электронное научно-техническое издание

НАУКА и ОБРАЗОВАНИЕ

Эл № ФС77 - 30569. Государственная регистрация №0421100025. ISSN 1994-0408

Метод выгрузки страниц памяти на основе обнаружения регулярных обращений

77-30569/342072

 # 04, апрель 2012

В.Н. Семенов, В.А. Крищенко

УДК 004.451.43:004.451.352

МГТУ им. Н.Э. Баумана
v.semenov@ivideon.ru, kva@bmstu.ru

Введение

Вопрос выбора страницы физической памяти для её последующей выгрузки в своп-файл является важным для подсистемы виртуальной памяти. При использования ОС Linux для некоторых рабочих нагрузок наблюдается существенное снижение производительности даже при незначительных превышениях объема распределенной виртуальной памяти над объемом доступной физической памяти.

В операционных системах на основе ядра Linux современных версий для управления виртуальной памятью используется разновидность метода LRU (вытеснение наиболее долго неиспользуемой страницы) в виде алгоритма PFRA [1]. Известно, что такие методы замещения страниц вызывают снижение производительности при последовательном сканировании и циклических обращениях к памяти [2].

Для решения данной проблемы необходимо установить возможные причины такого поведения, а затем разработать и реализовать модифицированный метод замещения страниц оперативной памяти в ОС Linux.

Статья организована следующим образом. В разд. 1 описывается проблема с существующим методом и проводятся предварительный опыт. В разд. 2 приведено описание разработанного метода, а в разд. 3 — его реализация для ядра Linux. Наконец, в разд. 4 представлены результаты экспериментов с программной реализацией метода.

1. Проблемы замещения страниц в ядре Linux

В худшем случае, когда система располагает N страницами физической памяти, а в рабочем множестве процесса содержится $N+1$ страница, при циклическом доступе к памяти в методе LRU страницей-кандидатом на выгрузку будет та страница, к которой произойдет следующее обращение (рис. 1).

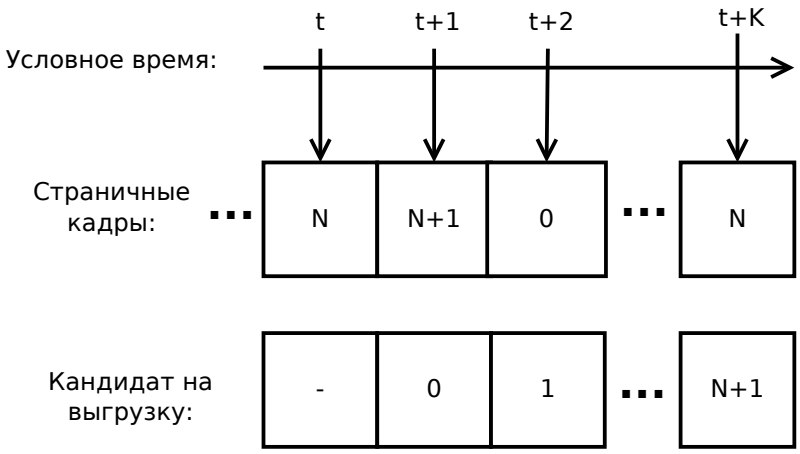

Рис. 1. Проблема циклического обращения

Поскольку в алгоритме LRU выгрузке подвергается страница, к которой наиболее давно производилось последнее обращение, при первом обращении к странице P_{N-1} кандидатом на выгрузку будет страница P_0. В силу циклического характера обращений следующей страницей, к которой процесс запросит доступ, будет выгруженная P_0. При этом выгрузке подвергнется страница P_1 как наиболее ««старая»» и т.д.

Для подтверждения существования описанной проблемы циклических обращений к памяти была разработана тестовая программа, осуществляющая циклический доступ к области памяти указанного размера. На рис. 2 показана зависимость времени её от размера обрабатываемой области памяти. Для сравнения представлены показатели тестовой программы, производящей то же количество обращений по случайным адресам такой же области памяти. Измерения проводились на машине с ядром Linux 2.6.28 и 128 Мб оперативной памяти.

Как видно из рис. 2, в случае циклических обращений производительность LRU-замещения резко падает после прохождения границы в 100% доступной памяти. Это связано с резко

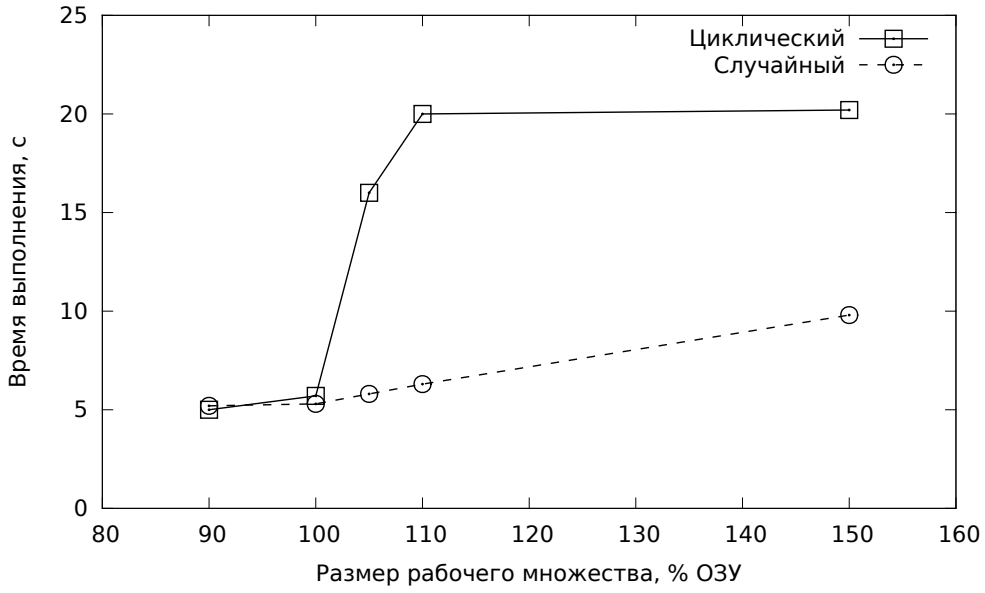

Рис. 2. Проблема циклических обращений

возросшим количеством страничных исключений: при шаге в одну страницу памяти (4 Кб) каждое обращение вызывало операцию замещения страницы. По этой же причине при размерах рабочего множества более 110% физической памяти время выполнения практически не изменяется.

2. Модифицированный метод замещения страниц

Разрабатываемый метод замещения страниц должен обеспечивать следующее:

- определение процессов, работающих с памятью по циклической или сканирующей схеме;
- переключение стратегии замещения для отдельных процессов;
- контроль выполнения предположений о характере работы процессов с памятью.

Схема метода в нотации IDEF0 представлена на рис. 3. Модуль анализа обращений к памяти должен отслеживать обращения процессов в оперативной памяти и на основе полученной информации выявлять те процессы, для которых характерен регулярный (циклический или сканирующий) характер работы с памятью.

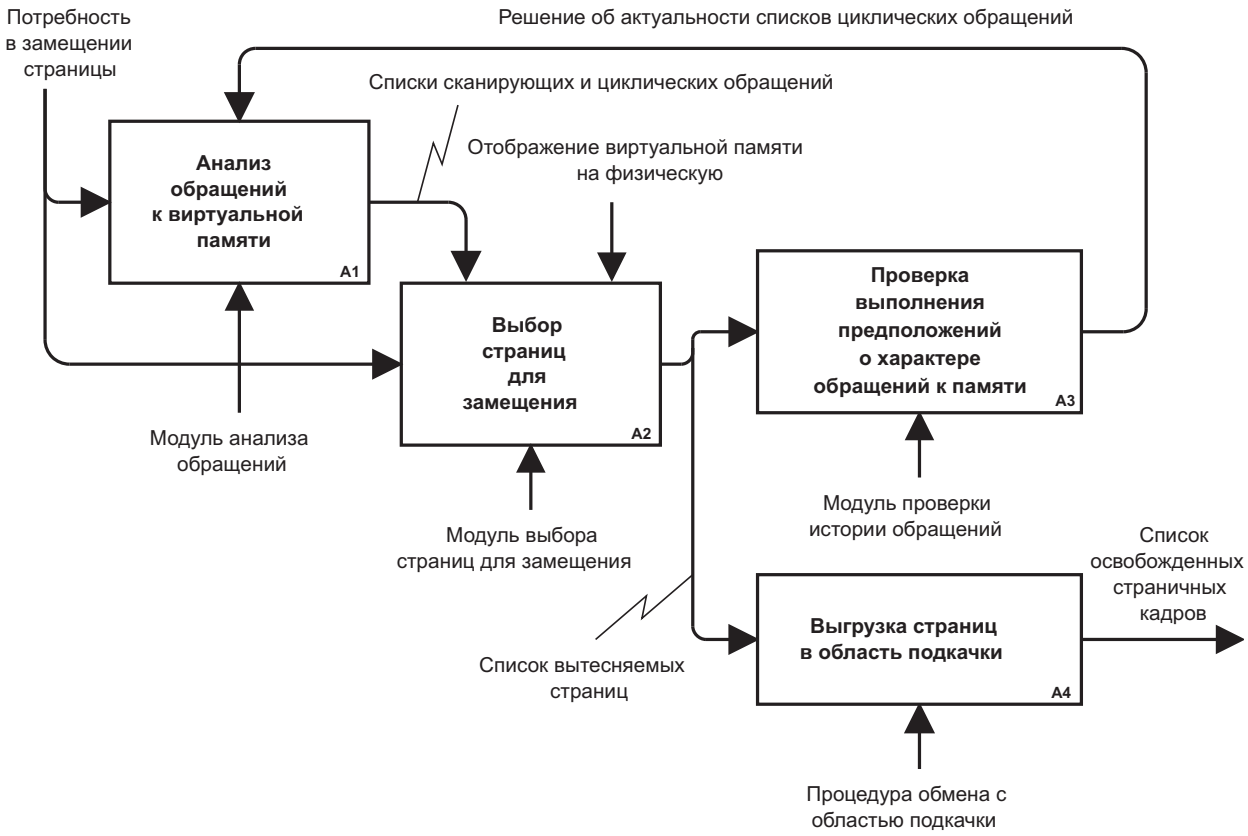

Рис. 3. Функциональная схема метода

Модуль замещения страниц осуществляет управление присутствием в памяти страниц памяти процессов. В результате работы этого блока формируются списки страниц, подлежащих выгрузке при очередном страничном исключении. Замещение страниц оперативной

памяти состоит из выбора кандидата на замещение и обмена содержимым страничных кадров между областью подкачки и физической памятью.

Для того чтобы ядро ОС могло отследить обращение процесса к оперативной памяти, должно произойти страничное исключение из-за отсутствия страницы в физической памяти. Разрабатываемый метод должен использовать информацию о страничных исключениях для распознавания двух типов регулярных обращений к памяти:

• сканирующие обращения — обращения по монотонно возрастающим или убывающим адресам;

• циклические обращения — сканирующие обращения, выполняемые периодически.

Из определения сканирующих обращений следует, что они определяются адресами первого и последнего обращений. Для циклических обращений также необходимо иметь информацию о периоде сканирования. В самом общем виде алгоритм работы блока обнаружения регулярных обращений имеет вид, показанный на рис. 4.

Выбор страниц-кандидатов на замещение производится в соответствии со следующими принципами:

• наиболее желательными кандидатами на замещение являются страницы из областей, к которым обнаружены сканирующие обращения;

• следующими приоритету являются области, к которым обращаются циклически;

• среди областей с циклическими обращениями бóльшим приоритетом обладают более ”инертные” (т.е. включающие в себя наибольшее количество страниц и/или имеющие наибольший интервал между сканированиями);

• при прочих равных условиях предпочтительна выгрузка страниц, принадлежащих тому же процессу, в контексте которого произошло страничное исключение.

В случае, если процесс обладает выявленными сканирующим или циклическим характером обращений к памяти, предпочтительным является страницы-кандидата из множества страниц, принадлежащих циклам этого процесса. Если некоторая страница принадлежит одному из этих циклов, то предпочтительной является выгрузка другой страницы из этого же цикла. В том случае, если цикл не подходит для выбора кандидата по количеству страниц или интервалу сканирования, выбирается максимальный цикл того же процесса. Если же неподходящим является и этот цикл, то выбирается максимально ««инертный»» цикл из всех, обнаруженных в системе.

Если страница не принадлежит процессу с обнаруженными циклическими обращениями к памяти, то сразу реализуется последний из перечисленных вариантов. Наконец, если не обнаружено ни одного подходящего процесса, выбор кандидата осуществляется имеющимся алгоритмом Linux PFRA по схеме LRU.

Худшим случаем работы данного метода замещения является ситуация, когда это предположение о циклическом сканировании выполняется в течение достаточно долгого времени, а затем изменяется направление обращений в пределах того же буфера. Для предотвращения возникновения подобных ситуаций необходимо предусмотреть механизм отслеживания

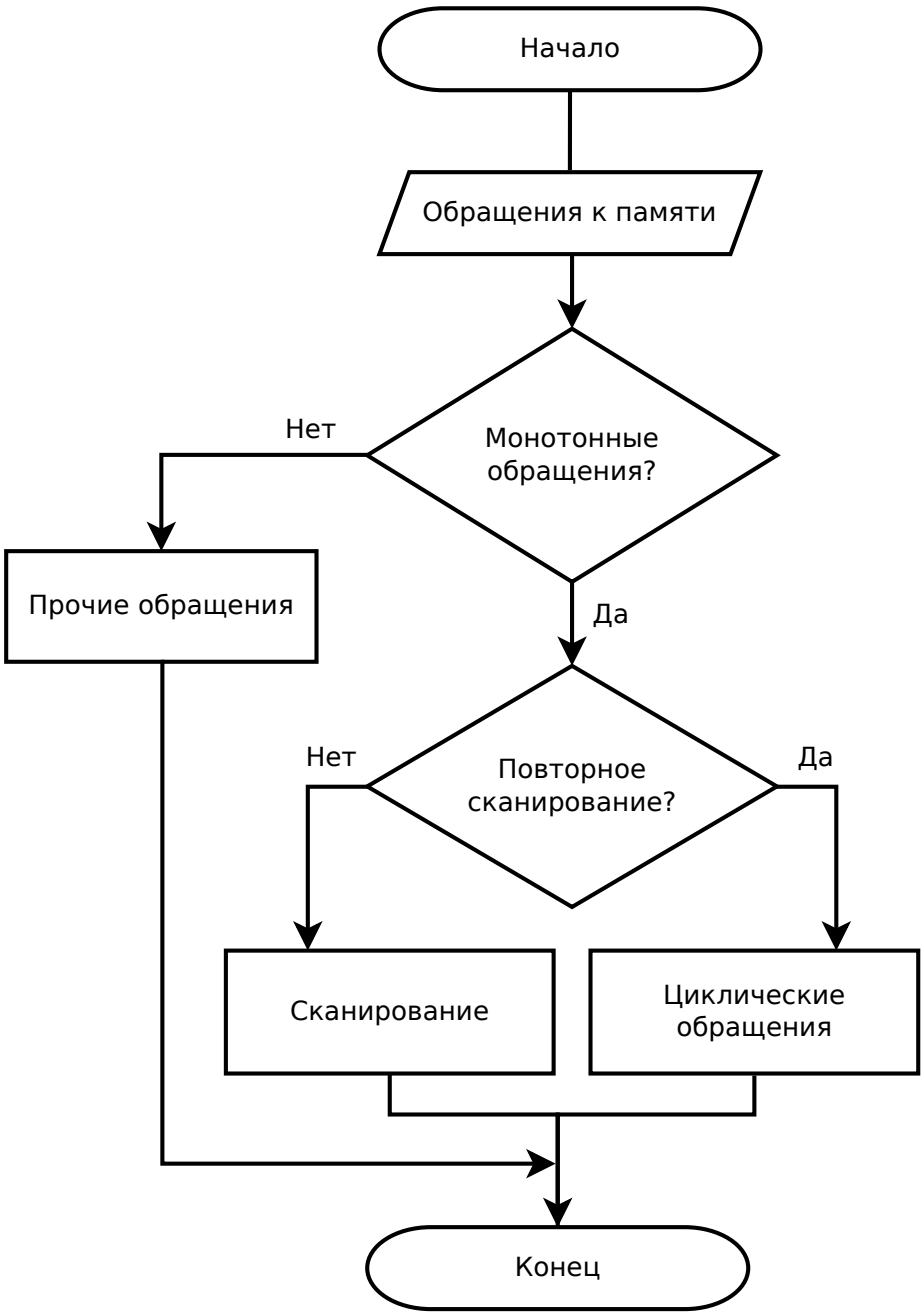

Рис. 4. Алгоритм обнаружения регулярных обращений

прекращения циклических или сканирующих обращений в пределах страниц, находящихся в диапазоне ранее обнаруженного цикла.

В качестве такого механизма в разрабатываемом методе могут использоваться следующие эвристики:

- если страничные исключения часто происходят на недавно выгруженных страницах некоторого цикла, такой цикл следует считать недействительным;
- если часто происходят обращения к страницам, располагающимся внутри некоторого цикла, не соответствующие ранее обнаруженному порядку обхода в цикле, такой цикл следует считать недействительным.

3. Реализация метода замещения

Отслеживание обращений к памяти возможно в точке обработки страничных исключений в ядре. В ядре Linux 2.6 такой точкой является функция **handle_pte_fault()**.

При возникновении страничного исключения на вход поступает линейный адрес страницы, при обращении к которой произошло исключение. Входным параметром функции **handle_pte_fault** является также структура **mm_struct** процесса, в контексте которого произошло страничное исключение. Из этой структуры может быть определен адрес дескриптора соответствующего процесса.

Следовательно, информации, доступной в контексте обработчика страничных исключений, достаточно для определения сканирующих и циклических обращений.

Из того, что отслеживаются виртуальные адреса обращений, следует, что информация об обнаруженных циклах должна быть связана с соответствующими процессами, а не может быть глобальной, как списки страниц LRU в методе PFRA. Структура данных, необходимая для хранения информации об одном цикле имеет следующий вид:

```
struct loop_info {
    unsigned long start_address;
    unsigned long end_address;
    unsigned long last;
    unsigned long period;
    unsigned long pages_present;
    unsigned long bad;
};
```

Поле **pages_present** содержит счетчик страниц диапазона, присутствующих в физической памяти. Это значение уменьшается на единицу при выгрузке страницы из диапазона в область подкачки.

Процесс может обладать выраженным циклическим поведением на нескольких областях, например, на нескольких таблицах БД. Следовательно, необходимо для каждого процесса хранить список структур **loop_info**. Таким образом, реализация метода замещения использует по два списка на процесс:

- **scans_list** — список сканирующих обращений;
- **loops_list** — список циклических обращений.

Указатель на начало списка обнаруженных циклов помещается в дескриптор соответствующего процесса.

Процедура анализа обращений к памяти должна производить на основе поступающих в функцию **handle_pte_fault** адресов, приводящих к исключениям, обнаружение сканирующих и циклических обращений. При очередном обращении к памяти процесс, в контексте которого произошло исключение, определяется по значению макроса **current**.

При срабатывании обработчика страничного исключения по адресу **address** производится следующая последовательность действий. В первую очередь производится обход списка

current-»loops_list. Если **address** принадлежит одному из ранее обнаруженных циклов и при этом выполняется условие **address == loop-»start**, то значение периода этого цикла обновляется: оно устанавливается равным разности между текущим временем и предыдущим обращением к первой странице цикла.

Если же адрес не принадлежит ни одному из циклов, производится аналогичная проверка для списка **current-»scans_list**. Если **address == cycle-»end + 1**, то соответствующее сканирование расширяется на одну страницу. Если же происходит повторное обращение к одной из страниц диапазона сканирования, то:

• если **address == scan-»start**, то запись переносится в список циклов с простановкой соответствующего значения периода;

• если **address » scan-»start && address « scan-»end**, то начало диапазона сканирования устанавливается равным **address**.

Наконец, если страница с адресом **address** не входит ни в один диапазон сканирования, проверяется её принадлежность диапазону сканирования **current_scan**. Если страница входит в него, диапазон расширяется. В противном случае, если ширина диапазона **current_scan** превышает пороговое значение, то адрес **current_scan** переносится в список **scans_list**. После выполнения этой операции начинается отсчет нового текущего сканирования с адреса **address**.

Для ускорения выбора процесса, из адресного пространства которого будет производиться выбор кандидата на выгрузку, вводятся глобальные списки **scan_proc** и **loop_proc**, в которые помещаются указатели на дескрипторы процессов, для которых обнаружены сканирующие или циклические обращения. Порядок обхода списков совпадает с порядком обхода при анализе обращений: в первую очередь выгрузке подлежат сканирующие обращения, затем — циклические.

Для контроля выполнения предположений о характере обращений к памяти с каждым процессом связывается список недавно выгруженных страниц **recent**. В этот список попадают только страницы, выгруженные из областей с циклическими и сканирующими обращениями. Страницы, выгруженные по схеме LRU, не обрабатываются.

Если при очередном страничном исключении страница обнаруживается в списке недавно выгруженных, в информации о соответствующем цикле увеличивается параметр **bad**.

4. Результаты экспериментов

Для исследования производительности разработанного метода использовались программы, относящиеся к различным классам рабочих нагрузок. В качестве вычислительных задач взяты стандартные тесты производительности набора инструментов криптографии OpenSSL, тест производительности библиотеки boost::graph над графами большого размера и распространённый тест умножения матриц MATMUL.

В качестве задачи, нагруженной по памяти, взята выборка данных из таблиц размером до 500 тыс. записей и их соединений в СУБД PostgreSQL 9.0. В качестве сбалансированных задач взяты компиляция исходных кодов ядра ОС Linux компилятором GCC 4.4.

Все рабочие нагрузки выполнялись на виртуальных машинах QEMU 0.12.0 со 64, 128, 256, 512 Мбайт оперативной памяти и одним процессором. В качестве операционной системы использовался специально подготовленный дистрибутив Debian на базе стандартного и модифицированного ядер Linux 2.6.28. Виртуальные машины запускались на машине с ОС Ubuntu Desktop 11.04, оснащенной 4 Гб физической памяти и процессором Intel Core2 Quad-8200.

Для оценки временных затрат, связанных с работой модифицированного метода замещения, был использован синтетический тест библиотекой boost::graph. Обращения к памяти в нем не имеют выраженного циклического или сканирующего характера. Данный тест реализует худший случай с точки зрения созданного метода замещения, поскольку им не будут обнаруживаться сканирующие и циклические обращения. Как видно из рис. 5, максимальная разница по времени выполнения рабочей нагрузки **boost::graph** при стандартном и модифицированном методами замещения составляет около 6%.

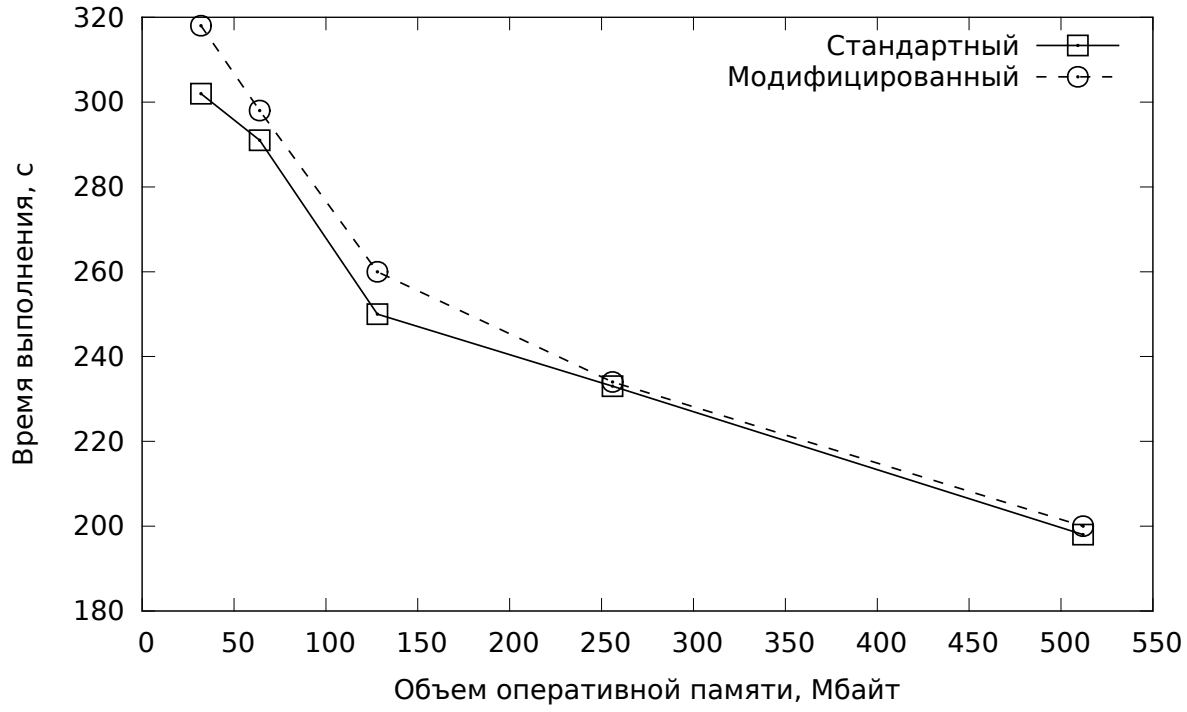

Рис. 5. Рабочая нагрузка: boost::graph

Результаты экспериментов по оценки влияния метода замещения страниц на время выполнения рабочих нагрузок с интенсивным вводом-выводом показаны на рис. 6, метод показал незначительный проигрыш.

В случае нагрузки нагрузки GCC + PostgreSQL метод показал небольшой выигрыш, поскольку задача соединения записей привела к сканирующим обращениям (рис. 7).

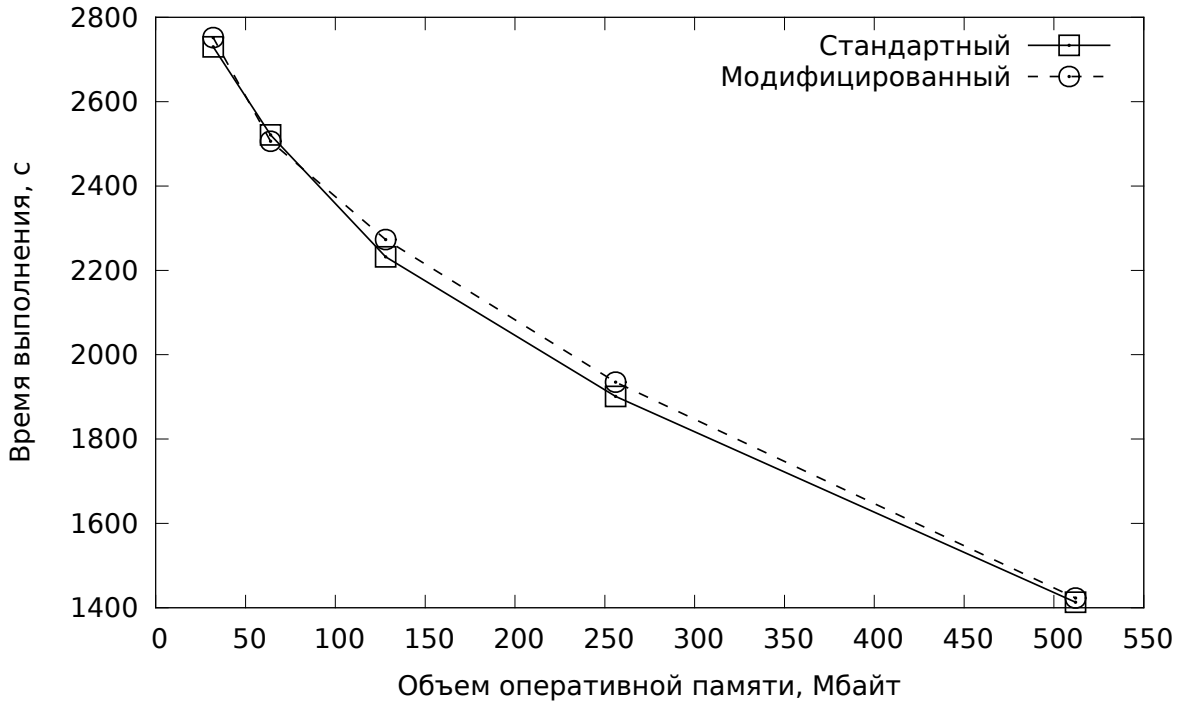

Рис. 6. Оценка влияния на ввод-вывод: GCC

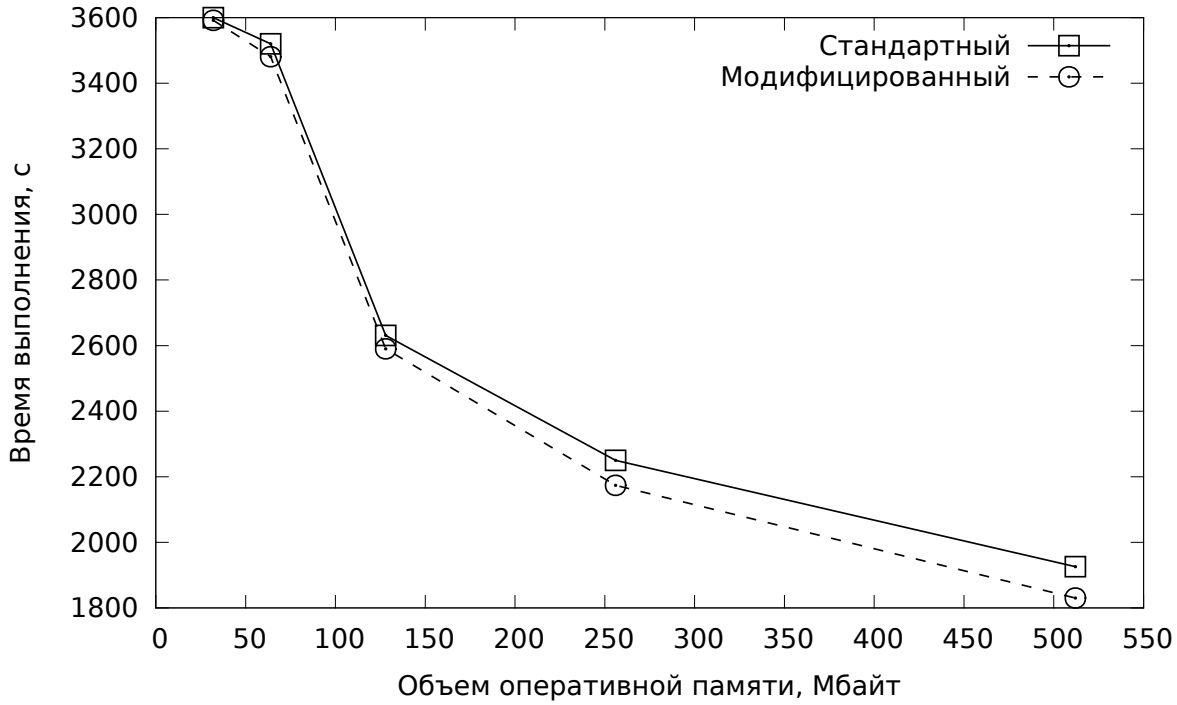

Рис. 7. Рабочая нагрузка: GCC + PostgreSQL

Для экспериментального определения производительности в условиях равномерной загрузки использовался комплект одновременно выполняющихся рабочих нагрузок OpenSSL + MATMUL + GCC. Результат эксперимента показан на рис. 8. В этом эксперименте был обнаружен существенный (до 27%) выигрыш производительности системы, поскольку обращения к памяти в тесте умножения матриц близки к ожидаемым.

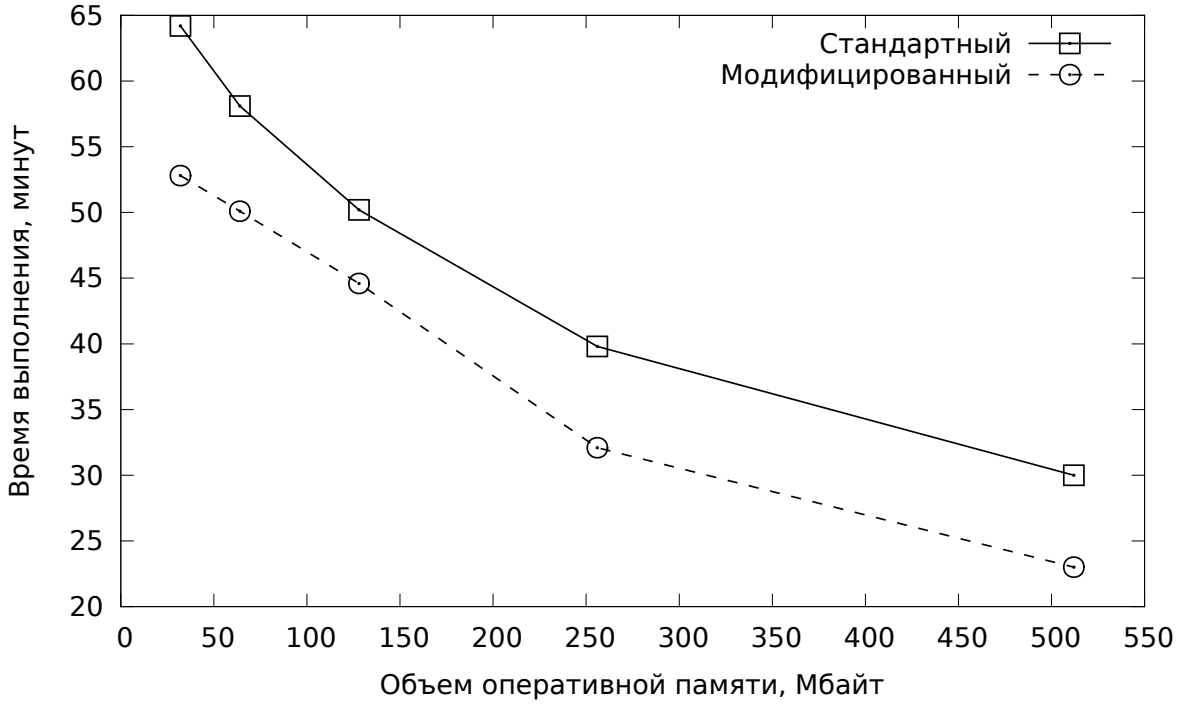

Рис. 8. Рабочая нагрузка: OpenSSL + MATMUL + GCC

Заключение

Был разработан и реализован метод замещения страниц, отслеживающий некоторые монотонные обращения процессов к памяти. Предложенный метод замещения страниц обеспечивает повышение производительности до 27% на некоторых рабочих нагрузках. Выявленный худший случай замедления работы системы при использовании модифицированного метода составляет 6%.

Список литературы

1. *Bovet D.*, *Cesatti M.* Understanding the Linux Kernel, Third Edition. O'Reilly Media, 2005. 944 p.
2. *O'Neil E.J.*, *O'Neil P.E.*, *Weikum G.* The LRU-K Page Replacement Algorithm for Database Disk Buffering // Proceedings of the 1993 ACM SIGMOD Conference, 1993. Pp. 297–306.

electronic scientific and technical periodical

SCIENCE and EDUCATION

El № FS77 - 30569. №0421100025. ISSN 1994-0408

Page frame reclaiming method with scanning access detection

77-30569/342072

 # 04, April 2012

V.N. Semenov, V.A. Krischenko

Bauman Moscow State Technical University
v.semenov@ivideon.ru, kva@bmstu.ru

We describe problems with the current LRU-based page reclaiming algorithm used by Linux kernel that arise in the case of a regular memory access pattern. Based on the results of some preliminary experiment, a pattern detection method is proposed. The method uses page fault history as an input. The developed reclaiming algorithm and its implementation for Linux kernel are described. Experiments results with the method software implementation of this algorithm are presented.

References

1. *Bovet D.*, *Cesatti M.* Understanding the Linux Kernel, Third Edition. O'Reilly Media, 2005. 944 p.
2. *O'Neil E.J.*, *O'Neil P.E.*, *Weikum G.* The LRU-K Page Replacement Algorithm for Database Disk Buffering // Proceedings of the 1993 ACM SIGMOD Conference, 1993. Pp. 297–306.