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 ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Оптимизация метода сжатия страниц оперативной памяти в ядре Linux

Студент группы **ИУ7-83Б**		**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

Тут точно нужно что-то поменять

Расчетно-пояснительная записка 32 с., 6 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

Тут точно нужно что-то поменять

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 центральные процессорные устройства (ЦПУ) 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) [4]. Так, например, скорость работы системы с мощным ЦПУ, но с маленьким количеством ОЗУ может быть маленькой – несмотря на быстроту, ЦПУ чисто ожидает подсистему ввода/вывода [4]. Существует несколько способов увеличения количества оперативной памяти. Один из способов заключается в физическом увеличении количества планок ОЗУ в системе. Данный способ подразумевает покупку и установку планок ОЗУ, что требует денежных затрат. Кроме физического способа увеличения количества памяти, существуют программные способы увеличения количества ОЗУ, например, сжатие данных. Данный способ требует только вычислительные мощности ЦПУ [2], а как было указано ранее, ЦПУ часто простаивает в ожидании операций ввода/вывода. Данный факт позволяет направить простаивающие вычислительные мощности ЦПУ на обработку операций сжатия оперативной памяти. Целью данной работы является разработка оптимизации метода сжатия страниц оперативной памяти в ядре Linux.

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

Для достижения поставленной цели необходимо решить следующие задачи:

- рассмотреть методы увеличения количества оперативной памяти;
- дать характеристику архитектурам ядер современных операционных систем;
- изучить подходы, структуры данных и API [3] в ядре Linux, позволяющие

управлять оперативной памятью;

- описать работу модуля сжатия оперативной памяти в ядре Linux;
- разработать оптимизацию для данного модуля;
- спроектировать структуру программного обеспечения, реализующего оптимизацию модуля сжатия оперативной памяти;
- разработать программное обеспечение для данной оптимизации;
- проверить работоспособность разработанного ПО.

1 Аналитическая часть

В данном разделе производится анализ предметной области. Даётся характеристика архитектурам современных ядрер операционных систем. Проводится краткий обзор ядра Linux, структур данных и его API. Описывается работа модуля ядра zram [17], позволяющего хранить страницы виртуальной памяти в сжатом виде оперативной памяти.

1.1 Постановка задачи

Модуль ядра zram хранит оперативную память в сжатом в виде. При каждом обращении системы к участкам оперативной памяти, происходит преобразование данных: сжатие, при попытке записи данных в оперативную память, или возвращение к исходному (из сжатого) виду, при попытке чтения данных. Для проведения любого из этих преобразований центральному процессорному устройству необходимо затратить какое-то количество времени. При проведении сжатия, возможно вариант когда размер выходных (сжатых) данных не изменился или остался исходным [10]: процессорное время потрачено, а выигрыш в размере сжатых данных оказался минимален. Не всегда процент сжатия данных прямо пропорционален машинному времени, потраченному на эти преобразования [10].

Таким образом, оптимизация метода сжатия страниц оперативной памяти должна определять (непосредственно до момента сжатия), соответствуют ли затраты процессорного времени, потраченные на сжатие данных, выигрышу в размере сжатых данных. Разработанный метод должен заранее определять, нужно ли сжимать страницу или хранить ее в несжатом виде.

1.2 ЦПУ и оперативная память

Оперативная память ([5]) – 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 памя-

ти во время работы компьютера хранится выполняемый машинный код и данные, обрабатываемые центральным процессорным устройством (англ. central processing unit, CPU [6]).

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

- электроэнергия – каждая новая установленная планка увеличивает потребление энергии компьютера [7];
- физические ограничения – количество слотов для планок на материнской плате ограничено;
- стоимость – каждая планка стоит денег.

Альтернативным решением является использование системы подкачки страниц (англ. paging [8]) – специальный механизм операционный системы, при котором отдельные фрагменты памяти (мало используемые или полностью не активные) перемещаются из оперативной памяти во вторичное 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 [9].

Ещё одним решением является сжатие данных прямо в оперативной памяти. Данный подход не имеет минусов выделенных у первого решения. Он реализуется программно и не требует закупки дополнительных планок ОЗУ. Кроме того, благодаря тому что сжатие не требует работы с внешним накопителем, скорость обработки данных увеличивается что позволяет избавиться от минуса (время работы) выделенного у второго решения. 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 отмечено во введении, чаще всего ЦПУ простаивает и не выполняет никакой работы, ожидая подсистему ввода/вывода.

1.3 Сжатие данных

Сжатие данных (англ. data compression [2]) – это способ (алгоритмический) преобразования информации в другую форму, чаще всего более компактную [10]. 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 [11]. В данной работе будут рассмотрены только алгоритмы сжатия данных без потерь: при работе с оперативной память терять данные непозволительно. На рисунке 1 представлена концептуальная модель сжатия данных без потерь.

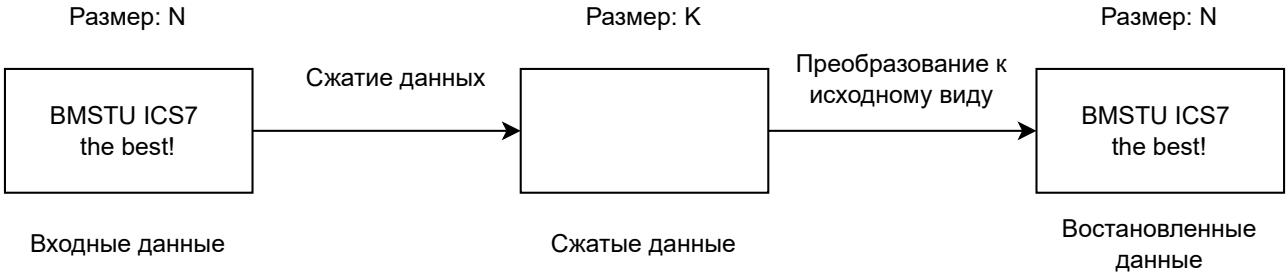

Рисунок 1 – Схема сжатия данных без потерь

1.3.1 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.

1.3.2 Требования к алгоритмам сжатия

К алгоритмам сжатия могут применяться несколько требований:

- скорость преобразования к сжатому виду;
- скорость преобразования к исходному виду;
- степень сжатия;

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.

1.4 Алгоритмы сжатия

Главный принцип алгоритмов сжатия данных алгоритмов берет во внимание то, что в любом наборе (не случайном) данных, информация частично (либо,в редких случаях, полностью) повторяется [11]. С помощью математических методов можно определить вероятность повторения определенной комбинации байт. После этого можно создать создать некоторые коды, которые будут соответствовать наиболее распространяющимся наборам байт. Данные коды можно создать различными способами: энтропийное кодирование, кодирование повто-

ров и сжатие с помощью словаря. С помощью указанных подходов возможно устранить дублирующую информацию из файла и уменьшить его итоговый размер (в сжатом виде).

1.4.1 Энтропийное кодирование

Данный подход основывается на предположении о том, что до кодирования отдельные элементы последовательности данных имеют различную вероятность появления. После преобразований (кодирования) в результирующей последовательности вероятности появления отдельных элементов практически одинаковы [11]. Таким образом, энтропийное кодирование усредняет вероятности появления элементов в закодированное последовательности байт.

1.4.2 Кодирование повторов

Кодирование повторов подразумевает замену повторяющихся символов (серии) на один символ и число его повторов. Серия – последовательность данных, состоящих из одинаковых символов. Так, например, строка
`a = IIIIIUUSEEVEEENNNNNNNN` может быть преобразована к виду
`b = 5I2U1S2E1V3E8N`. В таком случае, коэффициент сжатия k будет равен $k = \frac{len(a)}{len(b)} = \frac{22}{14} = 1.57$

Такое кодирование эффективно для наборов данных, содержащих большое количество серий – например, простых графических изображений.

1.4.3 Сжатие с помощью словаря

При использовании сжатия данных с помощью словаря данные делятся на слова, а сами слова в исходном наборе данных заменяются на их индексы в словаре. Чаще всего, словарь пополняется словами из исходной последовательности данных в процессе сжатия.

Ключевым параметром такого метода сжатия является размер словаря. С одной стороны, чем больше его размер, тем больше эффективность сжатия. Однако, для неоднородного набора данных большой размер словаря может оказаться вреден. Например, при резком изменении типа данных большая часть словаря будет заполнена неактуальными словами. Кроме того, чем больше раз-

мер словаря, тем больше нужно дополнительной памяти для хранения слов в нём.

Благодаря тому, что алгоритмическая сложность доступа к элементам словаря константа, алгоритмы сжатия с использованием словаря приводят данные к исходному виду быстрее, чем алгоритмы с использованием подходов, которые были описаны выше [11].

1.5 Ядра операционных систем

Ядро операционной системы – это программное обеспечение, которое предоставляет базовый функционал для всех остальных частей операционной системы, управляет аппаратным обеспечением и распределяет системные ресурсы [12]. Ядра операционных систем можно разделить на две группы: с монолитным 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 [12]. Из-за отсутствия издержек, таких как синхронизация процессов и передача данных между ними, монолитные ядра операционных систем являются более производительным, чем микроядра. На рисунке 2 представлена концептуальная модель операционной системы с монолитном ядром.

Реализация микроядра подразумевает отказ от одного большого процесса. Все функции ядра разделяются на несколько процессов, которые обычно называют серверами [12]. При этом, в привилегированном режиме могут работать лишь те сервера (процессы), которым этот режим необходим, а остальные сервера 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

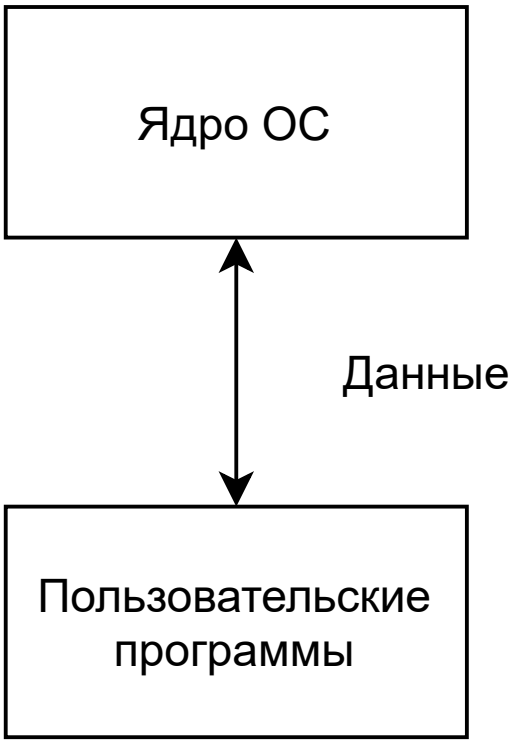

Рисунок 2 – Схема ОС с монолитным ядром

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 [13]), 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 3 представлена концептуальная схема операционной системы с монолитном ядром.

1.6 Ядро Linux

Ядро Linux [14] – 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 являет-

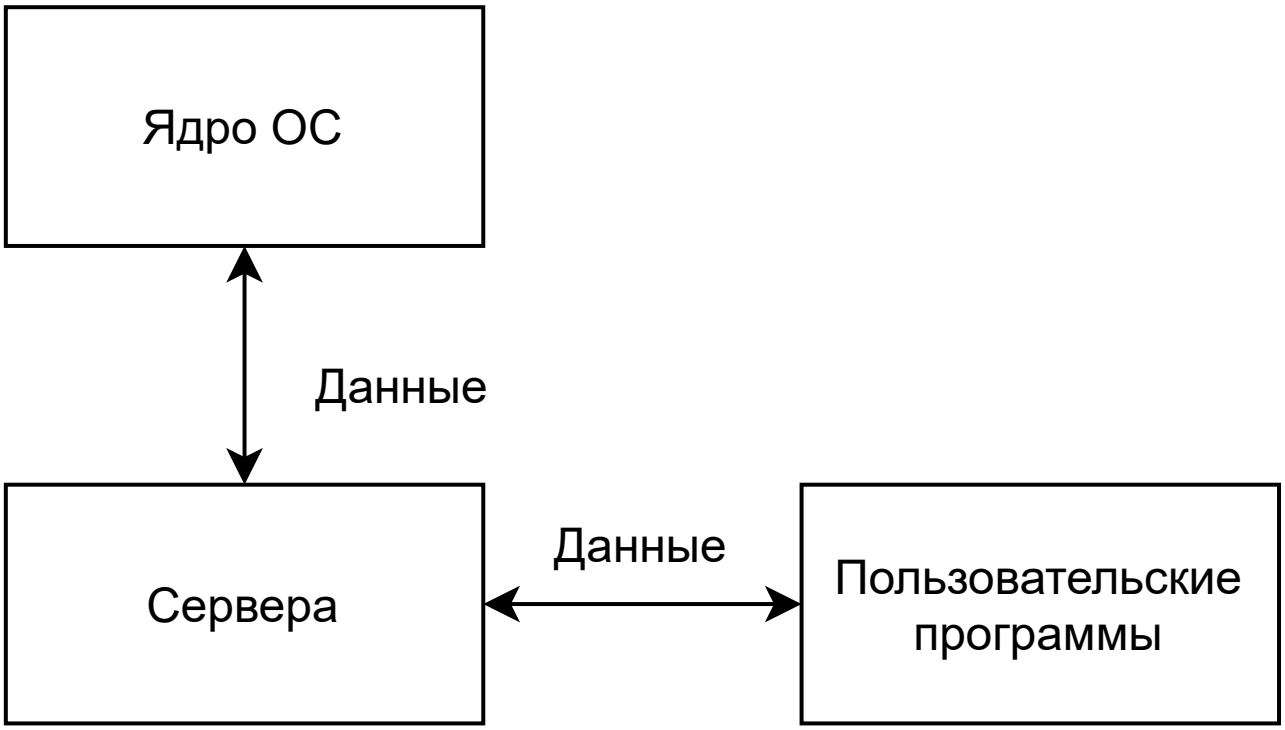

Рисунок 3 – Схема ОС с микроядром

ся 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) [12]. 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 разработ-

ки [12].

1.6.1 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 интерфейс, скрывающий интерфейс физических устройств;
- сетевые интерфейсы – обеспечивают работу с сетевым оборудованием;
- межпроцессная подсистема – механизмы межпроцессного взаимодействия.

1.6.2 Страничная организация памяти

Работа с памятью 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 [12]. Размер страницы это фиксированная константа `PAGE_SIZE`, которая на большинстве современных архитектур, поддерживаемых Linux, составляет 4 Кб [16]. Страничная организация памяти позволяет реализовать внутри ядра Linux механизм виртуальной памяти.

1.6.3 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.

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

Листинг 1: Структура `struct page`

```
struct page {
    unsigned long flags;
    union {
        struct {
            struct list_head lru;
            struct address_space *mapping;
            pgoff_t index;
            unsigned long private;
        };
    };

    union {
        atomic_t _mapcount;
        unsigned int page_type;
        unsigned int active;
        int units;
    };

    atomic_t _refcount;

#if defined(WANT_PAGE_VIRTUAL)
    void *virtual;
#endif
    int _last_cpupid;
} _struct_page_alignment;
```

Ниже представлено описание наиболее важных полей данной структуры:

- `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;
- `_last_cpupid` – номер последнего ЦПУ, использовавшего данную страницу.

Рассматриваемая структура данных в ядре используется для учёта всей физической памяти.

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. Функция 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 4 представлена схема управления страницами памяти в ядре Linux.

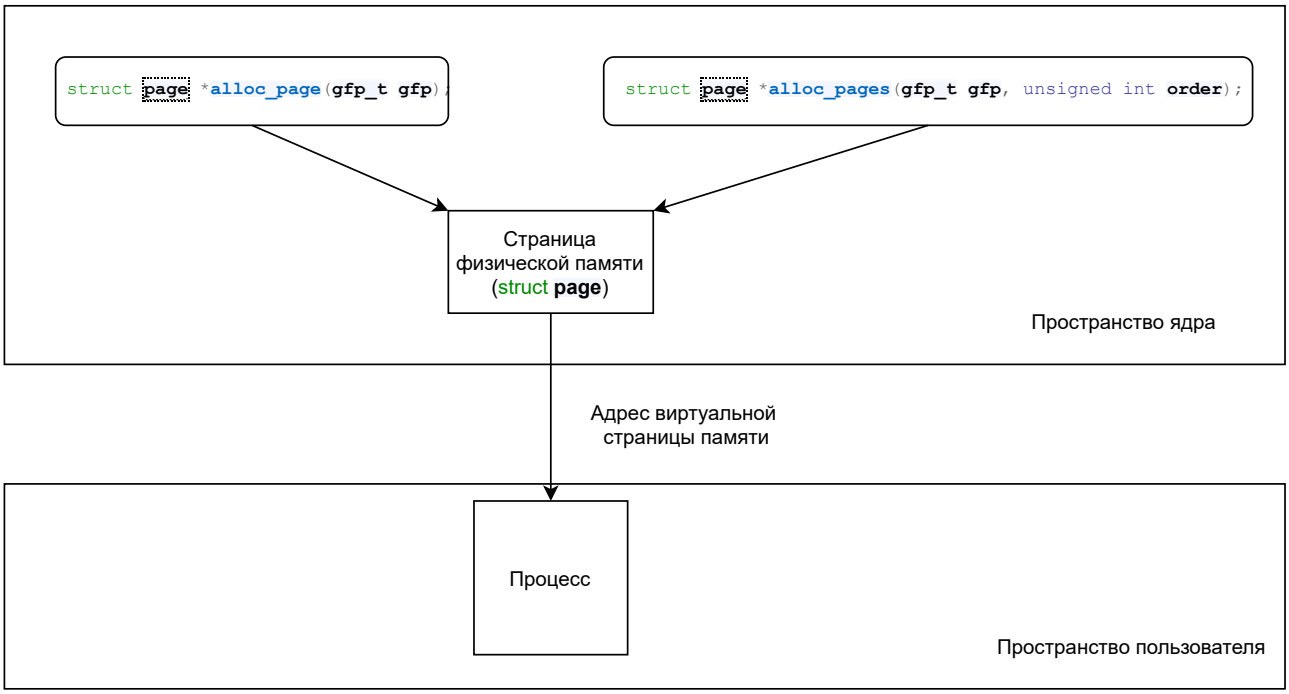

Рисунок 4 – Схема управления памятью

1.6.4 Подкачка страниц

Подкачка страниц – специальный механизм внутри ядра Linux, который реализован благодаря механизму виртуальной памяти, при котором отдельные страницы оперативной памяти записываются не в ОЗУ, а, чаще всего, в специальное вторичное хранилище, например, жесткий диск. Ядро само выбирает, какие страницы памяти попадут в это хранилище. Чаще всего, это неактивные или реже всего использующиеся страницы памяти.

Выгруженные из оперативной памяти страницы могут храниться как в специальном разделе жесткого диска, так и в файле. Пространство, где хранятся эти страницы, называется swap-пространством. При попытке обратиться к выгруженной странице, в ядре происходит исключительная ситуация, называемая page fault [21]. Обработчик прерываний обрабатывает данный запрос и перемещает страницу обратно в оперативную память. На рисунке **??** представлена схема работы подсистемы подкачки страниц в ядре Linux.

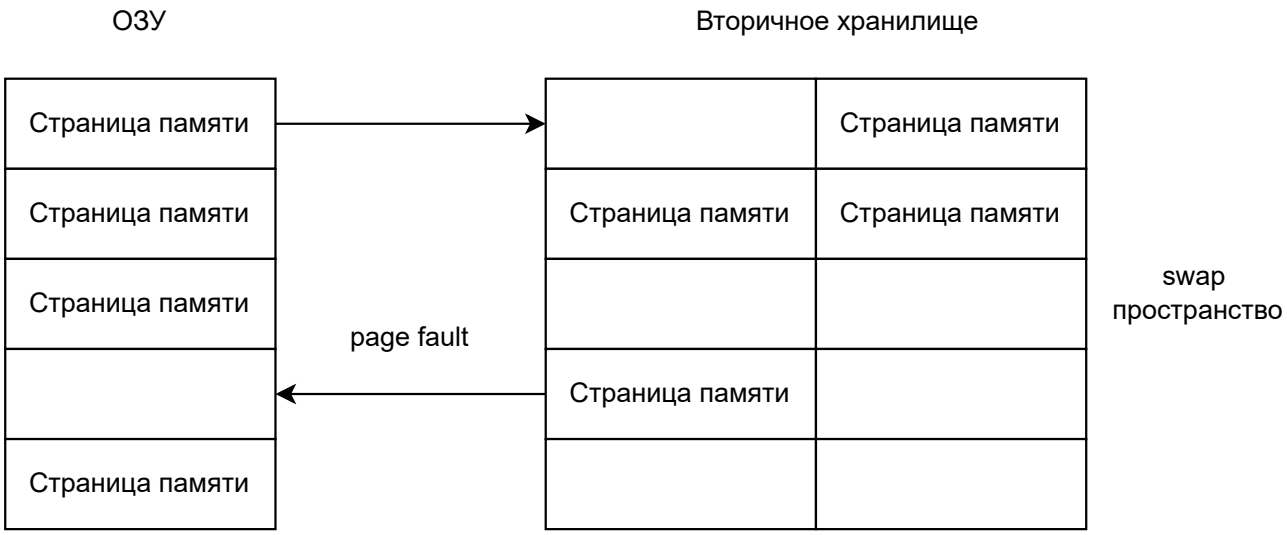

Рисунок 5 – Подсистема подкачки страниц

1.7 Модуль ядра zram

Модуль ядра zram создает в оперативной памяти специальное блочное устройство. Такое устройство можно создать из пространства пользователя. Страницы памяти, попадающие в это устройство, сжимаются и хранятся прямо в оперативной памяти. Такой подход обеспечивает экономию памяти и быстрый ввод/вывод [17].

Чаще всего, созданное блочное устройство zram устанавливают как в качестве подсистемы подкачки. Механизм swap отправляет страницу в zram через подсистему блочного ввода-вывода. Сжатая страница внутри zram идентифицируется уникальным ключом $zkey$ (2):

$$zkey = device\_id + offset \tag{2}$$

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 (когда происходит прерывание page fault), устройство zram уведомляется, о том, что необходимо привести запрашиваемую страницу с идентификатор $zkey$ преобразовать к исход-

ному виду и вернуть её системе. Пока данные находятся в сжатом состоянии, система не может прочитать или записать какие-либо отдельные байты из этой сжатой последовательности. Современные алгоритмы могут сжимать любое количество последовательных байт. Несмотря на это, модуль ядра zram использует работает лишь с виртуальными страницами памяти.

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. В модуле ядре zram существует специальное API, доступное из пространства пользователя, для выбора алгоритма сжатия. Данный модуль поддерживает несколько алгоритмов сжатия, например DEFLATE [18], LZ4 [19], zstd [20].

Из-за того что размер страницы достаточно большой (4 Кб), сжатие и восстановление данных – 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 [4]. 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. Степень сжатия страницы описывается следующей формулой (3):

$$k = \frac{PAGE\_SIZE}{zsize}, \tag{3}$$

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

Чаще всего, размер сжатой страницы меньше чем PAGE_SIZE, поэтому, обычно, коэффициент сжатия меньше единицы. Но, в некоторых случаях, коэффициент сжатия может быть больше единицы. В среднем страница памяти в ядре сжимается в два раза [4], то есть коэффициент сжатия $k = \frac{1}{2}$.

Неудачная попытка записать блок в подсистему блочного вывода приводит к большим накладным расходам [4]. 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 с помощью 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;
```

```
17      u64 bd_wb_limit;
18      struct block_device *bdev;
19      unsigned long *bitmap;
20      unsigned long nr_pages;
21  #endif
22  #ifdef CONFIG_ZRAM_MEMORY_TRACKING
23      struct dentry *debugfs_dir;
24  #endif
25  };
```

Данная структура хранит в себе массив сжатых страниц, семафор, с помощью которого достигается синхронизация при параллельной обработке [4], название алгоритма, который будет производить сжатие, размер блочного устройства, статистику по проведенным операциям и другие поля.

Для хранения информации о сжатых страницах (их размер и данные) используется массив структур. Это необходимо, для того чтобы найти эту страницу для её дальнейшего преобразования к исходному (не сжатому) виду. Элементом массива является 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 корректной параллельной обработки страниц используется семафор `struct rw_semaphore`.

zram устраняет дубликаты страниц, которые состоят из одного и того же элемента. В таком случае, сохраняется лишь один этот элемент. Несмотря на то, чтобы определить, состоит ли страница полностью из одинаковых байт, требуются накладные расходы, эти расходы чаще всего небольшие, по сравнению с затратами на сжатие [4].

На рисунке 6 представлена модель взаимодействия модуля zram, ядра и пользователя.

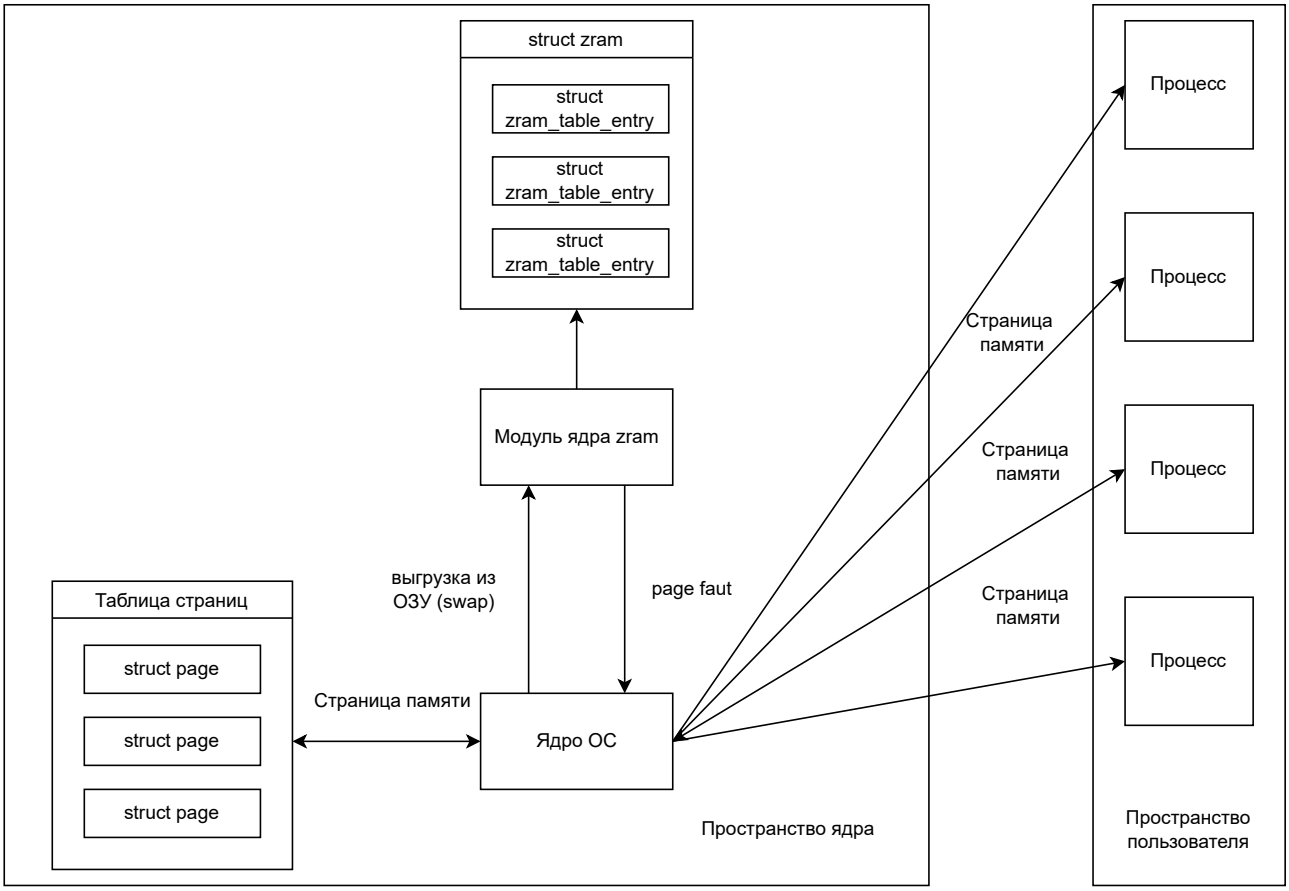

Рисунок 6 – Модель взаимодействия zram, ядра и пользователя

1.7.1 Опция writeback

Модуль ядра zram предоставляет возможно опционально включить специальную опцию writeback, которая позволяет сохранять неиспользуемые или несжимаемые страницы во вторичное хранилище, а не в оперативную память. Для реализации данного функционала структура `struct zram` хранит в себе указатель на структуру `struct block_device` (см. листинг 4). Структура `struct block_device` представлена на листинге 6.

Данная структура является описанием некоторого блочного устройства внутри ядра Linux. В случае модуля zram, структура `struct block_device` описывает вторичное хранилище, в которое будут отправляться несжимаемые или неиспользуемые страницы оперативной памяти при включенной опции writeback.

Листинг 6: Структура `struct block_device`

```
struct block_device {
    sector_t bd_start_sect;
    sector_t bd_nr_sectors;
    struct disk_stats __percpu *bd_stats;
    unsigned long bd_stamp;
    bool bd_read_only;
    dev_t bd_dev;
    int bd_openers;
    struct inode *bd_inode;
    struct super_block *bd_super;
    void *bd_claiming;
    struct device bd_device;
    void *bd_holder;
    int bd_holders;
    bool bd_write_holder;
    struct kobject *bd_holder_dir;
    u8 bd_partno;
```

```
18          spinlock_t bd_size_lock
19          struct gendisk *bd_disk;
20          struct request_queue *bd_queue;
21
22          int bd_fsfreeze_count;
23          struct mutex bd_fsfreeze_mutex;
24          struct super_block *bd_fsfreeze_sb;
25
26          struct partition_meta_info *bd_meta_info;
27      } __randomize_layout;
```

1.7.2 Схема работы

1.7.3 Структуры данных

1.8 Вывод

В данном разделе был проведен анализ предметной области.

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 API управления 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. Data compression | computing - Encyclopedia Britannica [Электронный ресурс]. – Режим доступа: https://www.britannica.com/technology/data-compression, свободный – (24.03.2021)

3. API | computer programming - Encyclopedia Britannica [Электронный ресурс]. – Режим доступа: https://www.britannica.com/technology/API, свободный – (24.03.2021)

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

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

6. An Introduction to Information Processing, 1986. с. 45 - 71.

7. 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)

8. Theoretical Computer Science, 15 July 1997. Esteban Feuerstein. с. 75 - 90.

9. 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)

10. Encyclopedia of Information Systems, 2002. Khalid Sayood. с. 5 - 27.

11. Industrial Control Technology. Peng Zhang, 2008. с. 675 - 774.

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

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

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

15. 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)

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

17. 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)

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

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

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

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