Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Исследование метода сжатия оперативной памяти в ядре Linux

Студент группы **ИУ7-73Б** _____ **А. В. Романов**
(Подпись, дата) (И.О. Фамилия)

Руководитель ВКР _____ **А. А. Оленев**
(Подпись, дата) (И.О. Фамилия)

Нормоконтролер _____ **Ю. В. Строганов**
(Подпись, дата) (И.О. Фамилия)

***2022** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 25 с., 4 рис., 0 табл., 19 ист., 0 прил.

Рассмотрены понятия сжатия данных, оперативной памяти и виртуальной памяти. Характеризованы современные ядра операционных систем: с монолитной и микроядерной архитектурой. Проведён краткий обзор ядра Linux, структур данных и функций отвечающих за управление памятью внутри ядра. Описана работа модуля zRam, позволяющего хранить страницы виртуальной памяти в сжатом виде оперативной памяти.

КЛЮЧЕВЫЕ СЛОВА

сжатие, оперативная память, Linux, zRam, операционные системы

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1) ЦПУ (центральное процессорное устройство) – электронный блок либо интегральная схема, исполняющая машинные инструкции (код программ), главная часть аппаратного обеспечения компьютера;
2) ОЗУ (оперативно запоминающие устройство) – техническое устройство, компонент аппаратного обеспечения компьютера, реализующий функции оперативной памяти.

ВВЕДЕНИЕ

В последнее десятилетие центральное процессорное устройство достигли своей пиковой тактовой частоты – около 5 Ггц, и этот предел будет преодолён ещё не скоро [1]. Из-за того что ЦПУ стали настолько быстрыми, становится нередка ситуация, когда процессор не выполняет инструкции, а ждёт, пока данные переместятся с диска в оперативное запоминающее устройство (или наоборот) [2]. Так, например, скорость работы системы с мощным ЦПУ, но с маленьким количеством ОЗУ может быть маленькой – несмотря на быстроту, ЦПУ будет часто будет ожидать подсистему ввода/вывода [2].

Цель работы – изучить метод сжатия страниц виртуальной памяти в оперативной памяти в ядре Linux.

Для достижения поставленной цели потребуется:

- описать термины предметной области и обозначить проблему;
- дать характеристику архитектуры ядер операционных систем;
- изучить подходы, структуры данных и функции в ядре Linux, позволяющие управлять оперативной памятью;
- описать работу модуля сжатия оперативной памяти в ядре Linux.

1 Аналитическая часть

В данном разделе рассматривается понятие сжатия данных, оперативной памяти и виртуальной памяти. Даётся характеристика архитектурам современным ядрам операционных систем. Проводится краткий обзор ядра Linux, структур данных и функций позволяющих ему управлять памятью. Описывается работа модуля zRam, позволяющего хранить страницы виртуальной памяти в сжатом виде оперативной памяти.

1.1 ЦПУ и оперативная память

Оперативная память ([3]) – компонент, который позволяет компьютеру кратковременно хранить данные и осуществлять быстрый доступ к ним. Функции оперативной памяти реализуются с помощью специального технического устройства – оперативного запоминающего устройства (ОЗУ). В этой памяти во время работы компьютера хранится выполняемый машинный код и данные, обрабатываемые центральным процессорным устройством (англ. central processing unit, CPU [4]).

Проблему объема оперативной памяти компьютера можно решить увеличив количество планок ОЗУ. Но, у данного подхода есть свои минусы:

- электроэнергия – каждая новая установленная планка увеличивает потребление энергии компьютера [5];
- физические ограничения – количество слотов для планок на материнской плате ограничено;
- стоимость – планка стоит денег.

Альтернативным решением является использование системы подкачки страниц (англ. paging [6]) – специальный механизм операционный системы, при котором отдельные фрагменты памяти (мало используемые или полностью не активные) перемещаются из оперативной памяти во вторичной хранилище, например, на жёсткий диск или другой внешний накопитель. Несмотря на то что количество ОЗУ в таком случае увеличивается, доступ к таким участкам памяти

замедляется – системе приходится работать (читать и писать) с внешним запоминающим устройством, а такие устройства работают значительно медленнее ОЗУ [7].

Ещё одним решением является сжатие данных прямо в оперативной памяти. Данный подход выигрывает у первого рассмотренного решения, так как он реализуется программно и не требует закупки дополнительных планок ОЗУ. Кроме того, этот подход выигрывает и у второго рассмотренного решения, благодаря тому что сжатие не требует работы с внешним накопителем, за счёт чего скорость обработки данных увеличивается. Сжатие данных так же имеет свои минусы, главным из которых является потребность в вычислительных ресурсах. ЦПУ должно выполнять инструкции для сжатия. Но, как и было описано в начале данного раздела, обычно ЦПУ как раз простаивает и не выполняет никакой работы, ожидая ввод/вывод.

1.2 Сжатие данных

Сжатие данных (англ. data compression [8]) – это способ (алгоритмический) преобразования информации в другую форму, обычно более компактную. Сжатие основано на устранение избыточности, которая содержится в исходных данных. Примером является повторение в исходном тексте некоторых фрагментов. Такая избыточность устраняется заменой повторяющейся последовательности ссылкой на уже закодированный фрагмент с указанием его длины. Другой вид избыточности связан с тем, что некоторые значения в сжимаемых данных встречаются чаще других. Сокращение объёма данных достигается за счёт замены часто встречающихся данных короткими кодовыми словами, а редких – длинными.

Все методы сжатия данных делятся на два класса: без потерь и с потерями. Сжатие данных без потерь позволяет полностью восстановить сжатые данные, в отличии от сжатия с потерями. Первый вариант чаще всего используют для сжатия текстовых данных и компьютерных программ, а сжатие с потерями используют для сокращения аудио- или видеоданных – для таких данных не

требуется полное соответствие исходным данным. Алгоритмы сжатия данных с потерями обладают большей эффективностью, чем алгоритмы без потерь [9].

1.2.1 Коэффициент сжатия

Коэффициент сжатия – характеристика алгоритмов сжатия данных, которая определяется формулой (1):

$$k = \frac{V_{original}}{V_{compressed}} \tag{1}$$

где $V_{original}$ – объём исходных данных, а $V_{compressed}$ – сжатых.

Чем выше коэффициент сжатия, тем алгоритм эффективнее. При этом:

- если $k = 1$, то алгоритм сжатия данных не производит. Сжатые данные по размеру равны исходным;
- если $k < 1$, то алгоритм сжатия данных создаёт еще больший (по размеру) блок данных, чем исходный.

1.2.2 Требования к алгоритмам сжатия

К алгоритмам сжатия могут применяться несколько требований:

- скорость сжатия;
- скорость декомпрессии;
- степень сжатия;
- эффективность программной реализации.

Для каждой конкретной задачи будут важны одни требования и менее важны другие. Так, например, для сжатия больших видеоданных с их последующим хранением, важнее будет степень сжатия данных, а не скорость работы алгоритма.

1.3 Ядра операционных систем

Ядро операционной системы – это программное обеспечение, которое предоставляет базовые функции для всех остальных частей операционной системы, управляет аппаратным обеспечением и распределяет системные ресурсы [10]. Ядра операционных систем можно разделить на две группы: с моно-

литным или с микроядром.

Монолитное ядро является самым простым, оно реализовано в виде одного большого процесса, который выполняется в одном адресном пространстве, все службы ядра находятся и выполняются в одном адресном пространстве. Благодаря этому, взаимодействия в ядре осуществляются очень просто, так, например, ядро может вызывать функции непосредственно, как это делают пользовательские приложения [10]. Из-за отсутствия издержек, таких как синхронизация процессов и передача данных между ними, монолитные ядра операционных систем являются более производительным, чем микроядра. На рисунке 1 представлена концептуальная схема операционной системы с монолитном ядром.

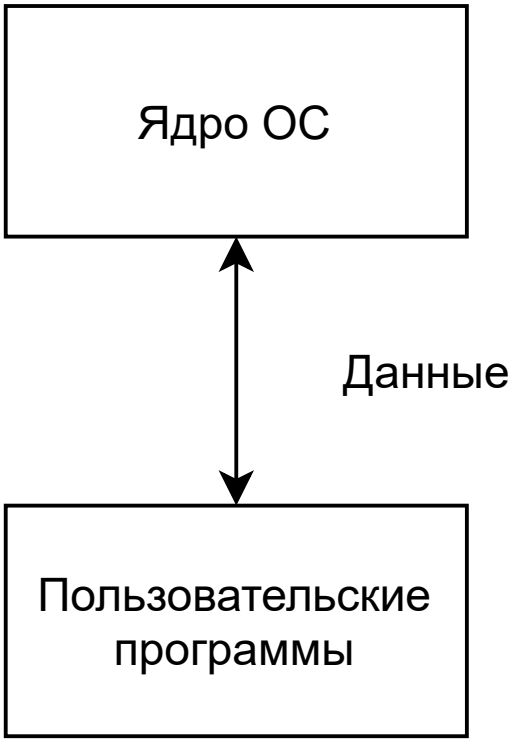

Рисунок 1 – Схема ОС с монолитным ядром

Реализация микроядра подразумевает отказ от одного большого процесса. Все функции ядра разделяются на несколько процессов, которые обычно называют серверами [10]. При этом, в привилегированном режиме могут работать лишь те сервера (процессы), которым этот режим необходим, а остальные сер-

вера работают в непривилегированном (пользовательском) режиме. К первым типам процессов можно отнести, например, механизмы управления памятью компьютера, а ко вторым драйвера устройств. При таком подходе все сервера изолированы и работают в независящем друг от друга адресном пространстве, то есть прямой вызов функций (как в монолитном ядре) невозможен. Все взаимодействия между серверами происходит с помощью механизма межпроцессного взаимодействия (англ. Interprocess Communcation, IPC [11]), а обеспечивает передачу данных само ядро (больше никаких действий в системе оно не выполняет). Такой подход позволяет предотвратить возможность выхода из строя одного сервера при выходе из строя другого и позволяет по по мере необходимости одному серверу выгрузить из памяти другой. Но, как уже было отмечено выше, такая модель имеет более низкую производительность из-за накладных расходов на IPC. На рисунке 2 представлена концептуальная схема операционной системы с монолитном ядром.

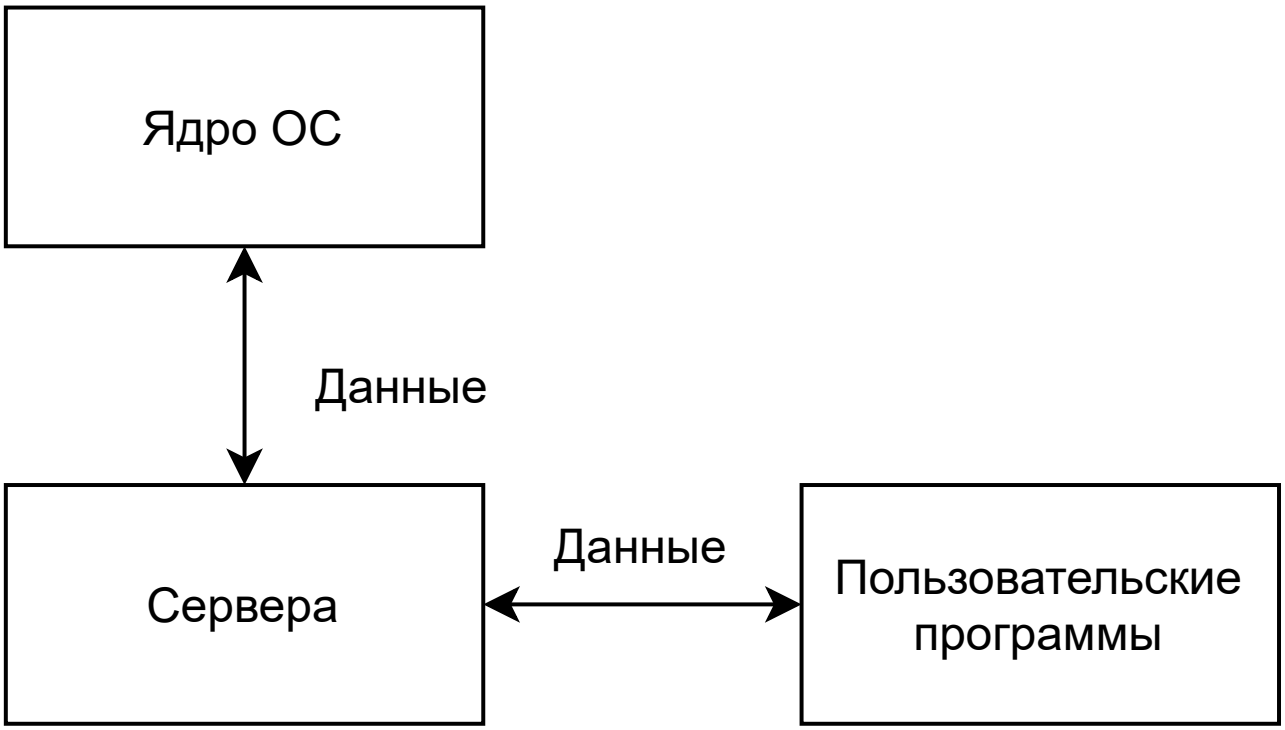

Рисунок 2 – Схема ОС с микроядром

1.4 Ядро Linux

Ядро Linux [12] – ядро операционной системы с открытым исходным кодом, распространяющееся как свободное программное обеспечение. Именно из-за этого в данной работе будет рассмотрено данное ядро. Ядро Linux является монолитным. Но, несмотря на это, при разработке ядра была из микроядерной модели были позаимствованы некоторые решения: модульный принцип построения, приоритетное планирование самого себя, поддержка многопоточного режима и динамической загрузки в ядро внешних бинарных файлов (модулей ядра) [10]. Linux не использует никаких функций микроядерной модели (IPC), которые приводят к снижению производительности системы.

Ниже представлены отличительные черты ядра Linux:

- потоки ничем не отличаются от обычных процессов. С точки зрения ядра все процессы одинаковы и лишь некоторые из них делят ресурсы между собой;
- динамическая загрузка модулей ядра;
- приоритетное планирование. Ядро способно прервать выполнение текущего процесса, даже если оно выполняется в режиме ядра;
- объектно-ориентированная модель устройств. Ядро поддерживает классы устройств и события;
- ядро Linux является результатом свободной и открытой модели разработки [10].

1.4.1 Основные компоненты ядра

Ядро Linux состоит из нескольких основных компонентов:

- планировщик процессов – определяет, какой из процессов должен выполняться, в какой момент и как долго;
- менеджер памяти – обеспечивает работу виртуальной памяти и непосредственно доступ к ней;
- виртуальная файловая система – специальный единый файловый интер-

фейс, скрывающий интерфейс физических устройств;

- сетевые интерфейсы – обеспечивают работу с сетевым оборудованием;
- межпроцессная подсистема – механизмы межпроцессного взаимодействия.

1.4.2 Виртуальная память

Виртуальная память – специальный механизм организации памяти, при котором процессы работают физическими адресами напрямую, а с виртуальными. С помощью такого подхода в ядре Linux реализуется защита адресного пространства процессов, так, например, процесс не может получить доступ к памяти другого процесса и внести туда изменения.

Работа с память в ядре Linux организована с помощью примитива страниц. Страница памяти – набор некоторого количества байт. Хотя наименьшими адресуемыми единицами памяти являются байт и машинное слово, такой подход не является удобным [10].

В ядре каждая физическая страница памяти представляется в виде структуры `struct page`. Рассматриваемая структура с наиболее важными полями представлена в листинге 1.

Листинг 1: Структура `struct page`

```
1  struct page {
2      unsigned long flags;
3      union {
4          struct {
5              struct list_head lru;
6              struct address_space *mapping;
7              pgoff_t index;
8              unsigned long private;
9          };
10     };
11
12     union {
13         atomic_t _mapcount;
```

```
14              unsigned int page_type;
15              unsigned int active;
16              int units;
17          };
18
19          atomic_t _refcount;
20
21      #if defined(WANT_PAGE_VIRTUAL)
22          void *virtual;
23      #endif
24      } _struct_page_alignment;
```

Рассмотрим некоторые поля данной структуры подробнее:

- `flags` – флаги, которые хранят информацию о состоянии страницы в текущий момент, например, находится ли данная страница в кэше или нет. Каждый флаг представляет из себя один бит;
- `_refcount` – счётчик ссылок на страницу. Если значение этого счётчика равно -1, это означает что страница нигде не используется;
- `virtual` – виртуальный адрес страницы, соответствует адресу страницы в виртуальной памяти ядра.

Рассматриваемая структура данных в ядре используется для учёта всей физической памяти, и определения какая область (страницы) памяти свободна, а какая нет.

В ядре Linux предусмотрен специальный низкоуровневый механизм для выделения памяти и несколько интерфейсов доступа к ней. Прототип функции основной функции выделения памяти представлен в листинге 2.

Листинг 2: Прототип функции `alloc_pages`

```
1   struct page *alloc_pages(gfp_t gfp_mask, unsigned int order);
```

С помощью функции `alloc_pages` можно выделить 2^{order} смежных страниц физической памяти. Возвращает указатель на структуру `struct page`, которая соответствует первой выделенной странице памяти. Для выделения одной страницы памяти используется функция `alloc_page` (листинг 3).

Листинг 3: Прототип функции `alloc_page`

```
1  struct page *alloc_page(gfp_t gfp_mask);
```

На рисунке 3 представлена схема управления страницами памяти в ядре Linux.

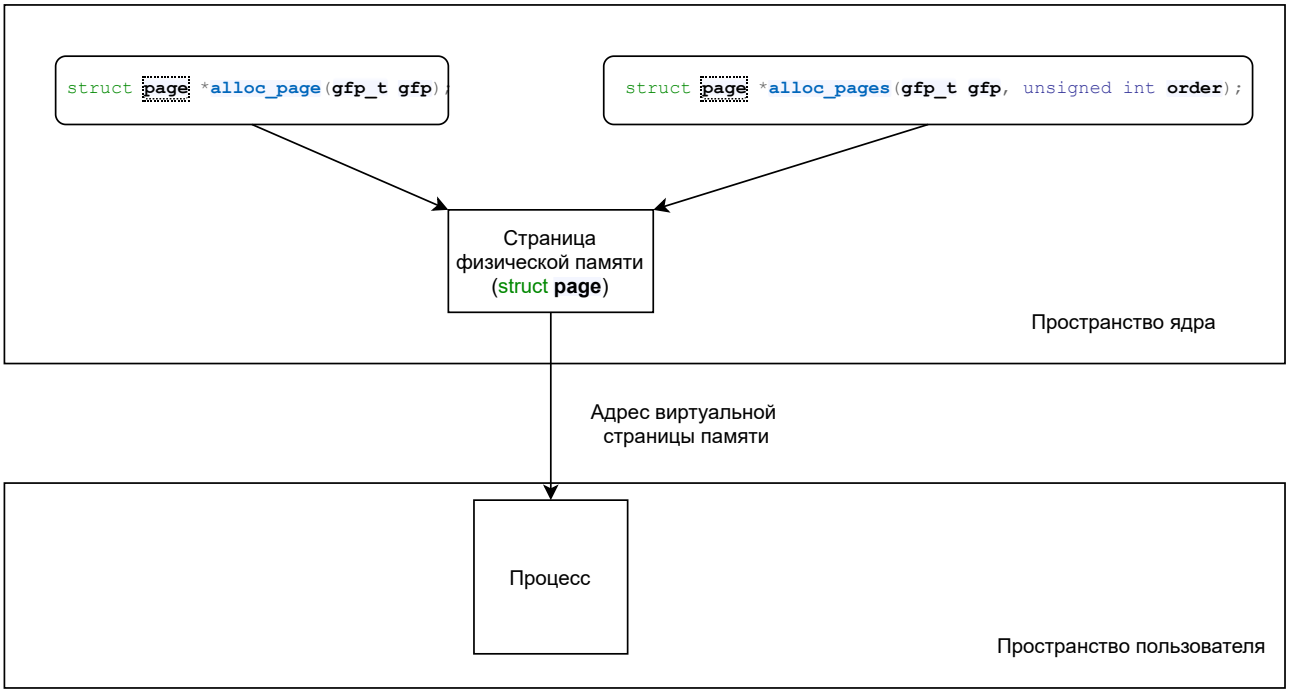

Рисунок 3 – Схема управления памятью

1.4.3 Сжатие памяти в ядре Linux

Для сжатия данных в оперативной памяти, ядро берёт последовательность байт в памяти, сжимает их, записывает сжатую версию обратно в ОЗУ и хранит их до тех пор, пока эти данные не потребуются системе. Пока данные находятся в сжатом состоянии, система не может прочитать или записать какие-либо от-

дельные байты из этой сжатой последовательности. Когда данные потребуется системе вновь, система восстанавливает сжатую последовательность.

Современные алгоритмы могут сжимать любое количество последовательных байт. Несмотря на это, в ядре удобно использовать фиксированную единицу сжатия памяти. Единицей хранения в ядре была выбрана страница памяти (англ. memory page [13]) – фиксированная константа `PAGE_SIZE`, которая на большинстве современных архитектур, поддерживаемых Linux, составляет 4 Кб [14].

Для достижения высокой степени сжатия требуется выполнение большого количества команд ЦПУ, тогда как менее эффективное сжатие может выполняться быстрее. В ядре необходимо добиться баланса между временем и степенью сжатия. Кроме того, важно чтобы выбор алгоритма оставался гибким. Например для выполнения одной задачи подойдёт один алгоритм сжатия, а для второй другой.

Из-за того что размер страницы достаточно большой (4 Кб), сжатие и восстановление данных – достаточно дорогостоящие операции, поэтому необходимо ограничить количество этих операций. Нужно тщательно выбирать, какие страницы стоит сжимать, а какие нет. Алгоритм, реализованный в ядре Linux, определяющий какие страницы нужно сжимать, выбирает те страницы, которые вероятно будут использоваться снова, но вряд ли будут использоваться в ближайшем будущем [2]. Такая реализация позволяет не тратить всё время ЦПУ на многократное сжатие и распаковку страниц. Кроме того, необходимо чтобы ядро могло идентифицировать сжатую страницу – иначе её невозможно найти и распаковать.

Размер страницы, к которой был применён алгоритм сжатия зависит от данных на исходной странице и его трудно предсказать. Степень сжатия страницы описывается следующей формулой (2):

$$k = \frac{PAGE\_SIZE}{zsize} \tag{2}$$

где `PAGE_SIZE` размер страницы памяти в системе, а `zsize` – размер сжатой страницы.

Обычно размер сжатой страницы меньше чем `PAGE_SIZE`, поэтому, обычно, коэффициент сжатия меньше единицы. Но, в некоторых случаях, коэффициент сжатия может быть больше единицы, и для решения этой проблемы необходимо, чтобы алгоритм реализованный в ядре мог найти выход из такой ситуации. В среднем страница памяти в ядре сжимается в два раза [2], то есть коэффициент сжатия $k = \frac{1}{2}$.

1.4.4 Модуль zRam

zRam [15] – модуль ядра Linux, который использует сжатое блочное устройство в оперативной памяти. Данный модуль поддерживает несколько алгоритмов сжатия (на выбор пользователя), например DEFLATE [16], LZ4 [17], ZSTD [18].

Блочное устройство zRam представляет из себя swap-устройство, создаваемый в пространстве пользователя, который является пролетаризированным по сравнению с другими устройствами. Подсистема подкачки отправляет страницу на устройство zRam через подсистему блочного ввода-вывода. Сжатая страница идентифицируется уникальным ключом $zkey$ (3):

$$zkey = device\_id + offset \tag{3}$$

где $device\_id$ – идентификатор устройства подкачки, а $offset$ – смещение страницы.

Когда система подкачки определяет, что какая-либо из страниц, хранящихся в устройстве подачки, необходима пользователю (с помощью page fault [19]), устройство zRam уведомляется, о том что необходимо привести запрашиваемую страницу с идентификатор $zkey$ к исходному виду (распаковать) и вернуть её системе.

Неудачная попытка записать блок в подсистему блочного вывода приво-

дит к большим накладным расходам [2]. По этой причине, при сохранении страницы памяти со степенью сжатия $k < 1$, zRam сохраняет исходную страницу памяти (не сжатую) в ОЗУ, что в конечном итоге не приводит к экономии места.

Модуль в ядре zRam описывается одним экземпляром структуры `struct zram` (листинг 4)

Листинг 4: Структура `struct zram`

```
struct zram {
    struct zram_table_entry *table;
    struct zs_pool *mem_pool;
    struct zcomp *comp;
    struct gendisk *disk;
    struct rw_semaphore init_lock;
    unsigned long limit_pages;

    struct zram_stats stats;
    u64 disksize;
    char compressor[CRYPTO_MAX_ALG_NAME];
    bool claim;
#ifdef CONFIG_ZRAM_WRITEBACK
    struct file *backing_dev;
    spinlock_t wb_limit_lock;
    bool wb_limit_enable;
    u64 bd_wb_limit;
    struct block_device *bdev;
    unsigned long *bitmap;
    unsigned long nr_pages;
#endif
#ifdef CONFIG_ZRAM_MEMORY_TRACKING
    struct dentry *debugfs_dir;
#endif
};
```

Рассматриваемая структура хранит в себе таблицу страниц, семафор, с помощью которого достигается синхронизация при параллельной обработке [2], название алгоритма, которые будет производить сжатие, размер диска, статистику и другие поля.

Для хранения информации о сжатых страницах (их размер и данные) используется специальная таблица. Это необходимо, для того чтобы найти эту страницу для её дальнейшего преобразования к исходному (не сжатому) виду. Для этого используется структура `struct zram_table_entry`, которая представлена в листинге 5

Листинг 5: Структура `struct zram_table_entry`

```
1  struct zram_table_entry {
2      union {
3          unsigned long handle;
4          unsigned long element;
5      };
6      unsigned long flags;
7  #ifdef CONFIG_ZRAM_MEMORY_TRACKING
8      ktime_t ac_time;
9  #endif
```

Для идентификации страниц и управления данными данный модуль использует прямой поиск по таблице страниц (массив из структур `struct zram_table_entry`), которая хранится в структуре `struct zram`.

Для параллельная обработки страниц используется семафор `struct rw_semaphore`.

zRam устраняет дубликаты страниц, которые содержат только нулевые байты. Модуль сохраняет лишь одну такую страницу и все остальные такие же страницы ссылаются на неё, не проводя сжатие этих страниц. Несмотря на то,

чтобы определить, состоит ли страница полностью из нулевых байт, требуются накладные расходы, эти расходы чаще всего небольшие, по сравнению с затратами на сжатие [2].

На рисунке 4 представлена модель взаимодействия модуля zram, ядра и пользователя.

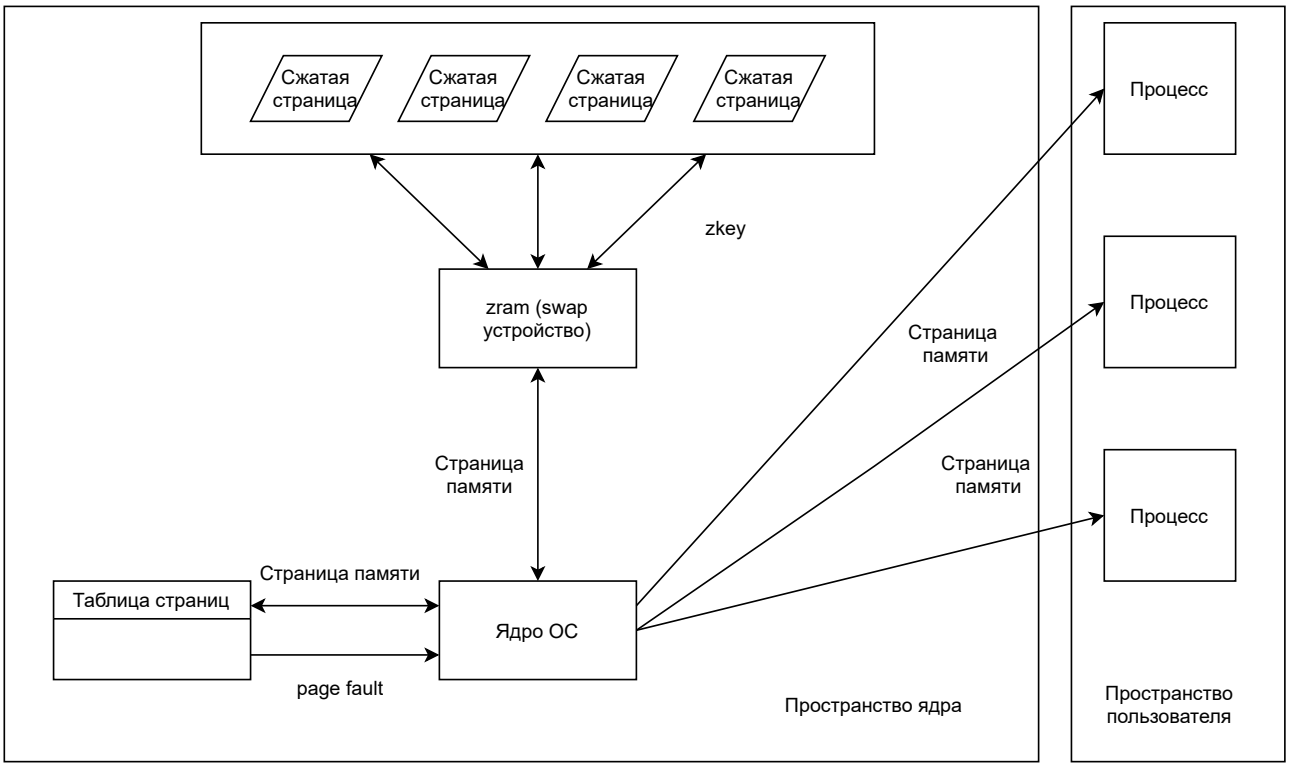

Рисунок 4 – Модель взаимодействия zram, ядра и пользователя

Вывод

В данном разделе были рассмотрены понятия сжатия данных, оперативной памяти и виртуальной памяти.

Были характеризованы современные ядра операционных систем: с монолитной и микроядерной архитектурой.

Проведён краткий обзор ядра Linux, структур данных и функций отвечающих за управление памятью внутри ядра.

Описана работа модуля zRam, позволяющего хранить страницы виртуальной памяти в сжатом виде оперативной памяти.

ЗАКЛЮЧЕНИЕ

Были рассмотрены понятия предметной области: сжатия данных, оперативной памяти и виртуальной памяти.

Были характеризованы современные ядра операционных систем: с монолитной и микроядерной архитектурой, описаны их плюсы и минусы.

Проведён краткий обзор ядра Linux, структур данных и функций отвечающих за управление памятью внутри ядра: `struct page`, `alloc_pages()`, `alloc_page()`.

Описана работа модуля zRam, позволяющего хранить страницы виртуальной памяти в сжатом виде оперативной памяти. Рассмотрены основные структуры данных, использующиеся в этом модуле: `struct zram` и `struct zram_table_entry`.

Таким образом, цель работы, изучить метод сжатия страниц виртуальной памяти в оперативной памяти в ядре Linux, была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Why Aren’t CPUs Getting Faster? - Apple Gazette [Электронный ресурс]. – Режим доступа: https://applegazette.com/mac/why-arent-cpus-getting-faster/, свободный – (10.11.2021)

2. In-kernel memory compression [Электронный ресурс]. – Режим доступа: https://lwn.net/Articles/545244/, свободный – (10.11.2021)

3. Designing Embedded Systems with 32-Bit PIC Microcontrollers and MikroC, 2014. Dogan Ibrahim. c. 1 - 39.

4. An Introduction to Information Processing, 1986. c. 45 - 71.

5. What are the negative effects on increasing RAM of a PC? [Электронный ресурс]. – Режим доступа: https://www.quora.com/What-are-the-negative-effects-on-increasing-RAM-of-a-PC, свободный – (10.11.2021)

6. Theoretical Computer Science, 15 July 1997. Esteban Feuerstein. c. 75 - 90.

7. SSD vs. HDD | Speed, Capacity, Performance & Lifespan | AVG [Электронный ресурс]. – Режим доступа: https://www.avg.com/en/signal/ssd-hdd-which-is-best, свободный – (10.11.2021)

8. Encyclopedia of Information Systems, 2002. Khalid Sayood. c. 5 - 27.

9. Industrial Control Technology. Peng Zhang, 2008. c. 675 - 774.

10. Ядро Linux. Описание процесса разработки. Третье издание, 2019. Роберт Лав. с. 25 - 36.

11. Linux Kernel 2.4 Internals: IPC mechanisms [Электронный ресурс]. – Режим доступа: https://tldp.org/LDP/lki/lki-5.html, свободный – (15.11.2021)

12. What is Linux? – The Linux Kernel Documentation [Электронный ресурс]. – Режим доступа: https://www.linux.com/what-is-linux/, свободный – (10.11.2021)

13. Memory Management — The Linux Kernel Documentation [Электронный ресурс]. – Режим доступа: https://www.kernel.org/doc/html/latest/admin-guide/mm/index.html, свободный – (10.11.2021)

14. Using 4KB Page Size for Virtual Memory is Obsolete [Электронный ресурс]. – Режим доступа: https://ieeexplore.ieee.org/document/5211562, свободный – (10.11.2021)

15. zram: Compressed RAM-based block devices - The Linux Kernel Documentation [Электронный ресурс]. – Режим доступа: https://www.kernel.org/doc/html/latest/admin-guide/blockdev/zram.html, свободный – (10.11.2021)

16. DEFLATE Compressed Data Format Specification version 1.3 [Электронный ресурс]. – Режим доступа: https://www.w3.org/Graphics/PNG/RFC-1951, свободный – (20.11.2021)

17. lz4/lz4: Extremely Fast Compression algorithm - GitHub [Электронный ресурс]. – Режим доступа: https://github.com/lz4/lz4, свободный – (20.11.2021)

18. facebook/zstd: Zstandard - Fast real-time compression algorithm [Электронный ресурс]. – Режим доступа: https://github.com/facebook/zstd, свободный – (20.11.2021)

19. Kernel level exception handling – The Linux Kernel Documentation [Электронный ресурс]. – Режим доступа: https://github.com/facebook/zstd, свободный – (20.11.2021)