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 _____ _____
(Подпись, дата) (И.О. Фамилия)

***2022** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 60 с., 11 рис., 4 табл., 37 ист.

Объектом исследования данной работы является сжатие страниц оперативной памяти в ядре Linux. Сжатие данных, коэффициент сжатия которых будет приближен к единице, практически бесполезно. Другими словами, попытки сжать данные, которые сжимаются плохо, или не сжимаются вообще, лишь тратят машинное время и практически не дают никакого преимущества. Целью этой работы является исследование метода сжатия страниц оперативной памяти в ядре Linux и разработка оптимизации для данного метода.

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 подсистемой памяти;
— описать работу модуля сжатия оперативной памяти в ядре Linux;
— разработать оптимизацию для данного модуля;
— спроектировать структуру программного обеспечения, реализующего оптимизацию модуля сжатия оперативной памяти;
— сравнить метод сжатия страниц оперативной памяти с оптимизацией и без.

Поставленная цель достигнута: в ходе дипломной работы был разработана оптимизация метода сжатия страниц оперативной памяти в ядре Linux. Разработанная оптимизация повышает скорость работы метода сжатия страниц: это достигается благодаря незначительным жертвам в размере сжатых данных.

КЛЮЧЕВЫЕ СЛОВА

сжатие данных, оперативная память, linux, zram, операционные систе-

мы.

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.
3) Сжатие данных – это способ (алгоритмический) преобразования информации в другую форму, чаще всего более компактную.

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 исследование метода сжатия страниц оперативной памяти в ядре Linux и разработка его оптимизации.

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 подсистемой памяти;
- описать работу модуля сжатия оперативной памяти в ядре Linux;

- разработать оптимизацию для данного модуля;
- спроектировать структуру программного обеспечения, реализующего оптимизацию модуля сжатия оперативной памяти;
- сравнить метод сжатия страниц оперативной памяти с оптимизацией и без.

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 [20], 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.

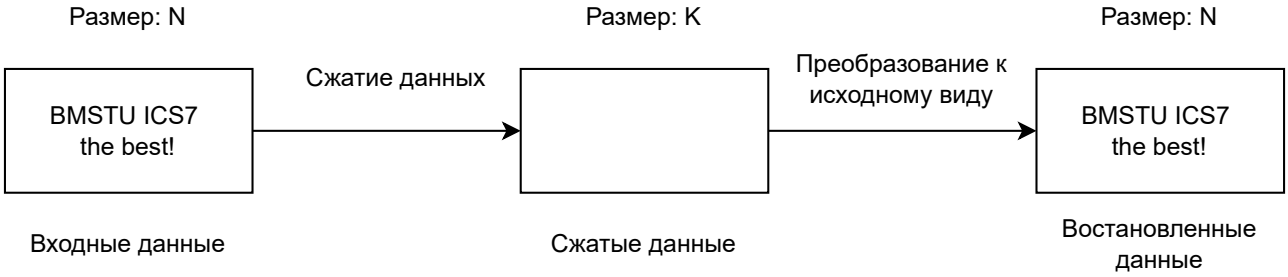

Рисунок 1 – Схема сжатия данных без потерь

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 Информационная энтропия

Информационная энтропия – мера неупорядоченности или неопределенности состояния некоторой системы, описываемой данными. Вычисляется по формуле Шеннона (2):

$$H(x) = -K \sum_{i=1}^{n} p(x_i) log_2 p(x_i), \tag{2}$$

где $p(x_i)$ – вероятность i-го состояния системы (значения принимаемого переменной), n – число состояний системы (значений, принимаемых переменной), K – положительная константа. Эта величина также называется средней энтропией сообщения.

Величина, представленная на формуле 3, называется частной энтропией и характеризует только i-е состояние системы.

$$H(x) = -log_2(p_i). \tag{3}$$

Функция 2 – единственная функция, которая удовлетворяет ряду требований, необходимые для измерения энтропии:

1) $H(p_1, ..., p_n)$ определена и непрерывна для всех $p1, ..., p_n$, где $p_i \in [0, 1]$ для всех $i = 1, ..., n$ и $p_1 + ... + p_n = 1$. Таким образом, функция зависит только от распределения вероятностей, но не от алфавита;
2) для целых положительных n, должно выполняться неравенство 4:

$$H(\frac{1}{n}, ..., \frac{1}{n}) < H(\frac{1}{n+1}, ..., \frac{1}{n+1}); \tag{4}$$

3) для целых положительных b_i, где $b_1 + ... + b_k = n$, должно выполняться равенство 5:

$$H(\frac{1}{n}, ..., \frac{1}{n}) = H(\frac{b_1}{n}, ..., \frac{b_k}{n}) + \sum_{i=1}^{k} \frac{b_i}{n} H(\frac{1}{b_i}, ..., \frac{1}{b_i}) \quad (5)$$

Энтропия измеряется в битах, натах, тритах или хартли, в зависимости от основания логарифма, использующегося при вычислении энтропии. Логарифм используется в связи с тем, что он аддитивен для независимых источников. Так, например, энтропия события – броска монеты, равна 1 биту (формула 6), а энтропия k бросков составит k бит.

$$H(x) = -2(\frac{1}{2} log_2 \frac{1}{2}) = -log_2 \frac{1}{2} = log_2 2 = 1. \quad (6)$$

В обычном представлении $log_2 n$ бит требуется для представления переменной, принимающей n значений, если n является степенью 2. Если значения равновероятны, то энтропия в битах будет равна значению n.

1.5.1 Бинарная энтропия

Бинарная энтропия – частный случай информационной энтропии, различие составляет лишь в том, что x (формула 3) может принимать только два значения: 0 или 1. Определяется как энтропия процесса Бернулли [36] с вероятностью p одного одного из двух значений. Рассчитывается по формуле Хартли (7):

$$i = log_2 N, \quad (7)$$

где N – мощность алфавита, i – количество информации в каждом символе сообщения. Для случайно величины x, принимающей n независимых случайных значений x_i с вероятностями $p_i (i = 1, ..., n)$ формула Хартли переходи в формулу Шеннона (2).

На рисунке 2 представлена функция бинарной энтропии.

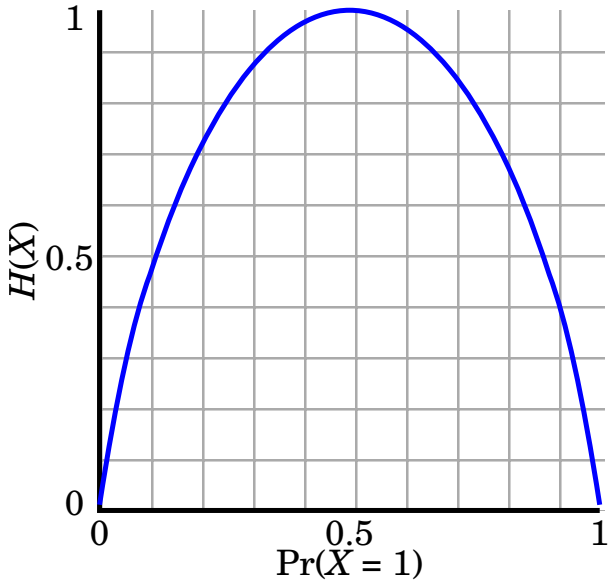

Рисунок 2 – Функция бинарной энтропии

1.5.2 Энтропия и сжатие данных

Энтропия набора данных является мерой количества содержащейся в нем информации [12]. Информационная энтропия, подсчитанная для данных, размер которых известен, можно использовать чтобы получить теоретическую границу того, насколько эти данные могут быть сжаты [12]. Таким образом, рассчитав энтропию для некоторых данных, можно предсказать их примерный размер после проведения преобразования сжатия.

Для вычисления энтропии набора данных, в среднем, требуется в несколько раз меньше машинного времени, чем для сжатия этого набора данных [12]. Этот факт можно использовать для ускорения процесса сжатия, если алгоритм сжатия вызывается повторно. Например если данные, которые нужно сжать, разбиты на участки, что может быть актуально при параллельной обработке данных. Можно считать энтропию для каждого такого участка, и на основе полученного значения принимать решение, сжимать этот участок или хранить в несжатом виде. В таком случае, время сжатия увеличиться, но при этом итоговый коэффициент сжатия данных будет уменьшен. Можно сделать вывод, что чем больше таких участков и чем больше энтропия сжимаемых участков, тем

больше будет выигрыш во времени и меньше проигрыш в сжатии данных.

1.6 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 [13]. 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 [13]. Из-за отсутствия издержек, таких как синхронизация процессов и передача данных между ними, монолитные ядра операционных систем являются более производительным, чем микроядра. На рисунке 3 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 [13]. 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 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 [14]), а обеспечивает передачу данных само ядро (больше никаких действий в системе оно не вы-

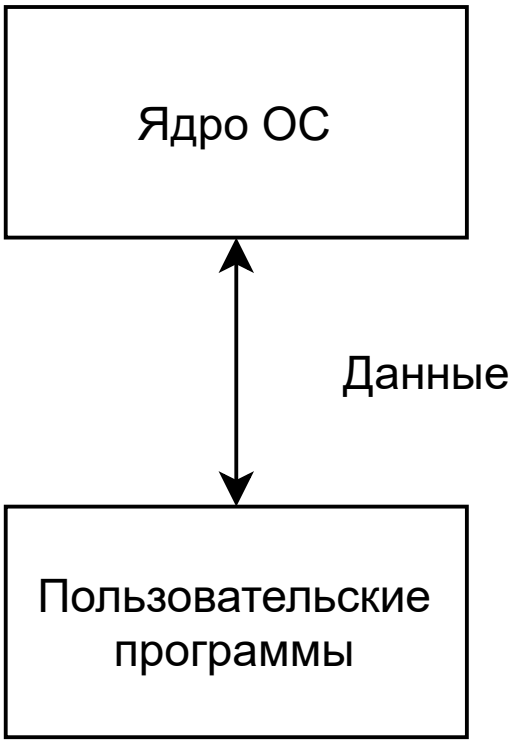

Рисунок 3 – Схема ОС с монолитным ядром

полняет). 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 4 представлена концептуальная схема операционной системы с монолитном ядром.

1.7 Ядро Linux

Ядро Linux [15] – 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) [13]. Linux не использует никаких функций микроядерной модели (IPC),

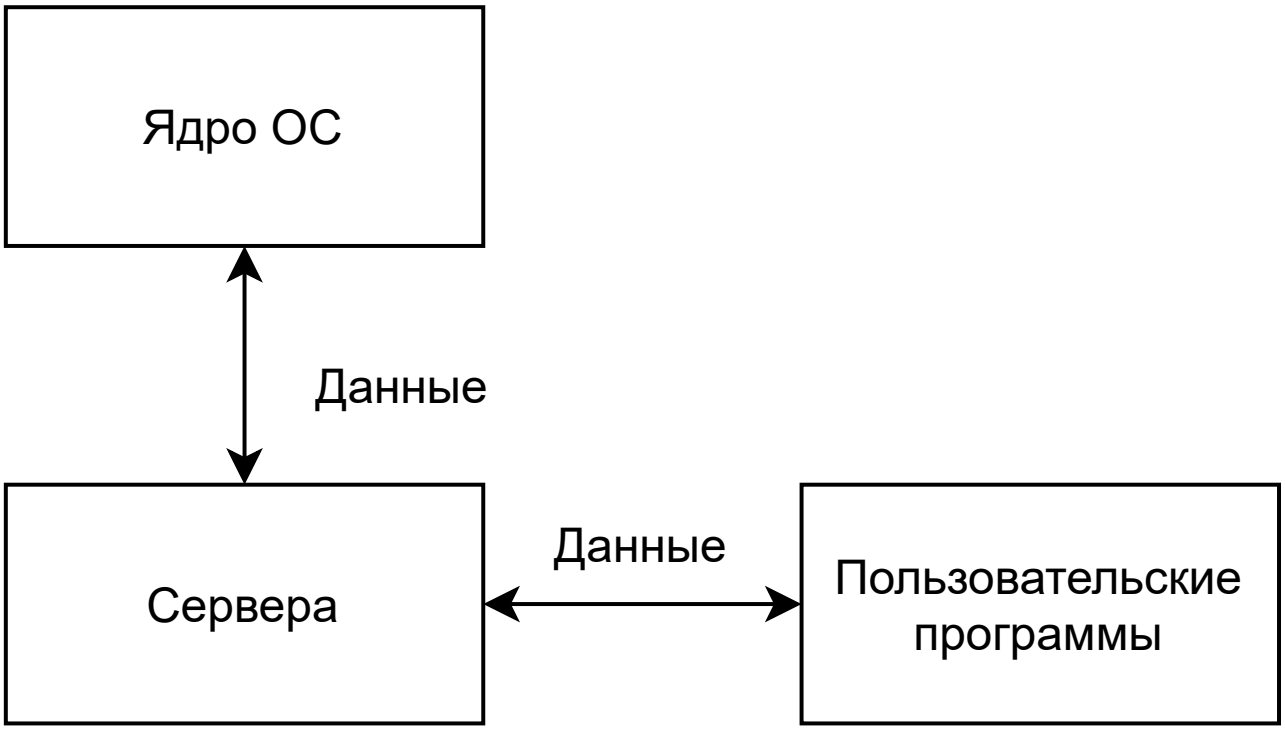

Рисунок 4 – Схема ОС с микроядром

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 [13].

1.7.1 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.

1.7.2 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 [13]. Размер страницы это фиксированная константа `PAGE_SIZE`, которая на большинстве современных архитектур, поддерживаемых Linux, составляет 4 Кб [17]. Страничная организация памяти позволяет реализовать внутри ядра Linux механизм виртуальной памяти.

1.7.3 Виртуальная память

Виртуальная память – специальный механизм организации памяти, при котором процессы работают не с 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 5 представлена схема управления страницами памяти в ядре Linux.

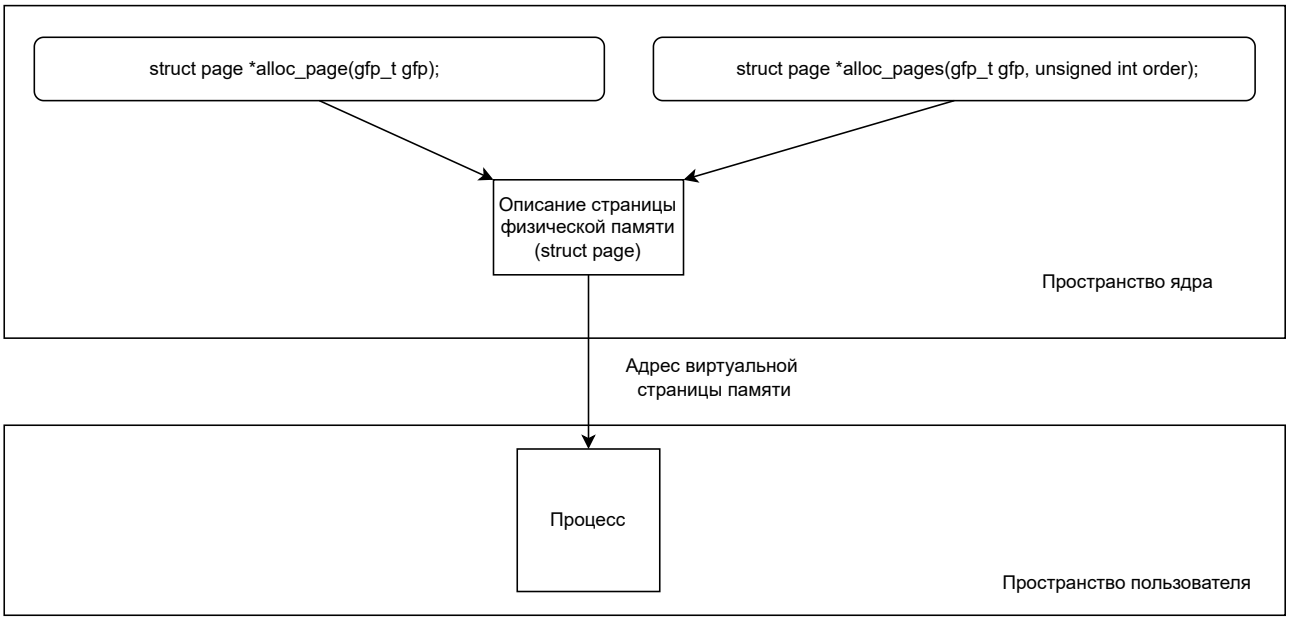

Рисунок 5 – Схема управления памятью

1.7.4 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 [28]. Обработчик прерываний обрабатывает данный запрос и перемещает страницу обратно в оперативную память. На рисунке 6 представлена схема работы подсистемы подкачки страниц в ядре Linux.

1.7.5 Блочные устройства

Блочное устройство – специальный интерфейс в ядре Linux, обеспечивающий доступ к реальному или виртуальному устройству [18]. Представляет

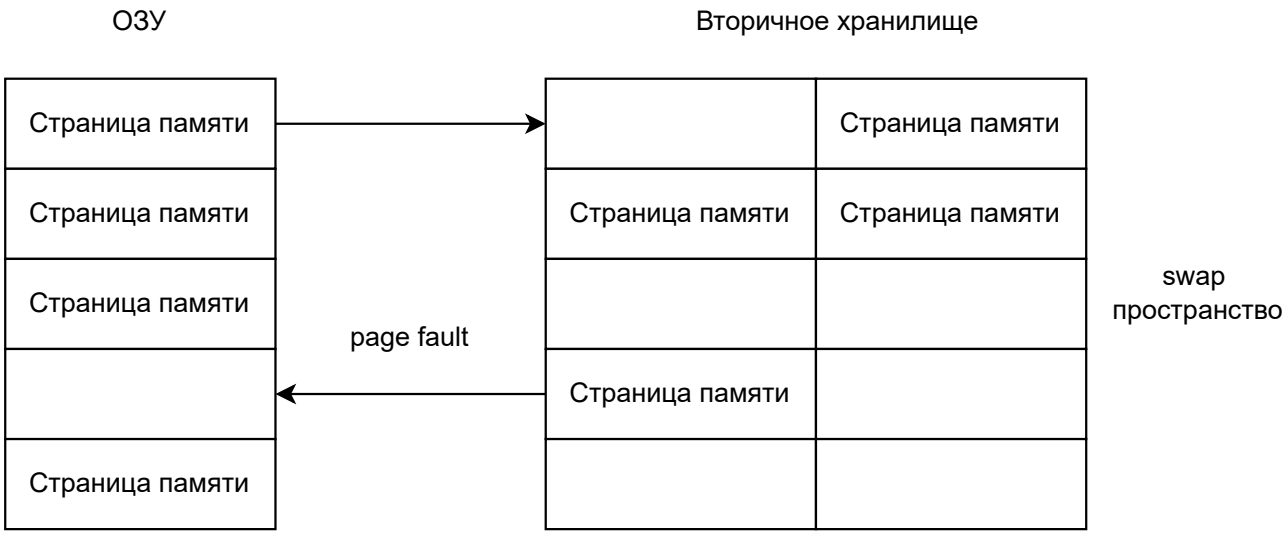

Рисунок 6 – Подсистема подкачки страниц

из себя файл в файловой системе. Блочные устройства характеризуются случайным доступом к данным, организованным в блоки фиксированного размера. Примерами таких устройств являются жестки диски, приводы CD-ROM, RAM-диски и т.д. Скорость блочных устройств, как правило, намного выше, чем скорость символьных устройств [19]. В символьных устройствах данные обрабатываются последовательно, без возможности произвольного доступа и по одному символу. Ядро Linux предоставляет специальное API для работы с блочными устройствами.

Работать с блочными устройствами сложнее, чем с символьными. Символьные устройства имеют одну текущую позицию, в то время как блочные устройства должны иметь возможность перемещаться на любую позицию в устройстве, для того чтобы обеспечить произвольный доступ к данным. Для упрощения работы с блочными устройствами ядро Linux представляет подсистему называемую подсистемой блочного ввода-вывода.

С точки зрения ядра наименьшей логической единицей адресацией является блок. Хотя физическое устройство может быть адресовано на уровне сектора, ядро выполняет все дисковые операции с использованием блоков. Поскольку наименьшей единицей физической адресации является сектор, размер

блока должен быть кратен размеру сектора. Кроме того, размер блока должен выполнять требование 8 (размер должен быть кратен цифре два, возведенную в любую степень) и не может превышать размер страницы памяти (PAGE_SIZE).

$$block\_size \; mod \; 2^N = 0, N \in [0, \text{inf}] \tag{8}$$

Размер блока может варьироваться в зависимости от используемой файловой системы, наиболее распространенными значениями являются 512 байт, 1 килобайт или 4 килобайта [18].

На рисунке 7 представлена схема взаимодействия блочного и физического устройства.

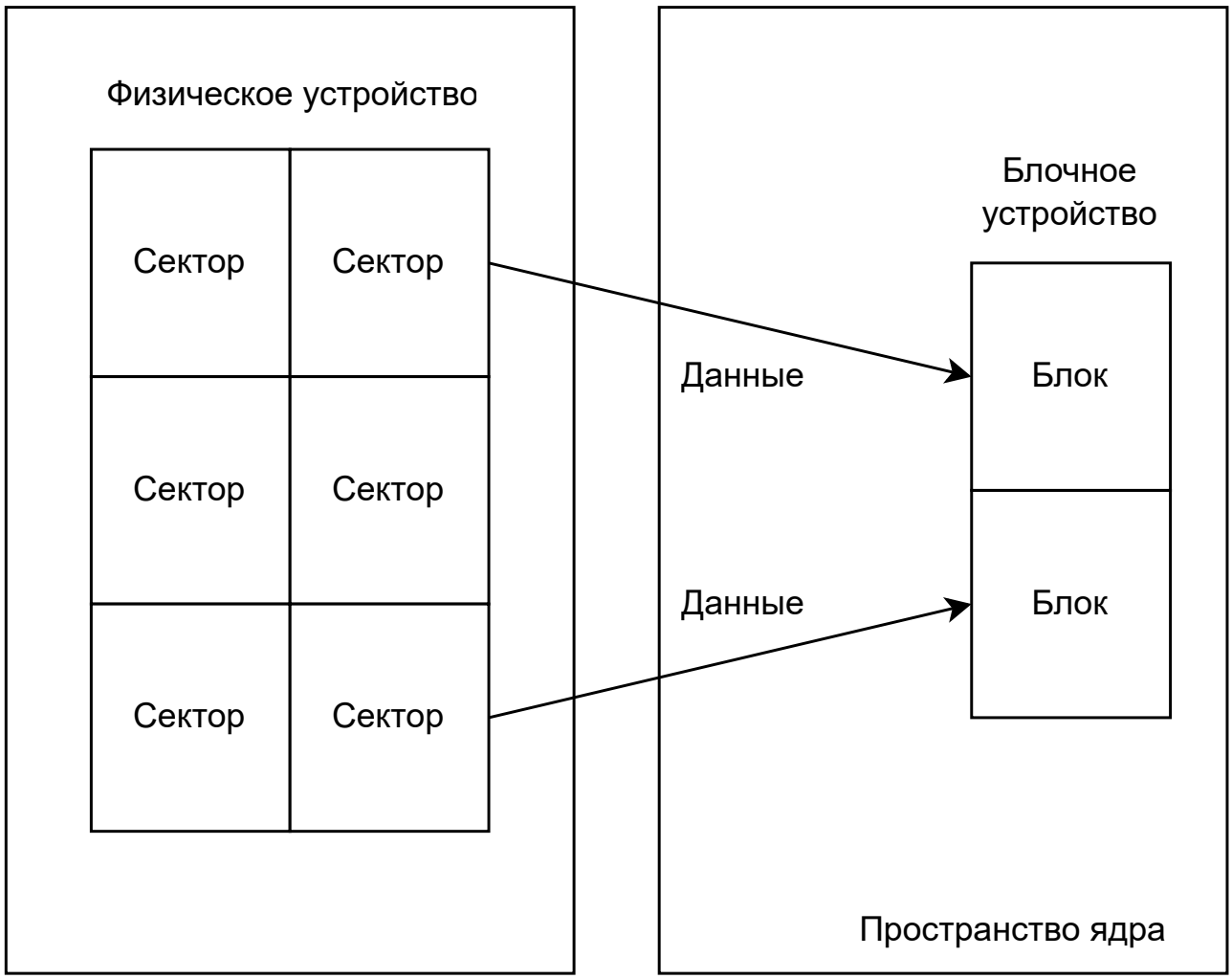

Рисунок 7 – Схема взаимодействия блочного и физического устройства

1.8 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, по сравнению с использованием дисковых хранилищ [20].

Созданное блочное устройство zram можно установить в качестве системы подкачки (swap) или в качестве в блочного устройства для файловых хранилищ, хранящихся в оперативной памяти. Примером такого хранилища является tmpfs [21].

Но, чаще всего, zram используется в паре с swap. Механизм swap отправляет страницу в zram через подсистему блочного ввода-вывода. Сжатая страница внутри zram идентифицируется уникальным ключом $zkey$ (9):

$$zkey = device\_id + offset \tag{9}$$

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 исходному 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 выпол-

няться 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 [22], LZ4 [23], LZO [24], LZO-RLE[25] 842 [26], zstd [27].

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 (10):

$$k = \frac{PAGE\_SIZE}{zsize}, \tag{10}$$

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.

В ядре модуль 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;
    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
```

```
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
10  }
```

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 8 представлена модель взаимодействия модуля zram, ядра и пользователя.

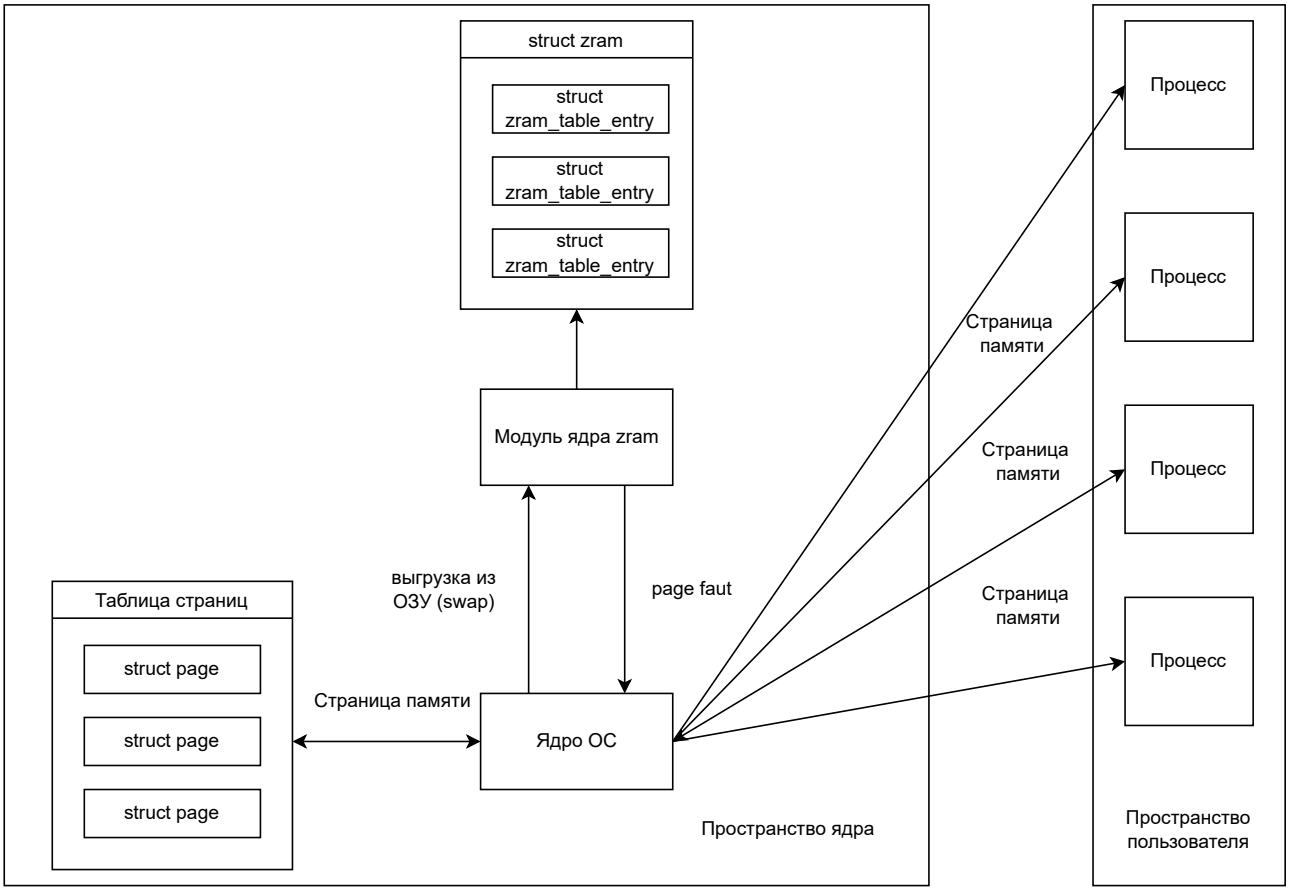

Рисунок 8 – Модель взаимодействия zram, ядра и пользователя

1.8.1 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;
    spinlock_t bd_size_lock
    struct gendisk *bd_disk;
    struct request_queue *bd_queue;

    int bd_fsfreeze_count;
    struct mutex bd_fsfreeze_mutex;
    struct super_block *bd_fsfreeze_sb;

```

```
26          struct partition_meta_info *bd_meta_info;
27      } __randomize_layout;
```

1.9 Вывод

В данном разделе был проведен анализ предметной области: были рассмотрены понятия сжатия данных, оперативной памяти и информационной энтропии. Была дана характеристика алгоритмам сжатия.

Были характеризованы современные ядра операционных систем: с монолитной и микроядерной архитектурой. Проведён краткий обзор ядра Linux, структур данных и API управления подсистемой памяти внутри ядра.

Была описана работа модуля ядра zram, позволяющего хранить страницы виртуальной памяти в сжатом виде оперативной памяти: приведен обзор используемых структур и схему взаимодействия zram, ядра и пользователя.

2 Конструкторская часть

В данном разделе переставлена архитектура программного обеспечения (модуля ядра zram). Приведен алгоритм вычисления информационной энтропии для страниц оперативной памяти и алгоритм записи страниц в блочном устройстве.

Алгоритмы рассчитаны на организацию кода в блоки, в которых могут быть несколько точек выхода и одна точка входа.

2.1 Архитектура программного обеспечения

На рисунке 9 представлена архитектура блочного устройства zram (в виде вызываемых функций).

Ниже приведено описание каждой из функций, представленных на рисунке 9:

- zram_rw_page() – эта функция является точкой входа для каждой страницы попавшей в блочное устройство. В этой функции происходят подготовительные вызовы: например, уведомление о начале I/O операции. После выполнения всех подготовительных операций, вызывается функция zram_bvec_rw();
- zram_bvec_rw() – в зависимости от операции, которую необходимо произвести прочитать сжатую страницу из блочного устройства или сжать и сохранить страницу, в этой функции вызывается функция zram_bvec_rw() или zram_bvec_write() соответственно;
- zram_bvec_read() – эта функция является обёрткой над функцией __zram_bvec_read(). Выполняет некоторые подготовительные действия и вызывает следующую функцию.
- __zram_bvec_read() –поиск индекса необходимой страницы, которую нужно вернуть, в блочном устройстве, вызов функции zram_get_element();
- zram_get_element() – возвращает сжатую страницу, хранящуюся в блоч-

ном устройстве zram;

- zcomp_decompress() – проводит преобразование сжатой страницы к исходному виду;
- zram_bvec_write() – эта функция является обёрткой над функцией __zram_bvec_write(). Выполняет некоторые подготовительные действия и вызывает следующую функцию.
- __zram_bvec_write() – запись сжатой страницы в блочное устройство, вызов zcomp_compress();
- zcomp_compress() – сжимает страницу памяти, подаваемую на вход;
- zram_put_element() – сохраняет сжатую страницу в блочном устройстве zram.

2.2 Алгоритмы

Ниже представлены алгоритм вычисления информационной энтропии для страницы оперативной памяти и алгоритм функции обработки (записи) страниц оперативной памяти в блочном устройстве zram.

2.2.1 Алгоритм вычисления информационной энтропии

Входные данные: Массив байт размером $PAGE\_SIZE$: src.

Выходные данные: Информационная энтропия для входного массива байт: $entropy\_sum$

Алгоритм 1 Алгоритм вычисления информационной энтропии для страницы памяти

1: $src \leftarrow$ массив байт размером $PAGE\_SIZE$
2: $entropy\_sum \leftarrow 0$
3: $entropy\_count[256] \leftarrow 0$
4: **for** $i \in [0, PAGE\_SIZE]$ **do**
5: $\quad index \leftarrow src[i]$
6: $\quad entropy\_count[index] \leftarrow entropy\_count[index] + 1$
7: **end for**
8: $powered \leftarrow pow(PAGE\_SIZE, 4)$ ▷ Возведение в 4 степень необходимо для увеличения точности вычисления логарифма
9: $sz\_base \leftarrow ilog2(powered)$ ▷ Вычисление логарифма (в целых числах)
10: **for** $i \in [0, 256]$ **do**
11: $\quad cnt \leftarrow entropy\_count[i]$
12: $\quad$ **if** $cnt > 0$ **then**
13: $\quad\quad powered \leftarrow pow(cnt, 4)$
14: $\quad\quad entropy\_sum = entropy\_sum + cnt * (sz\_base - ilog2(powered))$
15: $\quad$ **end if**
16: **end for**
17: **return** $entropy\_sum$

2.2.2 Алгоритм функции записи страниц памяти

Входные данные:

- структура описывающая модуль zram: $zram$;
- вектор, хранящий обрабатываемую страницу: $bvec$
- индекс сжимаемой страницы внутри блочного устройства: $index$

Выходные данные: дескриптор описывающий записанную (сжатую) страницу: $desc$

Алгоритм 2 Алгоритм функции записи страниц оперативной памяти в блочном устройстве zram

1: $page \leftarrow get\_page(bvec)$ ▷ Получить физическое описание обрабатываемой страницы
2: $mem \leftarrow kmap\_atomic(page)$ ▷ Получить байты, хранящиеся на этой странице памяти
3: **if** $page\_same\_filled(mem)$ **then**
4: $desc \leftarrow mem[0]$
5: $kunmap\_atomic(mem)$
6: **return** $desc$
7: **end if**
8: $kunmap\_atomic(mem)$
9: $zstrm \leftarrow zcomp\_stream\_get(zram)$ ▷ Захватить блокировку
10: $src \leftarrow kmap\_atomic(page)$
11: $entropy \leftarrow shannon\_entropy(src)$
12: **if** $entropy > ZRAM\_THRESHOLD$ **then** ▷ Сохранить страницу в несжатом виде
13: $comp\_len \leftarrow PAGE\_SIZE$
14: **else**
15: $comp\_len \leftarrow zcomp\_compress(zram, src)$ ▷ Сжать массив байт
16: $kunmap\_atomic(src)$
17: **end if**
18: $handle \leftarrow zs\_malloc(zram, comp_len)$ ▷ Аллоцировать в аллокаторе объект размером $comp\_len$
19: $dst \leftarrow zs\_map\_object(zram, handle)$ ▷ Получить аллоцированный объект
20: $src \leftarrow buffer$
21: **if** $comp\_ken = PAGE\_SIZE$ **then**
22: $src \leftarrow kmap\_atomic(page)$ ▷ Если страница не сжимаемая, сохранить ее в том виде, какая она есть
23: **end if**
24: $memcpy(dst, src, comp\_len)$
25: **if** $comp\_ken = PAGE\_SIZE$ **then**
26: $kunmap\_atomic(page)$
27: **end if**
28: $zcomp\_stream\_put(zram)$ ▷ Освободить блокировку
29: $desc \leftarrow dst$
30: $zs\_unmap\_object(zram, handle)$
31: **return** $desc$

2.3 Вывод

Была представлена архитектура модуля ядра zram. Были разработаны алгоритм вычисления информационной энтропии для страниц оперативной памяти и алгоритм записи страниц в блочном устройстве.

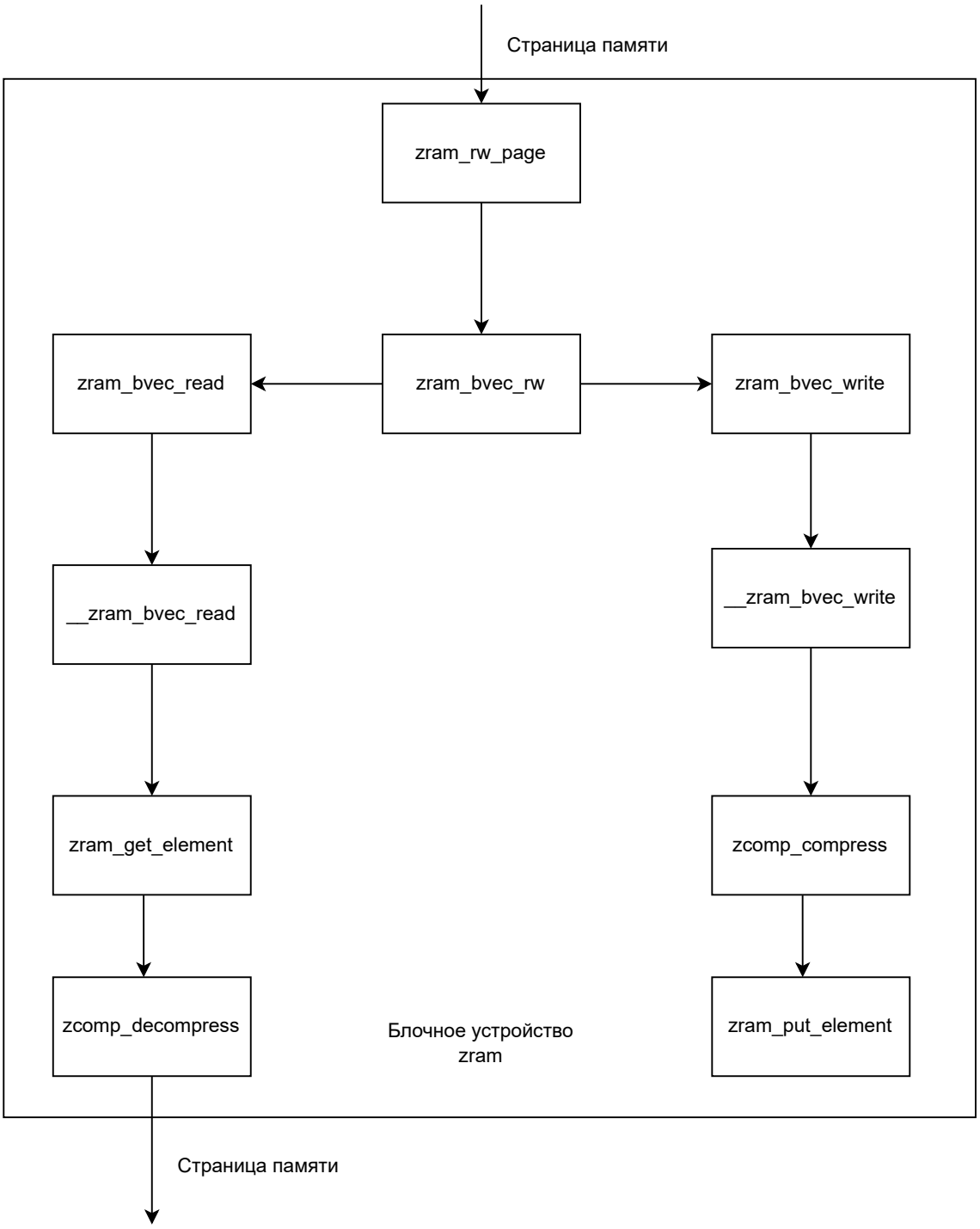

Рисунок 9 – Архитектура модуля ядра zram

3 Технологическая часть

В данном разделе описываются средства разработки программного обеспечения и требования к нему. Приводится структура разработанного ПО.

3.1 Выбор средств разработки

3.1.1 Выбор языка программирования

Ядро ОС Linux написано на языке программирования C [29] с использованием стандарта C89/C90. Встроенный драйвер ядра Linux, zram, так же написан на языке программирования C. Программное обеспечение представляет из себя модификацию модуля ядра zram, в связи с чем для реализации программного обеспечения был выбран язык программирования C версии стандарта C89. В модуле ядра zram, как и в ядре Linux, используется процедурный подход к программированию. Язык C поддерживает процедурную парадигму программирования.

3.1.2 Версия ядра Linux

В качестве версии ядра Linux была выбрана версия 5.17.5. На момент написания дипломной работы, эта версия является самой актуальной версией ядра Linux. Данный факт позволяет использовать все возможности ядра при разработке программного обеспечения.

3.2 Сборка программного обеспечения

Разработанное программное обеспечение является часть ядра Linux. Для сборки проекта используется специальная утилита make [30], позволяющая автоматизировать сборку ядра. make является кроссплатформенной системой автоматизации сборки программного обеспечения из исходного кода. make позволяет существенно ускорить процесс сборки проекта. Так, например, при изменении одного исходного файла проекта, заново будет собран в объектный файл лишь этот исходный файл, а не все файлы проекта.

Для сборки и включения модуля zram в итоговый образ ядра Linux, в конфигурационном файле ядра необходимо включить опции, указанные в листинге

7. Ниже приведено детальное описание включаемых опций:

- `CONFIG_ZRAM` – опция, включающая поддержку модуля zram в итоговый образ ядра Linux. Без этой опции, использовать zram будет невозможно;
- `CONFIG_ZRAM_DEF_COMP_ZSTD` и `CONFIG_ZRAM_DEF_COMP="zstd"` – выбрать алгоритм сжатия, который будет использоваться при сжатии страниц оперативной памяти. В данном случае, выбран алгоритм сжатия zstd;
- `CONFIG_ZRAM_ENTROPY` – эта опция включает поддержку энтропийной оптимизации модуля zram.
- `CONFIG_ZRAM_ENTROPY_THRESHOLD=100000` – данный параметр задает границу для отсекаемых страниц. Страницы, энтропия которых выше данного значения, будут храниться не в сжатом виде. Данное значение для каждого алгоритма сжатия выбирается оптимальным образом автоматически, но, при этом, имеется возможность установить его в ручную.

Разработанное программное обеспечение представлено в виде опции. Эту опцию можно выключать и отключать на стадии сборки ядра.

Листинг 7: Опции, которые необходимо добавить в конфигурационный файл ядра

```
1 CONFIG_ZRAM=y
2 CONFIG_ZRAM_DEF_COMP_ZSTD=y
3 CONFIG_ZRAM_DEF_COMP="zstd"
4 CONFIG_ZRAM_ENTROPY=y
5 CONFIG_ZRAM_ENTROPY_THRESHOLD=100000
```

3.3 Требования к вычислительной системе

Разработанное программное обеспечение представляет из себя модификацию ядра Linux. Для сборки и установки ядра в систему требуются следующие библиотеки и утилиты, представленные в таблице 1.

Таблица 1 – Таблица ПО, необходимого для сборки и установки ядра

ПО	**Минимальная версия**
gcc	3.2
GNU make	3.80
binutils	2.12
util-linux	2.10o
module-init-tools	0.9.10
e2fsprogs	1.41.4
jfsutils	1.1.3
reiserfsprogs	3.6.3
xfsprogs	2.6.0
squashfs-tools	4.0
btrfs-progs	0.18
pcmciautils	004
quota-tools	3.09
PPP	2.4.0
isdn4k-utils	3.1pre1
nfs-utils	1.0.5
procps	3.2.0
oprofile	0.9
udev	081
grub	0.93
mcelog	0.6
iptables	1.4.2
openssl, libcrypto	1.0.0
bc	1.2

3.4 Структура программного обеспечения

Разработанное ПО представляет из себя функцию вычисления информационной энтропии, её вызов перед попыткой сжатия страницы и сравнение с пороговым значением. Ниже описывается эта функция и модификация функции, в которой происходит её вызов и принятие решение о дальнейшем хранении страницы памяти.

3.4.1 Функция вычисления информационной энтропии

Функция shannon_entropy в качестве единственного входного параметра получает указатель на массив байт размером PAGE_SIZE. Возвращает информационную энтропию, подсчитанную по формуле 2, для данного набора байт.

Для вычисления логарифма используется встроенная функция ядра ilog2. В ядре Linux нет поддержки чисел с плавающей запятой, поэтому функция ilog2 работает только с целыми числами. Передаваемый параметр в функцию ilog2 (p_i) возводится в 4 степень для увлечения точности вычислений.

Пример реализации функции shannon_entropy() для модуля zram представлен в листинге 8.

Листинг 8: Функция shannon_entropy()

```
static inline u32 ilog2_w(u64 n)
{
    return ilog2(n * n * n * n);
}

static inline s32 shannon_entropy(const u8 *src)
{
    s32 entropy_sum = 0;
    u32 sz_base, i;
    u16 entropy_count[256] = { 0 };

    for (i = 0; i < PAGE_SIZE; ++i)
        entropy_count[src[i]]++;

    sz_base = ilog2_w(PAGE_SIZE);
    for (i = 0; i < ARRAY_SIZE(entropy_count); ++i) {
        if (entropy_count[i] > 0) {
            s32 p = entropy_count[i];

            entropy_sum += p * (sz_base - ilog2_w((u64)p));
        }
    }

    return entropy_sum;
}
```

3.4.2 Принятие решение о дальнейшем хранении страницы

Сжатие страницы происходит в функции zcomp_compress(), которая, в свою очередь вызывается в функции __zram_bvec_write(). Вызов функции

shannon_entropy() необходимо произвести до сжатия данных, хранящихся на странице памяти, поэтому функция __zram_bvec_write() была модифицирована.

Перед вызовом zcomp_compress() для каждой попавшей в эту функцию страницы считается информационная энтропия. Если вычисленное значение выше порогового значения CONFIG_ZRAM_ENTROPY_THRESHOLD, объявленного с помощью директивы #define, то эта страница помечается как несжимаемая в памяти. В обратном случае, происходит вызов zcomp_compress() и данные, находящиеся на этой странице памяти, сжимаются.

Пример реализации модификации функции __zram_bvec_write() модуля zram представлен в листинге 9.

Листинг 9: Функция __zram_bvec_write()

```
1  static int __zram_bvec_write(struct zram *zram,
2      struct bio_vec *bvec,
3      u32 index,
4      struct bio *bio) {
5  ...
6      zstrm = zcomp_stream_get(zram->comp);
7      src = kmap_atomic(page);
8
9  #ifdef CONFIG_ZRAM_ENTROPY
10     entropy = shannon_entropy((const u8 *)src);
11     if (entropy > CONFIG_ZRAM_ENTROPY_THRESHOLD)
12         comp_len = PAGE_SIZE;
13     else
14         ret = zcomp_compress(zstrm, src, &comp_len);
15 #else
16     ret = zcomp_compress(zstrm, src, &comp_len);
17 #endif
18
19     kunmap_atomic(src);
```

```
20      . . .
21      }
```

3.5 Руководство пользователя

Для запуска разработанного ПО необходимо запущенное ядро Linux, собранное с опциями указанными в главе 3.3. Список включенных опций на уже запущенном ядре Linux можно узнать с помощью команды zcat /proc/config.gz/.

Собрать ядро Linux из исходных файлов можно с помощью команды make -j$(nproc). До сборки необходимо убедиться, что в конфигурационном файле, имеющем имя .config и находящемся в корне проекта, включены опции указанные в главе 3.2. Для того чтобы загрузчик мог использовать собранный образ ядра, необходимо установить его с помощью команд make modules_install и make install.

Для того чтобы инициализировать и задать размер блочного устройства zram необходимо использовать команду: echo 512M > /sys/block/zram0/disksize. В данном примере размер устройства равен 512 мегабайтам. Задавать размер, превышающий размер ОЗУ более чем в 2 раза не имеет смысла: в среднем коэффициент сжатия равен двум [20].

Для того чтобы установить блочное устройство zram в качестве устройства, которое будет использовать в подкачке страниц, необходимо использовать следующие команды: mkswap /dev/zram0 и swapon /dev/zram0.

3.6 Вывод

В данном разделе были описаны средства разработки программного обеспечения и требования к ПО. Была приведена структура разработанного ПО.

4 Исследовательская часть

В рамках дипломной работы было проведено исследование сравнения времени обработки и коэффициент сжатия файлов в блочном устройстве zram с оптимизацией и без. Результаты исследования представлены в данном разделе.

4.1 Описание используемых данных

Для исследования работоспособности разработанного программного обеспечения и оценки времени обработки данных и коэффициента их сжатия были выбраны бинарные файлы различного типа и разного размера. В исследовании использовались файлы типа pdf (portable document format [31]), apk (android package [32]), случайные файлы из домашней директории, сохраненные в единый файл с помощью утилиты tar [33], библиотеки расположенные в корневой директории в папке /lib, так же упакованные с помощью утилиты tar. Сжатие директории /lib с библиотеками является попыткой воспроизвести процесс загрузки библиотек в оперативную память. Размер файлов составляет 400 мегабайт, 430 мегабайт, 5 гигабайт и 17 гигабайт соответственно. Все исследования были проведены для алгоритма сжатия zstd.

4.2 Вычисление порогового значения энтропии

Параметр конфигурации ядра CONFIG_ZRAM_ENTROPY_THRESHOLD это некоторое положительное, целое число. Страницы, энтропия которых больше этого параметра, будут помечены как несжимаемые и будут храниться в zram в несжатом виде. Этот параметр задается статически, на стадии сборки ядра, поэтому необходимо подобрать оптимальное значение для каждого из алгоритмов сжатия.

Для определения оптимального порогового значения из модуля ядра zram были собраны данные в формате указанном в таблице 2.

Каждая строка дает некоторую характеристику странице оперативной памяти, попавшую в блочное устройство zram:

- первый столбец – номер строки;

Таблица 2 – Пример csv файла с данными, необходимыми для вычисления параметра CONFIG_ZRAM_ENTROPY_THRESHOLD

	compressor_time	**entropy_time**	**entropy**	**compress**
0	192734	6693	11502	14.2222
1	565457	7364	16815	6.4341
2	2591472	6975	39072	3.8496
4	1417756	5680	51755	2.4380
3	1310464	5149	71668	1.8686

- второй столбец – машинное время (в тиках), потраченное на сжатие этой страницы;
- третий столбец – машинное время (в тиках), потраченное на вычисление энтропии для данной страницы;
- четвертый столбец – вычисленное значение энтропии;
- пятый столбец – коэффициент сжатия этой страницы.

Все данные в исходном файле отсортированы по коэффициенту сжатия страницы (по возрастанию). На основе этих данных были построены графики формата изображенного на рисунках 10 и 11.

На рисунке 10 представлены данные о изначальном и сжатом размере страниц. По оси x отложено количество страниц, по оси y размер страниц (в байтах). Прерывистая линия – кумулятивная сумма столбца compressed_size (сжатый размер страницы). Сплошная линия – кумулятивная сумма столбца orig_size (размер страницы не в сжатом виде).

На рисунке 11 представлены данные о времени, потраченном на сжатие и вычисление энтропии. По оси x отложено количество страниц, по оси y время (в тиках). Прерывистая линия – кумулятивная сумма столбца compressed_size (время, потраченное на сжатие страницы). Сплошная линия – кумулятивная сумма столбца entropy_time (время, потраченное на вычисление энтропии).

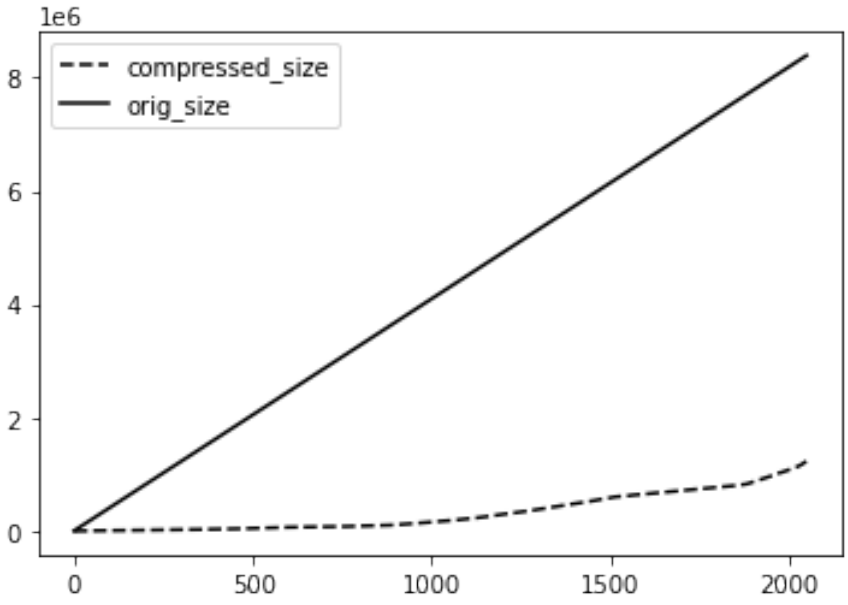

Рисунок 10 – Кумулятивные суммы сжатого и несжатого размера страниц

Проведем прямую из точки графика кумулятивной суммы времени вычисления энтропии, перпендикулярную оси x (см. рис. 11). Примем $x = 1500$. Проведем второй перпендикуляр к оси y из начала прямой, проведенной как перпендикуляр к оси x. На рассматриваемом графике, пересечения второй проведенной прямой и оси y будет в значении $y = 0.7e6$. Далее, необходимо спроецировать верхний график, находящийся справа от первого перпендикуляра до пересечения с точкой начала этой прямой. Пересечение проекции с осью y будет в $y = 4e6$.

Таким образом, если сжимать лишь страницы, энтропия которых меньше энтропии страницы находящейся в исходных данных под номером 1500 (данные отсортированы по увеличению коэффициенту сжатия), размер сжатых данных будет равен $4e6$ килобайт. В таком случае объем сжатых данных станет больше в два раза, то есть коэффициент сжатия уменьшится в два раза.

Проведем аналогичный перпендикуляр для графика 10, примем $x = 1500$. Можно сделать вывод, что в таком случае, выигрыш по времени составит 25%. Для данного набора данных можно сделать вывод, что не сжимая страницы, энтропия которых больше энтропии страницы с номером 1500, можно добиться

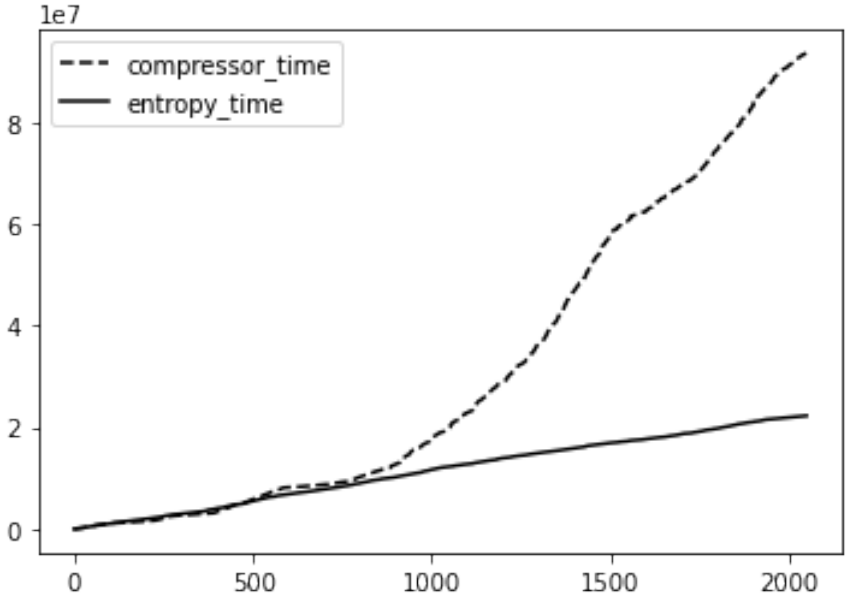

Рисунок 11 – Кумулятивные суммы времени затраченного на подсчёт энтропии и сжатия страницы

ускорения обработки данных на 25%, при этом увеличив объем сжатых данных на 200%.

В результате эксперимента в качестве значения порогового значения энтропии (для алгоритма zstd) было выбрано значение 100 000.

4.3 Методика проведения исследования

Для исследования необходимо произвести замеры количества машинных инструкций, времени сжатия и коэффициента сжатия данных с использованием энтропийной оптимизации и без. Количество машинных инструкций и время выполнения подсчитывается с помощью утилиты perf [34], а коэффициент сжатия данных с использованием встроенной в модуль zram статистики.

В таблице 3 представлены результаты сравнения коэффициентов сжатия с включенной разработанной оптимизацией и без. В первом столбце указано, включена энтропийная оптимизация или нет. В ячейках таблицы указан исходный размер данных, сжатый и коэффициент сжатия.

В таблице 4 представлено сравнение времени выполнения и количества машинных инструкций с включенной оптимизацией и без.

Таблица 3 – Таблица сравнения коэффициентов сжатия с оптимизаций и без

Патч	Файл	Размер на входе, кб	Размер на выходе, кб	Коэф. сжатия
Да	pdf	397.464	396.535	1.002
Нет	pdf	397.464	395.870	1.004
Да	apk	421.340	327.992	1.284
Нет	apk	421.340	282.331	1.492
Да	tar	5.153.692	4.131.741	1.247
Нет	tar	5.153.692	4.117.655	1.251
Да	tar (/lib)	16.527.126	5.921.855	2.791
Нет	tar (/lib)	16.527.126	5.594.516	2.954

Таблица 4 – Таблица сравнения количества машинных инструкций с оптимизаций и без

Патч	Файл	Размер на входе, кб	Время сжатия, с	Кол-во инструкций
Да	pdf	397.464	0.572	4.641.658.347
Нет	pdf	397.464	2.098	17.187.405.320
Да	apk	421.340	1.632	13.055.231.312
Нет	apk	421.340	3.196	24.345.720.313
Да	tar	5.153.692	7.066	35.039.839.769
Нет	tar	5.153.692	11.955	76.763.123.464
Да	tar (/lib)	16.527.126	16.461s	71.366.805.304
Нет	tar (/lib)	16.527.126	21.872s	112.664.002.524

4.4 Вывод

В результате исследования было установлено что коэффициент сжатия сильно зависит от входных данных. Так, например, файл формата pdf практически не сжался как с включенной энтропийной оптимизацией, так и без.

Было подобран пороговое значения энтропии для алгоритма zstd – 100 000. Таким образом, страницы, энтропия которых более 100 000 будут храниться в блочном устройстве несжатые.

Оптимизация метода сжатия ускоряет процесс преобразования данных от 1.5 до 4 раз. Чем меньше процент сжатия исходных данных без включенной оптимизации, тем больше ускоряется процесс сжатия с включенной оптимизацией. Файл формата pdf потерял маленькую долю сжатия (коэффициент сжатия стал 1.002, вместо 1.004), но при этом, процесс преобразования данных ускорился в 4 раза.

Разносортные данные упакованные в единый файл сжимаются в среднем на 25%. Сжатие с энтропийной оптимизацией происходит в 1.9 раза быстрее, чем без. При этом, потеря в сжатии составляет менее процента.

При сжатии директории /lib, в которой хранятся различные системные библиотеки, 25%-ый выигрыш по времени достигается с помощью потери 5.5% сжатого объема (итогового размера).

Подводя итог, можно сделать вывод, что разработанная оптимизация программного обеспечения ускоряет процесс сжатия в несколько раз (1.5-4 раза), при этом потеря в сжатии не крайне малы: от 1% до 16%, в зависимости от входных данных.

Полученные результаты, вместе с модификацией модуля zram, были отправлены в качестве RFC (англ. request for comments [35]) письма мейнтейнерам модуля ядра zram [37].

ЗАКЛЮЧЕНИЕ

Это надо поменять

В ходе выполнения дипломной работы был исследован метод сжатия страниц оперативной памяти в ядре Linux и была разработана его оптимизации.

Для достижения поставленной цели были решены следующие задачи:

- рассмотрены методы увеличения количества оперативной памяти;
- дана характеристика архитектурам ядер современных операционных систем;
- изучены подходы, структуры данных и API [3] в ядре Linux, позволяющие управлять подсистемой памяти;
- описана работа модуля сжатия оперативной памяти в ядре Linux;
- разработана оптимизацию для данного модуля;
- спроектирована структура программного обеспечения, реализующего оптимизацию модуля сжатия оперативной памяти;
- сравнен метод сжатия страниц оперативной памяти с оптимизацией и без.

Разработанная оптимизация была отправлена в качестве RFC письма разработчикам модуля ядра zram. В качестве дальнейшего развития было предложено вычисление порогового значения информационной энтропии не статически (на стадии компиляции), а динамически (во время работы блочного устройства).

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. IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 26, NO. 2, FEBRUARY 2007, Relationship Between Entropy and Test Data Compression. Kedarnath J. Balakrishnan. с. 386.

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

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

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

16. 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)

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

18. Block Device Drivers — The Linux Kernel documentation [Электронный ресурс]. – Режим доступа: https://linux-kernel-labs.github.io/refs/heads/master/labs/block_device_drivers.html, свободный – (10.03.2022)

19. Character Device Drivers — The Linux Kernel documentation [Электронный ресурс]. – Режим доступа: https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html, свободный – (10.03.2022)

20. 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)

21. Tmpfs — The Linux Kernel documentation [Электронный ресурс]. – Режим доступа: https://www.kernel.org/doc/html/latest/filesystems/tmpfs.html, свободный – (10.03.2022)

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

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

24. LZO – a real-time data compression library | GitHub [Электронный ресурс]. – Режим доступа: https://github.com/nemequ/lzo/blob/master/doc/LZO.TXT, свободный – (20.11.2021)

25. lzo-rle - Linux Kernel | Github [Электронный ресурс]. – Режим доступа: https://github.com/torvalds/linux/blob/master/crypto/lzo-rle.c – (27.04.2022)

26. Algorithm 842: A set of GMRES routines for real and complex arithmetics on high performance computers [Электронный ресурс]. – Режим доступа: https://dl.acm.org/doi/10.1145/1067967.1067970, свободный – (20.11.2021)

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

28. Kernel level exception handling – The Linux Kernel Documentation [Электрон-

ный ресурс]. – Режим доступа: https://github.com/facebook/zstd, свободный – (20.11.2021)

29. ISO/IEC 9899:1990 - Programming languages — C

30. Make - GNU Project - Free Software Foundation [Электронный ресурс]. – Режим доступа: https://www.gnu.org/software/make/, свободный – (27.04.2022)

31. What is a PDF? Portable Document Format | Adobe Acrobat [Электронный ресурс]. – Режим доступа: https://www.adobe.com/acrobat/about-adobe-pdf.html, свободный – (27.04.2022)

32. What is APK file (Android Package Kit file format)? - TechTarget [Электронный ресурс]. – Режим доступа: https://www.techtarget.com/whatis/definition/APK-file-Android-Package-Kit-file-format, свободный – (27.04.2022)

33. tar(1) - Linux manual page - man7.org [Электронный ресурс]. – Режим доступа: https://man7.org/linux/man-pages/man1/tar.1.html, свободный – (27.04.2022)

34. Perf Wiki | Linux Kernel [Электронный ресурс]. – Режим доступа: https://perf.wiki.kernel.org/index.php/Main_Page, свободный – (27.04.2022)

35. PatchTipsAndTricks - Linux Kernel Newbies [Электронный ресурс]. – Режим доступа: https://kernelnewbies.org/PatchTipsAndTricks, свободный – (27.04.2022)

36. Bernoulli Process - an overview | ScienceDirect Topics [Электронный ресурс]. – Режим доступа: https://www.sciencedirect.com/topics/mathematics/bernoulli-process, свободный – (27.04.2022)

37. [RFC,v1] zram: experimental patch with entropy calculation | Patchwork [Электронный ресурс]. – Режим доступа: https://patchwork.kernel.org/project/linux-

block/patch/20220520171309.26768-1-avromanov@sberdevices.ru/, свободный – (27.05.2022)