Оглавление

ВВЕДЕНИЕ

Планирование – это управление распределением ресурсов процессора между различными конкурирующими процессами путем передачи им управления согласно некоторой стратегии планирования. За планирование процессов в операционных системах отвечают планировщики задач, каждый из которых может использовать собственный уникальный алгоритм планирования процессов.

Linux – операционная система мягкого реального времени. Классы планирования реального времени, блокировка памяти, разделяемая память и сигналы реального времени получили поддержку в Linux с самых первых дней. Очереди сообщений POSIX, часы и таймеры поддерживаются в ядре версии 2.6. Асинхронный ввод/вывод также поддерживается с самых первых дней, но эта реализация была полностью сделана в библиотеке языка Си пользовательского пространства. Linux версии 2.6 имеет поддержку AIO (Asynchronous I/O) в ядре. Библиотека языка C GNU и glibc также претерпели изменения для поддержки этих расширений реального времени. Для обеспечения лучшей поддержки в Linux POSIX.1b ядро и glibc работают вместе.

Открытый исходный код Linux позволяет самостоятельно изучить особенности системы.

Данная курсовая работа направлена на изучение особенностей планирования процессов реального времени, таких как воспроизведение видео и аудио, и разработку программного обеспечения для мониторинга приоритетов, времени выполнения и простоя процессов на операционной системе Linux. В результате работы будет представлен анализ особенностей планирования при воспроизведении аудио и видеофайлов.

1. Аналитический раздел

В данном разделе приведена формализация цели, понятие процесса реального времени и описание работы планировщика реального времени. Также представлены структуры ядра и пояснения к их полям, которые понадобятся при решении поставленной задачи. Описаны способы определения принадлежности задачи к группе задач реального времени и рассмотрена их реализация.

1.1 Формализация цели

Цель работы – разработать загружаемый модуль ядра для мониторинга приоритетов, времени выполнения и простоя процессов на ОС Linux и проанализировать с использованием данного модуля воспроизведение аудиофайлов и видеофайлов.

Для достижения поставленной цели потребуется:

1) проанализировать структуры ядра, позволяющие определить приоритет, время выполнения и простоя процессов;
2) проанализировать методы передачи информации из модуля ядра в пространство пользователя;
3) спроектировать и реализовать загружаемый модуль ядра;
4) проанализировать с использованием реализованного модуля воспроизведение аудиофайлов и видеофайлов.

1.2 Понятие процесса реального времени

Процесс реального времени – это процесс, который имеет приоритет над обычными процессами. В Linux планирование процессов реального времени возложено на так называемый RT Scheduler (Real Time Scheduler – планировщик реального времени), а обычные процессы обрабатываются с использованием CFS Scheduler (Completely Fair Scheduler – совершенно честный планировщик). Работа данных планировщиков строится на работе с группами процессов, которые расположены в очередях, которые, в свою очередь, организованы в дерево. [1]

1.3 Описание работы RT Scheduler

Проблема планирования в режиме реального времени заключается в том, что группы процессов должны полагаться на постоянство объема полосы пропускания (например, времени исполнения). Для планирования несколь-

ких групп задач в реальном времени может потребоваться назначение, так называемого, гарантированного доступного времени исполнения. [2]

В качестве решения поставленной проблемы доступное процессорное время делится посредством указания того, сколько времени на исполнение даётся на указанный период. Данное время выделяется для каждой группы процессов реального времени, причём каждая группа может исполнятся только в свое выделенное время. [2]

Время, не выделенное группам реального времени или неиспользованное ими, будет выделено задачами с обычным приоритетом (SCHED_OTHER). [2]

Для примера можно рассмотреть следующую задачу: рендерер реального времени с фиксированным количеством кадров должен выдавать 25 кадров в секунду, что дает период 0.04 секунды на каждый кадр. Если перед рендерером также стоит параллельная задача проигрывания музыки и ответа на ввод, оставляя около 80% доступного процессорного времени, предназначенного для графики, то для этой группы можно выделить время выполнения $0.8 \cdot 0.04 = 0.032$ секунды.

Таким образом, графическая группа будет иметь период 0.04 секунды с ограничением времени выполнения 0.032 секунды. Если аудиопотоку необходимо заполнять буфер DMA (Direct Memory Access, прямой доступ к памяти) каждые 0.005 секунд, но для этого требуется около 3% времени процессора, ему может быть выделено время выполнения $0.03 \cdot 0.005 = 0.00015$ секунд. Таким образом, данная группа может быть запланирована с периодом 0.005 секунд с временем выполнения 0.00015 секунд.

Оставшееся процессорное время будет использовано для ввода данных пользователем и других задач.

Однако на текущий момент приведенный выше пример еще не реализован полностью в силу того, что отсутствует реализация планировщика EDF (Erlienst Deadline First scheduling, алгоритм планирования по ближайшему сроку завершения) для использования неоднородных периодов. [2]

1.4 Структуры ядра

Современные операционные системы предоставляют пользователю фундаментальные концепции, такие как, файл или процесс. [3]

С использованием документации представляется возможность полу-

чить доступ к данным концепциям и изучить работу системы изнутри.

1.4.1 task_struct

Процесс состоит из нескольких компонентов [3]:

- стек процесса;
- регистры процессора, в которые загружены ключевые переменные (зависит от архитектуры);
- адресное пространство;
- ресурсы: дескрипторы открытых файлов, ожидающие обработки сигналы;
- управляющие структуры ядра ОС.

Структура в ядре Linux, соответствующая каждому процессу, – task_struct. Она определена в файле include/linux/sched.h. Все процессы существующие в системе процессы объединены в кольцевой список. [3]

Стоит отметить, что данная структура занимает в памяти порядка 1.7 килобайт.

Поля структуры содержат информацию о процессе, которую можно поделить на несколько категорий [3]:

- поля, отвечающие за общую информацию о процессе (PID, exit_code, PPID);
- поля, востребованные планировщиком задач (prio, static_prio, timeslice);
- поля, связанные с безопасностью (uid, gid).

Структура task_struct для Linux v5.16rc8 представлена в приложении (см. Приложение 1).

Далее будут отмечены наиболее информативные в проводимой работе поля данной структуры и их назначение.

pid (Process Identifier) – уникальный идентификатор процесса. Каждый процесс в операционной системе имеет свой уникальный идентификатор, по которому можно получить информацию об этом процессе, а также направить ему управляющий сигнал или завершить [4].

prio, static_prio, normal_prio, rt_priority – приоритеты процесса.

prio – это значение, которое использует планировщик задач при выборе процесса. Чем ниже значение данной переменной, тем выше приоритет про-

цесса (может принимать значения от 0 до 139, то есть MAX_PRIO, значение которого вычисляется с использованием переменной MAX_RT_PRIO со значением 100) [5]. Также данный приоритет может быть поделён на два интервала:

- от 0 до 99 – процесс реального времени;
- от 100 до 139 – обычный процесс.

Также определены функции определения приоритета процесса, которые приведены в листинге 1.

Листинг 1: Функции определения приоритета процесса, определенные в /kernel/sched.c

Из предоставленного листинга видно, что для процессов реального времени значение приоритета определяется с использованием поля prio, а в ином случае – static_prio.

static_prio не изменяется ядром при работе планировщика, однако оно может быть изменено с использованием пользовательского приоритета nice. Макрос для изменения данного приоритета предоставлен в листинге 2.

Листинг 2: Макрос для изменения static_prio.

Таким образом, значение статического приоритета может быть изменено с использованием вызова макроса NICE_TO_PRIO(nice).

normal_prio зависит от статического приоритета и политики планировщика задач. Для процессов не реального времени данное значение равняется значению статического приоритета static_prio. Для процессов реального времени данное значение равняется значению, вычисленному с использованием максимального значения приоритета процесса реального времени и, непосредственно, его rt_priority. Функция вычисления нормального приоритета предоставлена в листинге 3.

Листинг 3: Функция вычисления нормального приоритета.

Важно отметить факт того, что, чем больше значение rt_priority, тем выше приоритет процесса.

1.4.2 sched_info

sched_info – структура, которая предоставляет информацию о планировании процесса:

- количество запусков процесса на исполнение центральным процессором;
- количество времени, проведенного в ожидании на исполнение;
- время последнего запуска процесса на исполнение центральным процессором;
- время последнего добавления процесса в очередь на исполнение.

Данная структура предоставлена в листинге 5.

Листинг 4: Структура sched_info.

utime – это время, проведенное в режиме пользователя и затраченное на запуск команд. Данное значение включает в себя только время, затраченное центральным процессором, и не включает в себя время, проведенное процессом в очереди на исполнение.

stime – это время процессора, затраченное на выполнение системных вызовов при исполнении процесса.

1.4.3 sched_rt_entity

sched_rt_entity – это структура, используемая для группового планирования процессов реального времени, которое включает в себя деревья групп и их очередей. Данное решение позволяет определять выделяемую производительность процессора для определенных групп процессов.

Структура sched_rt_entity предоставлена в листинге 5.

Листинг 5: Структура sched_rt_entity.

В данной структуре поле parent указывает на процесс более высокого уровня в дереве. rt_rq – это очередь, в которой процесс находится, а my_q – это очередь дочерних процессов.

Поле timeout изменяется так называемым watchdog таймером и используется для проверки того, что процесс не занимает процессор дольше, чем это задано в RLIMIT_RTTIME.

1.5 Способы определения принадлежности задачи к группе задач реального времени

В файле /include/linux/sched/rt.h определены функции, позволяющие определить, является ли процесс задачей реального времени.

1.5.1 Функция rt_prio

Реализация данной функции приведена в листинге 6.

Листинг 6: Функция rt_prio.

Представленная функция определяет, является ли процесс задачей реального времени, с использованием значения приоритета. Приоритет задачи сравнивается с максимальным значением приоритета для задачи реального времени. В случае, если условие истинно, процесс относится к задачам реального времени и возвращается единица.

Важно отметить, что макрос unlikely в данном случае применяется для оптимизации скорости выполнения сравнения.

1.5.2 Функция rt_task

Реализация данной функции приведена в листинге 7.

Листинг 7: Функция rt_task.

rt_task является оберточной функцией для вызова функции rt_prio. Возвращает единицу в том случае, если процесс является задачей реального времени.

1.5.3 Функция task_is_realtime

Реализация данной функции приведена в листинге 8.

Листинг 8: Функция task_is_realtime.

task_is_realtime анализирует политику планировщика из преданной структуры task_struct. SCHED_FIFO и SCHED_RR являются, так называемыми, политиками реального времени.

SCHED_FIFO – это политика панирования реального времени ”первый вошел, первый вышел”. Данный алгоритм планирования не использует интервалов времени, а процесс выполняется до завершения, если он не заблокирован запросом ввода-вывода, вытеснен высокоприоритетным процессом, или он добровольно не отказывается от процессора. [6]

Следует обратить внимание на то, что процесс SCHED_FIFO, вытесненный другим процессом с более высоким приоритетом, остается во граве списка с его приоритетом и возобновит выполнение, как только все процессы с более высоким приоритетом будут вновь заблокированы. Также, когда процесс SCHED_FIFO готов к выполнению (например, после пробуждения от операции блокировки), он будет вставлен в конец списка с его приоритетом. [6]

Вызов sched_setscheduler или sched_setparam поставит процесс SCHED_FIFO в начало списка. Как следствие, это может вытеснить исполняющийся в данный момент процесс, если его приоритет такой же, как и у исполняющегося процесса. [6]

SCHED_RR – это циклическая (Round-Robin) политика планирования реального времени. Она похожа на SCHED_FIFO с той лишь разницей, что процессу SCHED_RR разрешено работать как максимум время кванта. Если процесс SCHED_RR исчерпывает свой квант времени, он помещается в конец списка с его приоритетом. [6]

Процесс SCHED_RR, который был вытеснен процессом с более высоким приоритетом, завершит оставшуюся часть своего кванта времени после возобновления выполнения. [6]

Планировщик класса SCHED_DEADLINE реализует алгоритм EDF (Earliest Deadline First), который основан на идее выбора для выполнения из очереди ожидающих процессов задачи, наиболее близкой к истечению крайнего расчетного времени. SCHED_DEADLINE поддерживает обеспечение работы процессов, требующих выполнения операций в режиме реального времени, предоставляя для подобных задач гарантированное время выполнения, независимо от общего количества обслуживаемых процессов, и реализуя возможность резервирования пропускной способности процессора для процессов. [7]

Как видно из реализации функции, оставшиеся политики планировщика задач не относятся к алгоритмам планирования реального времени.

1.6 Передача данных из пространства ядра в пространство пользователя

Поставленная задача подразумевает под собой передачу информации из пространства ядра в пространство пользователя.

В Linux для передачи данных из пространства ядра в пространство пользователя зачастую используется виртуальная файловая система procfs.

Ключевая идея данной концепции состоит в том, чтобы выделить некоторую часть файловой системы, являющейся общей для всех файловых систем, и поместить ее код на отдельный уровень, из которого вызываются расположенные ниже конкретные файловые системы с целью фактического управления данными. [8]

Все относящиеся к файлам системные вызовы направляются для первичной обработки в адрес виртуальной файловой системы. Эти вызовы, поступающие от пользовательских процессов, являются стандартными POSIX-вызовами, такими как open, read, write, lseek и так далее. Таким образом, VFS обладает “верхним” интерфейсом к пользовательским процессам. [8]

У VFS также существует “нижний” интерфейс к конкретной файловой системе. Этот интерфейс состоит из нескольких десятков вызовов функций, которые VFS способна направлять к каждой файловой системе для достижения конечного результата. Таким образом, чтобы создать новую файловую систему, работающую с VFS, ее разработчики должны предоставить вызовы функций, необходимых VFS. [8]

В Linux procfs предоставляет все ресурсы для реализации интерфейса между пространством пользователя и пространством ядра.

Структура proc_ops определена в файле linux/proc_fs.h и содержит в себе указатели на функции драйвера, которые отвечают за выполнение различных операций с устройством. Поля структуры представлены в листинге 9.

Листинг 9: Структура proc_ops.

Функция copy_to_user, определенная в файле linux/uaccess.h, позволяет копировать блоки данных из пространства ядра в пространство пользователя. Возвращает количество байт, которые не удалось скопировать. [9]

Реализация функции приведена в листинге 10.

Листинг 10: Реализация функции copy_to_user.

Вывод

В разделе была формализована задача, а также приведено понятие процесса реального времени. Была описана работа RT Scheduler, или планировщика задач реального времени.

Также приведены структуры ядра, информация о полях которых потребуется для решения поставленной задачи: task_struct, sched_info, sched_rt_entity. Определены особенности и различия полей prio, static_prio и normal_prio, что в дальнейшем помогло понять работу функций rt_prio, rt_task и task_is_realtime, которые были отнесены к способам определения принадлежности задач к группе задач реального времени.

Была описана концепция виртуальной файловой системы, приведена

структура proc_ops и реализация функции copy_to_user.

2. Конструкторский раздел

В данном разделе представлены требования к программному обеспечению, сведения о реализуемом модуле и дополнительном программном обеспечении. Также приведены схемы, описывающие работу компонентов.

2.1 Требования к программному обеспечению

Требуется реализовать загружаемый модуль ядра для мониторинга приоритетов, времени выполнения и простоя процессов, который будет получать информацию о процессах в режиме ядра и предоставлять доступ к данной информации через procfs.

В целях упрощения работы с данным модулем также потребуется реализовать дополнительную программу, выполняющую загрузку модуля в систему, а также предоставляющую получаемую из procfs информацию без дополнительных манипуляций пользователя.

2.2 Модуль логирования процессов реального времени

Для хранения информации о приоритетах, времени выполнения и простоя процессов реального времени выделяется массив символов.

Листинг 11: Массив символов журнала.

Для того, чтобы не допускать переполнений при записи в журнал, каждый раз перед занесением данных в лог проверяется, может ли он в данный момент вместить новые данные.

Листинг 12: Функция проверки переполнения.

Листинг 13: Пример использования функции проверки переполнения.

Как видно из предоставленного листинга 13 в случае обнаружения переполнения модуль не прекращает свое выполнение.

В листинге 14 приведено заполнение структуры proc_ops. Требуется, чтобы функция yaRead включала в себя вызов функции copy_to_user в целях передачи информации из журнала в пространство пользователя.

Листинг 14: Заполнение структуры proc_ops.

В листинге 15 предоставлены функции инициализации и завершения работы модуля.

Листинг 15: Функции инициализации и завершения работы модуля.

В данном фрагменте можно увидеть, что в функции инициализации мо-

дуля требуется запустить отдельный поток для журналирования, так как в ином случае инициализация не будет завершена до окончания выполнения требующегося количества итераций сбора информации.

2.3 Программа загрузки модуля и получения информации

Программа загрузки модуля и вывода полученной информации должна включать в себя два системных вызова: загрузки и выгрузки модуля. Также требуется получить доступ к файлу, создаваемому модулем в procfs. Данное действие может быть произведено с использованием вызова cat.

В листингах 16–18 предоставлены фрагменты выполнения указанных вызовов.

Листинг 16: Загрузка модуля в систему.

Листинг 17: Выгрузка модуля из системы.

Листинг 18: Выполнения вызова cat.

Функция popen в листинге 18 открывает процесс, создавая канал, производя fork и вызывая командную оболочку. Возвращаемое значение данный функции – это поток ввода-вывода. При этом будет возвращен NULL, если вызовы fork или pipe завершились ошибкой или если невозможно выделить необходимый для этого объем памяти.

2.4 Схемы работы модуля и дополнительного программного обеспечения

На рисунках 2.1–2.5 предоставлены схемы работы модуля и дополнительного программного обеспечения.

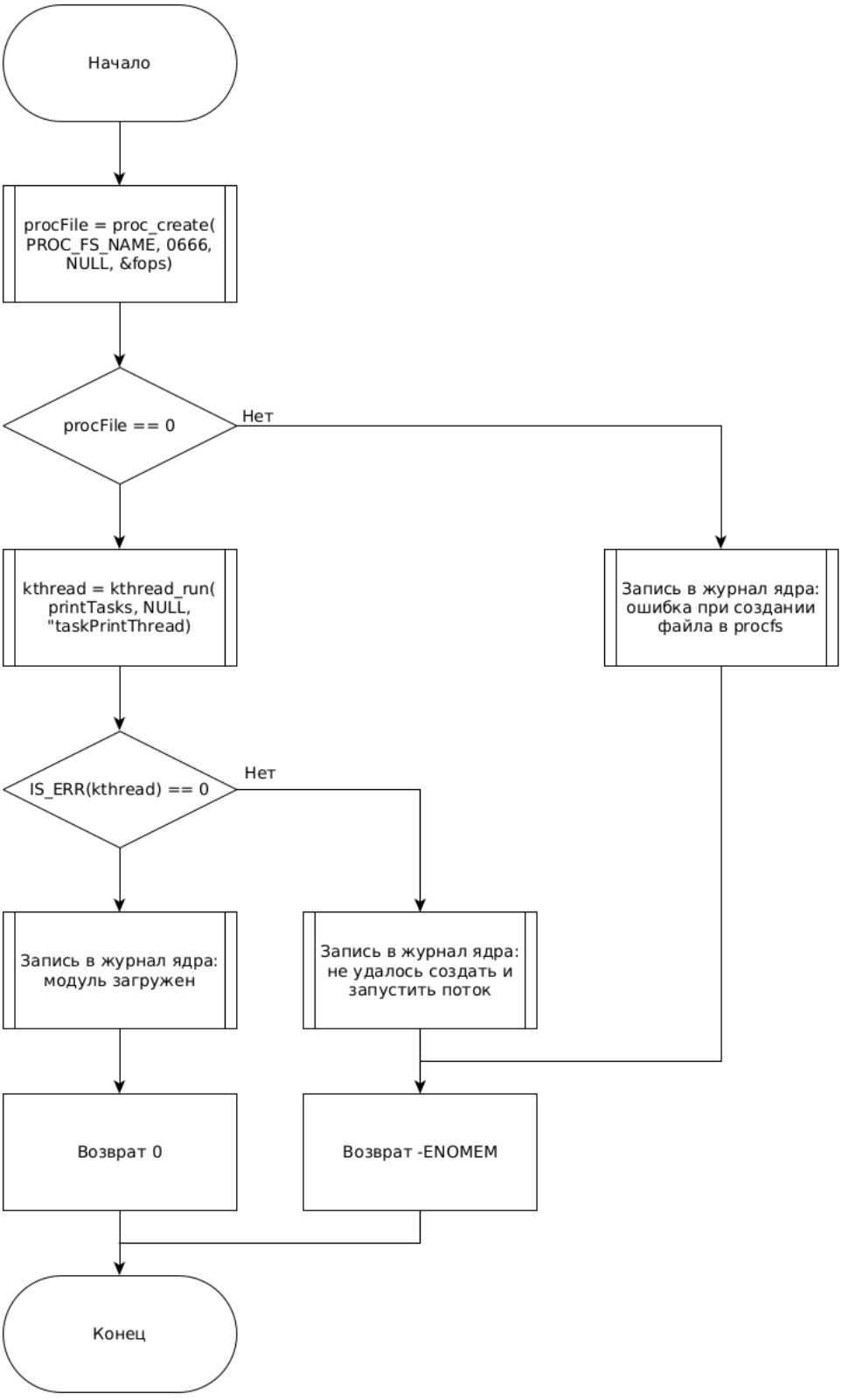


Рис. 2.1: Схема функции инициализации модуля.

Рис. 2.2: Схема функции завершения работы модуля.

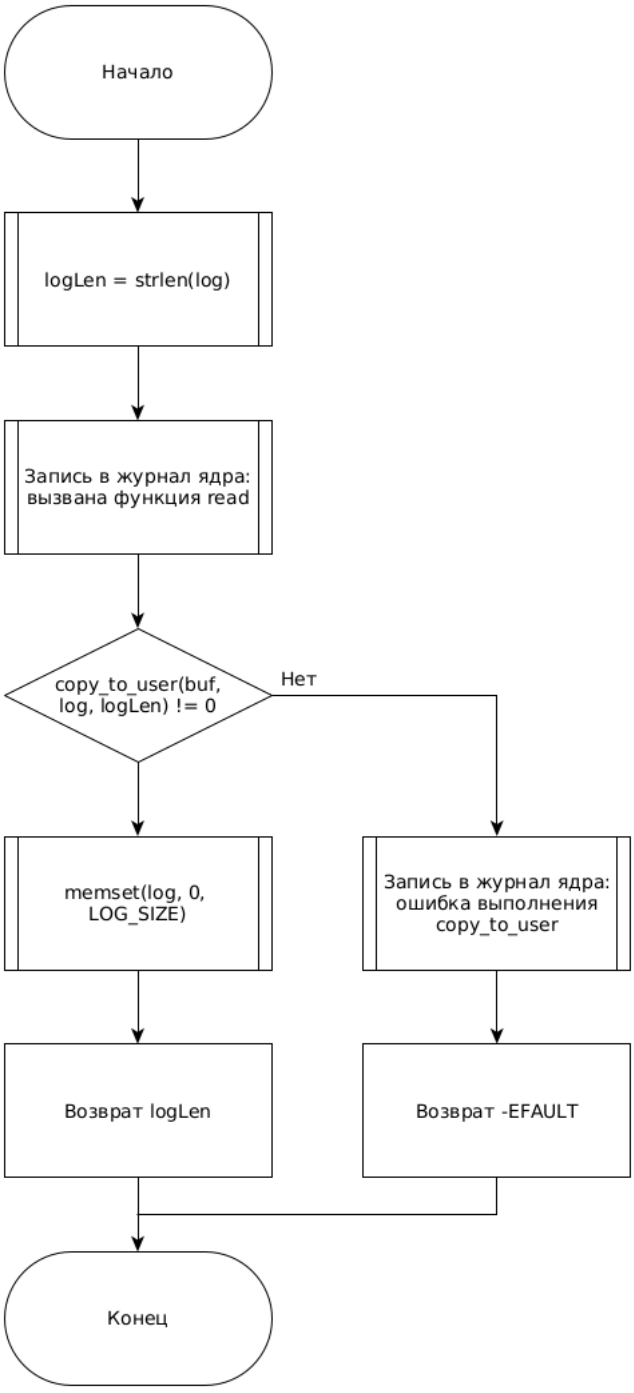


Рис. 2.3: Схема функции yaRead обработки чтения из /proc.

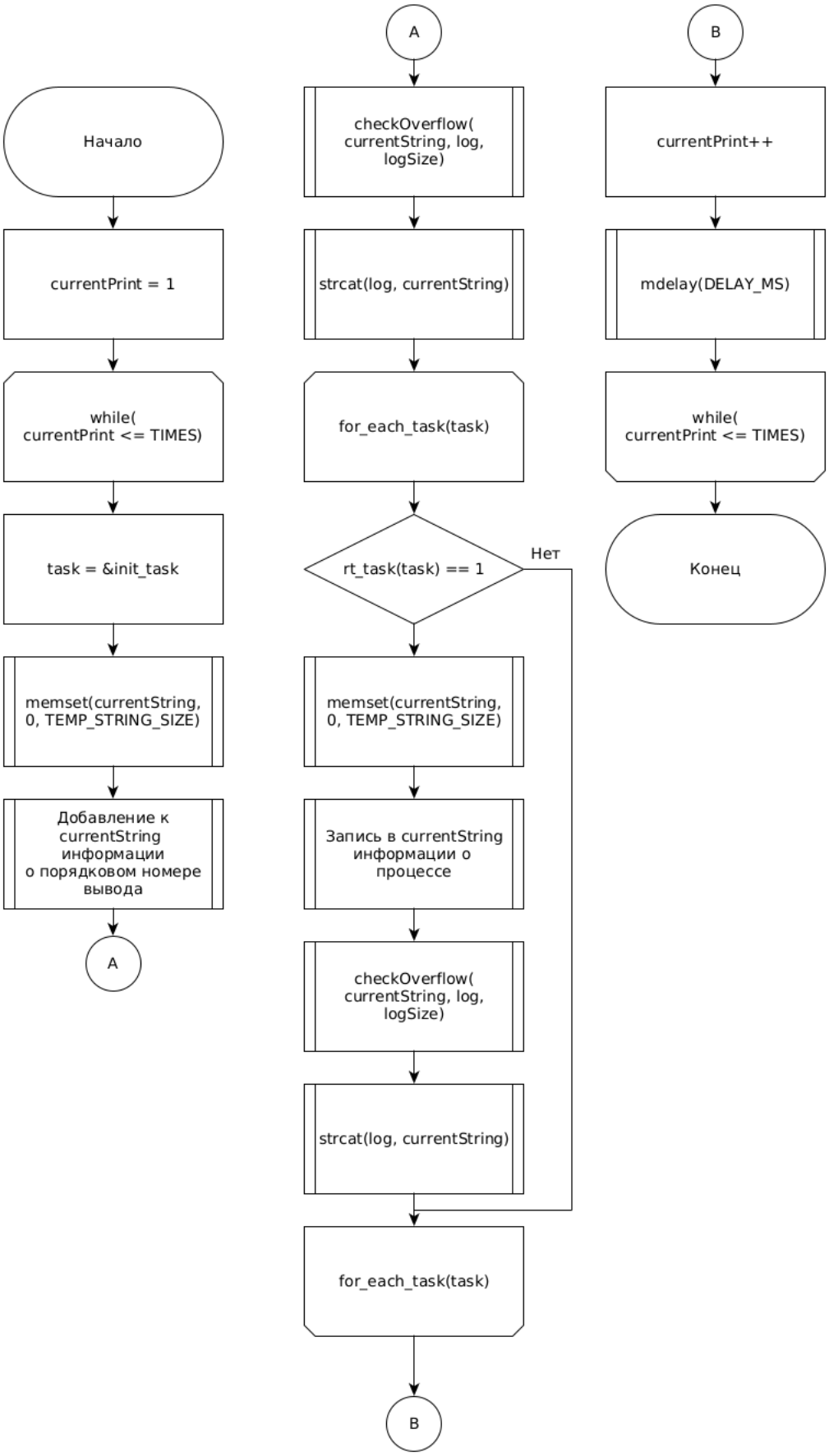


Рис. 2.4: Схема функции printTasks вывода информации о процессах реального времени.

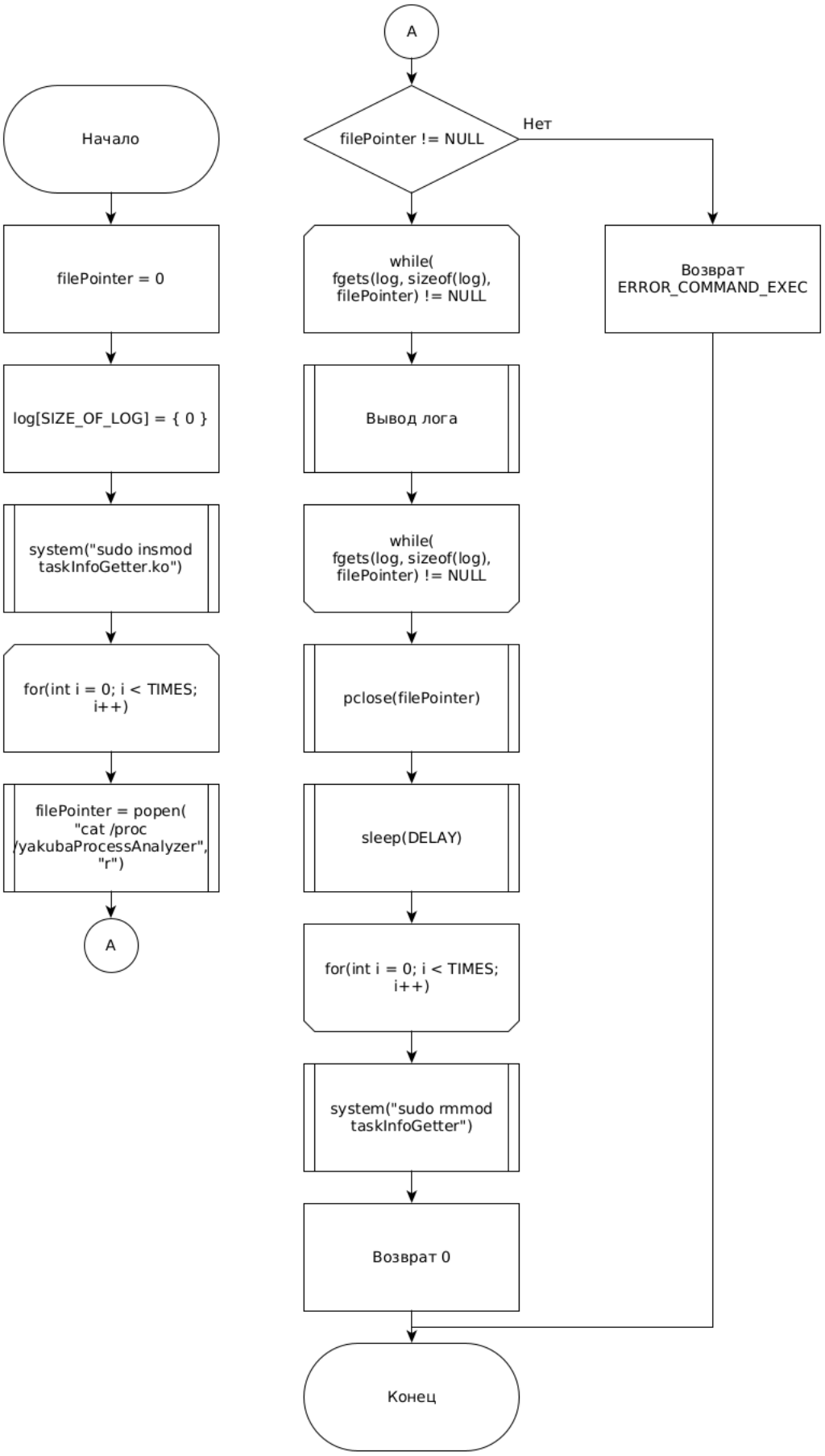


Рис. 2.5: Схема работы программы загрузки модуля и получения информации.

Вывод

В разделе были представлены требования к программному обеспечению. Было обозначено, что требуется реализовать загружаемый модуль ядра для мониторинга приоритетов, времени выполнения и простоя процессов. Также было решено реализовать дополнительную программу загрузки модуля в систему и вывода лога.

Были приведены основные сведения о компонентах, а также схемы, описывающие их работу.

3. Технологический раздел

В данном разделе представлен выбор и обоснование языка программирования и среды разработки, а также листинг реализованных компонентов. Также приведена демонстрация работы.

3.1 Выбор и обоснование языка программирования и среды разработки

При написании программного кода использовался язык программирования C [10].

В качестве среды разработки использовалась “Visual Studio Code” [11].

Данный выбор обусловлен следующими факторами:

- наличие плагинов для написания программ, работающих на уровне ядра,
- широкий функционал,
- ПО с открытым исходным кодом.

3.2 Программа загрузки модуля и получения информации

В листинге 19 предоставлена реализация программы загрузки модуля и получения информации.

Листинг 19: Программа загрузки модуля и получения информации.

3.3 Модуль логирования процессов реального времени

В листинге 20 предоставлена реализация модуля логирования процессов реального времени.

Листинг 20: Модуль логирования процессов реального времени.

3.4 Makefile

В листинге 21 предоставлено содержание Makefile для сборки компонентов.

Листинг 21: Содержание Makefile.

3.5 Демонстрация работы

На рисунках 3.1–3.3 предоставлена демонстрация работы программы загрузки модуля и получения информации.

На рисунке 3.4 предоставлено содержание журнала ядра при загрузке и выгрузке модуля из системы.

Рис. 3.1: Вывод при запуске starterLogger.c (первая итерация).

Рис. 3.2: Вывод при запуске starterLogger.c (вторая итерация).

Рис. 3.3: Вывод при запуске starterLogger.c (третья итерация).

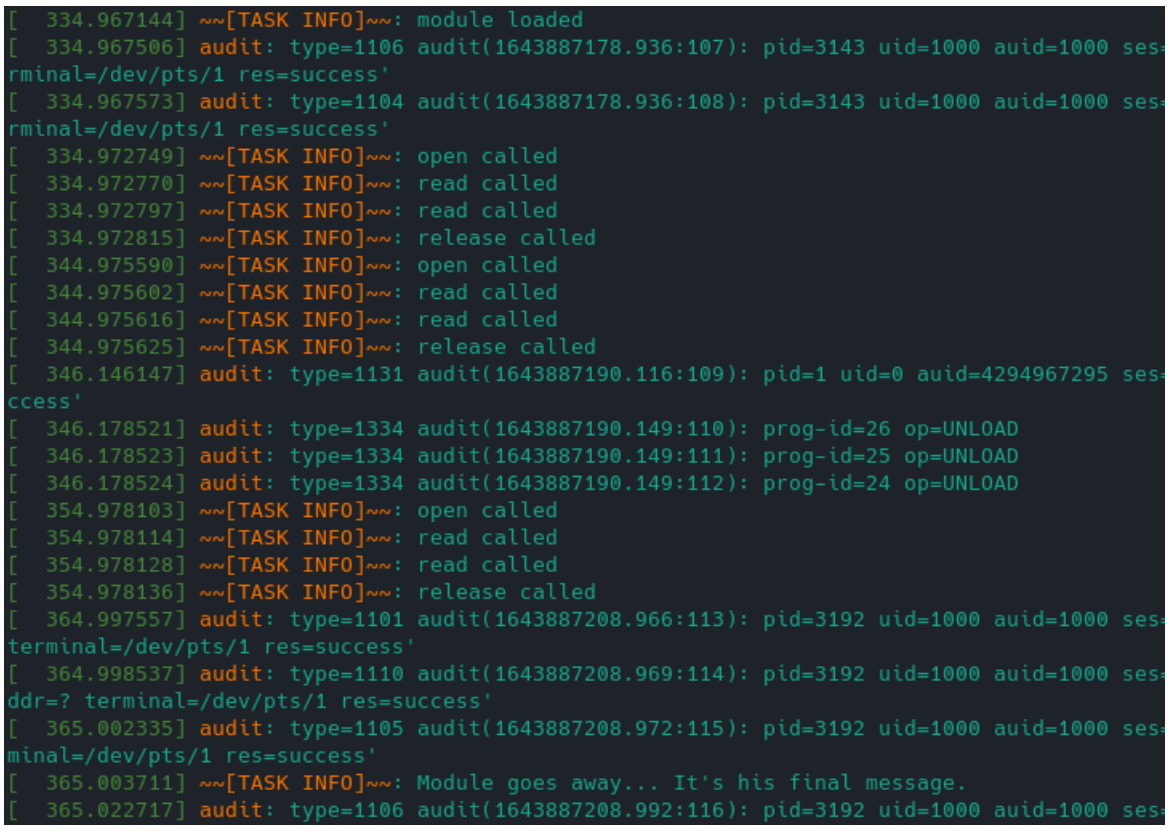


Рис. 3.4: Содержание журнала ядра.

Вывод

В качестве средств реализации был выбран язык программирования C. Используемая среда разработки – “Visual Studio Code”.

Были реализованы модуль логирования процессов реального времени и программа загрузки модуля и получения информации, представлены их листинги. Была проведена демонстрация работы реализованного программного обеспечения.

4. Исследовательский раздел

В данном разделе...

4.1 Технические характеристики системы

Исследование проводилось на персональном компьютере со следующими характеристиками:

- процессор Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz,
- операционная система Linux (дистрибутив Manjaro Linux, версия ядра Linux 5.13.19-2-MANJARO, архитектура x86-64),
- 16 Гб оперативной памяти.

Вывод разработанных компонентов сохранялся в отдельные текстовые файлы и в дальнейшем, для определения разницы, анализировался с использованием утилиты sdiff.

Для определения того, какие процессы задействуют директорию или файл, использовалась утилита lsof.

4.2 6 итераций вывода информации о процессах реального времени с разницей в 10 секунд при воспроизведении аудио с использованием MPlayer

Перед началом исследований было проверено, что аудиофайл не задействован никакими другими процессами, как это показано на рисунке 4.1.

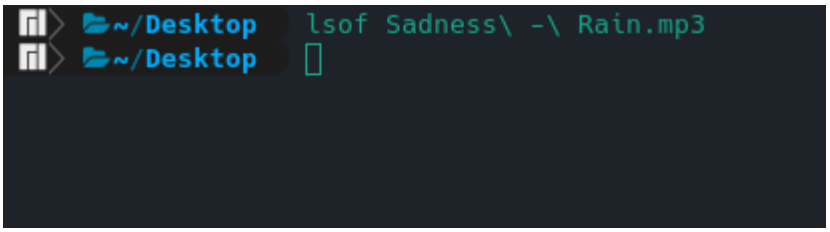


Рис. 4.1: Подтверждение того, что файл не задействован никакими другими процессами.

При запуске проигрывания с использованием MPlayer, утилита lsof указывает на то, что файл используется процессом с именем mplayer (рисунок 4.2).

Рис. 4.2: Результат запуска утилиты lsof при воспроизведении файла с использованием Mplayer.

При проведении исследования, первые 10 секунд композиция не запускается. Это связано с тем, что перерывы между получением информации о процессах в модуле заданы данным интервалом времени. Таким образом можно будет наблюдать возможное появление нового процесса, либо динамику изменения полей отдельных task_struct, отнесенных к задачам реального времени.

В силу того, что файл вывода содержит 1666 строк, его представление здесь нецелесообразно. При анализе результатов было обнаружено, что среди процессов, отнесенных к задачам реального времени, отсутствует процесс с идентификационным номером 18767. Причем, при использовании утилиты pstree у данного процесса обнаруживается лишь один потомок, у которого более нет потомков. Важно отметить также тот факт, что завершение потомка влечет за собой и завершение родительского процесса.

4.3 6 итераций вывода информации о процессах реального времени с разницей в 10 секунд при воспроизведении аудио с использованием VLC Media Player

Для достоверности результатов также будет попытка использовать VLC Media Player. Данное исследование проводилось также, как и предыдущее.

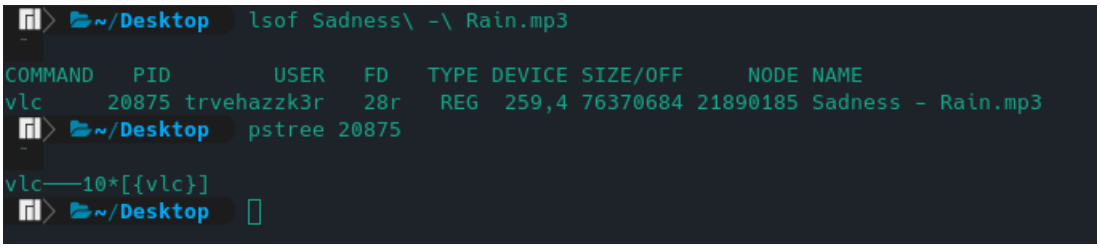


Рис. 4.3: Результат запуска утилит lsof и pstree при воспроизведении файла с использованием VLC Media Player.

В силу того, что файл вывода содержит 1666 строк, его представление здесь нецелесообразно. При анализе результатов было обнаружено отсутствие среди проанализированных процессов задачи с идентификационным

номером 20875.

Таким образом для дальнейшей работы потребуется проверить, совпадут ли данные выводы при использовании функции task_is_realtime, которая определяет принадлежность процесса к задачам реального времени по политике планировщика.

4.4 6 итераций вывода информации с использованием функции task_is_realtime с разницей в 10 секунд при воспроизведении аудио с использованием MPlayer и VLC Media Player

Исходный код загружаемого модуля был изменен, как это указано в листинге 22.

Листинг 22: Измененная часть вывода в модуле.

При анализе результатов было обнаружено отсутствие среди приведенных процессов задачи с идентификационным номером, совпадающим с номером процессов, отвечающих за воспроизведение ни в случае использования MPlayer, ни в случае использования VLC Media Player.

Таким образом для дальнейшей работы потребуется убрать условие выборки определения принадлежности процесса к задачам реального времени и проанализировать поля структуры task_struct.

4.5 6 итераций вывода информации о всех процессах в системе с разницей в 10 секунд при воспроизведении аудио с использованием VLC Media Player.

Исходный код загружаемого модуля был изменен, как это указано в листинге 23.

Листинг 23: Измененная часть вывода в модуле.

При проведении исследования производились две остановки воспроизведения на 5 секунде и 15 секунде. Время каждой остановки составляло 6–7 секунд. Процесс, отвечающий за воспроизведение, имел идентификационный номер 7702. В итоге, в выводе была получена информация, предоставленная в листинге 25.

Листинг 24: Информация о процессе vlc.

Видно, что со временем динамика отсутствует. Таким образом, можно сделать вывод, что, либо процесс был определен неверно, либо сам VLC Media Player обладает более сложным поведением.

4.6 6 итераций вывода информации о всех процессах в системе с разницей в 10 секунд при воспроизведении аудио с использованием MPlayer.

При воспроизведении остановок не производилось. Процесс, отвечающий за воспроизведение, имел идентификационный номер 9126. При этом он имел дочерний процесс 9127. В итоге была получена информация, предоставленная в листинге **??**.

Листинг 25: Информация о процессе mplayer.

Из предоставленной информации можно видеть, что в структуре изменяются поля delay, utime и stime. Причем значение поля static_prio равняется значению поля normal_prio, что означает, что процесс не относится к задаче реального времени.

В динамике заметно возрастание времени, проведенного в режиме пользователя и затраченное на запуск команд (utime), а также времени процессора, затраченного на выполнение системных вызовов при исполнении процесса (stime). В среднем за каждые 10 секунд utime увеличивался на 105 340 32 тика, в то время как stime в среднем увеличивался на 4 438 620.5 тика, что на ≈ 58 % меньше.

Важно отметить, что никакой информации о процессе с номером 9127 обнаружено не было.

В сложившейся ситуации также было проверено, что MPlayer является клиентом PulseAudio (многофункциональный звуковой сервер, предназначенный для работы в качестве прослойки между приложениями и аппаратными устройствами, либо ALSA (Advanced Linux Sound Architecture) или OSS (Open Sound System)), с использованием утилиты pacmd, в которой была вызвана команда list-clients. Результат вывода предоставлен в листинге 26. Идентификационный номер также совпал.

Листинг 26: Информация, полученная с использованием утилиты pacmd.

Полученные результаты говорят о том, что сами программы воспроизведения не создают процессы, относящиеся к задачам реального времени. В источнике [12] говорится о том, что планирование SCHED_FIFO может быть использовано лишь для потоков ввода-вывода. Таким образом, только потоки ввода-вывода PulseAudio выполняются в режиме реального времени, а управляющий поток, так называемый MainLoop, обычно остается обычным

потоком.

Из предоставленной информации становится ясно, что потребуется провести анализ потоков, создаваемых при воспроизведении аудио.

4.7 Исследование потоков, создаваемых MPlayer.

При запуске воспроизведения команда ps может позволить получить информацию о потоках приложения, что предоставлено на рисунке 4.4.

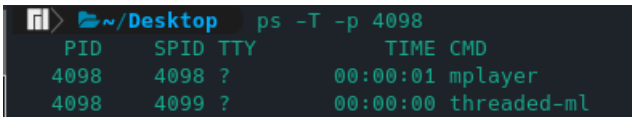


Рис. 4.4: Информация о потоках приложения.

На предоставленном изображении 4.4 SPID – это идентификационный номер потока. Таким образом, потребуется получить некоторую информацию о потоке, которая может быть предоставлена утилитой top.

Как можно видеть из вывода утилиты top 4.5, которая в реальном времени выводит информацию о процессах, ни процесс pulseaudio, ни процесс mplayer не имеют приоритета (PR), равного значению rt. На изображении PR обозначает приоритет планирования задачи, где rt будет означать, что процесс относится к классу задач реального времени. Однако можно заметить, что приоритет для pulseaudio имеет значение 9, в отличие от mplayer. Таким образом, pulseaudio имеет приоритет выполнения выше.

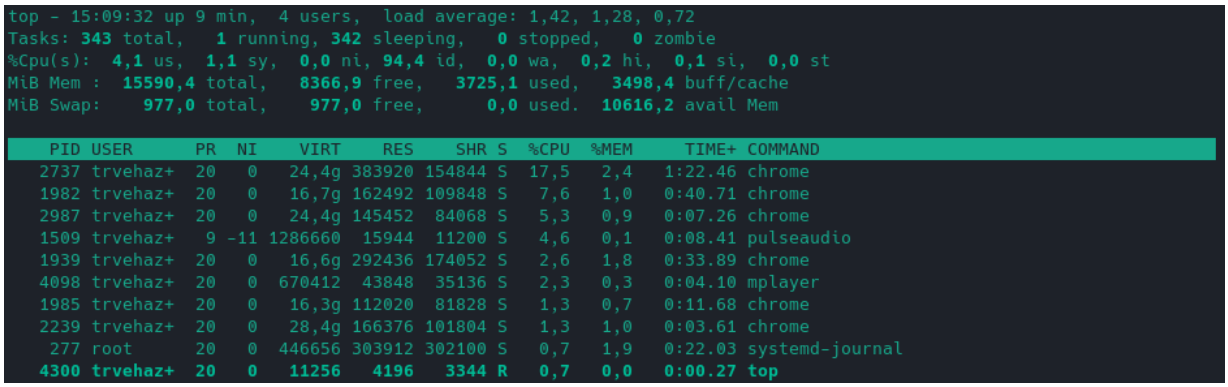


Рис. 4.5: Информация, полученная с использованием утилиты top.

Также данная утилита может использоваться для вывода информации о потоках в системе, что предоставлено на картинке 4.6.

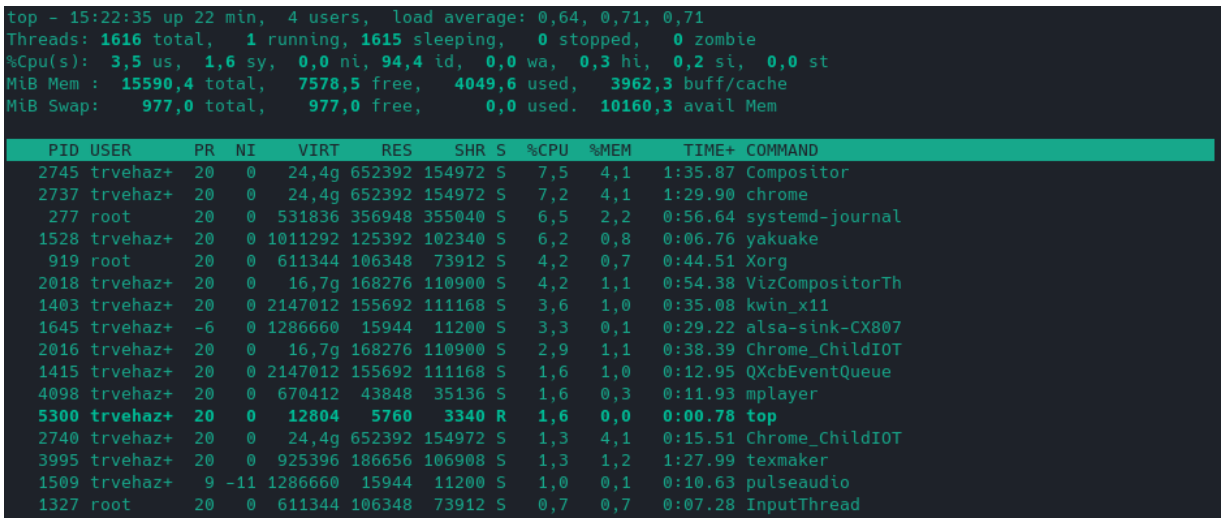
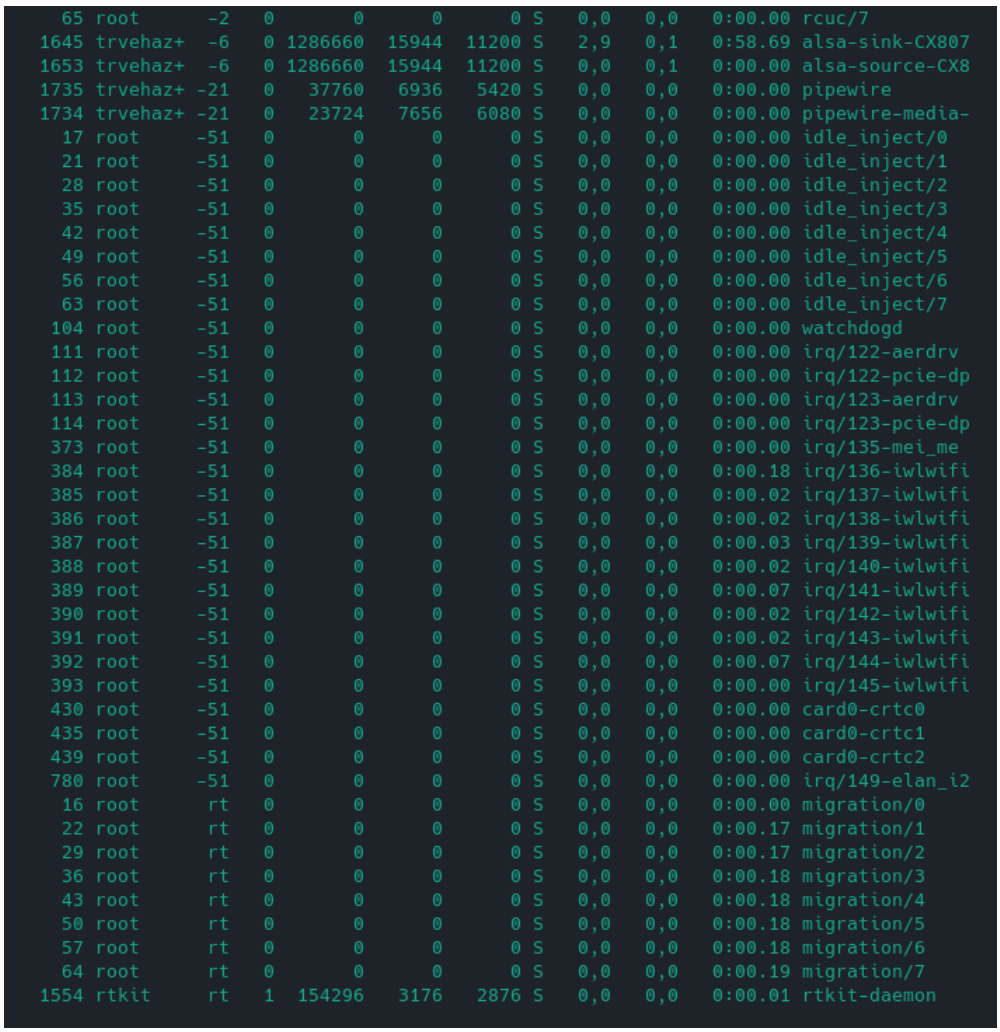


Рис. 4.6: Информация о потоках, полученная с использованием утилиты top.

На изображении видно присутствие в системе потока, относящегося к команде alsa-sink-CX807 с отрицательным приоритетом выполнения. Согласно документации PulseAudio [13] модуль alsa-sink отвечает за обработку воспроизведения на устройствах, поддерживающих ALSA, то есть с использованием звуковой карты.

На изображении 4.7 можно увидеть, что в текущий момент в системе присутствует лишь 9 потоков с приоритетом rt. Причём потоки alsa-sink-CX807 и alsa-source-CX8 выполняются с одними из наивысших приоритетов в системе при факте того, что всего в выводе присутствует 1599 потоков.



```
   65 root      -2   0       0      0      0 S   0,0   0,0   0:00.00 rcuc/7
 1645 trvehaz+  -6   0 1286660  15944  11200 S   2,9   0,1   0:58.69 alsa-sink-CX807
 1653 trvehaz+  -6   0 1286660  15944  11200 S   0,0   0,1   0:00.00 alsa-source-CX8
 1735 trvehaz+ -21   0   37760   6936   5420 S   0,0   0,0   0:00.00 pipewire
 1734 trvehaz+ -21   0   23724   7656   6080 S   0,0   0,0   0:00.00 pipewire-media-
   17 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/0
   21 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/1
   28 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/2
   35 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/3
   42 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/4
   49 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/5
   56 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/6
   63 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/7
  104 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 watchdogd
  111 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 irq/122-aerdrv
  112 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 irq/122-pcie-dp
  113 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 irq/123-aerdrv
  114 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 irq/123-pcie-dp
  373 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 irq/135-mei_me
  384 root     -51   0       0      0      0 S   0,0   0,0   0:00.18 irq/136-iwlwifi
  385 root     -51   0       0      0      0 S   0,0   0,0   0:00.02 irq/137-iwlwifi
  386 root     -51   0       0      0      0 S   0,0   0,0   0:00.02 irq/138-iwlwifi
  387 root     -51   0       0      0      0 S   0,0   0,0   0:00.03 irq/139-iwlwifi
  388 root     -51   0       0      0      0 S   0,0   0,0   0:00.02 irq/140-iwlwifi
  389 root     -51   0       0      0      0 S   0,0   0,0   0:00.07 irq/141-iwlwifi
  390 root     -51   0       0      0      0 S   0,0   0,0   0:00.02 irq/142-iwlwifi
  391 root     -51   0       0      0      0 S   0,0   0,0   0:00.02 irq/143-iwlwifi
  392 root     -51   0       0      0      0 S   0,0   0,0   0:00.07 irq/144-iwlwifi
  393 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 irq/145-iwlwifi
  430 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 card0-crtc0
  435 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 card0-crtc1
  439 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 card0-crtc2
  780 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 irq/149-elan_i2
   16 root      rt   0       0      0      0 S   0,0   0,0   0:00.00 migration/0
   22 root      rt   0       0      0      0 S   0,0   0,0   0:00.17 migration/1
   29 root      rt   0       0      0      0 S   0,0   0,0   0:00.17 migration/2
   36 root      rt   0       0      0      0 S   0,0   0,0   0:00.18 migration/3
   43 root      rt   0       0      0      0 S   0,0   0,0   0:00.18 migration/4
   50 root      rt   0       0      0      0 S   0,0   0,0   0:00.18 migration/5
   57 root      rt   0       0      0      0 S   0,0   0,0   0:00.18 migration/6
   64 root      rt   0       0      0      0 S   0,0   0,0   0:00.19 migration/7
 1554 rtkit     rt   1  154296   3176   2876 S   0,0   0,0   0:00.01 rtkit-daemon
```

Рис. 4.7: Информация о потоках, полученная с использованием утилиты top.

Занятен факт присутствия в выводе rtkit-daemon, которая является системной службой, меняющей по запросу политику планирования пользовательских процессов и потоков на SCHED_RR, то есть режим планирования в реальном времени. Отсюда и возникает факт того, что в системе сложно увидеть, что процесс проигрывания может выполняться в реальном времени, так как RealtimeKit предназначен для использования в качестве безопасного механизма, позволяющего обычным пользовательским процессам частично являться задачами реального времени.

Также из представленного изображения можно увидеть, что в системе присутствует пользователь rtkit.

Вывод

В разделе...

ЗАКЛЮЧЕНИЕ

Во время выполнения курсового проекта были достигнуты поставленные задачи:

- хоп,
- хоп,
- хоп,
- хоп,
- хоп.

Проведённая аналитическая работа позволила...

В результате работы, проведенной в конструкторском разделе, были приведены... Также была определена схема работы...

Для реализации в качестве используемого языка программирования был выбран ЯП ..., а в качестве среды разработки – ...

В результате работы было...

В ходе выполнения поставленных задач были получены знания в области ..., а также изучены...

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Список литературы

1. Fatal Errors: Linux Process Scheduling-Real-Time Scheduler Learning Notes [Электронный ресурс]. Режим доступа: https://www.fatalerrors.org/a/linux-process-scheduling-real-time-scheduler_learning-notes.html (дата обращения 20.12.2021).

2. The Linux Kernel documentation [Электронный ресурс]. Режим доступа: https://www.kernel.org (дата обращения 20.12.2021).

3. А. Кирьянов Д. Модель процессов в современных операционных системах и их реализация в ОС Linux // Сервис в России и за рубежом. 2007. № 1.

4. Проект Losst [Электронный ресурс]. Режим доступа: https://losst.ru (дата обращения 20.12.2021).

5. Programmer All: Linux kernel learning notes (6) [Электронный ресурс]. Режим доступа: https://www.programmerall.com/article/63151049863/ (дата обращения 20.12.2021).

6. Разработка и внедрение системы на встраиваемом Linux: Планирование процессов [Электронный ресурс]. Режим доступа: http://dmilvdv.narod.ru/Translate/ELSDD/index.html (дата обращения 20.12.2021).

7. Проект OpenNET [Электронный ресурс]. Режим доступа: https://www.opennet.ru/opennews/art.shtml?num=38906 (дата обращения 20.12.2021).

8. Таненбаум Х. Бос Х. Современные операционные системы. СПб.: Питер, 2015. с. 1120.

9. Лекции университета Бернингема [Электронный ресурс]. Режим доступа: https://www.birmingham.ac.uk/schools/computer-science/index.aspx (дата обращения 22.12.2021).

10. Спецификация языка C [Электронный ресурс]. Режим доступа: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf (дата обращения 22.12.2021).

11. Документация Visual Studio Code [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs (дата обращения 22.12.2021).

12. Hacker News [Электронный ресурс]. Режим доступа: https://news.ycombinator.com/item?id=21926545#:~:text=Please%20note%20that%20only%20the,high%2Dpriority%20option%20is%20orthogonal. (дата обращения 10.01.2022).

13. FreeDesctop.org [Электронный ресурс]. Режим доступа: https://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/User/Modules/#module-alsa-sink (дата обращения 10.01.2022).

ПРИЛОЖЕНИЕ 1

Листинг 27: Структура ядра task_struct