

Оглавление

Введение	4
1 Аналитический раздел	5
1.1 Постановка задачи	5
1.2 Описание работы RT Scheduler	5
1.3 Анализ структур ядра, предоставляющих информацию о приоритетах, времени выполнения и простоя процессов	6
1.3.1 task_struct	6
1.3.2 sched_info	11
1.4 Анализ способов определения принадлежности задачи к группе задач реального времени	12
1.4.1 Функция rt_prio	12
1.4.2 Функция rt_task	12
1.4.3 Функция task_is_realtime	13
1.5 Передача данных из пространства ядра в пространство пользователя	13
2 Конструкторский раздел	16
2.1 Диаграммы состояний (IDEF0)	16
2.2 Алгоритмы определения и логирования приоритетов, времени выполнения и простоя процессов	16
2.3 Алгоритм предоставления информации о процессах пользователю	18
2.4 Структура программного обеспечения	20
3 Технологический раздел	21
3.1 Выбор и обоснование языка и среды программирования	21
3.2 Реализация алгоритмов определения и логирования приоритетов, времени выполнения и простоя процессов.	21
3.3 Реализация алгоритма предоставления информации о процессах пользователю	23
3.4 Makefile	25
3.5 Демонстрация работы	25
4 Исследовательский раздел	30
4.1 Условия исследований	30
4.2 6 итераций вывода информации о процессах реального времени с разницей в 10 секунд при воспроизведении аудио с использованием MPlayer	30
4.3 6 итераций вывода информации о процессах реального времени с разницей в 10 секунд при воспроизведении аудио с использованием VLC Media Player	31
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	32

4.5	6 итераций вывода информации о всех процессах в системе с разницей в 10 секунд при воспроизведении аудио с использованием VLC Media Player. . . .	33
4.6	6 итераций вывода информации о всех процессах в системе с разницей в 10 секунд при воспроизведении аудио с использованием MPlayer.	36
4.7	Исследование потоков, создаваемых MPlayer.	39
4.8	Исследование потоков при проигрывании видео с использованием VLC Media Player	41
ЗАКЛЮЧЕНИЕ		43
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		44
ПРИЛОЖЕНИЕ А. Исходный код программы		46

ВВЕДЕНИЕ

Для системы имеют большое значение приоритеты, время простоя и выполнения процессов. Эти значения используются для планирования. За планирование процессов в операционных системах отвечают планировщики задач, каждый из которых может использовать собственный уникальный алгоритм планирования процессов.

Linux является операционной системой 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 приведена 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.

Данная курсовая работа посвящена вопросам определения приоритетов, времени простоя выполнения процессов в операционной системе Linux, а также их мониторингу.

1. Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу по курсу “Операционные Системы” необходимо разработать загружаемый модуль ядра, позволяющий отслеживать состояния приоритетов, времени выполнения и простоя процессов в операционной системе Linux и проанализировать с использованием данного модуля процессы реального времени.

Для решения поставленной задачи необходимо:

- 1) проанализировать и выбрать структуры ядра, содержащие необходимую информацию;
- 2) проанализировать и выбрать методы передачи информации из модуля ядра в пространство пользователя;
- 3) разработать алгоритмы и структуру программного обеспечения;
- 4) проанализировать разработанное программное обеспечение.

1.2 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 нескольких 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 исполняться только в свое выделенное время. [2]

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

Рассмотрим пример: 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 груп-

пы 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 (Earliest Deadline First scheduling, алгоритм планирования по ближайшему сроку завершения) для использования неоднородных периодов. [2]

1.3 Анализ структур ядра, предоставляющих информацию о приоритетах, времени выполнения и простоя процессов

1.3.1 task_struct

Структура в ядре Linux, соответствующая каждому процессу, – `task_struct`. Она определена в файле `linux/sched.h`. [3]

Далее будут отмечены требующиеся в работе поля данной структуры (в Linux v5.16rc8).

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 его [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 процесса (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 описаны функции определения приоритета процесса, которые приведены в листинге 1.

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

```

1  #include "sched_idletask.c"
2  #include "sched_fair.c"
3  #include "sched_rt.c"
4  #ifdef CONFIG_SCHED_DEBUG
5  #include "sched_debug.c"
6  #endif
7
8  /*
9   * __normal_prio - return the priority that is based on
10   ↪ the static prio
11   */
12 static inline int __normal_prio(struct task_struct *p) //
13 ↪ _NORMAL_PRIO function, return static priority value
14 {
15     return p->static_prio;
16 }
17
18 /*
19  * Calculate the expected normal priority: i.e. priority
20  * without taking RT-inheritance into account. Might be
21  * boosted by interactivity modifiers. Changes upon fork,
22  * setprio syscalls, and whenever the interactivity
23  * estimator recalculates.
24  */
25 static inline int normal_prio(struct task_struct *p) //
26 ↪ NORMAL_PRIO function
27 {
28     int prio;
29
30     if (task_has_rt_policy(p)) // The task_has_rt_policy
31 ↪ function, the determination process is a real-time
32 ↪ process, if the real-time process, returns 1,
33 ↪ otherwise returns 0

```

```

28     prio = MAX_RT_PRIO-1 - p->rt_priority; // The
        ↳ process is real-time process, and the PRIO
        ↳ value is related to the real-time priority
        ↳ value: PRIO = MAX_RT_PRIO -1 - P-> rt_priority
29 else
30     prio = __normal_prio(p); // The process is a
        ↳ non-real-time process, then the PRIO value is
        ↳ a static priority value, that is, PRIO = P->
        ↳ static_prio
31     return prio;
32 }
33
34 /*
35  * Calculate the current priority, i.e. the priority
36  * taken into account by the scheduler. This value might
37  * be boosted by RT tasks, or might be boosted by
38  * interactivity modifiers. Will be RT if the task got
39  * RT-boosted. If not then it returns p->normal_prio.
40  */
41 static int effective_prio(struct task_struct *p) // The
        ↳ Effective_Prio function, the effective priority of the
        ↳ calculation process, the PRIO value, this value is the
        ↳ priority value used by the final scheduler
42 {
43     p->normal_prio = normal_prio(p); // Calculate the
        ↳ value of Normal_PRIO
44     /*
45      * If we are RT tasks or we were boosted to RT
        ↳ priority,
46      * keep the priority unchanged. Otherwise, update
        ↳ priority
47      * to the normal priority:
48      */
49     if (!rt_prio(p->prio))
50         return p->normal_prio; // If the process is a
        ↳ non-real-time process, return normal_prio
        ↳ value, at this time Normal_Prio = Static_Prio
51     return p->prio; // Otherwise, the return value is
        ↳ constant, still PRIO value, at this time, PRIO =
        ↳ MAX_RT_PRIO -1 - P-> RT_Priority
52 }

```

```

53
54  /*****
55  void set_user_nice(struct task_struct *p, long nice)
56  {
57      ...
58      p->prio = effective_prio(p); // In the function
        ↳ set_user_nice, call the Effective_Prio function to
        ↳ set the process's PRIO value.
59      ...
60  }

```

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`.

```

1  /*
2   * Convert user-nice values [ -20 ... 0 ... 19 ]
3   * to static priority [ MAX_RT_PRIO..MAX_PRIO-1 ],
4   * and back.
5   */
6  #define NICE_TO_PRIO(nice)      (MAX_RT_PRIO + (nice) + 20)
7  #define PRIO_TO_NICE(prio)      ((prio) - MAX_RT_PRIO - 20)
8  #define TASK_NICE(p)            PRIO_TO_NICE((p)->static_prio)
9
10 /*
11  * 'User priority' is the nice value converted to
12  * ↳ something we
13  * can work with better when scaling various scheduler
14  * ↳ parameters,
15  * it's a [ 0 ... 39 ] range.
16  */
17 #define USER_PRIO(p)            ((p) - MAX_RT_PRIO)
18 #define TASK_USER_PRIO(p)       USER_PRIO((p)->static_prio)

```



```

17  #define MAX_USER_PRIO          (USER_PRIO(MAX_PRIO))
18
19  / *** /
20  p->static_prio = NICE_TO_PRIO(nice);

```

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.

```

1  static inline int normal_prio(struct task_struct *p) //
   ↳ NORMAL_PRIO function
2  {
3      int prio;
4
5      if (task_has_rt_policy(p)) // The task_has_rt_policy
   ↳ function, the determination process is a real-time
   ↳ process, if the real-time process, returns 1,
   ↳ otherwise returns 0
6          prio = MAX_RT_PRIO-1 - p->rt_priority; // The
   ↳ process is real-time process, and the PRIO
   ↳ value is related to the real-time priority
   ↳ value: PRIO = MAX_RT_PRIO -1 - P-> rt_priority
7      else
8          prio = __normal_prio(p); // The process is a
   ↳ non-real-time process, then the PRIO value is
   ↳ a static priority value, that is, PRIO = P->
   ↳ static_prio
9      return prio;

```

```
10     }
```

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

1.3.2 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 4.

Листинг 4: Структура `sched_info`.

```
1 struct sched_info {
2 #ifdef CONFIG_SCHED_INFO
3     /* Cumulative counters: */
4
5     /* # of times we have run on this CPU: */
6     unsigned long          pcount;
7
8     /* Time spent waiting on a runqueue: */
9     unsigned long long      run_delay;
10
11     /* Timestamps: */
12
13     /* When did we last run on a CPU? */
14     unsigned long long      last_arrival;
15
16     /* When were we last queued to run? */
17     unsigned long long      last_queued;
18
19 #endif /* CONFIG_SCHED_INFO */
20 };
```

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 Анализ способов определения принадлежности задачи к группе задач реального времени

1.4.1 Функция `rt_prio`

Реализация функции `rt_prio` приведена в листинге 5.

Листинг 5: Функция `rt_prio`.

```
1 static inline int rt_prio(int prio)
2 {
3     if (unlikely(prio < MAX_RT_PRIO))
4         return 1;
5     return 0;
6 }
```

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.

1.4.2 Функция `rt_task`

Реализация данной функции приведена в листинге 6.

Листинг 6: Функция `rt_task`.

```
1 static inline int rt_task(struct task_struct *p)
2 {
```

```

3     return rt_prio(p->prio);
4 }

```

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

1.4.3 Функция `task_is_realtime`

Реализация данной функции приведена в листинге 7.

Листинг 7: Функция `task_is_realtime`.

```

1  static inline bool task_is_realtime(struct task_struct
2  ↪   *tsk)
3  {
4      int policy = tsk->policy;
5
6      if (policy == SCHED_FIFO || policy == SCHED_RR)
7          return true;
8      if (policy == SCHED_DEADLINE)
9          return true;
10     return false;
11 }

```

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

1.5 Передача данных из пространства ядра в пространство пользователя

В Linux для передачи данных из пространства ядра в пространство пользователя зачастую используется виртуальная файловая система `procfs`, которая предоставляет системные вызовы для реализации интерфейса между двумя этими пространствами.

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 8.

Листинг 8: Структура proc_ops.

```
1 struct proc_ops {
2     unsigned int proc_flags;
3     int (*proc_open)(struct inode *, struct file *);
4     ssize_t (*proc_read)(struct file *, char __user *,
5         ↪ size_t, loff_t *);
6     ssize_t (*proc_read_iter)(struct kiocb *, struct
7         ↪ iov_iter *);
8     ssize_t (*proc_write)(struct file *, const char
9         ↪ __user *, size_t, loff_t *);
10    /* mandatory unless nonseekable_open() or equivalent
11        ↪ is used */
12    loff_t (*proc_lseek)(struct file *, loff_t, int);
13    int (*proc_release)(struct inode *, struct file *);
14    __poll_t (*proc_poll)(struct file *, struct
15        ↪ poll_table_struct *);
16    long (*proc_ioctl)(struct file *, unsigned int,
17        ↪ unsigned long);
18    #ifdef CONFIG_COMPAT
19        long (*proc_compat_ioctl)(struct file *, unsigned
20            ↪ int, unsigned long);
21    #endif
22    int (*proc_mmap)(struct file *, struct
23        ↪ vm_area_struct *);
24    unsigned long (*proc_get_unmapped_area)(struct file *,
25        ↪ unsigned long, unsigned long, unsigned long,
26        ↪ unsigned long);
27 } __randomize_layout;
```

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 9.

Листинг 9: Реализация функции `copy_to_user`.

```
1 static __always_inline unsigned long __must_check
```

```

2 | copy_to_user(void __user *to, const void *from, unsigned
   | ↪ long n)
3 | {
4 |     if (likely(check_copy_size(from, n, true)))
5 |         n = _copy_to_user(to, from, n);
6 |     return n;
7 | }

```

Выводы

В результате анализа кода ядра определены структуры, содержащие необходимую информацию: `task_struct` (поля `pid`, `prio`, `static_prio`, `normal_prio`) и `sched_info` (поля `run_delay`, `utime`, `stime`). Также были определены функции, позволяющие определить, относится ли процесс к задачам реального времени: `rt_prio`, `rt_task` и `task_is_realtime`.

2. Конструкторский раздел

2.1 Диаграммы состояний (IDEF0)

На рисунках 2.1 и 2.2 показаны соответственно нулевой и первый уровни диаграммы IDEF0, отображающие процесс мониторинга приоритетов, времени выполнения и простоя процессов.

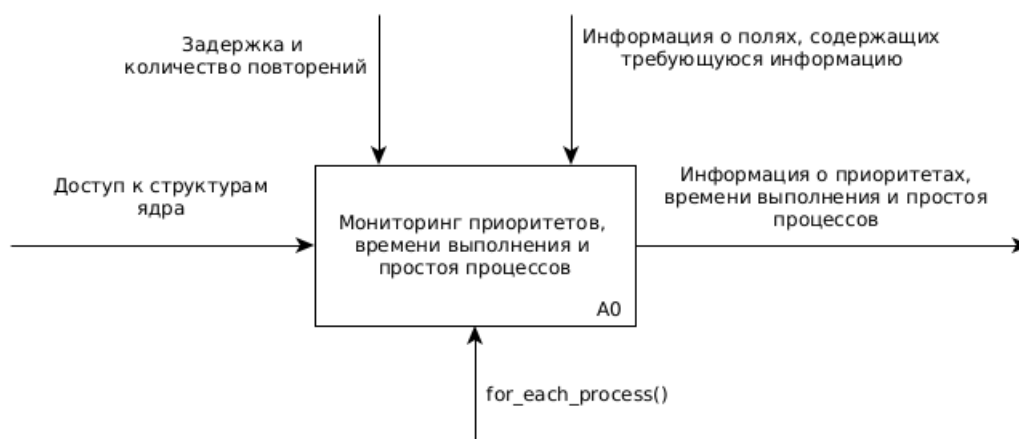


Рис. 2.1: IDEF0 нулевого уровня.

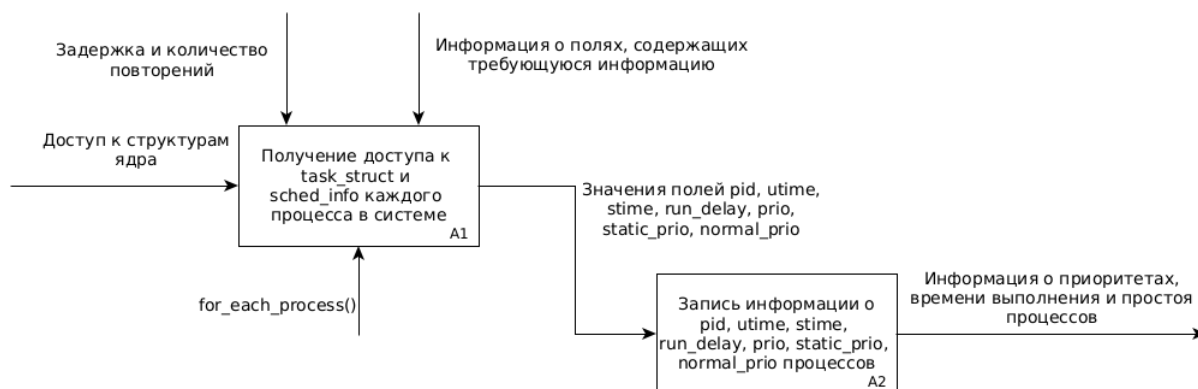


Рис. 2.2: IDEF0 первого уровня.

2.2 Алгоритмы определения и логирования приоритетов, времени выполнения и простоя процессов

На рисунках 2.4 и 2.3 представлены алгоритмы определения и логирования приоритетов, времени выполнения и простоя процессов.

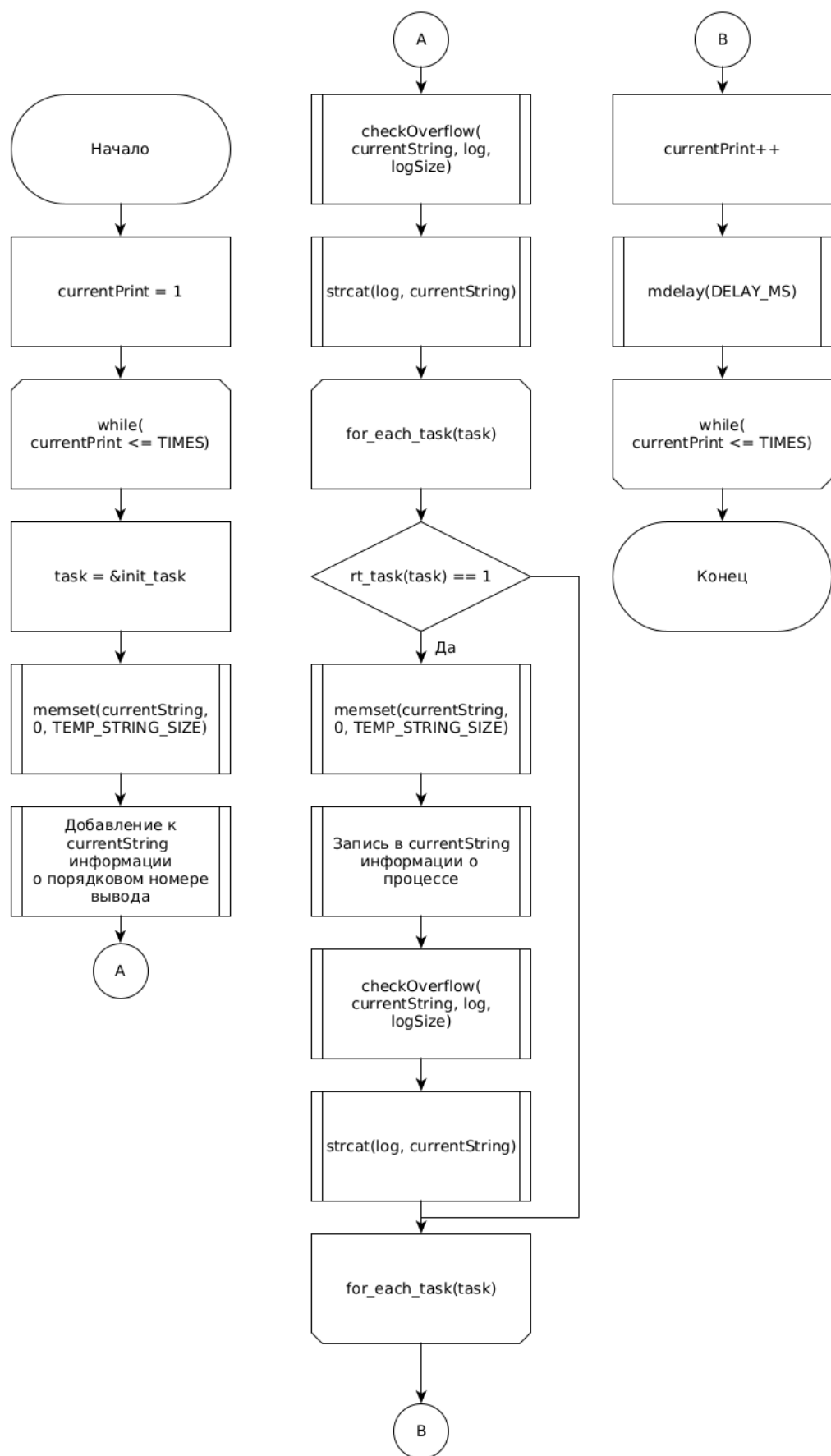


Рис. 2.3: Алгоритм определения приоритетов, времени выполнения и простоя процессов.

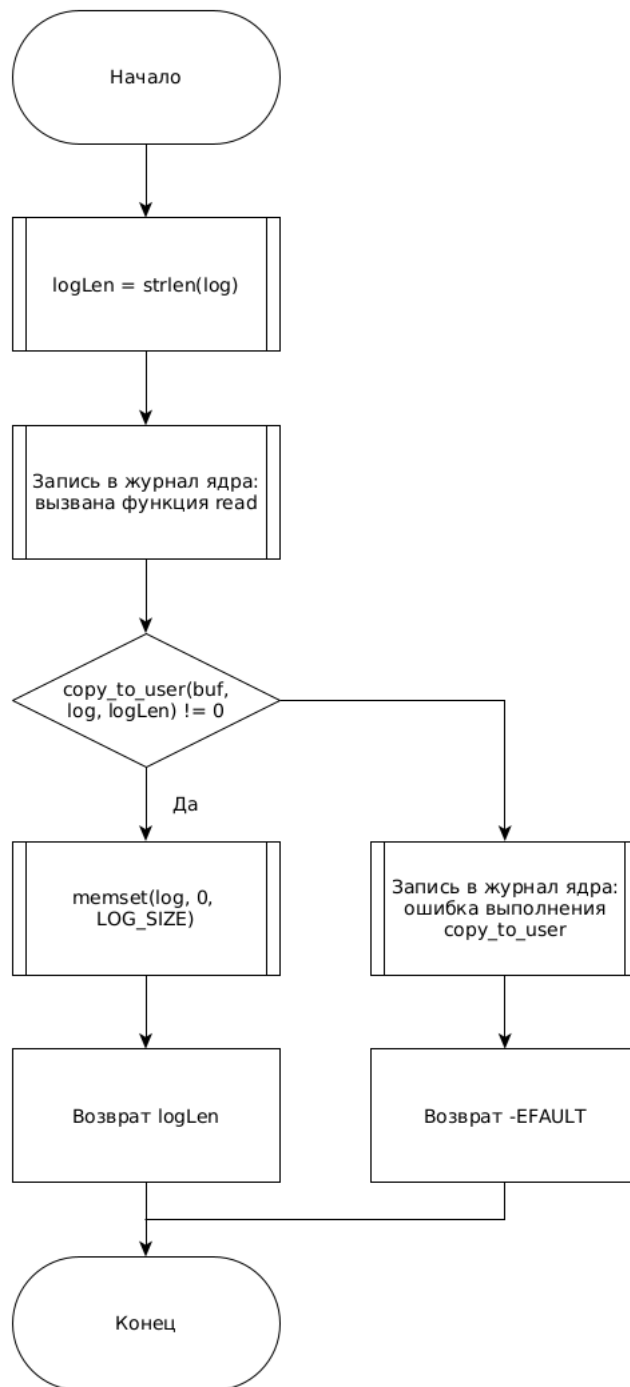


Рис. 2.4: Алгоритм логирования приоритетов, времени выполнения и простоя процессов.

2.3 Алгоритм предоставления информации о процессах пользователю

На рисунке 2.5 представлен алгоритм предоставления информации о приоритетах, времени выполнения и простоя процессов пользователю.

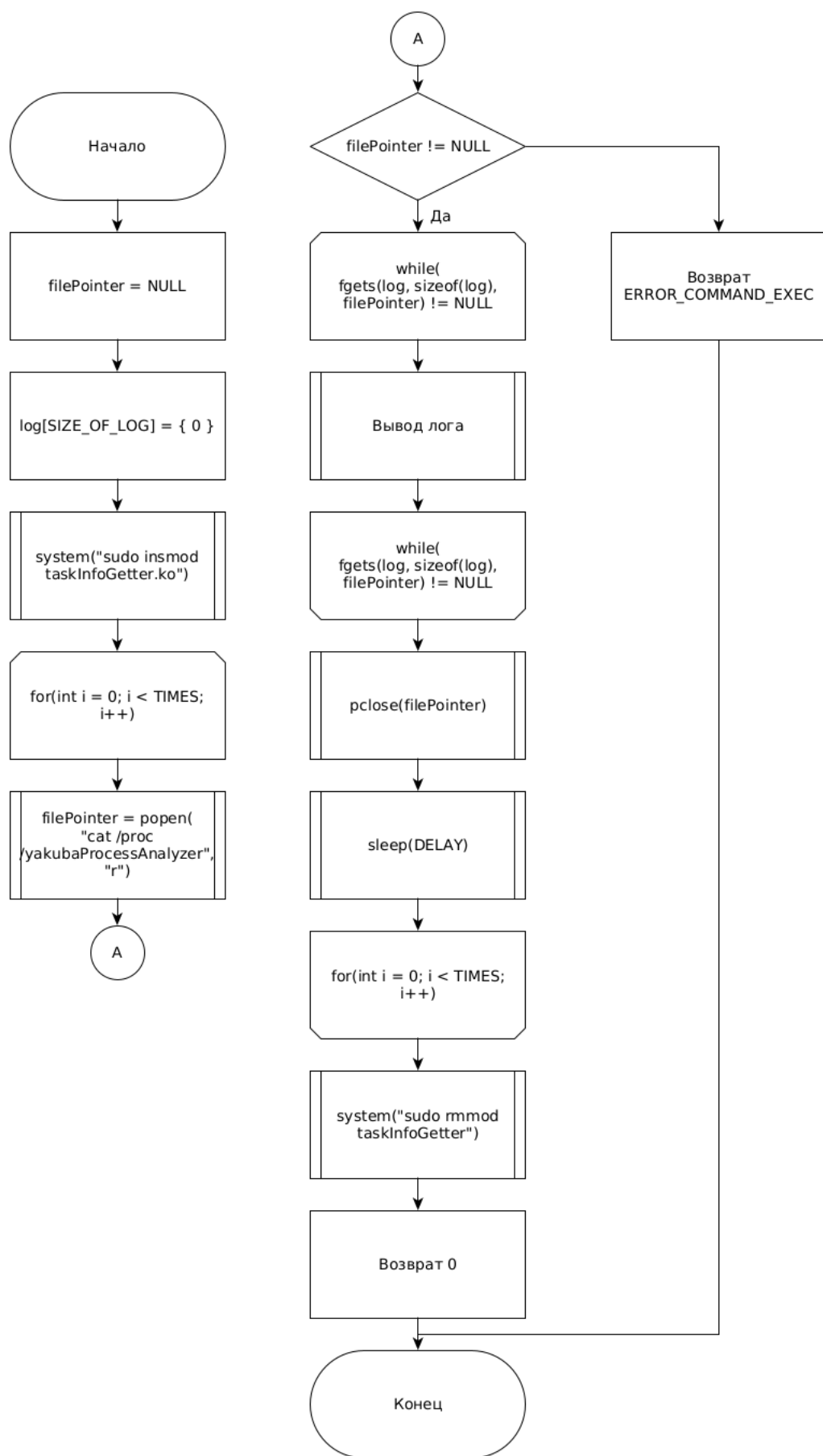


Рис. 2.5: Алгоритм предоставления информации о приоритетах, времени выполнения и простоя процессов пользователю.

2.4 Структура программного обеспечения

На рисунке 2.6 предоставлена структура программного обеспечения.



Рис. 2.6: Структура программного обеспечения.

3. Технологический раздел

3.1 Выбор и обоснование языка и среды программирования

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,
- программное обеспечение с открытым исходным кодом.

3.2 Реализация алгоритмов определения и логирования приоритетов, времени выполнения и простоя процессов.

В листинге 10 предоставлена реализация алгоритмов определения и логирования приоритетов, времени выполнения и простоя процессов.

Листинг 10: Реализация алгоритмов определения и логирования приоритетов, времени выполнения и простоя процессов.

```
43 static int printTasks(void *arg)
44 {
45     struct task_struct *task;
46     size_t currentPrint = 1;
47
48     char currentString[TEMP_STRING_SIZE];
49
50     while (currentPrint <= TIMES)
51     {
52         task = &init_task;
53
54         memset(currentString, 0, TEMP_STRING_SIZE);
55         snprintf(currentString, TEMP_STRING_SIZE,
56                  ↪ "~~~~~: %lu TIME\n",
57                  ↪ currentPrint);
58
59         checkOverflow(currentString, log, LOG_SIZE);
60
61         strcat(log, currentString);
```

```

60
61     for_each_process(task)
62     {
63         if (rt_task(task))
64         {
65             memset(currentString, 0,
66                 ↪ TEMP_STRING_SIZE);
67             snprintf(currentString, TEMP_STRING_SIZE,
68                 "procID: %-5d, name: %15s\nprio:
69                 ↪ %3d, static_prio: %3d,
70                 ↪ normal_prio (with "
71                 "scheduler policy): %3d,
72                 ↪ realtime_prio: %3d\n"
73                 "delay: %10lld\nutime: %10lld
74                 ↪ (ticks), stime: %15lld
75                 ↪ (ticks)\n"
76                 "Sched_rt_entity: timeout: %ld,
77                 ↪ watchdog_stamp: %ld,
78                 ↪ time_slice: %d\n\n",
79                 task->pid, task->comm,
80                 ↪ task->prio,
81                 ↪ task->static_prio,
82                 ↪ task->normal_prio,
83                 ↪ task->rt_priority,
84                 task->sched_info.run_delay,
85                 ↪ task->utime, task->stime,
86                 ↪ task->rt.timeout,
87                 task->rt.watchdog_stamp,
88                 ↪ task->rt.time_slice);
89
90             checkOverflow(currentString, log,
91                 ↪ LOG_SIZE);
92
93             strcat(log, currentString);
94         }
95     }
96
97     currentPrint++;
98     mdelay(DELAY_MS);
99 }

```

```

85     return 0;
86 }
87
88 static int yaOpen(struct inode *spInode, struct file
    ↪ *spFile)
89 {
90     printk(KERN_INFO "%s open called\n", PREFIX);
91
92     try_module_get(THIS_MODULE);
93
94     return 0;
95 }
96
97 static ssize_t yaRead(struct file *filep, char __user
    ↪ *buf, size_t count, loff_t *offp)
98 {
99     ssize_t logLen = strlen(log);
100
101     printk(KERN_INFO "%s read called\n", PREFIX);
102
103     if (copy_to_user(buf, log, logLen))
104     {
105         printk(KERN_ERR "%s copy_to_user error\n",
            ↪ PREFIX);
106
107         return -EFAULT;
108     }
109
110     memset(log, 0, LOG_SIZE);
111
112     return logLen;
113 }

```

3.3 Реализация алгоритма предоставления информации о процессах пользователю

В листинге 11 предоставлена реализация алгоритма предоставления информации о приоритетах, времени выполнения и простоя процессов пользователю.

Листинг 11: Реализация алгоритма предоставления информации о приоритетах, времени выполнения и простоя процессов пользователю.

```
17  int main(int argc, char *argv[])
18  {
19      FILE *filePointer = NULL;
20      char log[SIZE_OF_LOG] = {'\0'};
21
22      system("sudo insmod taskInfoGetter.ko");
23
24      for (int i = 0; i < TIMES; i++)
25      {
26          filePointer = popen("cat
27              ↪ /proc/yakubaProcessAnalyzer", "r");
28
29          if (filePointer == NULL)
30          {
31              printf("Error: can't execute cat for process
32              ↪ analyzer");
33              return ERROR_COMMAND_EXEC;
34          }
35
36          while (fgets(log, sizeof(log), filePointer) !=
37              ↪ NULL)
38          {
39              printf("%s", log);
40              fgets(log, sizeof(log), filePointer);
41              printf("%s", log);
42          }
43
44          pclose(filePointer);
45          sleep(DELAY);
46      }
47
48      system("sudo rmmod taskInfoGetter");
49
50      return EXIT_SUCCESS;
51  }
```

3.4 Makefile

В листинге 12 предоставлено содержание Makefile для сборки компонентов.

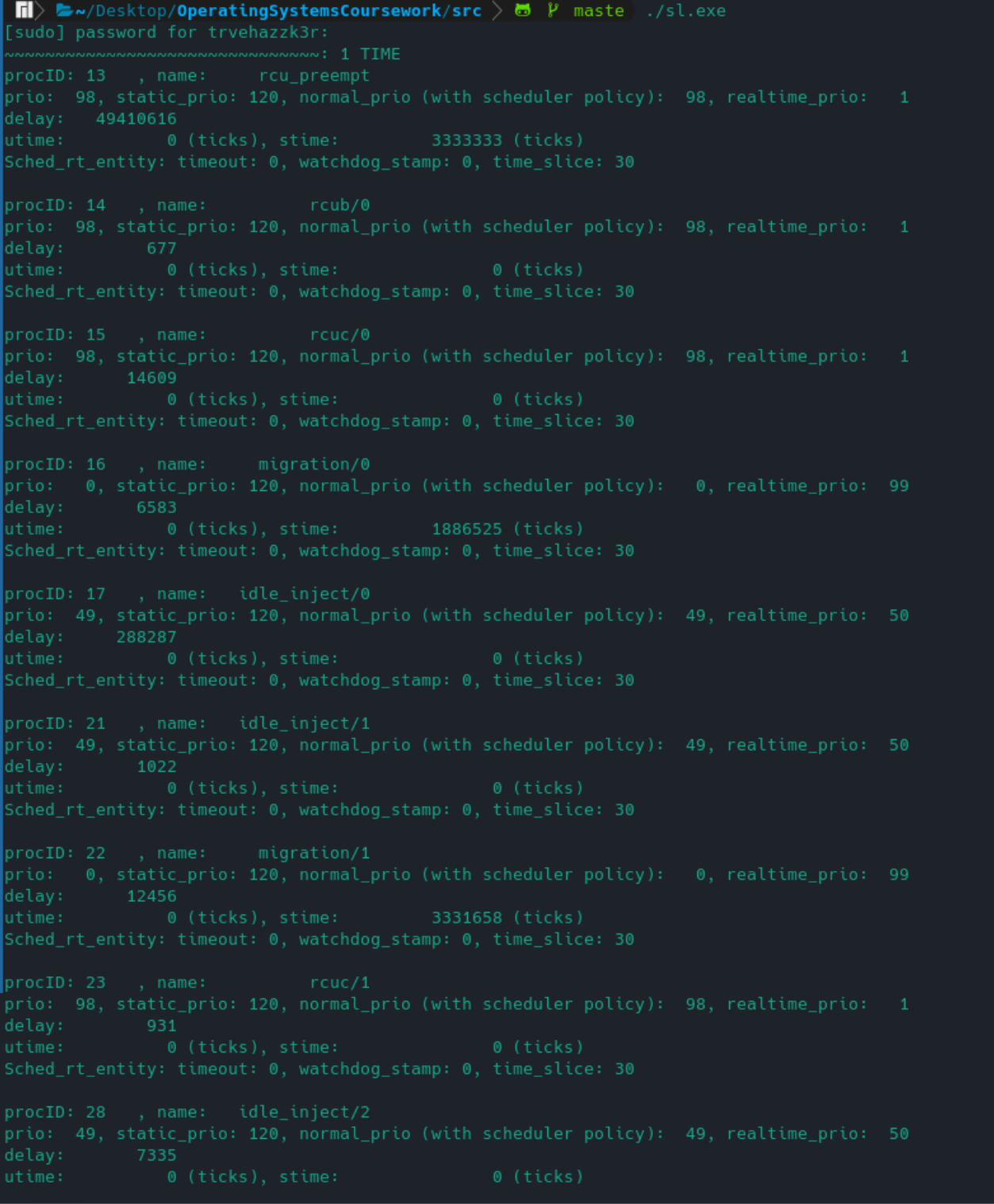
Листинг 12: Содержание Makefile.

```
1  ifneq ($(KERNELRELEASE),)
2      obj-m := taskInfoGetter.o
3      taskInfoGetter-objs := md.o
4  else
5      CURRENT = $(shell uname -r)
6      KDIR = /lib/modules/$(CURRENT)/build
7      PWD = $(shell pwd)
8
9  default:
10     $(MAKE) -C $(KDIR) M=$(PWD) modules
11     make clean
12
13  logger:
14     gcc starterLogger.c -Wall -Werror -o sl.exe
15
16  clean:
17     @rm -f *.o *.cmd *.flags *.mod.c *.order *.mod
18     ↪ *.symvers
19     @rm -f *.*.cmd *~ *.*~ TODO.*
20     @rm -fR .tmp*
21     @rm -rf .tmp_versions
22
23  disclean: clean
24     @rm *.ko *.symvers *.mod
25  endif
26
27  # sudo insmod md.ko
28  # lsmod | grep md
29  # sudo dmesg
30  # sudo rmmod md.ko
```

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.



```
~/Desktop/OperatingSystemsCoursework/src > P maste ./sl.exe
[sudo] password for trvehazzk3r:
~~~~~: 1 TIME
procID: 13 , name: rcu_preempt
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 49410616
utime: 0 (ticks), stime: 3333333 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 14 , name: rcub/0
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 677
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 15 , name: rcuc/0
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 14609
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 16 , name: migration/0
prio: 0, static_prio: 120, normal_prio (with scheduler policy): 0, realtime_prio: 99
delay: 6583
utime: 0 (ticks), stime: 1886525 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 17 , name: idle_inject/0
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 288287
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 21 , name: idle_inject/1
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 1022
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 22 , name: migration/1
prio: 0, static_prio: 120, normal_prio (with scheduler policy): 0, realtime_prio: 99
delay: 12456
utime: 0 (ticks), stime: 3331658 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 23 , name: rcuc/1
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 931
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 28 , name: idle_inject/2
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 7335
utime: 0 (ticks), stime: 0 (ticks)
```

Shell

src: sl.exe - Terminal

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

```

#####: 2 TIME
procID: 13 , name: rcu_preempt
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 49596120
utime: 0 (ticks), stime: 3333333 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 14 , name: rcub/0
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 677
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 15 , name: rcuc/0
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 14609
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 16 , name: migration/0
prio: 0, static_prio: 120, normal_prio (with scheduler policy): 0, realtime_prio: 99
delay: 6583
utime: 0 (ticks), stime: 1886525 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 17 , name: idle_inject/0
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 288287
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30


procID: 21 , name: idle_inject/1
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 1022
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 22 , name: migration/1
prio: 0, static_prio: 120, normal_prio (with scheduler policy): 0, realtime_prio: 99
delay: 12456
utime: 0 (ticks), stime: 3331658 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 23 , name: rcuc/1
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 931
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 28 , name: idle_inject/2
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 7335
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

```

 **Shell**


 src: zsh - Terminal

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

```

: 3 TIME
procID: 13 , name: rcu_preempt
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 49727587
utime: 0 (ticks), stime: 3333333 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 14 , name: rcub/0
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 677
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 15 , name: rcuc/0
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 14609
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 16 , name: migration/0
prio: 0, static_prio: 120, normal_prio (with scheduler policy): 0, realtime_prio: 99
delay: 6583
utime: 0 (ticks), stime: 1886525 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 17 , name: idle_inject/0
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 288287
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 21 , name: idle_inject/1
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 1022
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 22 , name: migration/1
prio: 0, static_prio: 120, normal_prio (with scheduler policy): 0, realtime_prio: 99
delay: 12456
utime: 0 (ticks), stime: 3331658 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 23 , name: rcuc/1
prio: 98, static_prio: 120, normal_prio (with scheduler policy): 98, realtime_prio: 1
delay: 931
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

procID: 28 , name: idle_inject/2
prio: 49, static_prio: 120, normal_prio (with scheduler policy): 49, realtime_prio: 50
delay: 7335
utime: 0 (ticks), stime: 0 (ticks)
Sched_rt_entity: timeout: 0, watchdog_stamp: 0, time_slice: 30

```

Shell

src: zsh - Terminal

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

```

[ 334.967144] ~~[TASK INFO]~~: module loaded
[ 334.967506] audit: type=1106 audit(1643887178.936:107): pid=3143 uid=1000 auid=1000 ses:
rminal=/dev/pts/1 res=success'
[ 334.967573] audit: type=1104 audit(1643887178.936:108): pid=3143 uid=1000 auid=1000 ses:
rminal=/dev/pts/1 res=success'
[ 334.972749] ~~[TASK INFO]~~: open called
[ 334.972770] ~~[TASK INFO]~~: read called
[ 334.972797] ~~[TASK INFO]~~: read called
[ 334.972815] ~~[TASK INFO]~~: release called
[ 344.975590] ~~[TASK INFO]~~: open called
[ 344.975602] ~~[TASK INFO]~~: read called
[ 344.975616] ~~[TASK INFO]~~: read called
[ 344.975625] ~~[TASK INFO]~~: release called
[ 346.146147] audit: type=1131 audit(1643887190.116:109): pid=1 uid=0 auid=4294967295 ses:
ccess'
[ 346.178521] audit: type=1334 audit(1643887190.149:110): prog-id=26 op=UNLOAD
[ 346.178523] audit: type=1334 audit(1643887190.149:111): prog-id=25 op=UNLOAD
[ 346.178524] audit: type=1334 audit(1643887190.149:112): prog-id=24 op=UNLOAD
[ 354.978103] ~~[TASK INFO]~~: open called
[ 354.978114] ~~[TASK INFO]~~: read called
[ 354.978128] ~~[TASK INFO]~~: read called
[ 354.978136] ~~[TASK INFO]~~: release called
[ 364.997557] audit: type=1101 audit(1643887208.966:113): pid=3192 uid=1000 auid=1000 ses:
terminal=/dev/pts/1 res=success'
[ 364.998537] audit: type=1110 audit(1643887208.969:114): pid=3192 uid=1000 auid=1000 ses:
ddr=? terminal=/dev/pts/1 res=success'
[ 365.002335] audit: type=1105 audit(1643887208.972:115): pid=3192 uid=1000 auid=1000 ses:
minal=/dev/pts/1 res=success'
[ 365.003711] ~~[TASK INFO]~~: Module goes away... It's his final message.
[ 365.022717] audit: type=1106 audit(1643887208.992:116): pid=3192 uid=1000 auid=1000 ses:

```

Рис. 3.4: Содержание журнала ядра.

4. Исследовательский раздел

4.1 Условия исследований

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.

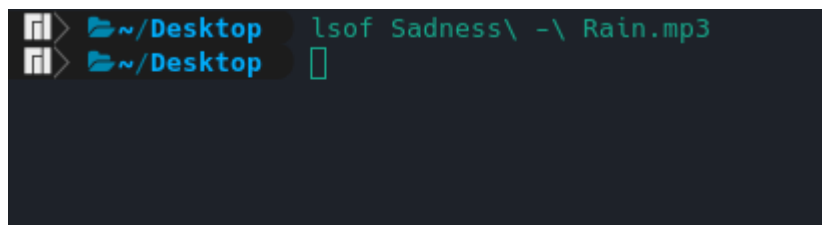


Рис. 4.1: Подтверждение того, что файл не задействован никакими другими процессами.

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 `ps` 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 произведена попытка воспользоваться VLC Media Player. Данное исследование проводилось также, как и предыдущее.



```
❏ > ~/Desktop lsof Sadness\ -\ Rain.mp3
COMMAND  PID    USER   FD   TYPE DEVICE SIZE/OFF  NODE NAME
vlc       20875 trvehazzk3r 28r  REG  259,4 76370684 21890185 Sadness - Rain.mp3
❏ > ~/Desktop pstree 20875
-
vlc-----10*[{vlc}]
❏ > ~/Desktop
```

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

В силу того, что файл вывода содержит 1666 строк, его представление здесь нецелесообразно. При анализе результатов среди проанализированных процессов отсутствовала задача 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 13.

Листинг 13: Измененная часть вывода в модуле.

```
63         if (task_is_realtime(task))
64         {
65             memset(currentString, 0,
66                 ↪ TEMP_STRING_SIZE);
67             snprintf(currentString, TEMP_STRING_SIZE,
68                 "procID: %-5d, name: %15s\nprio:
69                 ↪ %3d, static_prio: %3d,
70                 ↪ normal_prio (with "
71                 "scheduler policy): %3d,
72                 ↪ realtime_prio: %3d\n"
73                 "delay: %10lld\nutime: %10lld
74                 ↪ (ticks), stime: %15lld
75                 ↪ (ticks)\n"
76                 "Sched_rt_entity: timeout: %ld,
77                 ↪ watchdog_stamp: %ld,
78                 ↪ time_slice: %d\n\n",
79                 task->pid, task->comm,
80                 ↪ task->prio,
81                 ↪ task->static_prio,
82                 ↪ task->normal_prio,
83                 ↪ task->rt_priority,
84                 task->sched_info.run_delay,
85                 ↪ task->utime, task->stime,
86                 ↪ task->rt.timeout,
87                 task->rt.watchdog_stamp,
88                 ↪ task->rt.time_slice);
89
90             checkOverflow(currentString, log,
91                 ↪ LOG_SIZE);
92
93             strcat(log, currentString);
```

При анализе результатов среди приведенных процессов отсутствовали задачи с идентификационным номером, совпадающим с номерами процессов, отвечающих за воспроизведение.

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 14.

Листинг 14: Измененная часть вывода в модуле.

```

61     for_each_process(task)
62     {
63         // if (task_is_realtime(task))
64         // {
65         memset(currentString, 0, TEMP_STRING_SIZE);
66         snprintf(currentString, TEMP_STRING_SIZE,
67                 "procID: %-5d, name: %15s\nprio: %3d,
68                 ↪ static_prio: %3d, normal_prio
69                 ↪ (with "
70                 "scheduler policy): %3d,
71                 ↪ realtime_prio: %3d\n"
72                 "delay: %10lld\nutime: %10lld
73                 ↪ (ticks), stime: %15lld (ticks)\n"
74                 "Sched_rt_entity: timeout: %ld,
75                 ↪ watchdog_stamp: %ld, time_slice:
76                 ↪ %d\n\n",
77                 task->pid, task->comm, task->prio,
78                 ↪ task->static_prio,
79                 ↪ task->normal_prio,
80                 ↪ task->rt_priority,
81                 task->sched_info.run_delay,
82                 ↪ task->utime, task->stime,
83                 ↪ task->rt.timeout,

```



```

73         task->rt.watchdog_stamp,
           ↪ task->rt.time_slice);
74
75         checkOverflow(currentString, log, LOG_SIZE);
76
77         strcat(log, currentString);
78         // }
79     }

```

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 15.

Листинг 15: Информация о процессе vlc.

```

1  ~~~~~: 1 TIME
2  ...
3  procID: 7702 , name:          vlc
4  prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio: 0
5  delay:      288400
6  utime:   16552482 (ticks), stime:      19674870 (ticks)
7  Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30
8  ...
9
10 ~~~~~: 2 TIME
11 ...
12 procID: 7702 , name:          vlc
13 prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio: 0
14 delay:      288400
15 utime:   16552482 (ticks), stime:      19674870 (ticks)
16 Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30
17 ...
18

```

```

19 ~~~~~: 3 TIME
20 ...
21 procID: 7702 , name:          vlc
22 prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio:  0
23 delay:      288400
24 utime:   16552482 (ticks), stime:      19674870 (ticks)
25 Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30
26 ...
27
28 ~~~~~: 4 TIME
29 ...
30 procID: 7702 , name:          vlc
31 prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio:  0
32 delay:      288400
33 utime:   16552482 (ticks), stime:      19674870 (ticks)
34 Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30
35 ...
36
37 ~~~~~: 5 TIME
38 ...
39 procID: 7702 , name:          vlc
40 prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio:  0
41 delay:      288400
42 utime:   16552482 (ticks), stime:      19674870 (ticks)
43 Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30
44 ...
45
46 ~~~~~: 6 TIME
47 ...
48 procID: 7702 , name:          vlc
49 prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio:  0
50 delay:      288400
51 utime:   16552482 (ticks), stime:      19674870 (ticks)
52 Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30

```

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, представленная в листинге 16.

Листинг 16: Информация о процессе mplayer.

```

1  ~~~~~: 1 TIME
2  ...
3  procID: 9126 , name:          mplayer
4  prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio: 0
5  delay: 99583422
6  utime: 2398582148 (ticks), stime:          598783104 (ticks)
7  Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30
8  ...
9
10 ~~~~~: 2 TIME
11 ...
12 procID: 9126 , name:          mplayer
13 prio: 120, static_prio: 120, normal_prio (with scheduler
   ↪ policy): 120, realtime_prio: 0
14 delay: 99753204
15 utime: 2418484508 (ticks), stime:          598783104 (ticks)
16 Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
   ↪ time_slice: 30
17 ...
18
19 ~~~~~: 3 TIME

```

```

20  ...
21  procID: 9126 , name:          mplayer
22  prio: 120, static_prio: 120, normal_prio (with scheduler
    ↪ policy): 120, realtime_prio: 0
23  delay: 100071573
24  utime: 2421817841 (ticks), stime:          608761366 (ticks)
25  Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
    ↪ time_slice: 30
26  ...
27
28  ~~~~~: 4 TIME
29  ...
30  procID: 9126 , name:          mplayer
31  prio: 120, static_prio: 120, normal_prio (with scheduler
    ↪ policy): 120, realtime_prio: 0
32  delay: 100179471
33  utime: 2438474430 (ticks), stime:          618750383 (ticks)
34  Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
    ↪ time_slice: 30
35  ...
36
37  ~~~~~: 5 TIME
38  ...
39  procID: 9126 , name:          mplayer
40  prio: 120, static_prio: 120, normal_prio (with scheduler
    ↪ policy): 120, realtime_prio: 0
41  delay: 102045216
42  utime: 2448470429 (ticks), stime:          622081494 (ticks)
43  Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
    ↪ time_slice: 30
44  ...
45
46  ~~~~~: 6 TIME
47  ...
48  procID: 9126 , name:          mplayer
49  prio: 120, static_prio: 120, normal_prio (with scheduler
    ↪ policy): 120, realtime_prio: 0
50  delay: 102484167
51  utime: 2461786340 (ticks), stime:          625414827 (ticks)
52  Sched_rt_entity: timeout: 0, watchdog_stamp: 0,
    ↪ time_slice: 30

```

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 10 534 032 тика, в то время как `stime` в среднем увеличивался на 4 438 620.5 тика, что на $\approx 58\%$ 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 17. Идентификационный номер также совпал.

Листинг 17: Информация, полученная с использованием утилиты `pacmd`.

```
1 index: 13
2     driver: <protocol-native.c>
3     owner module: 12
4     properties:
5         application.name = "ALSA plug-in
6         ↪ [mplayer]"
7         native-protocol.peer = "UNIX socket
8         ↪ client"
9         native-protocol.version = "35"
10        application.process.id = "9126"
11        application.process.user = "trvehazzk3r"
12        application.process.host = "trvehazzk3r"
```

```

11     application.process.binary = "mplayer"
12     application.language = "C"
13     window.x11.display = ":0"
14     application.process.machine_id =
15         ↪ "b8d7673a22fe4ef6a2deb9f23ce0eafe"
16     application.process.session_id = "1"
17     application.icon_name = "mplayer"

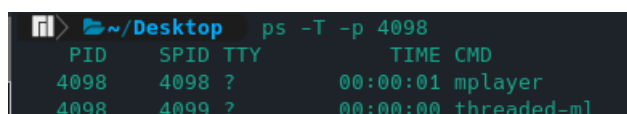
```

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`, 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.



```

~> ps -T -p 4098
  PID  SPID  TTY      TIME  CMD
  4098   4098  ?        00:00:01 mplayer
  4098   4099  ?        00:00:00 threaded-ml

```

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`, представленного на рисунке 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.

```
top - 15:09:32 up 9 min, 4 users, load average: 1,42, 1,28, 0,72
Tasks: 343 total, 1 running, 342 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4,1 us, 1,1 sy, 0,0 ni, 94,4 id, 0,0 wa, 0,2 hi, 0,1 si, 0,0 st
MiB Mem : 15590,4 total, 8366,9 free, 3725,1 used, 3498,4 buff/cache
MiB Swap: 977,0 total, 977,0 free, 0,0 used, 10616,2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2737	trvehaz+	20	0	24,4g	383920	154844	S	17,5	2,4	1:22.46	chrome
1982	trvehaz+	20	0	16,7g	162492	109848	S	7,6	1,0	0:40.71	chrome
2987	trvehaz+	20	0	24,4g	145452	84068	S	5,3	0,9	0:07.26	chrome
1509	trvehaz+	9	-11	1286660	15944	11200	S	4,6	0,1	0:08.41	pulseaudio
1939	trvehaz+	20	0	16,6g	292436	174052	S	2,6	1,8	0:33.89	chrome
4098	trvehaz+	20	0	670412	43848	35136	S	2,3	0,3	0:04.10	mplayer
1985	trvehaz+	20	0	16,3g	112020	81828	S	1,3	0,7	0:11.68	chrome
2239	trvehaz+	20	0	28,4g	166376	101804	S	1,3	1,0	0:03.61	chrome
277	root	20	0	446656	303912	302100	S	0,7	1,9	0:22.03	systemd-journal
4300	trvehaz+	20	0	11256	4196	3344	R	0,7	0,0	0:00.27	top

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.

```
top - 15:22:35 up 22 min, 4 users, load average: 0,64, 0,71, 0,71
Threads: 1616 total, 1 running, 1615 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3,5 us, 1,6 sy, 0,0 ni, 94,4 id, 0,0 wa, 0,3 hi, 0,2 si, 0,0 st
MiB Mem : 15590,4 total, 7578,5 free, 4049,6 used, 3962,3 buff/cache
MiB Swap: 977,0 total, 977,0 free, 0,0 used, 10160,3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2745	trvehaz+	20	0	24,4g	652392	154972	S	7,5	4,1	1:35.87	Compositor
2737	trvehaz+	20	0	24,4g	652392	154972	S	7,2	4,1	1:29.90	chrome
277	root	20	0	531836	356948	355040	S	6,5	2,2	0:56.64	systemd-journal
1528	trvehaz+	20	0	1011292	125392	102340	S	6,2	0,8	0:06.76	yakuake
919	root	20	0	611344	106348	73912	S	4,2	0,7	0:44.51	Xorg
2018	trvehaz+	20	0	16,7g	168276	110900	S	4,2	1,1	0:54.38	VizCompositorTh
1403	trvehaz+	20	0	2147012	155692	111168	S	3,6	1,0	0:35.08	kwin_x11
1645	trvehaz+	-6	0	1286660	15944	11200	S	3,3	0,1	0:29.22	alsa-sink-CX807
2016	trvehaz+	20	0	16,7g	168276	110900	S	2,9	1,1	0:38.39	Chrome_ChildIOT
1415	trvehaz+	20	0	2147012	155692	111168	S	1,6	1,0	0:12.95	QXcbEventQueue
4098	trvehaz+	20	0	670412	43848	35136	S	1,6	0,3	0:11.93	mplayer
5300	trvehaz+	20	0	12804	5760	3340	R	1,6	0,0	0:00.78	top
2740	trvehaz+	20	0	24,4g	652392	154972	S	1,3	4,1	0:15.51	Chrome_ChildIOT
3995	trvehaz+	20	0	925396	186656	106908	S	1,3	1,2	1:27.99	texmaker
1509	trvehaz+	9	-11	1286660	15944	11200	S	1,0	0,1	0:10.63	pulseaudio
1327	root	20	0	611344	106348	73912	S	0,7	0,7	0:07.28	InputThread

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.

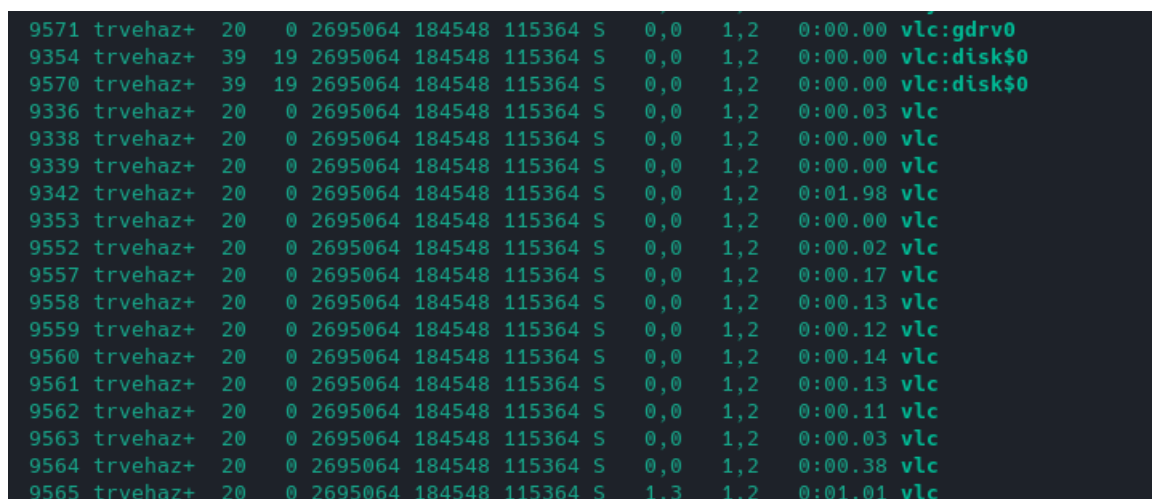
Занятен факт присутствия в выводе rtkit-daemon, который является системной службой, меняющей по запросу политику планирования пользовательских процессов и потоков на SCHED_RR, то есть режим планирования в реальном времени. Отсюда и возникает факт того, что в системе сложно увидеть, что процесс проигрывания 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.

4.8 Исследование потоков при проигрывании видео с использованием VLC Media Player

При проигрывании видеоряда с использованием VLC Media Player в выводе утилиты top было обнаружено несколько потоков, созданных коман-

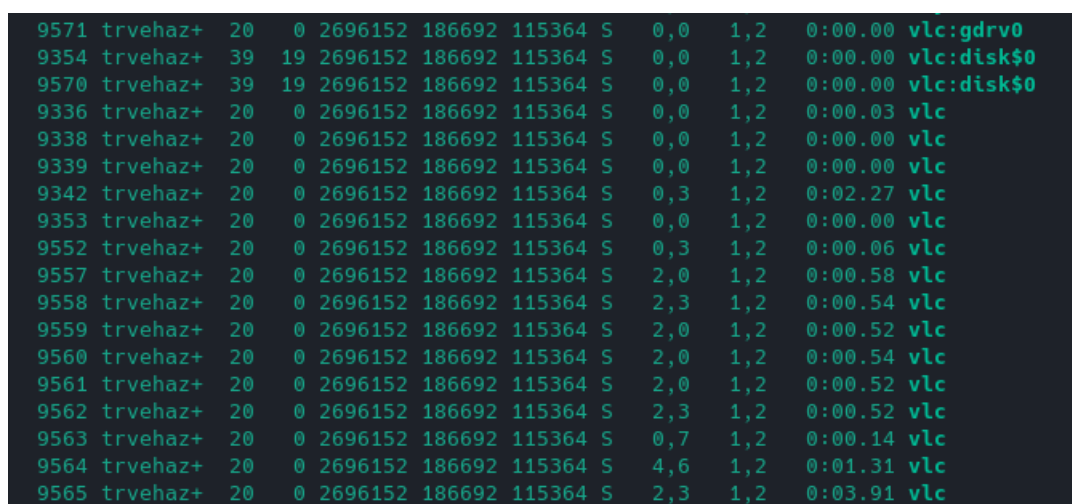
дой vlc, что представлено на изображении 4.8.



9571	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.00	vlc:gdrv0
9354	trvehaz+	39	19	2695064	184548	115364	S	0,0	1,2	0:00.00	vlc:disk\$0
9570	trvehaz+	39	19	2695064	184548	115364	S	0,0	1,2	0:00.00	vlc:disk\$0
9336	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.03	vlc
9338	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.00	vlc
9339	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.00	vlc
9342	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:01.98	vlc
9353	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.00	vlc
9552	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.02	vlc
9557	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.17	vlc
9558	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.13	vlc
9559	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.12	vlc
9560	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.14	vlc
9561	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.13	vlc
9562	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.11	vlc
9563	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.03	vlc
9564	trvehaz+	20	0	2695064	184548	115364	S	0,0	1,2	0:00.38	vlc
9565	trvehaz+	20	0	2695064	184548	115364	S	1,3	1,2	0:01.01	vlc

Рис. 4.8: Информация о потоках, полученная с использованием утилиты top.

Причем, если видео не проигрывается, данные потоки (кроме потока с идентификационным номером 9565) не получают приращения к времени использования процесса. Однако, при запуске видеоряда, информация о данном поле в потоках 9564, 9563, 9562, 9561, 9560, 9559, 9558, 9557, 9552 меняется, что показано на изображении 4.9.



9571	trvehaz+	20	0	2696152	186692	115364	S	0,0	1,2	0:00.00	vlc:gdrv0
9354	trvehaz+	39	19	2696152	186692	115364	S	0,0	1,2	0:00.00	vlc:disk\$0
9570	trvehaz+	39	19	2696152	186692	115364	S	0,0	1,2	0:00.00	vlc:disk\$0
9336	trvehaz+	20	0	2696152	186692	115364	S	0,0	1,2	0:00.03	vlc
9338	trvehaz+	20	0	2696152	186692	115364	S	0,0	1,2	0:00.00	vlc
9339	trvehaz+	20	0	2696152	186692	115364	S	0,0	1,2	0:00.00	vlc
9342	trvehaz+	20	0	2696152	186692	115364	S	0,3	1,2	0:02.27	vlc
9353	trvehaz+	20	0	2696152	186692	115364	S	0,0	1,2	0:00.00	vlc
9552	trvehaz+	20	0	2696152	186692	115364	S	0,3	1,2	0:00.06	vlc
9557	trvehaz+	20	0	2696152	186692	115364	S	2,0	1,2	0:00.58	vlc
9558	trvehaz+	20	0	2696152	186692	115364	S	2,3	1,2	0:00.54	vlc
9559	trvehaz+	20	0	2696152	186692	115364	S	2,0	1,2	0:00.52	vlc
9560	trvehaz+	20	0	2696152	186692	115364	S	2,0	1,2	0:00.54	vlc
9561	trvehaz+	20	0	2696152	186692	115364	S	2,0	1,2	0:00.52	vlc
9562	trvehaz+	20	0	2696152	186692	115364	S	2,3	1,2	0:00.52	vlc
9563	trvehaz+	20	0	2696152	186692	115364	S	0,7	1,2	0:00.14	vlc
9564	trvehaz+	20	0	2696152	186692	115364	S	4,6	1,2	0:01.31	vlc
9565	trvehaz+	20	0	2696152	186692	115364	S	2,3	1,2	0:03.91	vlc

Рис. 4.9: Информация о потоках, полученная с использованием утилиты top.

Таким образом, можно сделать вывод о том, что данное видео проигрывается с использованием девяти потоков VLC Media Player.

В остальном, сведения, находящиеся в системе, идентичны ситуации при проигрывании аудио.

ЗАКЛЮЧЕНИЕ

В соответствии с заданием на курсовую работу по курсу “Операционные Системы” был разработан загружаемый модуль ядра, позволяющий отслеживать состояния приоритетов, времени выполнения и простоя процессов в операционной системе Linux, а также были проанализированы процессы реального времени.

В процессе разработки были реализованы алгоритмы определения, логирования и предоставления информации пользователю о приоритетах, времени выполнения и простоя процессов.

В результате проведенных исследований было показано, что процессы программного обеспечения проигрывателей в системе не являются постоянными задачами реального времени.

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

ПРИЛОЖЕНИЕ А. Исходный код программы

Листинг 18: Содержимое файла md.c

```
1  #include <linux/delay.h>
2  #include <linux/init.h>
3  #include <linux/init_task.h>
4  #include <linux/kernel.h>
5  #include <linux/module.h>
6  #include <linux/proc_fs.h>
7
8  #include <linux/kthread.h>
9
10 #define PROC_FS_NAME "yakubaProcessAnalyzer"
11
12 #define PREFIX "~~[TASK INFO]~~:"
13
14 #define TEMP_STRING_SIZE 512
15
16 MODULE_LICENSE("GPL");
17 MODULE_AUTHOR("Yakuba D.");
18
19 #define TIMES 6
20 #define DELAY_MS 10 * 1000
21
22 static struct proc_dir_entry *procFile;
23
24 static struct task_struct *kthread;
25
26 #define LOG_SIZE 262144
27 static char log[LOG_SIZE] = { 0 };
28
29 static int checkOverflow(char *fString, char *sString, int
    ↪ maxSize)
30 {
31     int sumLen = strlen(fString) + strlen(sString);
32
33     if (sumLen >= maxSize)
34     {
```

```

35     printk(KERN_ERR "%s not enough space in log (%d
        ↳ needed but %d available)\n", PREFIX, sumLen,
        ↳ maxSize);
36
37     return -ENOMEM;
38 }
39
40     return 0;
41 }
42
43 static int printTasks(void *arg)
44 {
45     struct task_struct *task;
46     size_t currentPrint = 1;
47
48     char currentString[TEMP_STRING_SIZE];
49
50     while (currentPrint <= TIMES)
51     {
52         task = &init_task;
53
54         memset(currentString, 0, TEMP_STRING_SIZE);
55         snprintf(currentString, TEMP_STRING_SIZE,
            ↳ "~~~~~: %lu TIME\n",
            ↳ currentPrint);
56
57         checkOverflow(currentString, log, LOG_SIZE);
58
59         strcat(log, currentString);
60
61         for_each_process(task)
62         {
63             // if (task_is_realtime(task))
64             // {
65             memset(currentString, 0, TEMP_STRING_SIZE);
66             snprintf(currentString, TEMP_STRING_SIZE,
67                 "procID: %-5d, name: %15s\nprio: %3d,
                    ↳ static_prio: %3d, normal_prio
                    ↳ (with "
68                 "scheduler policy): %3d,
                    ↳ realtime_prio: %3d\n"

```

```

69         "delay: %10lld\nutime: %10lld
        ↪ (ticks), stime: %15lld (ticks)\n"
70     "Sched_rt_entity: timeout: %ld,
        ↪ watchdog_stamp: %ld, time_slice:
        ↪ %d\n\n",
71     task->pid, task->comm, task->prio,
        ↪ task->static_prio,
        ↪ task->normal_prio,
        ↪ task->rt_priority,
72     task->sched_info.run_delay,
        ↪ task->utime, task->stime,
        ↪ task->rt.timeout,
73     task->rt.watchdog_stamp,
        ↪ task->rt.time_slice);

74
75     checkOverflow(currentString, log, LOG_SIZE);
76
77     strcat(log, currentString);
78     // }
79 }
80
81     currentPrint++;
82     mdelay(DELAY_MS);
83 }
84
85     return 0;
86 }
87
88 static int yaOpen(struct inode *spInode, struct file
    ↪ *spFile)
89 {
90     printk(KERN_INFO "%s open called\n", PREFIX);
91
92     try_module_get(THIS_MODULE);
93
94     return 0;
95 }
96
97 static ssize_t yaRead(struct file *filep, char __user
    ↪ *buf, size_t count, loff_t *offp)
98 {

```

```

99         ssize_t logLen = strlen(log);
100
101         printk(KERN_INFO "%s read called\n", PREFIX);
102
103         if (copy_to_user(buf, log, logLen))
104         {
105             printk(KERN_ERR "%s copy_to_user error\n",
106                    ↪ PREFIX);
107
108             return -EFAULT;
109         }
110
111         memset(log, 0, LOG_SIZE);
112
113         return logLen;
114     }
115
116     static ssize_t yaWrite(struct file *file, const char
117     ↪ __user *buf, size_t len, loff_t *offp)
118     {
119         printk(KERN_INFO "%s write called\n", PREFIX);
120
121         return 0;
122     }
123
124     static int yaRelease(struct inode *spInode, struct file
125     ↪ *spFile)
126     {
127         printk(KERN_INFO "%s release called\n", PREFIX);
128
129         module_put(THIS_MODULE);
130
131         return 0;
132     }
133
134     static struct proc_ops fops = {proc_read : yaRead,
135     ↪ proc_write : yaWrite, proc_open : yaOpen, proc_release
136     ↪ : yaRelease};
137
138     static int __init md_init(void)
139     {

```



```

135     if (!(procFile = proc_create(PROC_FS_NAME, 0666, NULL,
    ↪      &fops)))
136     {
137         printk(KERN_ERR "%s proc_create error\n", PREFIX);
138
139         return -EFAULT;
140     }
141
142     kthread = kthread_run(printTasks, NULL,
    ↪      "taskPrintThread");
143     if (IS_ERR(kthread))
144     {
145         printk(KERN_ERR "%s kthread_run error\n", PREFIX);
146
147         return -EFAULT;
148     }
149
150     printk(KERN_INFO "%s module loaded\n", PREFIX);
151
152     return 0;
153 }
154
155 static void __exit md_exit(void)
156 {
157     remove_proc_entry(PROC_FS_NAME, NULL);
158
159     printk(KERN_INFO "%s Module goes away... It's his
    ↪      final message.\n", PREFIX);
160 }
161
162 module_init(md_init);
163 module_exit(md_exit);

```

Листинг 19: Содержимое файла starterLogger.c

```

1  #include <linux/sched/types.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>

```

```

5  #include <sys/syscall.h>
6  #include <unistd.h>
7
8  #include <errno.h>
9
10 #define SIZE_OF_LOG 8192
11
12 #define ERROR_COMMAND_EXEC 1
13
14 #define TIMES 6
15 #define DELAY 10
16
17 int main(int argc, char *argv[])
18 {
19     FILE *filePointer = NULL;
20     char log[SIZE_OF_LOG] = {'\0'};
21
22     system("sudo insmod taskInfoGetter.ko");
23
24     for (int i = 0; i < TIMES; i++)
25     {
26         filePointer = popen("cat
27         ↪ /proc/yakubaProcessAnalyzer", "r");
28
29         if (filePointer == NULL)
30         {
31             printf("Error: can't execute cat for process
32             ↪ analyzer");
33             return ERROR_COMMAND_EXEC;
34         }
35
36         while (fgets(log, sizeof(log), filePointer) !=
37         ↪ NULL)
38         {
39             printf("%s", log);
40             fgets(log, sizeof(log), filePointer);
41             printf("%s", log);
42         }
43
44         pclose(filePointer);
45         sleep(DELAY);

```

```
43     }
44
45     system("sudo rmmod taskInfoGetter");
46
47     return EXIT_SUCCESS;
48 }
```