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

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Загружаемый модуль ядра, отслеживающий процессы, использующие заданный пользователем файл»

Студент ИУ7-76Б _____ Авсюнин А. А.
(Подпись, дата) (Фамилия И. О.)

Руководитель курсовой работы _____ Рязанова Н. Ю.
(Подпись, дата) (Фамилия И. О.)

2024 г.

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

УТВЕРЖДАЮ
Заведующий кафедрой ИУ-7
_____ И. В. Рудаков
«16» сентября 2023 г.

З А Д А Н И Е
на выполнение курсовой работы

по теме

«Загружаемый модуль ядра, отслеживающий процессы, использующие заданный пользователем файл»

Студент группы ИУ7-76Б

Авсюнин Алексей Алексеевич

Направленность КР

учебная

График выполнения КР: 25% к 6 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание

Разработать загружаемый модуль ядра для получения информации о процессах, которые используют заданный пользователем файл. Обеспечить возможность посылки сигналов этим процессам.

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на **30-40** листах формата А4.

Дата выдачи задания «16» сентября 2023 г.

Руководитель Курсовой работы	_____	Рязанова Н. Ю.
	(Подпись, дата)	(Фамилия И. О.)
Студент	_____	Авсюнин А. А.
	(Подпись, дата)	(Фамилия И. О.)

Содержание

Введение

UNIX-подобные операционные системы являются одними из самых популярных в мире. На персональных компьютерах в 2023 году такие системы встречаются в у 20% [1] пользователей, а на телефонных устройствах около 98% [2] всех пользователей используют операционные системы семейства UNIX.

Процессы во время своей работы постоянно используют файлы. Над файлами производятся операции открытия и закрытия, записи и чтения. К одному файлу на диске несколько процессов могут иметь одновременный доступ. Такие процессы могут быть ничем не связаны между собой, их объединяет только факт использования одного и того же файла.

Целью данной работы является разработка загружаемого модуля ядра для получения информации о процессах, которые используют заданный пользователем файл. Необходимо обеспечить возможность посылки сигналов рассматриваемым процессам. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ структур и функций, предоставляющих возможность реализовать поставленную задачу;

2) разработать алгоритмы и структуру загружаемого модуля ядра, обеспечивающего отслеживание процессов, одновременно обращающихся к заданному пользователем файлу, и посылку сиганалов таким процессам;

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с техническим заданием на курсовую работу необходимо разработать загружаемый модуль ядра для получения информации о процессах, которые используют заданный пользователем файл. Также необходимо обеспечить возможность посылки сигналов рассматриваемым процессам. Для решения поставленной задачи необходимо:

1. провести анализ структур и функций, предоставляющих возможность реализовать поставленную задачу;
2. разработать алгоритмы и структуру загружаемого модуля ядра, в соответствии с поставленной задачей;
3. реализовать спроектированный модуль ядра;
4. протестировать работу реализованного модуля ядра.

К разрабатываемой программе предъявляются следующие требования:

1. взаимодействие с загружаемым модулем должно происходить из пространства пользователя;
2. необходимо передавать данные из пространства ядра в пространство пользователя или наоборот.

Программа будет разрабатываться для операционной системы Ubuntu [3] версии 20.04.4. В связи с этим она может не поддерживаться на более ранних версиях или на других операционных системах.

1.2 Анализ структур ядра

1.2.1 struct task_struct

task_struct — структура ядра, описывающая процесс в операционной системе. Она содержит всю информацию, необходимую ядру для управления процессом. В листинге 1.1 представлены необходимые для решения задачи фрагменты структуры task_struct [4].

Листинг 1.1 – Структура task_struct

```
1  struct task_struct {
2  // ...
3  unsigned int __state;
4  // ...
5  int prio;
6  // ...
7  pid_t pid;
8  // ...
9  struct task_struct __rcu *parent;
10 // ...
11 char comm[TASK_COMM_LEN];
12 // ...
13 struct files_struct *files;
14 // ...
```

Подробное описание представленного фрагмента структуры task_struct:

1. __state — состояние процесса;
2. prio — приоритет процесса;
3. pid — идентификатор процесса;
4. parent — указатель на структуру родительского процесса;
5. comm — имя исполняемого файла;
6. files — указатель на структуру, содержащую информацию об открытых файлах.

Для получения файловых дескрипторов процесса необходимо рассмотреть структуру files_struct [5]. В листингах 1.2–1.3 она полностью приведена.

Листинг 1.2 – Структура files_struct

```
1  struct files_struct {
2      atomic_t count;
3      bool resize_in_progress;
```

Листинг 1.3 – Структура files_struct

```
    wait_queue_head_t resize_wait;

    struct fdtable __rcu *fdt;
    struct fdtable fdtab;

    spinlock_t file_lock ____cacheline_aligned_in_smp;
    unsigned int next_fd;
    unsigned long close_on_exec_init[1];
    unsigned long open_fds_init[1];
    unsigned long full_fds_bits_init[1];
    struct file __rcu * fd_array[NR_OPEN_DEFAULT];
};
```

В данной структуре особого внимания заслуживает указатель fdt на структуру fdtable, именно в ней находится массив fd, содержащий структуры открытых файлов file для данного процесса. Ниже представлен листинг структуры fdtable [5].

Листинг 1.4 – Структура fdtable

```
struct fdtable {
    unsigned int max_fds;
    struct file __rcu **fd;
    unsigned long *close_on_exec;
    unsigned long *open_fds;
    unsigned long *full_fds_bits;
    struct rcu_head rcu;
};
```

1.2.2 struct file

struct file [6] — структура ядра, описывающая открытый файл. В листингах 1.5–1.6 полностью представлена данная структура.

Листинг 1.5 – Структура file

```
struct file {
    union {
        struct llist_node    f_llist;
        struct rcu_head      f_rcuhead;
```

Листинг 1.6 – Структура file

```
1         unsigned int        f_iocb_flags;
2     };
3
4     spinlock_t      f_lock;
5     fmode_t         f_mode;
6     atomic_long_t       f_count;
7     struct mutex        f_pos_lock;
8     loff_t          f_pos;
9     unsigned int        f_flags;
10    struct fown_struct  f_owner;
11    const struct cred   *f_cred;
12    struct file_ra_state    f_ra;
13    struct path     f_path;
14    struct inode        *f_inode;
15    const struct file_operations    *f_op;
16
17    u64         f_version;
18    #ifdef CONFIG_SECURITY
19    void            *f_security;
20    #endif
21    void            *private_data;
22
23    #ifdef CONFIG_EPOLL
24
25    struct hlist_head   *f_ep;
26    #endif
27    struct address_space    *f_mapping;
28    errseq_t        f_wb_err;
29    errseq_t        f_sb_err;
30 };
```

Для получения информации о том, какой файл представляет данная структура необходимо получить структуру dentry. Указатель на неё можно найти в структуре path, которая содержится в структуре file. В листинге 1.7 представлена структура path [5].

Листинг 1.7 – Структура path

```
1 struct path {
2     struct vfsmount *mnt;
3     struct dentry *dentry;
4 };
```

1.3 Анализ системных вызовов

1.3.1 Системный вызов kern_path

Пользователь загружаемого модуля будет передавать полный путь до файла, который он желает отследить, то есть найти все процессы, которые на момент запроса используют данный файл. По этой причине необходимо получить по полному имени файла указатель на структуру dentry данного файла. Данный функционал предоставляет системный вызов kern_path, заголовок которого представлен в листинге 1.8.

Листинг 1.8 – Заголовок системного вызова kern_path

```
1 int kern_path(const char *name, unsigned int flags, struct path *
    path);
```

На вход kern_path принимает name — имя файла; flags — флаги поиска элемента пути; path — структура path с результатом поиска. Системный вызов возвращает 0 в случае успеха и код ошибки при неудаче.

1.3.2 Системный вызов send_sig_info

В соответствии с заданием необходимо предоставить возможность посылать сигналы процессам, использующим заданный файл. Данный функционал предоставляет системный вызов send_sig_info, заголовок которого представлен в листинге 1.9.

Листинг 1.9 – Заголовок системного вызова send_sig_info

```
1 int send_sig_info(int sig, struct siginfo *info, struct task_struct
    *p)
```

На вход send_sig_info принимает sig — номер сигнала; info — структура, содержащая информацию о сигнале, возможно вместо структуры передать

одно из двух числовых значений [7]: 0, если сигнал послан из пространства пользователя, или 1, если сигнал послан из пространства ядра; p — структура процесса, которому отправляется сигнал. Системный вызов возвращает 0 в случае успеха и код ошибки при неудаче.

1.3.3 Системный вызов copy_from_user

В соответствии с требованиями необходимо передавать данные из пространства пользователя в пространство ядра. Данный функционал предоставляет системный вызов copy_from_user, заголовок которого представлен в листинге 1.10.

Листинг 1.10 – Заголовок системного вызова copy_from_user

```
1 long copy_from_user(void *to, const void __user *from, long n);
```

На вход copy_from_user принимает to — указатель на память в пространстве ядра, в которую будет осуществлено копирование; from — указатель на память в пространстве пользователя, из которой будет осуществлено копирование; n — размер копируемых данных. Системный вызов возвращает 0 в случае успеха и ненулевое значение при неудаче.

1.3.4 Системный вызов copy_to_user

В соответствии с требованиями необходимо передавать данные из пространства ядра в пространство пользователя. Данный функционал предоставляет системный вызов copy_to_user, заголовок которого представлен в листинге 1.11.

Листинг 1.11 – Заголовок системного вызова copy_to_user

```
1 long copy_to_user(void __user *to, const void *from, long n)
      ;
```

На вход copy_to_user принимает to — указатель на память в пространстве пользователя, в которую будет осуществлено копирование; from — указатель на память в пространстве ядра, из которой будет осуществлено копирование; n — размер копируемых данных. Системный вызов возвращает 0 в случае успеха и ненулевое значение при неудаче.

1.4 Интерфейс взаимодействия с модулем

Для взаимодействия с модулем ядра из пространства пользователя будут использоваться файлы, созданные в /proc. /proc — интерфейс, предоставляющий доступ к структурам ядра. Передача информации будет достигатся при помощи привычных обращений чтения и записи к файлам.

1.4.1 struct proc_ops

Для файлов, созданных в /proc сущетсвует специальная структура proc_ops, содержащая указатели на функции взаимодействия с файлом, такие как открытие, закрытие, чтение и запись, заменяющая аналогичную структуру file_operations для файлов на диске. В листинге 1.12 представлена структура proc_ops.

Листинг 1.12 – 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 *, size_t,
           loff_t *);
5      ssize_t (*proc_read_iter)(struct kiocb *, struct iov_iter *);
6      ssize_t (*proc_write)(struct file *, const char __user *, size_t
           , loff_t *);
7      loff_t  (*proc_lseek)(struct file *, loff_t, int);
8      int (*proc_release)(struct inode *, struct file *);
9      __poll_t (*proc_poll)(struct file *, struct poll_table_struct *)
           ;
10     long    (*proc_ioctl)(struct file *, unsigned int, unsigned long
           );
11     #ifdef CONFIG_COMPAT
12     long    (*proc_compat_ioctl)(struct file *, unsigned int,
           unsigned long);
13     #endif
14     int (*proc_mmap)(struct file *, struct vm_area_struct *);
15     unsigned long (*proc_get_unmapped_area)(struct file *, unsigned
           long, unsigned long, unsigned long, unsigned long);
16 };
```

Для изменения поведения файла при чтении или записи необходимо создать экземпляр структуры со своими функциями чтения и записи.

1.4.2 системный вызов proc_mkdir

Системный вызов proc_mkdir создаёт в /proc директорию. Его заголовок представлен в листинге 1.13.

Листинг 1.13 – Заголовок системного вызова proc_mkdir

```
1 struct proc_dir_entry *proc_mkdir(const char *name, struct
    proc_dir_entry *parent);
```

На вход proc_mkdir принимает name — имя создаваемой директории; parent — указатель на структуру proc_dir_entry, описывающую родительскую директорию, если равно NULL, то директория создаётся в корне. Системный вызов возвращает указатель на структуру proc_dir_entry созданной директории в случае успеха и NULL при неудаче.

1.4.3 системный вызов proc_create

Системный вызов proc_create создаёт в /proc файл. Его заголовок представлен в листинге 1.14.

Листинг 1.14 – Заголовок системного вызова proc_create

```
1 struct proc_dir_entry *proc_create(const char *name, umode_t mode,
    struct proc_dir_entry *parent,const struct file_operations *
    proc_fops);
```

На вход proc_create принимает name — имя создаваемого файла; parent — указатель на структуру proc_dir_entry, описывающую родительскую директорию, если равно NULL, то файл создаётся в корне; proc_fops — указатель на структуру с функциями работы с файлом. Системный вызов возвращает указатель на структуру proc_dir_entry созданного файла в случае успеха и NULL при неудаче.

1.4.4 системный вызов proc_symlink

Системный вызов proc_symlink создаёт в /proc символическую ссылку. Его заголовок представлен в листинге 1.15.

Листинг 1.15 – Заголовок системного вызова proc_symlink

```
1 struct proc_dir_entry *proc_symlink(const char *name,struct
    proc_dir_entry *parent,const char *dest);
```

На вход proc_symlink принимает name — имя создаваемой символической ссылки; parent — указатель на структуру proc_dir_entry, описывающую родительскую директорию, если равно NULL, то символическая ссылка создаётся в корне; dest — имя файла, для которого создаётся символическая ссылка. Системный вызов возвращает указатель на структуру proc_dir_entry созданной символической сслыки в случае успеха и NULL при неудаче.

2 Конструкторский раздел

2.1 Последовательность действий

На рисунке 2.1 представлена последовательность действий, выполняемых при загрузке модуля, а на рисунке 2.2 — при выгрузке модуля.

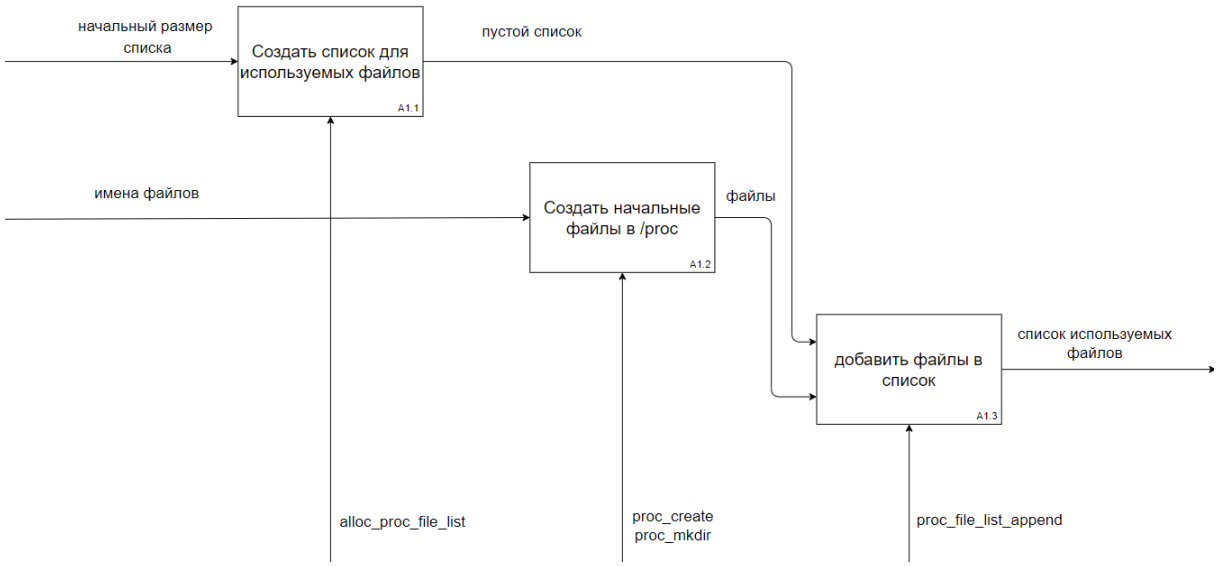

Рис. 2.1 – Последовательность действий при загрузке модуля

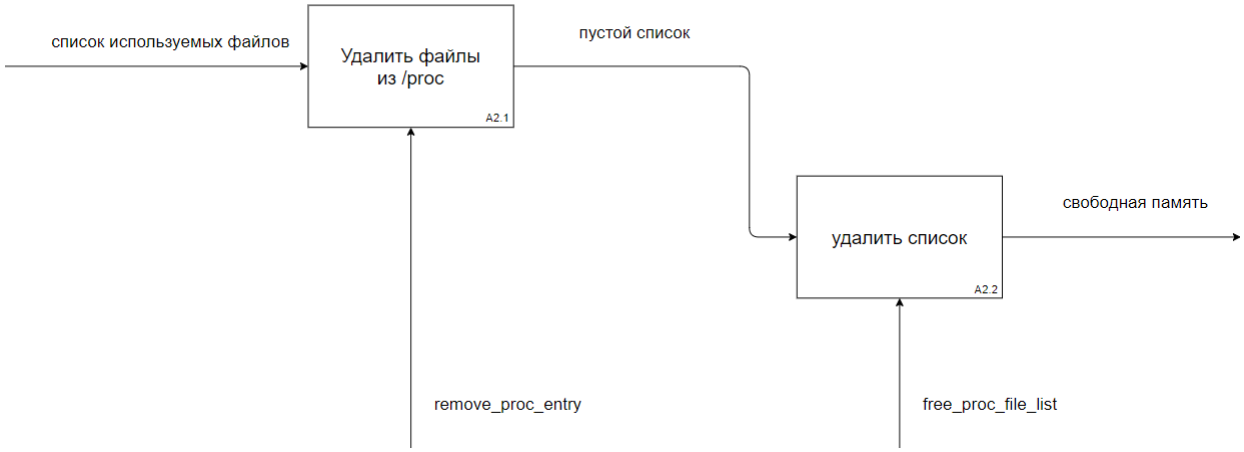

Рис. 2.2 – Последовательность действий при выгрузке модуля

При загрузке модуля ядра в /proc создаётся директория finder с файлами command и help. Файл help возможно только читать, он отображает справку по использованию разработанного модуля. В файл command необходимо записать имя файла для отслеживания. Последовательность выполняемых действий отображена на рисунке 2.3.

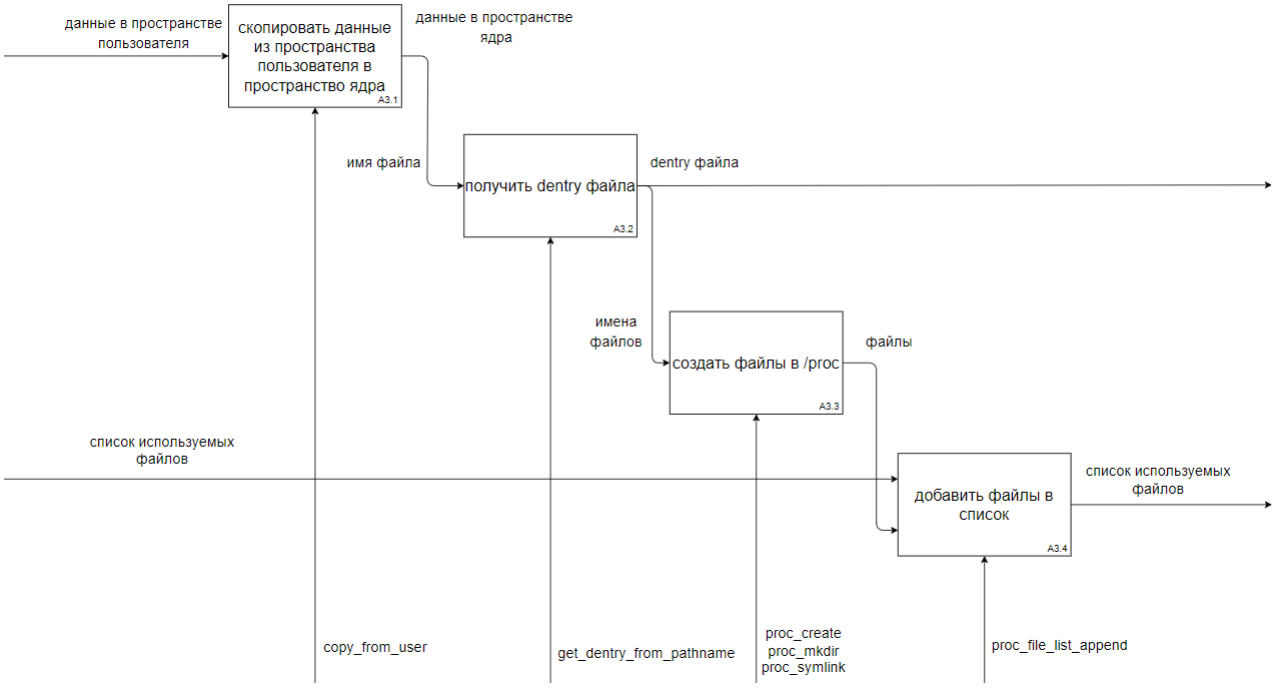

Рис. 2.3 – Последовательность действий при записи в файл command

После записи в файл command создаётся директория с именем отслеживаемого файла, в которой создаются файлы report и symlink. Файл symlink является символической ссылкой на отслеживаемый файл. При чтении файла report отображается информация о процессах, использующих отслеживаемый файл. Последовательность выполняемых действий отображена на рисунке 2.4.

При записи в файл report номера сигнала, данный сигнал рассылается всем процессам, использующим отслеживаемый файл. Последовательность действий отображена на рисунке 2.5.

Рис. 2.4 – Последовательность действий при чтении из файла report

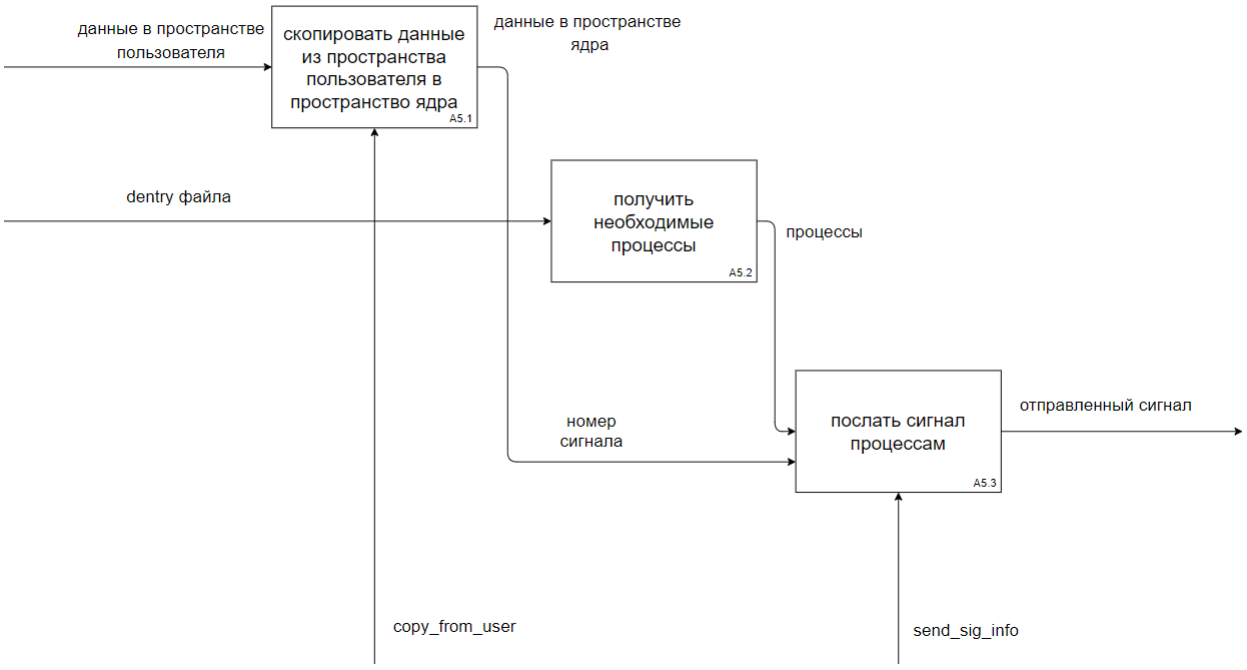

Рис. 2.5 – Последовательность действий при записи в файл report

2.2 Разработка алгоритмов

На рисунке 2.6 представлены схемы алгоритмов загрузки и выгрузки модуля.

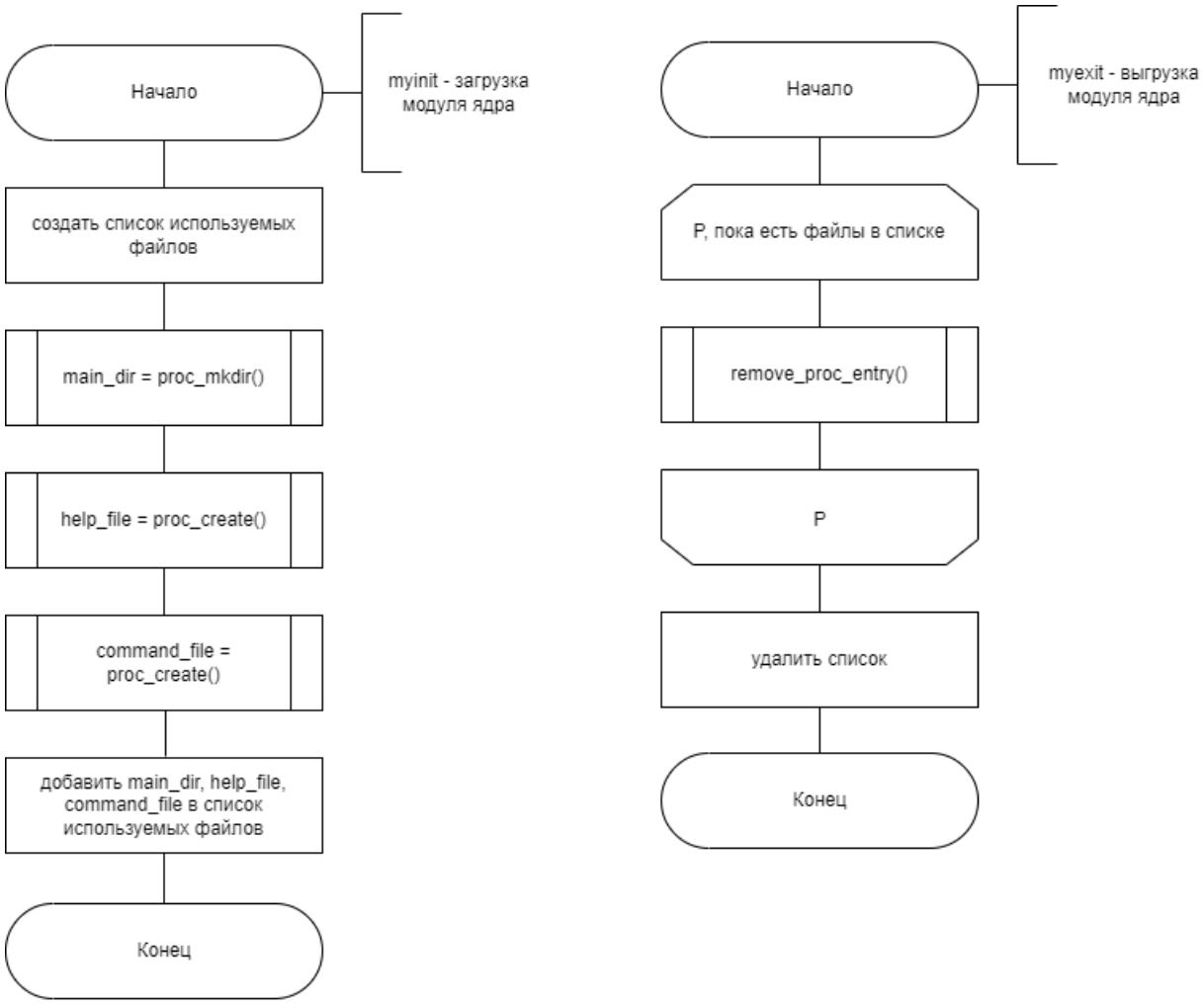

Рис. 2.6 – Схемы алгоритмов загрузки и выгрузки модуля

На рисунке 2.7 представлены схемы алгоритмов записи в файл command и чтения из файла report.

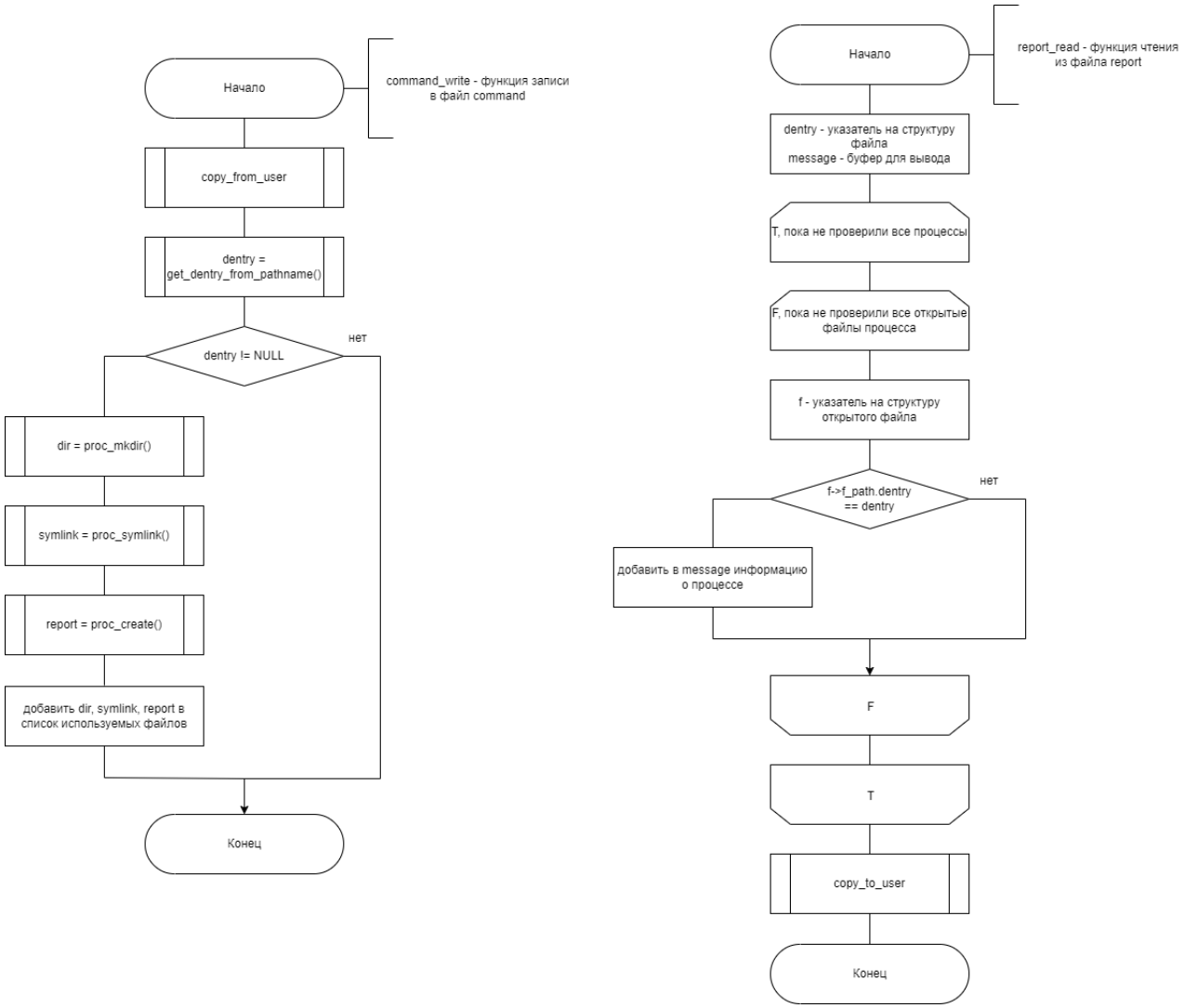

Рис. 2.7 – Схемы алгоритмов записи в файл command и чтения из файла report

На рисунке 2.8 представлена схема алгоритма записи в файл report.

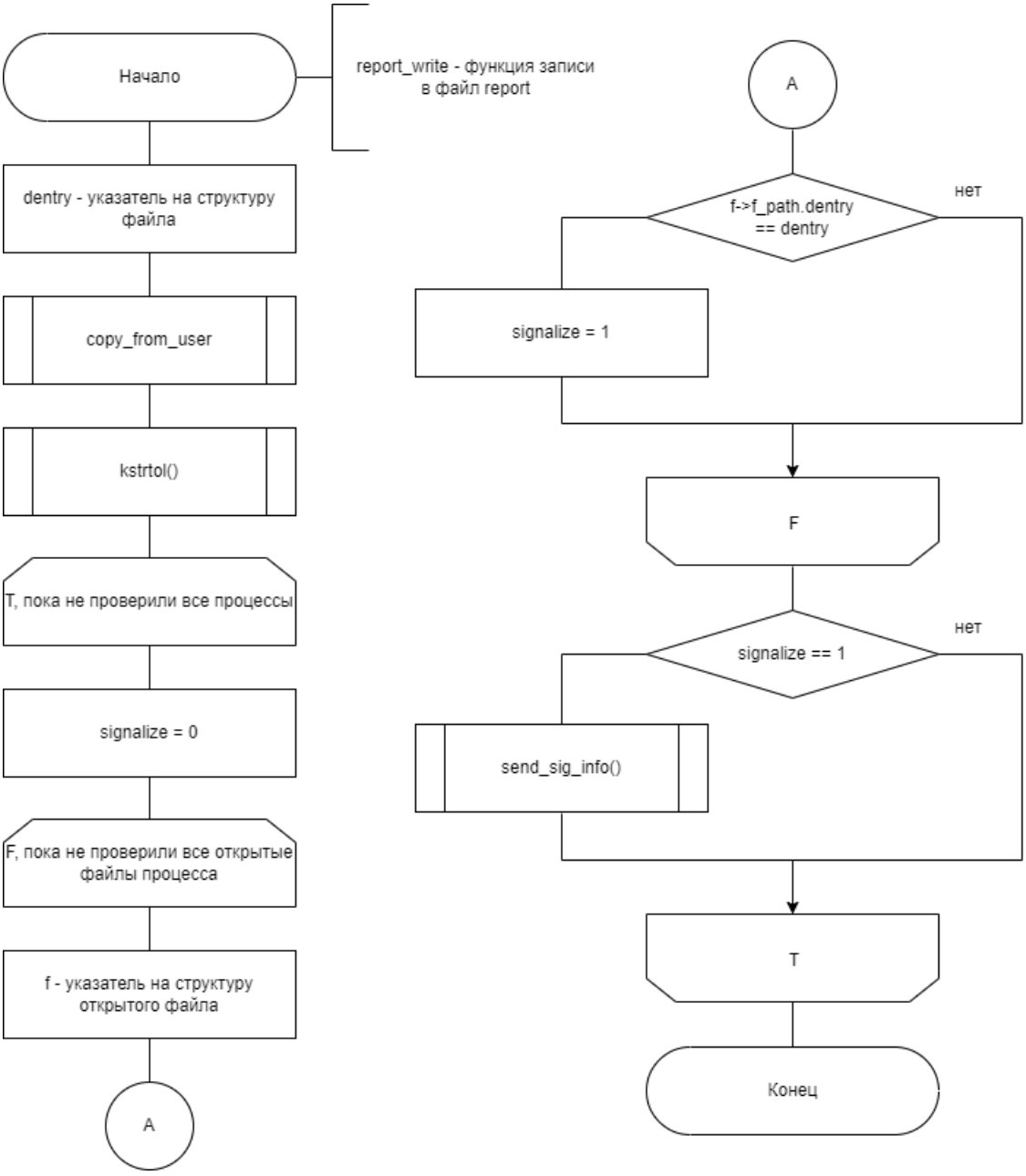

Рис. 2.8 – Схема алгоритма записи в файл report

3 Технологический раздел

3.1 Выбор языка и среды программирования

Для реализации ПО был выбран язык программирования Си [8], поскольку в нём есть все инструменты для реализации загружаемого модуля ядра. Средой программирования послужил графический редактор Visual Studio Code [9], так как в нём много плагинов, улучшающих процесс разработки.

3.2 Реализация загружаемого модуля

В листингах 3.1–3.2 представлена функция загрузки модуля, а в листинге 3.3 функция выгрузки модуля.

Листинг 3.1 – Функция загрузки модуля

```
1  static int __init myinit(void)
2  {
3      command_info[0] = 0;
4      if (alloc_proc_file_list(&proc_file_list, 10, 2))
5      {
6          return -ENOMEM;
7      }
8      if (!(main_dir = proc_mkdir(MAIN_DIR, NULL)))
9      {
10         free_proc_file_list(&proc_file_list);
11         return -ENOMEM;
12     }
13     if (!(help_file = proc_create(HELP_FILE, 0444, main_dir, &
           help_ops)))
14     {
15         free_proc_file_list(&proc_file_list);
16         remove_proc_entry(MAIN_DIR, NULL);
17         return -ENOMEM;
18     }
19     if (!(command_file = proc_create(COMMAND_FILE, 0646, main_dir, &
           command_ops)))
20     {
21         free_proc_file_list(&proc_file_list);
```

Листинг 3.2 – Функция загрузки модуля

```
1         remove_proc_entry(HELP_FILE, NULL);
2         remove_proc_entry(MAIN_DIR, NULL);
3         return -ENOMEM;
4     }
5     proc_file_list_append(&proc_file_list, MAIN_DIR, NULL);
6     proc_file_list_append(&proc_file_list, HELP_FILE, main_dir);
7     proc_file_list_append(&proc_file_list, COMMAND_FILE, main_dir);
8     return 0;
9 }
```

Листинг 3.3 – Функция выгрузки модуля

```
1  static void __exit myexit(void)
2  {
3      proc_file_t *pfile;
4      while ((pfile = proc_file_list_pop(&proc_file_list)))
5      {
6          remove_proc_entry(pfile->name, pfile->parent);
7          kfree(pfile->name);
8          kfree(pfile);
9      }
10     free_proc_file_list(&proc_file_list);
11 }
```

В листингах 3.4–3.5 представлена функция записи в файл command.

Листинг 3.4 – Функция записи в файл command

```
1  static ssize_t command_write(struct file *file, const char __user *
       ubuf, size_t count, loff_t *ppos)
2  {
3      if (dentry_library.len == MAX_LIBRARY_LEN)
4      {
5          sprintf(command_info, "Невозможно␣отслеживать␣более␣%d␣файлов\n"
               , MAX_LIBRARY_LEN);
6          return count;
7      }
8      char kbuf[10 * MAX_FILE_LEN + 1];
9      if (copy_from_user(kbuf, ubuf, count))
10     return -EFAULT;
```

Листинг 3.5 – Функция записи в файл command

```
1     kbuf[count - 1] = 0;
2     struct dentry *dentry = get_dentry_from_pathname(kbuf);
3     struct proc_dir_entry *dir;
4     struct proc_dir_entry *report_file;
5     int status = 0;
6     if (!dentry)
7     sprintf(command_info, "Данного␣пути␣не␣существует␣в␣системе");
8     else if (!(dir = proc_mkdir(dentry->d_name.name, main_dir)))
9     status = -ENOMEM;
10    else if (!(proc_symlink("symlink", dir, kbuf)))
11    {
12        remove_proc_entry(dentry->d_name.name, main_dir);
13        status = -ENOMEM;
14    }
15    else if (!(report_file = proc_create("report", 0646, dir, &
          report_ops)))
16    {
17        remove_proc_entry("symlink", dir);
18        remove_proc_entry(dentry->d_name.name, main_dir);
19        status = -ENOMEM;
20    }
21    else if (status)
22    sprintf(command_info, "Ошибка␣при␣создании␣директорий\n");
23    else
24    {
25        proc_file_list_append(&proc_file_list, dentry->d_name.name,
              main_dir);
26        proc_file_list_append(&proc_file_list, "symlink", dir);
27        proc_file_list_append(&proc_file_list, "report", dir);
28        dentry_library.array[(dentry_library.len)++] = dentry;
29        sprintf(command_info, "Путь␣распознан.␣Создана␣папка␣%s,␣
              proc_dir_entry=%p\n", dentry->d_name.name, (void *)
              report_file);
30    }
31    return count;
32 }
```

В листингах 3.6–3.7 представлена функция чтения из файла report.

Листинг 3.6 – Функция чтения из файла report

```
1  static ssize_t report_read(struct file *file, char __user *ubuf,
       size_t count, loff_t *ppos)
2  {
3      if (*ppos > 0)
4      return 0;
5
6      int len = 0;
7      char message[20 * MAX_FILE_LEN + 1];
8
9      int i = 0;
10     while (i < dentry_library.len && strcmp(dentry_library.array[i
         ]->d_name.name, file->f_path.dentry->d_parent->d_name.name))
11     ++i;
12     struct dentry *dentry = dentry_library.array[i];
13     if (!dentry)
14     {
15         sprintf(command_info, "Что - то пошло не так\n");
16         len += sprintf(message, "Что - то пошло не так\n");
17     }
18     else
19     {
20         sprintf(command_info, "Команда выполнена успешно\n");
21         len += sprintf(message + len, "%7s %7s %7s %7s %7s %7s\n", "
             PPID", "PID", "FD", "PRIO", "STATE", "COMMAND");
22         struct task_struct *task = &init_task;
23         do
24         {
25             struct files_struct *files = task->files;
26             struct fdtable *fdt = files->fdt;
27             if (fdt && files)
28             {
29                 for (int i = 0; i < files->next_fd; ++i)
30                 {
31                     struct file *f = (fdt->fd)[i];
32                     if (f->f_path.dentry == dentry)
33                     {
```

Листинг 3.7 – Функция чтения из файла report

```
                        len += sprintf(message + len, "%7d %7d %7d
                            %7d %7d %s\n", task->parent->pid, task->
                            pid, i, task->prio, task->__state, task->
                            comm);
                    }
                }
            }
        }
        while ((task = next_task(task)) != &init_task);
    }
    if (copy_to_user(ubuf, message, len))
    return -EFAULT;
    *ppos += len;
    return len;
}
```

В листингах 3.8–3.9 представлена функция записи в файл report.

Листинг 3.8 – Функция записи в файл report

```
static ssize_t report_write(struct file *file, const char __user *
    ubuf, size_t count, loff_t *ppos)
{
    int i = 0;
    while (i < dentry_library.len && strcmp(dentry_library.array[i
        ]->d_name.name, file->f_path.dentry->d_parent->d_name.name))
    ++i;
    struct dentry *dentry = dentry_library.array[i];
    if (!dentry)
    sprintf(command_info, "Что - то пошло не так\n");
    else
    {
        char kbuf[10 * MAX_FILE_LEN + 1];
        if (copy_from_user(kbuf, ubuf, count))
        {
            sprintf(command_info, "Не удалось скопировать данные из
                пользовательского режима\n");
            return -EFAULT;
        }
        kbuf[count - 1] = 0;
```

Листинг 3.9 – Функция записи в файл report

```
1         long sig;
2         int res = kstrtol(kbuf, 10, &sig);
3         if (res)
4         {
5             sprintf(command_info, "Введены␣некорректные␣данные\n");
6             return res;
7         }
8         struct task_struct *task = &init_task;
9         do
10        {
11            struct files_struct *files = task->files;
12            struct fdtable *fdt = files->fdt;
13            if (fdt && files)
14            {
15                int signalize = 0;
16                for (int i = 0; !res && i < files->next_fd; ++i)
17                {
18                    struct file *f = (fdt->fd)[i];
19                    if (f->f_path.dentry == dentry)
20                    {
21                        signalize = 1;
22                    }
23                }
24                if (signalize)
25                res = send_sig_info(sig, 1, task);
26            }
27        }
28        while (!res && (task = next_task(task)) != &init_task);
29    }
30    sprintf(command_info, "Команда␣выполнена␣успешно\n");
31    return count;
32 }
```

Для файлов были созданы экземпляры структуры proc_ops, они представлены в листинге 3.10.

Листинг 3.10 – Экземпляры структуры proc_ops

```
static struct proc_ops help_ops =
{
    .proc_read = help_read,
    .proc_open = myopen,
    .proc_release = myrelease,
};
static struct proc_ops command_ops =
{
    .proc_read = command_read,
    .proc_write = command_write,
    .proc_open = myopen,
    .proc_release = myrelease,
};
static struct proc_ops report_ops =
{
    .proc_read = report_read,
    .proc_write = report_write,
    .proc_open = myopen,
    .proc_release = myrelease,
};
```

Весь код программы представлен в Приложении А.

Для компиляции использовался makefile с наполнением, представленным в листингах 3.11–3.12.

Листинг 3.11 – Makefile проекта

```
TARGET := finder
OBJS := main.o list.o
obj-m += $(TARGET).o
$(TARGET)-objs := $(OBJS)

KDIR ?= /lib/modules/$(shell uname -r)/build

ccflags-y += -std=gnu18 -Wall

all:
    make -C $(KDIR) M=$(shell pwd) modules
```

Листинг 3.12 – Makefile проекта

```
1 $(TARGET).o: $(OBJS)
2     $(LD) -r -o $@ $(OBJS)
3
4 reader:
5     gcc -o reader.out reader.c
6
7 clean:
8     make -C $(KDIR) M=$(shell pwd) clean
```

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором запускалась программа:

1. операционная система Ubuntu, 20.04.4 [3];
2. память 8 ГБ;
3. процессор 2,4 ГГц 4-ядерный процессор Intel Core i5-1135G7 [10].

4.2 Исследование работы программы

Для исследования работы была разработана вспомогательная программа, которая открывает 10 раз один и тот же файл, а потом прерывается в ожидании ввода. Код вспомогательной программы представлен в листинге 4.1–4.2.

Листинг 4.1 – Вспомогательная программа

```
1  # include <stdio.h>
2  # include <signal.h>
3
4  int flag = 0;
5
6  void handler(int s)
7  {
8      flag = 1;
9  }
10
11 int main()
12 {
13     signal(SIGINT, handler);
14     FILE *fd[10];
15     for (int i = 0; i < 10; ++i)
16         fd[i] = fopen("qwerty.txt", "r");
17     char c;
18     scanf("%c", &c);
19     for (int i = 0; i < 10; ++i)
```

Листинг 4.2 – Вспомогательная программа

```
        fclose(fd[i]);
    if (flag)
        printf("Процесс принял сигнал SIGINT\n");
    return 0;
}
```

На рисунке 4.1 продемонстрирована работа загружамого модуля:

1. sudo insmod finder.ko — загрузка модуля;
2. cd /proc/finder — переход в рабочую директорию модуля;
3. echo "/home/alex/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw/src/qwerty.txt"> command — создание папки для отслеживания заданного файла;
4. cd qwerty.txt — переход в папку отслеживания;
5. cat report — чтение файла report;
6. echo "2"> report — посылка сигнала процессам;
7. sudo rmmod finder — выгрузка модуля.

```
alex@alex-HP-Laptop-15s-fq2xxx:~/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw$ make
make: *** No targets specified and no makefile found.  Stop.
alex@alex-HP-Laptop-15s-fq2xxx:~/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw$ cd src/
alex@alex-HP-Laptop-15s-fq2xxx:~/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw/src$ sudo insmod finder.ko
[sudo] password for alex:
alex@alex-HP-Laptop-15s-fq2xxx:~/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw/src$ sudo rmmod finder
alex@alex-HP-Laptop-15s-fq2xxx:~/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw/src$ sudo insmod finder.ko
alex@alex-HP-Laptop-15s-fq2xxx:~/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw/src$ cd /proc/finder/
alex@alex-HP-Laptop-15s-fq2xxx:/proc/finder$ echo "/home/alex/Bomonka/Semester_7/Operating_systems/bmstu-semester_7-os_cw/src/qwerty.txt" > command
alex@alex-HP-Laptop-15s-fq2xxx:/proc/finder$ cd qwerty.txt/
alex@alex-HP-Laptop-15s-fq2xxx:/proc/finder/qwerty.txt$ cat report
   PPID     PID      FD    PRIO   STATE COMMAND
   4507    4526       3     120       1 reader.out
   4507    4526       4     120       1 reader.out
   4507    4526       5     120       1 reader.out
   4507    4526       6     120       1 reader.out
   4507    4526       7     120       1 reader.out
   4507    4526       8     120       1 reader.out
   4507    4526       9     120       1 reader.out
   4507    4526      10     120       1 reader.out
   4507    4526      11     120       1 reader.out
   4507    4526      12     120       1 reader.out
alex@alex-HP-Laptop-15s-fq2xxx:/proc/finder/qwerty.txt$ echo "2" > report
alex@alex-HP-Laptop-15s-fq2xxx:/proc/finder/qwerty.txt$ sudo rmmod finder
alex@alex-HP-Laptop-15s-fq2xxx:/proc/finder/qwerty.txt$
```

Рис. 4.1 – Демонстрация работы программы

На рисунке 4.2 продемонстрирован приём сигнала вспомогательной программой.

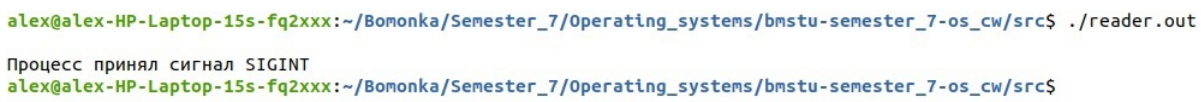

Рис. 4.2 – Демонстрация работы вспомогательной программы

Заключение

Цель, поставленная в начале, была достигнута: разработан загружаемый модуль ядра для получения информации о процессах, которые используют заданный пользователем файл, с возможностью посылки сигналов рассматриваемым процессам.

В ходе выполнения курсовой работы были решены следующие задачи:

1) произведён анализ структур и функций, предоставляющих возможность реализовать поставленную задачу;

2) разработаны алгоритмы и структуры загружаемого модуля ядра, в соответствии с поставленной задачей;

3) реализован спроектированный модуль ядра;

4) реализованный модуль ядра протестирован.

Список используемых источников

1. Desktop Operating System Market Share Worldwide [Электронный ресурс]. Режим доступа: https://gs.statcounter.com/os-market-share/desktop/worldwide (дата обращения: 15.01.2024).

2. Mobile Operating System Market Share Worldwide [Электронный ресурс]. Режим доступа: https://gs.statcounter.com/os-market-share/desktop/worldwide (дата обращения: 15.01.2024).

3. Enterprise Open Source and Linux | Ubuntu [Электронный ресурс]. Режим доступа: https://ubuntu.com/ (дата обращения: 15.01.2024).

4. Linux source code, struct task_struct [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h#L746 (дата обращения: 15.01.2024).

5. Linux source code [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source (дата обращения: 15.01.2024).

6. Linux source code, struct file [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h#L992 (дата обращения: 15.01.2024).

7. Halo Linux Services — Generating a signal [Электронный ресурс]. Режим доступа: https://www.halolinux.us/kernel-reference/generating-a-signal.html (дата обращения: 15.01.2024).

8. C language documentation [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/cpp/c-language/?view=msvc-170 (дата обращения: 15.01.2024).

9. Visual Studio Code [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/ (дата обращения: 15.01.2024).

10. Процессор Intel® Core™ i5 [Электронный ресурс]. Режим доступа: https://www.intel.com/processors/core/i5/docs (дата обращения: 15.12.2023).

ПРИЛОЖЕНИЕ А

Листинг 4.3 – Файл main.c

```
1  #include <linux/module.h>
2  #include <linux/kernel.h>
3  #include <linux/proc_fs.h>
4  #include <linux/init.h>
5  #include <linux/path.h>
6  #include <linux/fs_struct.h>
7  #include <linux/fs.h>
8  #include <linux/namei.h>
9  #include <linux/file.h>
10 #include <linux/fdtable.h>
11 #include <linux/signal.h>
12 #include <linux/version.h>
13 #include <linux/uaccess.h>
14 #include <linux/slab.h>
15
16 #include "help.h"
17 #include "list.h"
18
19 #if LINUX_VERSION_CODE >= KERNEL_VERSION(5,6,0)
20 #define HAVE_PROC_OPS
21 #endif
22
23 #ifdef HAVE_PROC_OPS
24 static struct proc_ops myops;
25 static struct proc_ops help_ops;
26 static struct proc_ops command_ops;
27 static struct proc_ops report_ops;
28 #else
29 static struct file_operations myops;
30 static struct file_operations help_ops;
31 static struct file_operations command_ops;
32 static struct file_operations report_ops;
33 #endif
34
35 extern char *help_info;
36
37 MODULE_LICENSE("GPL");
```

Листинг 4.4 – Файл main.c

```
1  MODULE_AUTHOR("ALEX");
2  #define BUFSIZE PAGE_SIZE
3  #define HELPBUF 100
4
5  # define MAIN_DIR "finder"
6  static struct proc_dir_entry *main_dir;
7  # define HELP_FILE "help"
8  static struct proc_dir_entry *help_file;
9  # define COMMAND_FILE "command"
10 static struct proc_dir_entry *command_file;
11 static proc_file_list_t proc_file_list;
12
13 # define MAX_LIBRARY_LEN 20
14
15 static struct dentry_library
16 {
17     int len;
18     struct dentry *array[MAX_LIBRARY_LEN];
19 } dentry_library = {.len = 0};
20
21 static char command_info[10 * MAX_FILE_LEN + 1];
22
23 static struct dentry *get_dentry_from_pathname(const char *pathname)
24 {
25     struct path path;
26     int error = kern_path(pathname, LOOKUP_FOLLOW, &path);
27     if (!error)
28     return path.dentry;
29     else
30     return NULL;
31 }
32
33 static ssize_t report_write(struct file *file, const char __user *
       ubuf, size_t count, loff_t *ppos)
34 {
35     int i = 0;
36     while (i < dentry_library.len && strcmp(dentry_library.array[i
           ]->d_name.name, file->f_path.dentry->d_parent->d_name.name))
37     ++i;
38     struct dentry *dentry = dentry_library.array[i];
```

Листинг 4.5 – Файл main.c

```
1     if (!dentry)
2     sprintf(command_info, "Чтото-пошло не так\n");
3     else
4     {
5         char kbuf[10 * MAX_FILE_LEN + 1];
6         if (copy_from_user(kbuf, ubuf, count))
7         {
8             sprintf(command_info, "Не удалось скопировать данные из
                  пользовательского режима\n");
9             return -EFAULT;
10        }
11        kbuf[count - 1] = 0;
12
13        long sig;
14        int res = kstrtol(kbuf, 10, &sig);
15        if (res)
16        {
17            sprintf(command_info, "Введены некорректные данные\n");
18            return res;
19        }
20        struct task_struct *task = &init_task;
21        do
22        {
23            struct files_struct *files = task->files;
24            struct fdtable *fdt = files->fdt;
25            if (fdt && files)
26            {
27                int signalize = 0;
28                for (int i = 0; !res && i < files->next_fd; ++i)
29                {
30                    struct file *f = (fdt->fd)[i];
31                    if (f->f_path.dentry == dentry)
32                    {
33                        signalize = 1;
34                    }
35                }
36                if (signalize)
37                res = send_sig_info(sig, 1, task);
38            }
39        }
```

Листинг 4.6 – Файл main.c

```
1           while (!res && (task = next_task(task)) != &init_task);
2       }
3       sprintf(command_info, "Команда␣выполнена␣успешно\n");
4       return count;
5   }
6
7   static ssize_t report_read(struct file *file, char __user *ubuf,
        size_t count, loff_t *ppos)
8   {
9       if (*ppos > 0)
10      return 0;
11
12      int len = 0;
13      char message[20 * MAX_FILE_LEN + 1];
14
15      int i = 0;
16      while (i < dentry_library.len && strcmp(dentry_library.array[i
            ]->d_name.name, file->f_path.dentry->d_parent->d_name.name))
17      ++i;
18      struct dentry *dentry = dentry_library.array[i];
19      if (!dentry)
20      {
21          sprintf(command_info, "Чтото-␣пошло␣не␣так\n");
22          len += sprintf(message, "Чтото-␣пошло␣не␣так\n");
23      }
24      else
25      {
26          sprintf(command_info, "Команда␣выполнена␣успешно\n");
27          len += sprintf(message + len, "%7s␣%7s␣%7s␣%7s␣%7s␣%7s\n", "
               PPID", "PID", "FD", "PRIO", "STATE", "COMMAND");
28          struct task_struct *task = &init_task;
29          do
30          {
31              struct files_struct *files = task->files;
32              struct fdtable *fdt = files->fdt;
33              if (fdt && files)
34              {
35                  for (int i = 0; i < files->next_fd; ++i)
36                  {
37                      struct file *f = (fdt->fd)[i];
```

Листинг 4.7 – Файл main.c

```
1                       if (f->f_path.dentry == dentry)
2                       {
3                           len += sprintf(message + len, "%7d %7d %7d
                                %7d %7d %s\n", task->parent->pid, task->
                                pid, i, task->prio, task->__state, task->
                                comm);
4                       }
5                   }
6               }
7           }
8           while ((task = next_task(task)) != &init_task);
9       }
10      if (copy_to_user(ubuf, message, len))
11      return -EFAULT;
12      *ppos += len;
13      return len;
14  }
15
16  static ssize_t command_write(struct file *file, const char __user *
        ubuf, size_t count, loff_t *ppos)
17  {
18      if (dentry_library.len == MAX_LIBRARY_LEN)
19      {
20          sprintf(command_info, "Невозможно отслеживать более %d файлов\n"
                , MAX_LIBRARY_LEN);
21          return count;
22      }
23      char kbuf[10 * MAX_FILE_LEN + 1];
24      if (copy_from_user(kbuf, ubuf, count))
25      return -EFAULT;
26      kbuf[count - 1] = 0;
27      struct dentry *dentry = get_dentry_from_pathname(kbuf);
28      struct proc_dir_entry *dir;
29      struct proc_dir_entry *report_file;
30      int status = 0;
31      if (!dentry)
32      sprintf(command_info, "Данного пути не существует в системе");
33      else if (!(dir = proc_mkdir(dentry->d_name.name, main_dir)))
34      status = -ENOMEM;
35      else if (!(proc_symlink("symlink", dir, kbuf)))
```

Листинг 4.8 – Файл main.c

```
1     {
2         remove_proc_entry(dentry->d_name.name, main_dir);
3         status = -ENOMEM;
4     }
5     else if (!(report_file = proc_create("report", 0646, dir, &
          report_ops)))
6     {
7         remove_proc_entry("symlink", dir);
8         remove_proc_entry(dentry->d_name.name, main_dir);
9         status = -ENOMEM;
10    }
11    else if (status)
12    sprintf(command_info, "Ошибка при создании директорий\n");
13    else
14    {
15        proc_file_list_append(&proc_file_list, dentry->d_name.name,
              main_dir);
16        proc_file_list_append(&proc_file_list, "symlink", dir);
17        proc_file_list_append(&proc_file_list, "report", dir);
18        dentry_library.array[(dentry_library.len)++] = dentry;
19        sprintf(command_info, "Путь распознан. Создана папка %s, 
              proc_dir_entry=%p\n", dentry->d_name.name, (void *)
              report_file);
20    }
21    return count;
22 }
23
24 static ssize_t command_read(struct file *file, char __user *ubuf,
       size_t count, loff_t *ppos)
25 {
26    if (*ppos > 0)
27    return 0;
28
29    int len;
30    if (command_info[0])
31    {
32        len = strlen(command_info);
33        if (copy_to_user(ubuf, command_info, len))
34        return -EFAULT;
35    }
```

Листинг 4.9 – Файл main.c

```
    else
    {
        len = strlen(help_info);
        if (copy_to_user(ubuf, help_info, len))
        return -EFAULT;
    }
    *ppos += len;
    return len;
}

static ssize_t help_read(struct file *file, char __user *ubuf,size_t
    count, loff_t *ppos)
{
    if (*ppos > 0)
    return 0;

    int len = strlen(help_info);
    if (copy_to_user(ubuf, help_info, len))
    return -EFAULT;
    *ppos += len;
    return len;
}

static int myopen(struct inode *inode, struct file *file)
{
    struct hlist_node *list = (inode->i_dentry).first;
    struct dentry *d = list_entry(list, struct dentry, d_u.d_alias);
    printk(KERN_INFO "file %s is opened", d->d_name.name);
    return 0;
}

static int myrelease(struct inode *inode, struct file *file)
{
    struct hlist_node *list = (inode->i_dentry).first;
    struct dentry *d = list_entry(list, struct dentry, d_u.d_alias);
    printk(KERN_INFO "file %s is released", d->d_name.name);
    return 0;
}
```

Листинг 4.10 – Файл main.c

```
#ifdef HAVE_PROC_OPS
static struct proc_ops help_ops =
{
    .proc_read = help_read,
    .proc_open = myopen,
    .proc_release = myrelease,
};
static struct proc_ops command_ops =
{
    .proc_read = command_read,
    .proc_write = command_write,
    .proc_open = myopen,
    .proc_release = myrelease,
};
static struct proc_ops report_ops =
{
    .proc_read = report_read,
    .proc_write = report_write,
    .proc_open = myopen,
    .proc_release = myrelease,
};
#else
static struct file_operations help_ops =
{
    .read = help_read,
    .open = myopen,
    .release = myrelease,
};
static struct file_operations command_ops =
{
    .read = command_read,
    .write = command_write,
    .open = myopen,
    .release = myrelease,
};
static struct file_operations report_ops =
{
    .read = report_read,
    .write = report_write,
    .open = myopen,
```

Листинг 4.11 – Файл main.c

```
1       .release = myrelease,
2   };
3   #endif
4
5   static int __init myinit(void)
6   {
7       command_info[0] = 0;
8       if (alloc_proc_file_list(&proc_file_list, 10, 2))
9       {
10          return -ENOMEM;
11      }
12      if (!(main_dir = proc_mkdir(MAIN_DIR, NULL)))
13      {
14          free_proc_file_list(&proc_file_list);
15          return -ENOMEM;
16      }
17      if (!(help_file = proc_create(HELP_FILE, 0444, main_dir, &
            help_ops)))
18      {
19          free_proc_file_list(&proc_file_list);
20          remove_proc_entry(MAIN_DIR, NULL);
21          return -ENOMEM;
22      }
23      if (!(command_file = proc_create(COMMAND_FILE, 0646, main_dir, &
            command_ops)))
24      {
25          free_proc_file_list(&proc_file_list);
26          remove_proc_entry(HELP_FILE, NULL);
27          remove_proc_entry(MAIN_DIR, NULL);
28          return -ENOMEM;
29      }
30      proc_file_list_append(&proc_file_list, MAIN_DIR, NULL);
31      proc_file_list_append(&proc_file_list, HELP_FILE, main_dir);
32      proc_file_list_append(&proc_file_list, COMMAND_FILE, main_dir);
33      return 0;
34  }
35
36  static void __exit myexit(void)
37  {
38      proc_file_t *pfile;
```

Листинг 4.12 – Файл main.c

```
    while ((pfile = proc_file_list_pop(&proc_file_list)))
    {
        remove_proc_entry(pfile->name, pfile->parent);
        kfree(pfile->name);
        kfree(pfile);
    }
    free_proc_file_list(&proc_file_list);
}

module_init(myinit);
module_exit(myexit);
```

Листинг 4.13 – Файл list.c

```
#include "list.h"

int alloc_proc_file_list(proc_file_list_t *list, int size, int k)
{
    list->array = kmalloc(size * sizeof(proc_file_t), GFP_KERNEL);
       // GFP_KERNEL – обычныйзапрос , которыйможноблокировать
    if (list->array)
    {
        list->k = k;
        list->size = size;
        list->len = 0;
        return 0;
    }
    return -1;
}

int proc_file_list_append(proc_file_list_t *list, const char *name,
   struct proc_dir_entry *pdentry)
{
    if (list->len == list->size)
    {
        proc_file_t *new_pointer = krealloc(list->array, list->size
           * list->k, GFP_KERNEL);
        if (!new_pointer)
        return -ENOMEM;
        list->array = new_pointer;
```

Листинг 4.14 – Файл list.c

```
1          list->size *= list->k;
2      }
3      char *new_name = kmalloc((MAX_FILE_LEN + 1) * sizeof(char),
           GFP_KERNEL);
4      if (!new_name)
5      return -ENOMEM;
6      strcpy(new_name, name);
7      list->array[list->len].name = new_name;
8      list->array[list->len].parent = pdentry;
9      ++(list->len);
10     return 0;
11 }
12
13 proc_file_t *proc_file_list_pop(proc_file_list_t *list)
14 {
15     if (!list->len)
16     return NULL;
17     proc_file_t *result = kmalloc(sizeof(proc_file_t), GFP_KERNEL);
18     if (!result)
19     return NULL;
20     --(list->len);
21     memcpy(result, list->array + list->len, sizeof(proc_file_t));
22     return result;
23 }
24
25 void free_proc_file_list(proc_file_list_t *list)
26 {
27     kfree(list->array);
28     list->array = NULL;
29 }
```

Листинг 4.15 – Файл list.h

```
1 # ifndef LIST_H
2
3 # define LIST_H
4
5 #include <linux/slab.h>
6 #include <linux/string.h>
```

Листинг 4.16 – Файл list.h

```
1  # define MAX_FILE_LEN 100
2
3  typedef struct proc_file_t
4  {
5      const char *name;
6      struct proc_dir_entry *parent;
7  } proc_file_t;
8
9  typedef struct proc_file_list_t
10 {
11     int size;
12     int len;
13     proc_file_t *array;
14     int k;
15 } proc_file_list_t;
16
17 int alloc_proc_file_list(proc_file_list_t *list, int size, int k);
18 int proc_file_list_append(proc_file_list_t *list, const char *name,
       struct proc_dir_entry *pdentry);
19 proc_file_t *proc_file_list_pop(proc_file_list_t *list);
20 void free_proc_file_list(proc_file_list_t *list);
21
22 # endif
```

Листинг 4.17 – Файл help.h

```
1  # ifndef HELP_H
2
3  # define HELP_H
4
5  static char *help_info = "Модуль␣ядра␣предназначен␣для␣взаимодействия␣с␣
       процессами,\n" \
6  "которые␣удерживают␣открытым␣определённый␣файл\n\n\n" \
7  "Для␣взаимодействия␣с␣модулем␣определён␣следующий␣порядок:\n\n" \
8  "Файл␣/proc/finder/command␣предназначен␣для␣создания␣отчёта␣о␣файле\n" \
9  "Отчёт␣создаётся␣командой:␣echo␣полноеимяфайла__␣>␣command\n" \
10 "После␣выполнения␣команды␣в␣дериктории␣/proc/finder␣создаётся␣директория\n"
       \
11 "с␣таким␣же␣именем,␣как␣и␣отслеживаемый␣файл,␣в␣ней␣находится␣символическая␣
       ссылка\n" \
```

Листинг 4.18 – Файл help.h

```
1 "на␣исследуемый␣файл␣и␣файл␣report.␣При␣чтении␣из␣файла␣report␣в␣
      терминале␣будет\n" \
2 "отображаться␣отчёт␣о␣процессах,␣использующих␣файл,␣а␣при␣записи␣в␣файл␣
      числа␣командой\n\n" \
3 "echo␣номерсигнала_␣>␣report\n\n" \
4 "всем␣процессам,␣использующим␣файл␣будет␣послан␣сигнал␣с␣номером␣номерсигнала_
      \n";
5 # endif
```