром варианте контроля *S(i)*, пока не обнаружен очередной ДЦ, к МА относят самую последнюю страницу (сегмент) в *R*(ц) (*i*).

Очевидно, что в первом случае уменьшается вероятность обменов страницами (сегментами) между уровнями памяти, но увеличивается средняя емкость ОП, используемая про­граммой при выполнении (среднее значение ).

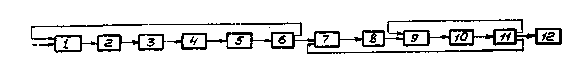
Сделаем одно важное замечание. Для рассматриваемых в этом параграфе и далее алгоритмов определения МА, пред­полагается, что емкость ОП *v* всегда достаточна для разме­щения всех страниц (сегментов), принадлежащих МА *S(t)* программы (). Теоретически такое требование не яв­ляется выполнимым, однако на практике соотношение между размерами выполняемых программ и емкостью ОП совре­менных вычислительных машин и систем позволяет утверж­дать, что вероятность *Р*() чрезвычайно мала. Поэто­му, хотя в том случае, когда , некоторые страницы (сегменты) *S(t)* удаляются из ОП, это оказывает несущест­венное влияние на результирующую вероятность обменов страницами (сегментами) между уровнями памяти при вы­полнении программ, определяемую принятым АМА.

Рис. 17. Граф-схема программы

В качестве примера рассмотрим реализацию програм­мы, граф межстраничных связей которой изображен на рис. 17:

Здесь *l1* = 1, 2, 3, 4, 5, 6; *l2* = 9, 10, 11.

На рис. 18 приведены кривые зависимости вероятности за­мещений страниц *Р3* от числа повторений *k*  циклических участков для данной реализации программы при использовании алгоритмов замещения *LRU* и *ОРТ*. Для каждого алго­ритма изображены 2 кривые, соответствующие двум значениям *v* емкости ОП (*v* =4 стр. и *v* = 5 стр.), выделяемой про­грамме при ее выполнении.

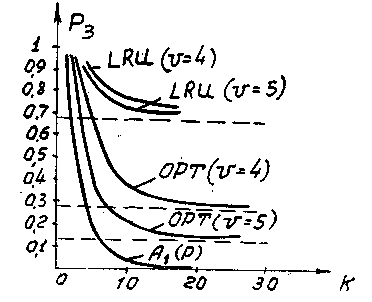
На этом же рисунке приведена зависимость от *k*  вероят­ности того, что при ссылке к очередной странице (сегменту) ее не окажется в ОП при условии, что определение *S(t)* осу­ществляется по алгоритму *А1(р)* (для *m*=1 признак *р* не имеет значения), блок-схема которого приведена на рис. 16.

Рис. 18. Зависимость р3 от *k*

Важно отметить, что среднее число на ссылку страниц ОП, которое не используется программой при ее выполнении в соответствии с алгоритмом *А1(р)*, т. е.

,

приблизительно равно 5 (*n* — число ссылок в *R*).

Таким образом, при одной и той же средней емкости ОП, используемой программой при выполнении АМА, основанные на анализе ДЦ, могут обеспечивать меньшую интенсивность обменов между уровнями памяти.

Это подтверждается обширным статистическим материа­лом, собранным путем моделирования процесса выполнения различных по сложности программ (см. работу, указанную в сноске 2).

АМА, основанные на анализе ДЦ, позволяют на 20 - 50% уменьшить по сравнению с алгоритмами *LRU, FIFO, ОРТ* интенсивность обменов информацией между уровнями памя­ти в зависимости от сложности выполняемых программ при одинаковой средней емкости ОП, используемой программой при выполнении.

АМА, основанные на анализе ДЦ, исключают «болтанку», т. е. частые обмены между уровнями памяти, позволяют луч­ше по сравнению с алгоритмами *LRU, FIFO* предсказывать «будущее» поведение программы; изменяя параметр *m* — число контролируемых ДЦ, можно активно управлять интен­сивностью обменов информацией между уровнями памяти в процессе выполнения программ.

Однако АМА, основанные на анализе ДЦ, более сложны, их использование на практике требует большой изобретатель­ности и применения аппаратных решений для того, чтобы время их «собственной» работы было сравнимо с алгоритма­ми *LRU* или *FIFO*.

Велико среднеквадратное отклонение *S(i)* для этих алго­ритмов по сравнению с алгоритмами замещения, у которых эта величина практически равна 0.

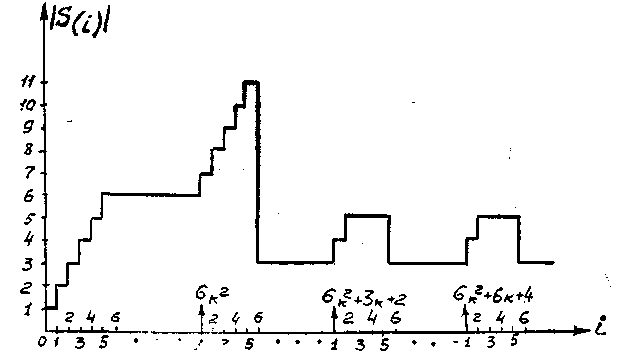


Рис. 19. Диаграмма изменения *S(i)*

На рис. 19 показано изменение |*S(i)*|—мощности *S(i)* в процессе выполнения программы предыдущего примера в со­ответствии с алгоритмом *А1(1)*. Легко проверить, что

*S(1)={1}, S(2)={1, 2}, S(3)={1,2,3}, S(4)={1,2,3,4}, S(5)={1,2,3,4,5}, S(6)=S(7)=…=S(6k2)={1,2,3,4,5,6}, S(6k2+1)={1,2,3,4,5,6,7}, S(6k2+2)={1,2,3,4,5,6,7,8},… S(6k2+6)=…= S(6k2+3k+3)={9,10,11}* и т. д.

На рис. 19 для сокращения обозначений над осью абсцисс указано «смещение», которое необходимо прибавлять к сле­дующим за ним числам под осью, чтобы получить порядко­вый номер соответствующей ссылки.

## 3.6. Особенности управления памятью в случае сегментного способа ее распределения

Проблема выбора свободного участка оперативной па­мяти для размещения загружаемой информации программ не возникает в случае страничного и сегментно-страничного способов ее распределения, так как безразлично, какие свобод­ные страницы ОП при этом будут использованы.

В случае сегментного способа распределения памяти раз­личие в размерах сегментов приводит к тому, что в процессе выполнения программ в ОП образуются разделенные находя­щимися в ОП сегментами свободные участки памяти различ­ного размера.

Более того, если не предусмотрено специальных мер, чис­ло свободных участков увеличивается со временем, а их раз­меры уменьшаются. В результате растет вероятность того, что при загрузке нового сегмента в ОП для него не найдется подходящих размеров свободной зоны и потребуется удаление из ОП или перемещение в ОП сегментов для формирования свободного участка памяти, большего или равного размеру загружаемого сегмента.

Анализ управления памятью в случае сегментного спосо­ба ее распределения показывает, что следующие взаимосвя­занные механизмы определяют эффективность ее использова­ния:

* алгоритмы определения множества активных сег­ментов;
* алгоритмы размещения сегментов в ОП;
* алгоритмы «уплотнения» памяти, формирующие сво­бодные участки требуемого размера.

Различные АМА, которые применяются или могут приме­няться в случае сегментного способа распределения памяти, подробно рассмотрены в параграфе 3.5.

Назначение алгоритмов размещения состоит в выборе сво­бодного участка подходящего размера при загрузке нового сегмента и размещении его таким образом, чтобы замедлить процесс «дробления» свободных участков и тем самым умень­шить число обменов сегментами между уровнями памяти при выполнении программы.

Пусть задана последовательность ссылок *R* = 1, 2, 3, 4, 5, 4, 6, 5 к сегментам 1, 2, 3, 4, 5 и 6 программы при ее выпол­нении; пусть емкость ОП равна 9 единицам, а размеры сегментов в порядке их нумерации равны 3, 4, 1, 2, 2 к 4 едини­цам. Предположим, что при выполнении программы согласно последовательности *R* удаление сегментов из ОП, если для очередного сегмента нет свободного участка подходящих раз­меров, осуществляется в соответствии с алгоритмом *LRU*.

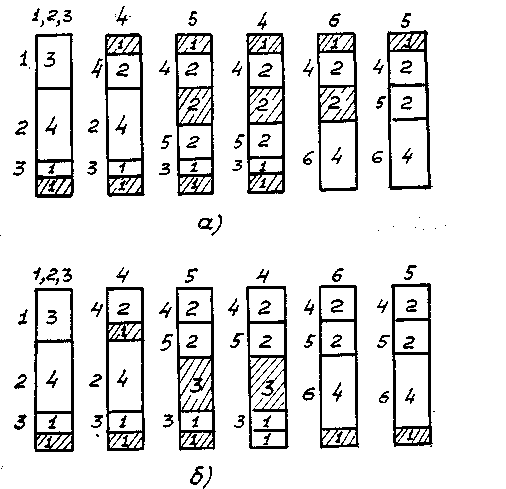


Рис. 20. Состояния памяти

На рис. 20, *а* и *б* показаны последовательные состояния памяти для каждой новой ссылки к сегменту в *R* для двух различных стратегий размещения загружаемых в ОП сегмен­тов.

На рис. 20 сверху над каждым состоянием памяти указа­ны соответствующие ссылки, а слева перечислены номера сегментов в соответствии с занимаемым ими местом в ОП; свободные участки заштрихованы.

Ясно, что вторая стратегия размещения предпочтительнее первой, так как уменьшает число перемещений сегментов между ОП и ВП при выполнении программы.

Как показывает приведенный пример, не только выбор свободного участка при загрузке нового сегмента в ОП, но также и то, к какому из сегментов он будет «плотно примы­кать», оказывает влияние на число обменов между уровнями памяти.

Для контроля состояния памяти в процессе работы в си­стеме организуются обычно два списка: список размещения находящихся в ОП сегментов, который используется для ад­ресации, и список свободных участков, который использует­ся для размещения новых сегментов в ОП.

Пусть *x1, х2,…, xm* — размеры свободных участков ОП и предположим, что поступает требование на размещение в ОП сегмента, имеющего размер *х*.

Приведем примеры двух алгоритмов размещения [4], за­служивающих внимания по практическим соображениям:

1. свободные участки *x1, х2,…, xm*  упорядочиваются в списке в порядке увеличения их размеров; при загрузке сег­мента *х* находится наименьший участок xi, , такой,  
   что .
2. свободные участки *x1, х2,…, xm*  организованы в списке в порядке увеличения их начальных адресов. При загрузке нового сегмента х просматривается список свободных участ­ков и первый свободный участок, удовлетворяющий условию , выбирается для размещения.

Первый алгоритм замедляет по сравнению со вторым про­цесс уменьшения (дробления) свободных участков при вы­полнении программы; второй — более прост в реализации и требует меньшего времени на выбор свободного участка.

Кроме того, чтобы свободные участки меньшего размера не собирались в начале списка, что имеет место после про­должительной работы второго алгоритма, этот список орга­низуется как циклический, с перемещаемой «стартовой» точ­кой, которая всегда указывает на свободный участок, следую­щий за последним выбранным.

Алгоритмы уплотнения информации в ОП так же, как и алгоритмы размещения, нацелены на уменьшение вероятно­сти того, что при занесении в ОП нового сегмента в ней не найдется свободного участка подходящих размеров.

Однако, если алгоритмы размещения достигают этой цели за счет соответствующего выбора свободного участка, алго­ритмы уплотнения либо сдвигают через определенные интер­валы времени всю находящуюся в ОП информацию к одной из ее границ, либо в момент загрузки нового сегмента путем перемещения некоторых сегментов в ОП образуют свободный участок нужного размера. Последнее имеет смысл, если для всех , но .

На рис. 20,*б* при ссылке к сегменту 6 можно было вместо удаления сегмента 3 переместить его «вплотную» к сегмен­ту 2, в результате чего образуется свободный участок, доста­точный для размещения сегмента 6.

На рис. 21 приведена блок-схема взаимодействия при уп­равлении сегментной памятью описанных алгоритмов: АМА, алгоритма размещения (АР) и алгоритма уплотнения памяти (АУП) при ссылке к очередному сегменту *r(ti)*. В блоке 1 алгоритм АМА определяет множество активных сегментов *S(ti)*, в блоке 2 — сегменты программы, не вошедшие в *S(ti)*, но загруженные ранее в ОП (принадлежащие *S(ti-1))* пере­писываются на устройства ВП.

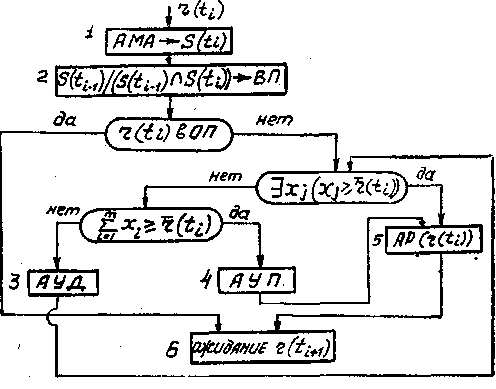


Рис. 21. Блок-схема алгоритма

Отметим, что здесь, как и ранее, рассматриваются только АМА по требованию. Поэтому при ссылке к сегменту *r(ti)* только для этого сегмента может потребоваться загрузка в ОП, если он не принадлежит *S(ti-1)*. Если *r(ti)* нет в ОП, т. е. *r(ti)* не принадлежит *S(ti-1)*, проверяется, есть ли в ОП сво­бодный участок *xj*, , такой, что , где — размер сегмента *r(ti)*.

В блоке 5 работает алгоритм размещения, в блоке 4 вы­полняется алгоритм уплотнения памяти, если суммарный раз­мер свободных участков достаточен для размещения *r(ti)*.

Наконец, если суммарный размер свободных участков в ОП меньше , т. е. емкость ОП *v* меньше суммарного размера сегментов, принадлежащих *S(ti)*, в блоке 3 алгоритм удаления (АУД) удаляет из ОП один или несколько сегментов, чтобы освободить память, необходимую для раз­мещения *r(ti)*.

Как и ранее, мы предполагаем, что вероятность *Р*() очень мала, чтобы алгоритмы удаления могли ока­зывать существенное влияние на вероятность обменов сегментами между уровнями памяти, определяемую, главным образом, АМА, АР и АУП.

Заметим, что возможны стратегии управления сегментной памятью, в которых алгоритмы уплотнения не применяются и вместо них используются алгоритмы удаления, если при за­грузке нового сегмента в ОП нет свободного участка доста­точного размера.

Анализ влияния алгоритмов АМА, АР и АУП на эффек­тивность использования памяти для случая сегментного спо­соба ее распределения представляет собой еще более слож­ную задачу, чем исследование эффективности АМА в стра­ничных системах памяти.

На рис. 22, *а* и *б* приведены примеры изменения состояний ОП при выполнении программы согласно последовательности ссылок к ее сегментам 1, 2, 3, 4, 5, 1, 2, 3, причем, размеры сегментов в порядке их нумерации равны 6, 2, 1, 2 и 4 едини­цам, а емкость ОП — 9 единицам.

На рис. 22,*а* показано изменение состояний памяти при использовании алгоритма замещения ОРТ в случае, если для загружаемого в ОП сегмента нет свободного участка до­статочных размеров. При этом АР играет здесь подчиненную роль и используется только после того, как в ОП путем уда­ления сегментов образован свободный участок, достаточный для размещения загружаемого сегмента.

На рис. 22,*б*, напротив, удаление сегментов из ОП не всегда следует принципу оптимальности, справедливому при замещении страниц в страничных системах памяти, а стратегия распределения памяти также учитывает размеры сегмен­тов, к которым будут ссылки в «будущем».

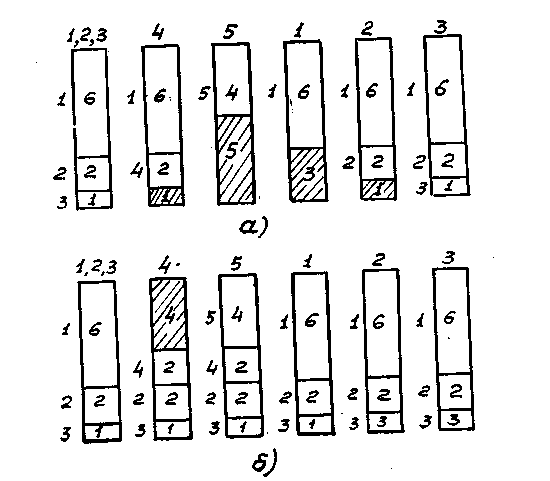


Рис. 22. Состояния памяти

В результате, в первом случае требуется 8 раз загружать сегменты в ОП, и 5 раз удалять сегменты из ОП; во втором случае число загрузок и удалений равно 6 и 3 соответственно.

В работе, которая указана в сноске 2, методами программного моделирования для различных по сложности программ было проведено исследование влияния на эффективность использования памяти различных алгоритмов АМА, АР и АУП.

Следующие выводы вытекают из анализа полученных в ней результатов:

1) АМА играют первостепенную роль в управлении сег­ментной памятью по сравнению с АР и АУП. Применение АМА, основанных на анализе динамических циклов и рассмотренных в параграфе 3.5, позволяет на 10 - 50% в зависи­мости от сложности выполняемых программ (см. пара­граф 3.4) уменьшить вероятность обменов сегментами между уровнями памяти по сравнению с алгоритмами замещения *LRU* и *FIFO* и независимо от применяемых АР и АУП. Важно, что этот эффект достигается при одинаковой средней емкости ОП, занимаемой программой при ее выполнении;

2) при применении одного из алгоритмов замещения *LRU, FIFO, ОРТ* в качестве АМА алгоритмы уплотнения памяти уменьшают всего на 10—15% вероятность обмена сегментами между уровнями памяти по сравнению со случаем, когда вместо этого всякий раз осуществляется удаление части сег­ментов из ОП для того, чтобы образовать свободный участок, достаточный для размещения загружаемого сегмента.

Большое теоретическое и практическое значение для це­лей управления памятью в случае сегментного способа ее распределения имело бы создание таких АМА, которые по своей эффективности не уступают алгоритмам, основанным на анализе динамических циклов, и в то же время не тре­буют применения АР, т. е. размещение сегментов в ОП при выполнении программы всегда осуществляется «плотно» без свободных участков между ними.

Ниже мы опишем один из таких АМА, предложенный ав­тором настоящей работы, и дадим оценку его эффективности.

Предположим, что программа организована из множества сегментов команд *Nк* и данных *N*д.

Пусть *R(tm) = r(t1), r(t2), … r(tm), m*≥1 — последова­тельность ссылок к различным сегментам, соответствующая некоторой реализации программы; *R*(к)(t*m*к) — последовательность ссылок к сегментам команд, полученная из *R(tm)*.

Введем понятие *динамической матрицы* *Д(t)* *следования* сегментов команд в последовательности *R*(к)(t*m*к).

Эта матрица имеет размер | *Nк* |×| *Nк* | и претерпевает из­менения только в моменты *tl*, *1≤ l ≤mк* перехода от одного сегмента команд к другому при выполнении программы.

Элемент матрицы *D(tl)* стоящий на пересечении ее *i*к-й строки и *j*к-го столбца равен 1 или 0 в зависимости от того, существует или нет в последовательности *R*(к)(t*l*) хотя бы одна пара ссылок *r(tv-n)* и *r(tv*), *v ≤ l*, *n*>0, таких, что *r(tv*) непосредственно следует за *r(tv-n)* и *r(tv-n)*= *i*к, *r(tv*) = *j*к.

Таким образом, матрица *D(tl)* является булевской матри­цей и фиксирует непосредственные переходы между сегментами команд при выполнении программы. Так как переход от сегмента команд *r(tv-n)*= *i*к к сегменту команд *r(tv*) = *j*к означает, что сегмент *j*к непосредственно следует за сегмен­том *i*к в графическом смысле, матрица *D(tl)* задает некоторый граф осуществленных переходов (связей) между сегментами команд программы, принадлежащих { *R*(к)(t*l*)}. Этот граф яв­ляется подграфом графа связей между сегментами команд программы (см. параграф 3.4) и аналогично последнему за­дает некоторое множество путей, соединяющих сегменты { *R*(к)(t*l*)}.

Пусть α(*r(tl*)) — подмножество сегментов множества {*R*(к)(t*l*)}, к которым существуют пути, исходящие от сег­мента *r(tl*), которые определяются по матрице *D(tl).*

Обозначим *S*к*(t)* и *S*д*(t)*—множество активных сегмен­тов команд и данных в момент времени *t* выполнения про­граммы; *S(t)= S*к*(t)* U *S*д*(t)*— МА программы в момент вре­мени *t*.

Как и ранее, предположим, что сегменты, принадлежащие *S(t)*, находятся в ОП при выполнении программы.

На рис. 23 приведена блок-схема алгоритма *А* контроля *S(t)* в соответствии с последовательностью ссылок *R(tm)* при выполнении программы. Этот алгоритм исключает появление свободных участков между сегментами *S(t)* при условии, что каждый новый сегмент программы, загружаемый в ОП, размещается «вплотную» (без промежутков) к сегментам про­граммы, находящимся в ОП.

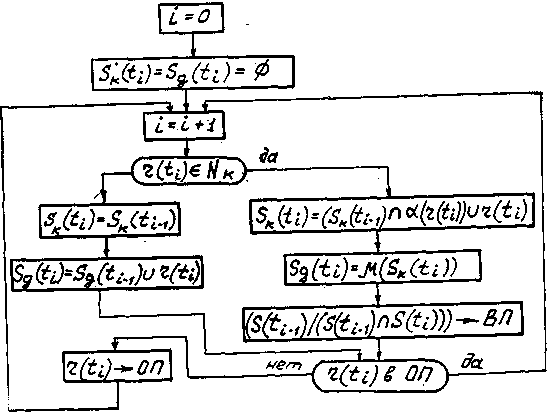


Рис. 23. Блок-схема алгоритма *А*

На рис. 23 µ(*S*к*(t)*) обозначает множество сегментов дан­ных программы, которые использовали при своем выполнении сегменты команд, принадлежащие *S*к*(t)*, с момента их по­следнего занесения в ОП. Операция «→» означает занесение информации в ОП или ВП; например, действия, выполняемые в одном из блоков алгоритма *А*, состоят в переписи из ОП в ВП сегментов программы, не принадлежащих *S(ti)*, но нахо­дящихся в ОП в момент предыдущей ссылки *ti-1*, т.е.принад­лежащих *S(ti-1)*.

Рассмотрим пример выполнения программы согласно последовательности ссылок *R*=1к, 1д, 1к, 1д, 2к, 2д, 3к, 2д, 1к, 1д, 3к, 2д, 4к, 3д, 3к, 2д, 4к, 3д, 4к.

Ниже приведена последовательность изменения состояний ОП в моменты очередной ссылки к сегментам в последова­тельности *R* при условии, что контроль *S(t)* осуществляется в соответствии с вышеприведенным алгоритмом *А*. Порядок перечисления сегментов соответствует порядку их занесения и размещения в ОП.

Состояния ОП:

Докажем, что приведенный алгоритм определения *S(t)* действительно исключает фрагментацию памяти, т. е. появле­ние свободных участков между сегментами программы, нахо­дящимися в ОП.

*Утверждение.* При всякой ссылке *r(ti)*=*q*, 1≤ *i* ≤*m*, к сегменту команд в последовательности *R(tm)* справедливо следующее соотношение:

*Доказательство.* Очевидно, условие для всех и *r(ti)*=*q* означает, что ссылка *r(ti)* соответ­ствует первому обращению к сегменту команд *q*.

В этом случае, по определению *D(ti)* и α(*r(ti*)), α(*r(ti*))=*q* и, следовательно,

Пусть, напротив, *r(ti)*— повторная ссылка к сегменту команд *q*; *r(ti-n), n≥1* —предыдущая ссылка к сегменту *q*, *r(ti-v)*, *v≤n* — ссылка, которая непосредственно предшест­вует ссылке *r(ti)* в последовательности *R*(к)(t*i*).

Для доказательства равенства *S*к*(ti)*= *S*к*(ti-1)*, во-пер­вых, заметим, что для всякой ссылки к сегменту команд *r(tj)* в *R(tm)* и для любой пары сегментов *а* и *b*, принадлежа­щих α(*r(tj)*), существует путь как от сегмента *а* к *b*, так и от сегмента *b* к *а*, определяемый по матрице *D(tj)*. Это следует из определения *D(tj)* и множества α(*r(tj)*). Поэтому и, так как

*r(ti)= r(ti-n)=q, n≥1, r(ti-v)* α(*r(ti)*) и α(*r(ti-v)*) α(*r(ti)*).

Во-вторых, поскольку не изменяется при каждой ссыл­ке к сегменту данных в *R(tm)* (см. блок-схему алгоритма контроля *S(t)* на рис. 23),

.

В свою очередь,

.

Подставляя правую часть последнего равенства вместо в предыдущем равенстве и используя теоретико-мно­жественные соотношения, определенные выше, получим тре­буемый результат.

*Следствие.* Если в момент ссылки к сегменту команд *r(ti)*

*S*к*(ti)= r(ti),*  то *S*д*(ti)=Ф.*

Этот результат следует из определения µ(*S*к*(ti)*) и контро­ля множества активных сегментов данных программы соглас­но рассмотренного алгоритма.

Доказанное утверждение позволяет перейти к эквивалент­ному представлению алгоритма контроля *S(t)*, изображенно­му на рис. 24 (алгоритм В), из которого следует возможность простой его реализации на практике.

*Утверждение*. Контроль *S(t)* и размещение сегментов в ОП, осуществляемые алгоритмом В, исключают фрагмента­цию ОП.

*Доказательство.* В силу принятой стратегий «плотного» размещения сегментов фрагментация ОП может возникать только при удалении сегментов программы из ОП в про­цессе ее выполнения.

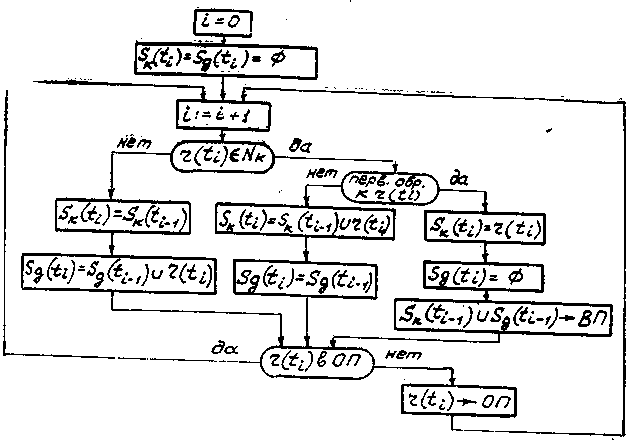


Рис. 24. Блок-схема алгоритма *В*

Однако, как это следует из алгоритма В, удаление сегмен­тов из ОП может происходить только при первой ссылке к не­которому сегменту команд в последовательности *R(tm)*. При этом МА программы становится равным только этому сег­менту, а все другие сегменты программы, находящиеся в ОП, удаляются, в силу чего фрагментация не имеет места.

Без доказательства сформулируем утверждение о верхней грани числа обменов сегментами между ОП и ВП для рассматриваемого алгоритма контроля МА. Под обменом в этом утверждении понимается сам факт обращения к ВП при выполнении программы без учета количества сегментов, считываемых или записываемых на устройства ВП.

*Утверждение.* Величина |*N*к|∙(|*N*к|+1))/2+|*N*к|∙|*N*д| является верхней гранью числа обменов между ОП и ВП при контроле МА в соответствии с алгоритмом В.

Ниже приведен пример последовательности ссылок, для которой данная верхняя грань достигается: *R*(к)= 1к, 2к, 1к, 3к, 1к, 2к, 4к, 1к, 2к, 3к, 5к, 1к, 2к, 3к, 4к.

Принято, что при каждой новой ссылке к сегменту команд в *R*(к) он использует все сегменты данных *N*д.

Предложенный алгоритм распределения сегментной памя­ти хорошо учитывает динамику поведения программы, осо­бенно в случае многократного повторения циклических участ­ков, исключает фрагментацию памяти и, как следует из его эквивалентного представления, чрезвычайно прост в реализа­ции.

## 3.7. Контроль многопрограммности, дисциплины обслуживания программ и распределение памяти

Как следует из данного в параграфе 3.1 качественного анализа общей схемы обслуживания программ в многопро­граммном режиме работы (рис. 5), производительность систе­мы при заданных ограничениях на ее ресурсы в большой степени определяется алгоритмами определения активной ин­формации программ и контроля многопрограммности.

В частности, при ограниченной пропускной способности устройств, связанных с обслуживанием операций ввода—вы­вода и обмена между уровнями памяти, среднее время про­стоя ЦП, по которому можно судить о производительности, должно уменьшаться с увеличением числа одновременно вы­полняемых программ. На практике число программ, одновре­менно выполняемых в системе, ограничено емкостью ОП. Применение стратегий динамического распределения памяти позволяет хранить в ОП не всю программу, а только часть ее, что ведет к увеличению числа программ в ОП. Однако стра­ничный, сегментно-страничный и сегментный способы динами­ческого распределения памяти требуют для своей реализа­ции выполнения такой непроизводительной операции, как обмен информацией между ОП и ВП в процессе выполнения программ, что, в свою очередь, может увеличивать время про­стоя ЦП.

Количественному анализу, устанавливающему взаимо­связь между производительностью многопрограммной работы систем с ограниченными ресурсами и эффективностью алго­ритмов определения МА и контроля уровня многопрограмм­ности, посвящен следующий параграф.

### 3.7.1. Анализ влияния на производительность многопрограммной работы

### систем с ограниченными ресурсами алгоритмов определения МА

### и числа программ в активном состоянии

В этом параграфе, используя аппарат теории массового обслуживания, мы проанализируем влияние на производи­тельность многопрограмм­ной работы алгоритмов оп­ределения МА и числа про­грамм в активном состоянии при заданных ограничениях на ресурсы в системе, такие, как емкость ОП *v*, пропуск­ная способность подсистем ввода — вывода и обмена информацией между уров­нями памяти (см. рис. 5).

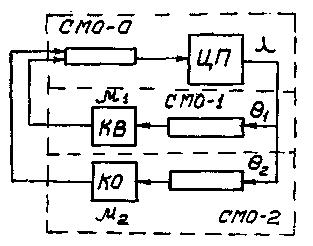
Для этой цели рассмот­рим упрощенную модель функционирования однопро­цессорной системы в режиме многопрограммной работы, изображенную на рис. 25.

Рис. 25. Модель ВС как системы массового обслуживания

Предполагается, что в этой системе в любой момент вре­мени *t* выполняется, т. е. находится в активном состоянии по­стоянное число *N1(t)=k* программ, что соответствует, напри­мер, режиму пакетной обработки с *k* программами в пакете.

Пусть *А* — применяемый в системе алгоритм определения МА; — МА *i*-й программы (1≤*i*≤*k*) в момент вре­мени *t* функционирования системы, определенное в соответст­вии с алгоритмом *А*. Как и ранее, будем рассматривать толь­ко АМА по требованию и предполагать, что в любой момент времени *t* страницы или сегменты каждой *i*-й программы, 1≤*i*≤*k*, принадлежащие находятся в ОП, т. е. выполняется условие . Таким образом, может изменяться только при ссылке к очередной странице или сегменту при выполнении программы, а необходимость загрузки в ОП возникает только при ссылке к странице или сегменту программы, не принадлежащим .

Рассмотрим процесс выполнения программ в исследуемой модели (рис. 25).

Программа, поступившая на обслуживание в ЦП, зани­мает его в течение интервала τ до момента времени *t*, такого что: 1) происходит ссылка к странице или сегменту, отсутст­вующим в ОП; 2) появляется команда ввода—вывода.

В первом случае программа переводится на обслуживание в очередь к КО для обмена с ВП, во втором случае — в очередь к КВ для ввода—вывода информации. После обслуживания в КО и КВ программа поступает в конец очереди к ЦП. ЦП в момент времени *t* переходит к выполнению сле­дующей программы из очереди программ к ЦП при условии, что она не пуста, в противном случае ЦП простаивает.

Описанный процесс функционирования системы можно представить в виде замкнутой сети массового обслуживания с *k* требованиями.

Анализ системы проведем при следующих предположе­ниях:

а) потоки обращений к каналам ввода—вывода и обмена при условии, что ЦП работает непрерывно, — независимые, пуассоновские с параметрами и соответственно. То есть при выполнении программы на ЦП она порождает и   
требований (команд) в единицу времени к КВ и КО соответ­ственно. Отсюда следует, что выходной поток ЦП — пуассоновский с параметром + и в этом потоке очередное требование с вероятностью поступает в очередь к КВ и с вероятностью — в очередь к КО. В силу сказанно­го, ЦП можно представить в виде обслуживающего прибора с интенсивностью обслуживания . На выходе ЦП поток раз­ветвляется на два потока в соответствии с вероятностями и ;

б) длительности обслуживания программ в КВ и КО — независимые случайные величины, распределенные по показательному закону с параметрами соответственно, из­меряемыми в количестве требований, обслуживаемых в еди­ницу времени;

в) время, затрачиваемое на постановку программы в оче­редь или на обслуживание, а также инициацию операций ввода—вывода и обмена, считается равным нулю;

г) программы в очередях обслуживаются соответствую­щим устройством в порядке поступления;

д) программы не покидают систему в процессе ее функ­ционирования.

Отметим, что принятые допущения о пуассоновском харак­тере обращений к КО и КВ и об экспоненциальном законе распределения времени обслуживания в КО и КВ дают воз­можность аналитического анализа модели и удовлетворитель­но согласуются с данными экспериментальных наблюдений работы реальных систем или их моделей.

Пусть — число обращений к КО и КВ, соот­ветствующее некоторой реализации программы *R* длины *m* (*m* — число страниц в *R*) и определению МА по алгоритму *А*. Тогда для достаточно большого *m* можно считать:

где — суммарное время выполнения программы на ЦП; — среднее значение интервала между двумя последова­тельными ссылками к страницам (сегментам) программы при выполнении ее на ЦП; =; =; — интенсивность обслуживания (смены) страниц (сегментов) на ЦП.

Определим в рассматриваемой модели вероятность про­стоя ЦП *P*пр, полагая, что значением (1— *P*пр) измеряется производительность многопрограммной работы системы.

При введенных допущениях модель является линейно-замкнутой показательной сетью, состоящей из трех систем массового обслуживания (СМО): СМО-0 (ЦП), СМО-1 (КВ) и СМО-2 (КО). Состояние сети полностью определяется вектором <a0(t), a1(t), a2(t)>, где ai(t) (*i*=0, 1, 2) — число требо­ваний в СМО-*i* в момент *t*.

Обозначим стационарные вероятности:

где — вероятность того, что в СМО-1 находится *i*, а в СМО-2 *j*-требований, и — вероятность того, что в СМО-*n* находится *m*-требований.

Для определения стационарных вероятностей состоя­ний сети воспользуемся известным результатом: установив­шийся режим в показательной сети представляет суперпози­цию установившихся режимов систем ее составляющих, на­груженных пуассоновскими потоками со стационарными интенсивностями. В данном случае это означает, что

,

где *s+i+j=k;*

),

*0≤s, m, n ≤k, s+m+n=k; с*—нормировочная константа.

Пусть γn, *n* = 0, 1, 2 — стационарная интенсивность пото­ка, проходящего через СМО-*n*. Тогда

Обозначим *а* = , тогда . Очевидно,

, 0≤*i≤k, n=*0, 1, 2.

После подстановки и вычисления *с* получим:

где ;

Искомая вероятность () простоя ЦП находится сумми­рованием:

Исследуем влияние на величин *k*,.

Обозначим через зависимость при фиксированных значениях . Можно пока­зать, что:

1) уменьшается при увеличении *k;*

2)

3)

На рис. 26 представлены семейства кривых для различных фиксированных значений .

Для трех нижних кривых α<1; для средних α >1; для трех верхних кривых α >1.

Таким образом, независимо от значения α =, т. е. независимо от соотношения интенсивности обращений к КВ и интенсивности обслуживания требований в КВ (пропускной способности КВ вместе с управляемыми им устройствами ввода—вывода) уменьшается при уменьшении при не­изменном числе выполняемых в системе программ.

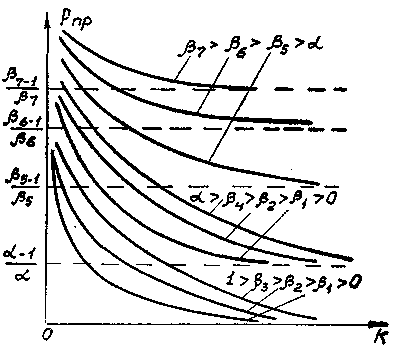


Рис. 26. Зависимости от *k*

Более того, даже для бесконечного числа выполняемых в системе программ (бесконечной емкости ОП) при значениях значением определяется нижняя граница (или верхняя граница производительности) многопрограммной ра­боты (верхние кривые на рис. 26).

Для заданной пропускной способности системы КО и ВП значение определяется отношением , в котором прямо зависит от применяемого АМА и, конечно, характеристик сложности выполняемых в системе программ (см. пара­граф 3.4).

Следовательно, АМА, которые приводят к уменьшению при той же средней емкости ОП, используемой программой при ее выполнении, в общем случае будут более эффективны при их использовании в режиме многопрограммной работы, так как уменьшают .

Этот анализ еще раз подтверждает, что применение алго­ритмов замещения в качестве АМА даже для больших зна­чений А (большой емкости ОП) может приводить к большим значениям простоя ЦП, так как эти алгоритмы определяют МА «вслепую», без учета особенностей поведения про­грамм.

На примере алгоритмов замещения страниц в качестве АМА исследуем зависимость от числа *k* выполняемых в системе программ при ограниченной емкости ОП *v*.

Пусть каждой из выполняемых в системе программ отво­дится постоянное число *х* страниц, равное ближайшему слева целому к значению *v/k* (*v* измеряется в страницах).

На рис. 13 приведена зависимость для алгоритмов замещения *LRU*, полученная из анализа статистических данных выполнения реальных программ в системах со страничной памятью.

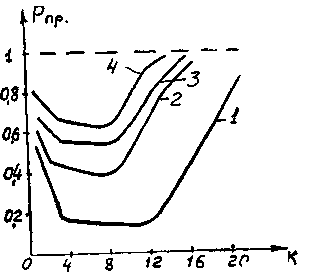


Рис. 27. Параметрические зависимости от *k*

На этом рисунке в качестве единицы времени принято среднее время выполнения на ЦП 1000 команд; например, =10 означает, что в среднем 10 из 1000 команд, выполняемых на ЦП, приводят к необходимости обмена между уровнями памяти.

Считая, что все *k* выполняемых в системе программ имеют одинаковую зависимость , можно построить зависимость , у которой для каждого нового значения *k* значение определяется согласно кривой рис. 13 по значению *х*, бли­жайшему слева целому к *v/k*.

На рис. 27 и 28 построены зависимости для различ­ных значений *v*, α и , взятых из области их практического изменения.

Для всех кривых на рис. 27 *v*=64 стр., =0,5; для кри­вой 1 α =1,2; для кривой 2 α =1,6; для кривых 3 и 4 α равно 2 и 3 соответственно.

Для всех кривых рис. 28 α =1,6; для кривых 1 и 2 *v*=64 стр., равно 0,5 и 0,1 соответственно; для кривых 3 и 4 (пунктирные) *v* =32 стр., а равно 0,5 и 0,1.

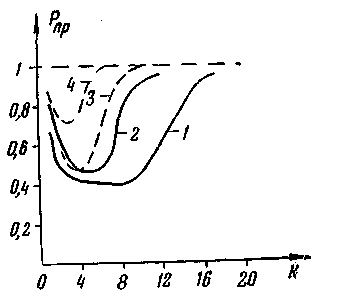


Рис. 28. Параметрические зависимо­сти от А

Анализ этих графиков показывает, что:

а) для каждой комбинации значений *v*, α и существует значение *k=k*опт, при котором принимает минимальное значение;

б) для заданного значения = может быть вы­делена область изменения *k*, *k1≥k≥k2*, такая, что и = для *k1≤k≤k2*;

в) при увеличении емкости *v* оперативной памяти интер­вал [*k1,k2*], определенный в п.*б*, увеличивается.

Очевидно, что результаты последнего анализа могут быть перенесены и на случай использования других АМА, в кото­рых размер МА программы изменяется во времени, если с большой вероятностью выполняется условие

.

Таким образом, как следует из вышеизложенного, конт­роль уровня многопрограммности и АМА действительно яв­ляются центральными механизмами в общей системе управ­ления многопрограммной работой, определяющими производительность всей системы.

### 3.7.2. Дисциплины обслуживания программ и распределение памяти

Рассмотренная в предыдущем параграфе модель представ­ляет собой один из наиболее простых режимов многопро­граммной работы с неизменным множеством *N1(t)* программ в активном состоянии.

В этой модели не учитываются возможность окончания в различные моменты времени программ и перевод в активное состояние новых программ, а также ситуации, когда в неко­торый момент функционирования системы нарушается усло­вие и некоторая программа (или несколько программ) должна освободить ОП.

На практике обычно ко времени обслуживания различ­ных программ предъявляются различные требования, что на­ходит отражение в использовании приоритетных дисциплин обслуживания. При этом программы, имеющие при поступле­нии в систему (см. рис. 5) более высокий приоритет, вытес­няют из активного состояния менее приоритетные программы.

В режиме коллективной работы с системой многих поль­зователей, известном под названием разделения времени, применяются специальные циклические дисциплины обслужи­вания программ, основанные на поочередном выделении кван­та времени для обслуживания всем находящимся в системе программам *N(t)*.

Все это приводит к дополнительному увеличению интен­сивности обменов информацией между уровнями памяти за счет частой смены программ в *N1(t)* и, как следствие, может значительно снижать производительность многопрограммной работы.

Поэтому при контроле множества программ *N1(t)* в актив­ном состоянии большое значение приобретает основанное на учете принятой дисциплины обслуживания правильное пред­сказание моментов времени, в которые программы переводят­ся из пассивного в активное состояние и обратно, а также выбор множества перемещаемых программ.

Пусть — скорость изменения размера МА в момент времени *t*, усредненная на множестве программ, принадлежа­щих *N1(t)*.

Полагая, что величина может хорошо служить для предсказания изменения в будущем, на примерах двух дисциплин обслуживания программ покажем, ка­ким образом может эффективно осуществляться контроль *N1(t).*

*1. Бесприоритетное обслуживание.*

Назовем так дисциплину обслуживания, которая имеет смысл в случае, если к обслуживанию программ в системе не предъявляется никаких временных требований, и которая оставляет наибольшую свободу варьирования порядка обслу­живания программ, находящихся в различных фазах (см. рис. 5) для достижения максимальной производительности.

Для этой дисциплины обслуживания множество *N1(t)* про­грамм, находящихся в активном состоянии в момент време­ни *t*, определяется из условия:

(1)

Если

(2)

то некоторое подмножество программ из *N2(t)* (из оче­реди 1 или 5) переводится в активное состояние (загружается в ОП). При этом, если загружаются программы из очереди 1, их множество активных страниц или сегментов первоначально определяется стратегией начальной загрузки (например, за­гружается первая страница (сегмент) команд программы).

При формировании множества программ могут учитываться различные соображения: планирование при на­чальной загрузке, обеспечивающее равномерную загрузку ЦП, КВ и КО; стремление иметь в активном состоянии мак­симальное число программ; первоначально обслуживать про­граммы из очереди 5 и пр.

Перевод программы из активного состояния в пассивное состояние происходит из очереди 4 в очередь 5 при условии, что суммарное количество требований на страницы (сегмен­ты) программ в очереди 4 ведет к нарушению условия (1). При этом выбор программы из очереди 4, страницы (сегмен­ты) МА которой будут удалены из ОП, может диктоваться также различными соображениями, например, размером МА. Кроме того, не обязательно удаление из ОП всех страниц или сегментов, принадлежащих МА программы.

Очевидно, что в качестве основного события, по которому инициируется переход программ из активного состояния в пассивное и наоборот, является либо окончание выполнения программы, либо прерывание ее при изменении МА.

*2. Приоритетное обслуживание.*

В этом случае предполагается, что всем программам в си­стеме приписано некоторое число из натурального ряда или приоритет величиной которого определяется срочность об­служивания программы.

Основная схема прохождения программ и требование со­блюдения условий (1) и (2) для этой дисциплины обслужи­вания остаются в силе. Однако перевод программ из актив­ного состояния в пассивное и наоборот теперь осуществляет­ся согласно приоритетам, присвоенным программам.

При этом при переводе программы из активного состояния в пассивное состояние выбирается в качестве кандидата про­грамма из очереди 4 с самым низким приоритетом. Аналогич­но, при переводе программы из пассивного состояния в актив­ное выбирается программа с самым высоким приоритетом, но при соблюдении условия (1).

Таким образом, хотя и не всегда программа самого низ­кого (высокого) приоритета переводится из активного (пас­сивного) состояния в пассивное (активное), получается компромиссное решение, направленное на достижение макси­мальной производительности системы для случая приоритет­ного обслуживания программ.

# Заключение

Рассмотренные в настоящем пособии вопросы структур­ной организации и распределения памяти затрагивают одну из самых сложных проблем управления цифровыми вычисли­тельными машинами и системами — проблему управления памятью.

Технический уровень современных средств вычислительной техники позволяет строить самые сложные и совершен­ные в структурном отношении системы памяти. Однако многие проблемы оптимального управления памятью требуют дальнейшего их исследования. В настоящем пособии память машины или системы рассматривалась до определенной сте­пени изолировано, как самостоятельная система, и еще пред­стоит большая работа по использованию известных теорети­ческих и экспериментальных данных, полученных при иссле­довании различных стратегий распределения памяти, при ре­шении общей задачи оптимального распределения ресурсов машины или системы.

# Упражнения

1. Пусть заданы вероятности *p1≥p2≥p3≥…≥pm* обраще­ний к каждой *i*-й, 1≤*i≤m*, странице программы при ее вы­полнении и емкость ОП, равная *n* страницам, *n<m*:

а) покажите, что предельная вероятность замещений стра­ниц при использовании алгоритма, который при замещении удаляет из ОП страницу с наименьшим значением *pi*, 1≤*i≤m,* выражается формулой

б) покажите, что алгоритм замещения страниц пункта *а* является оптимальным на множестве всевозможных алгорит­мов замещения.

2. Предложите пригодные для практического применения программные, программно-аппаратные и аппаратные реали­зации для каждого из рассмотренных в параграфах 3.5 и 3.6 алгоритмов определения МА. Дайте сравнительные времен­ные оценки реактивности предлагаемых .реализаций, т. е. вре­мени, необходимого для отработки требуемых в них функций.

3. Выведите предложенную в параграфе 3.5.1 формулу для замещений страниц циклического участка программы по алгоритму *ОРТ*. Обобщите результат на случай выполнения вложенных циклов.

4. Предложите способы организации программ, увеличи­вающие степень локализации ссылок к страницам (сегмен­там) три их выполнении.

# ЛИТЕРАТУРА

1. Единая система ЭВМ. Под ред. А. М. Ларионова. М., «Статистика», М., 1974.

2. Бертен Ж. и др. Работа ЭВМ с разделением времени. М., «Наука», 1970.

3. Кутепов В. П. Активное множество страниц программы и его пове­дение. Программирование. М., АН СССР, № 1, 1975.

4. Dennig P. J. Virtual memory “Computing Surveys”, 1970, N. 3, v. 2.

1. Напомним, что в случае страничного или сегментно-страничного способа распределения памяти обмен между уровнями памяти осуществ­ляется постранично; в случае сегментного способа распределения памяти единицей пересылаемой информации между уровнями памяти является сегмент. [↑](#footnote-ref-2)
2. Результаты исследования описаны в работе В. П. Пьянкова «Иссле­дование алгоритмов определения активной информации в страничных и сегментных системах памяти» (автореферат диссертации на соискание ученой степени канд. техн. наук, МЭИ, 1976). [↑](#footnote-ref-3)
3. Для сокращения здесь и ниже опускаются временные параметры в обозначении ДЦ, если это не ведет к недоразумениям. [↑](#footnote-ref-4)