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

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Драйвер устройства сканирования отпечатков пальцев Shenzhen Goodix Technology для операционной системы Linux»

Студент ИУ7-71Б (Группа) _____ (Подпись, дата) Корниенко К. Ю. (И. О. Фамилия)

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

2024 г.

РЕФЕРАТ

Расчетно-пояснительная записка 24 с., 4 рис., 0 табл., 10 источн., 4 прил.

Курсовая работа по дисциплине «Операционные системы» на тему «Драйвер устройства сканирования отпечатков пальцев Shenzhen Goodix Technology для операционной системы Linux», студента Корниенко К. Ю.

Ключевые слова: загружаемый модуль ядра; отпечаток пальца; ОС Linux; TLS; сканер отпечатков пальцев; драйвер устройства.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной работы — написать драйвер сканера отпечатка пальца.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с техническим заданием, требуется написать драйвер сканера отпечатка пальца Shenzhen Goodix Technology [1] для операционной системы Linux.

Для решения поставленной задачи необходимо:

— проанализировать особенности работы сканера отпечатка пальца;

— рассмотреть способы получения данных от устройства с использованием подсистемы USB [2];

— разработать драйвер в виде загружаемого модуля ядра;

— провести исследование разработанного драйвера.

1.2 Конфигурация устройства

Ниже приведена конфигурация устройства, полученная из отладочной файловой системы по пути `/sys/kernel/debug/devices` [3].

```
T:  Bus=03 Lev=01 Prnt=01 Port=03 Cnt=01 Dev#=  2 Spd=12   MxCh= 0
D:  Ver= 2.00 Cls=02(comm.) Sub=01 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=27c6 ProdID=5125 Rev= 2.00
S:  Manufacturer=Shenzhen Goodix Technology Co.,Ltd.
S:  Product=Goodix Fingerprint Device
S:  SerialNumber=00000000001A
C:* #Ifs= 2 Cfg#= 1 Atr=60 MxPwr=100mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=02(comm.) Sub=01 Prot=01 Driver=(none)
E:  Ad=82(I) Atr=03(Int.) MxPS=   8 Ivl=16ms
I:* If#= 1 Alt= 0 #EPs= 2 Cls=0a(data ) Sub=00 Prot=00 Driver=(none)
E:  Ad=01(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=81(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
```

Данное устройство имеет одну конфигурацию с двумя активными интерфейсами, первый из которых предназначен для передачи управляющих комманд, а второй – для передачи данных.

1.3 Подсистема USB

USB драйвер имеет две точки входа – `probe` и `disconnect` [4], которые вызываются подсистемой USB при подключении и отключении устройства, соответствующего данному драйверу.

Обмен информацией между подключенным устройством и драйвером осуществляется пакетами по запросу. Блок запроса USB (USB Request Block – URB) представляется структурой ядра `urb`:

```
struct urb {
    struct list_head urb_list;
    struct usb_device *dev; /* (in) pointer to associated device */
    unsigned int pipe; /* (in) pipe information */
    int status; /* (return) non-ISO status */
    void *transfer_buffer; /* (in) associated data buffer */
    dma_addr_t transfer_dma; /* (in) dma addr for transfer_buffer */
    u32 transfer_buffer_length; /* (in) data buffer length */
    u32 actual_length; /* (return) actual transfer length */
    int interval; /* (modify) transfer interval */
    void *context; /* (in) context for completion */
    usb_complete_t complete; /* (in) completion routine */
    /* ... */
};
```

1.3.1 Способы передачи URB пакетов

Передача URB пакетов по шине USB является дуплексной и может происходить в одной из четырех форм, в зависимости от подключенного устройства:

— `Control` – используется для передачи управляющих команд для настройки устройства или получения его статуса.

— `Bulk` – используется для обмена большими пакетами данных. Зачастую именно эта форма передачи используется устройствами на подобие сканеров и SCSI адаптеров.

— `Interrupt` – используется для запроса передачи небольших пакетов в режиме опроса устройства. Если был запрошен пакет в режиме

прерывания, то драйвер хост-контроллера автоматически будет повторять этот запрос с определенным интервалом (1–255 мс).

— `Isochronous` – используется для передачи данных в реальном времени с гарантированной пропускной способностью шины, но ненадежно. В общем случае изохронный тип используется для передачи аудио и видео информации.

Рассматриваемое устройство – сканер отпечатка пальца – поддерживает форматы передачи `Control` (для передачи управляющих команд) и `Bulk` (для передачи изображения считанного отпечатка пальца).

1.4 Особенности взаимодействия с устройством с использованием протокола TLS

Сканер отпечатков пальцев Shenzhen Goodix Technology для Linux требует использования протокола TLS [5] 1.2 для защиты передачи данных.

Для аутентификации в данном протоколе используются асимметричные алгоритмы шифрования [6] (открытый ключ – закрытый ключ), а для сохранения конфиденциальности – симметричные (с одним, секретным, ключом), также используются и сеансовые ключи, которые необходимы для каждого отдельного уникального защищенного сеанса.

В соответствии с протоколом TLS процесс инициации сеанса, также называемый «рукопожатием», состоит из следующих шагов:

1. Клиент (в данном случае – драйвер) отправляет запрос на безопасное соединение с сервером (устройством).

2. Сервер предоставляет цифровой сертификат, подтверждающий подлинность сервера.

3. Используя открытый ключ сервера, клиент и сервер устанавливают сеансовый ключ, с помощью которого шифруются данные на протяжении всего сеанса.

Установка защищенного соединение с устройством производится через отправку URB Control блоков.

1.5 Взаимодействие драйвера с пользовательскими программами

Одним из возможных способов передачи информации из пространства ядра в пространство пользователя является использование виртуальной файловой системы proc [7].

Операции, которые могут быть осуществлены с файлом определяются структурой `proc_ops`. В данной работе необходимо и достаточно реализовать следующие операции: `proc_open`, `proc_read`, `proc_lseek`, `proc_poll`, `proc_release`.

Так как изображение отпечатка пальца имеет существенный размер, пользовательское приложение может не обработать все данные за одну операцию чтения. В ядре Linux есть интерфейс файлов последовательностей (sequence file). Использование данного интерфейса избавляет разработчика загружаемого модуля ядра от необходимости учитывать описанную проблему.

В случае, если пользователь не прикладывает палец к сканеру, процесс должен быть заблокирован при попытке чтения. Необходимость блокировки процесса и ожидания появления нового события приводит к использованию очередей ожидания, которые представляются в ядре структурой `wait_queue_head`:

```
struct wait_queue_entry {
    unsigned int flags;
    void *private;
    wait_queue_func_t func;
    struct list_head entry;
};

struct wait_queue_head {
    spinlock_t lock;
    struct list_head head;
};
```

Блокировка процесса и ожидание возникновения события осуществляется вызовом макроса

`wait_event_interruptible(wq_head, condition)`.

Все ждущие в данной очереди процессы пробуждаются посредством вызова макроса `wake_up_all(wq_head)`, после чего будет анализировано вы-

ражение `condition` переданное при блокировке. Если оно ложно, то процесс вновь блокируется.

Для поддержки механизма опроса реализуется функция `proc_poll`.

Выводы

В данном разделе была представлена конфигурация устройства сканера, изложены основные моменты работы с подсистемой USB при проектировании драйверов USB-устройств в ОС Linux. Также рассмотрены особенности взаимодействия драйвера с устройством с использованием протокола TLS и обеспечением защищенного соединения. Обсуждены способы передачи данных из пространства ядра в пространство пользователя.

2 Конструкторский раздел

В данном разделе приведены ключевые алгоритмы использованные при написании драйвера сканера отпечатка пальца.

2.1 Структура ПО

Разрабатываемое ПО состоит из драйвера сканера отпечатка пальца и пользовательского приложения для тестирования работы драйвера.

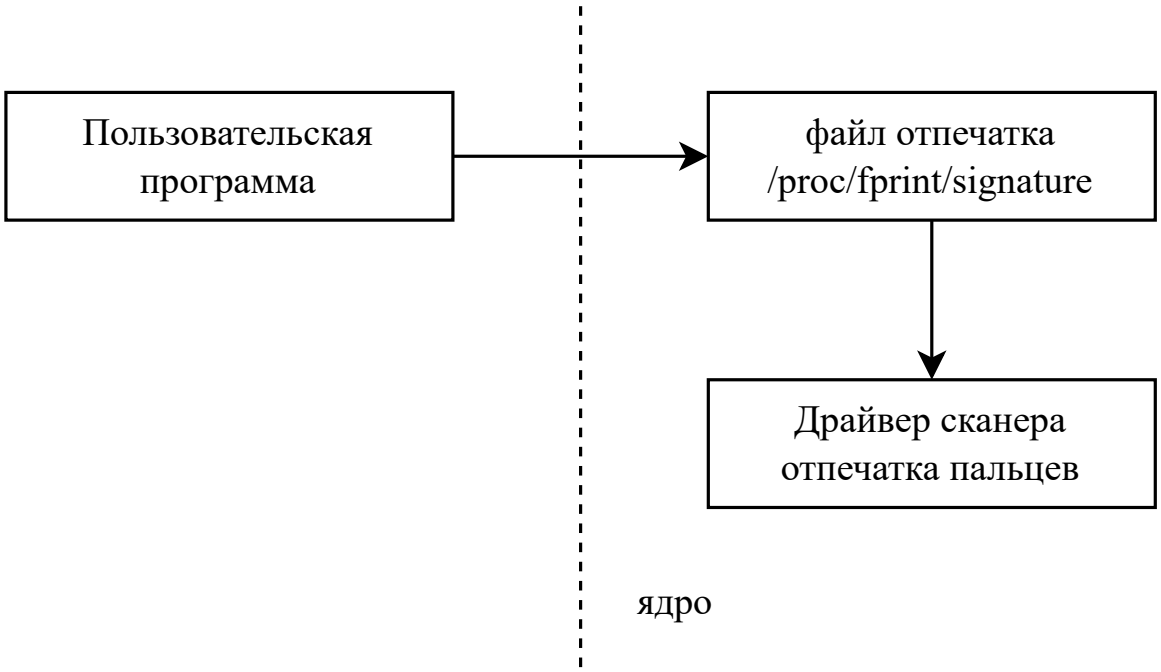

Рисунок 2.1 – Структура ПО

2.2 Алгоритм инициализации TLS-соединения

На рисунке 2.2 представлен алгоритм инициализации TLS соединения с устройством. PSK [8] – pre-shared key ciphersuites – предварительный общий ключ безопасности используемый в качестве сертификата.

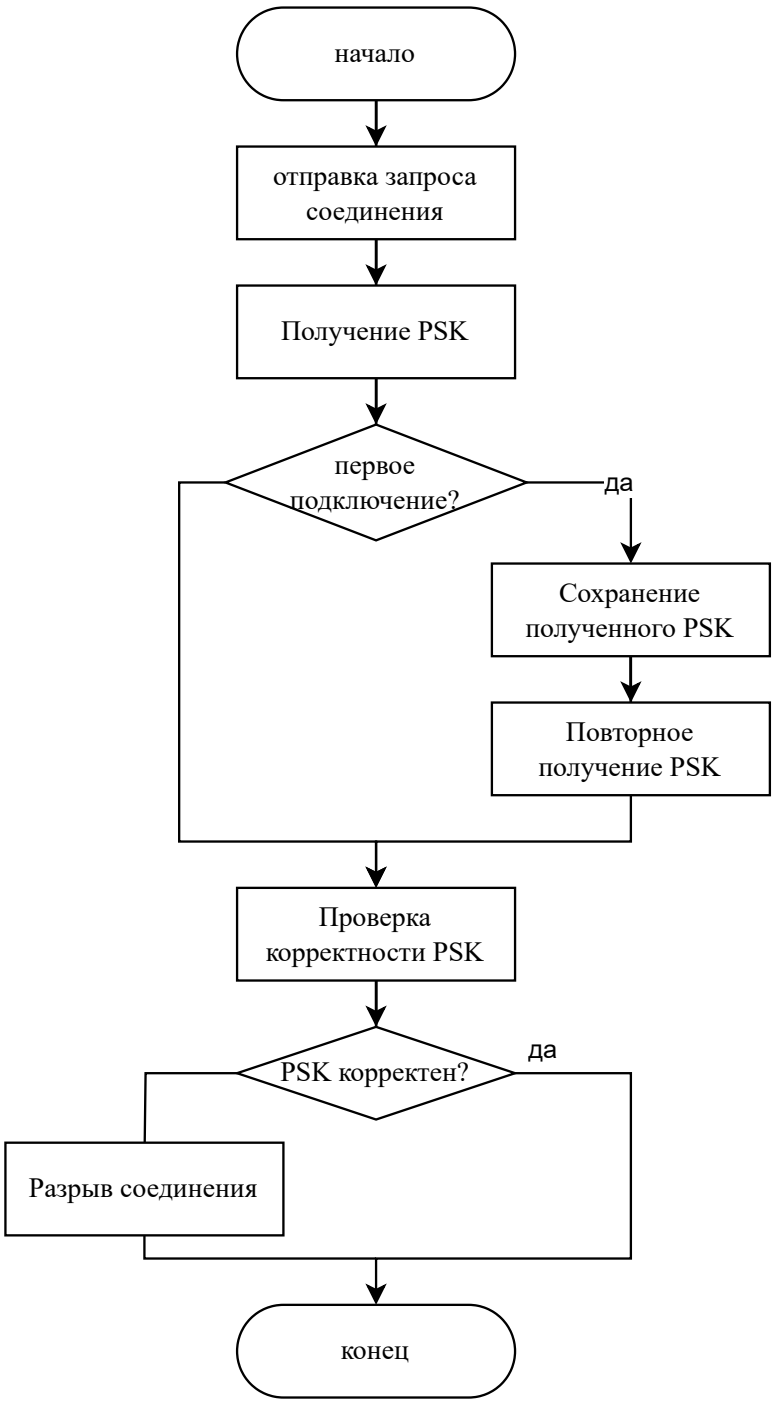

Рисунок 2.2 – Алгоритм инициализации соединения

2.3 Обработка пользовательских запросов

Сканер отпечатков пальцев является общим ресурсом для всех пользовательских приложений. Разрабатывая драйвер данного устройства важно рассмотреть все проблемы, которые могут возникнуть при обращении к драйверу различными пользовательскими приложениями в многопоточной системе. Множество возможных проблем включает в себя следующие:

1. несколько пользовательских программ могут одновременно запрашивать отпечаток;
2. несколько потоков одного процесса могут запрашивать отпечаток;
3. процесс(поток) может умереть недочитав отпечаток;
4. процесс(поток) может недочитать отпечаток.

В качестве решения для указанных проблем можно предложить следующие подходы.

Во-первых, для обеспечения эффективной синхронизации захвата и использования ресурсов сканера отпечатков пальцев могут быть применены мьютексы, которые позволят ограничить доступ к общему ресурсу только одному потоку или процессу в определенный момент времени.

Кроме того, для решения проблемы прерванного чтения отпечатков пальцев можно ввести механизм отслеживания времени, чтобы по истечении определенного периода запрос отпечатка пальца был принудительно завершен. Это позволит избежать затянутого доступа к ресурсу в случае, если пользовательское приложение не успевает оперативно обработать данные от сканера.

Таким образом, разработка драйвера для сканера отпечатков пальцев в многопоточной системе требует учета и обработки указанных проблем с использованием вышеуказанных решений.

2.4 Взаимодействие с ФС proc

В контексте разработки драйвера сканера отпечатка пальцев важно иметь возможность передавать считанный отпечаток из пространства ядра в пространство пользователя. Для этого можно использовать интерфейс файлов последовательностей, предоставляемый ядром Linux. При этом данные считанного отпечатка будут представлены как последовательность байт, которую можно передать из пространства ядра в пространство пользователя с помощью файлов в виртуальной файловой системе `proc`. Диаграмма вызовов функций, определенных на `seq_file` представлена на рисунке 2.3.

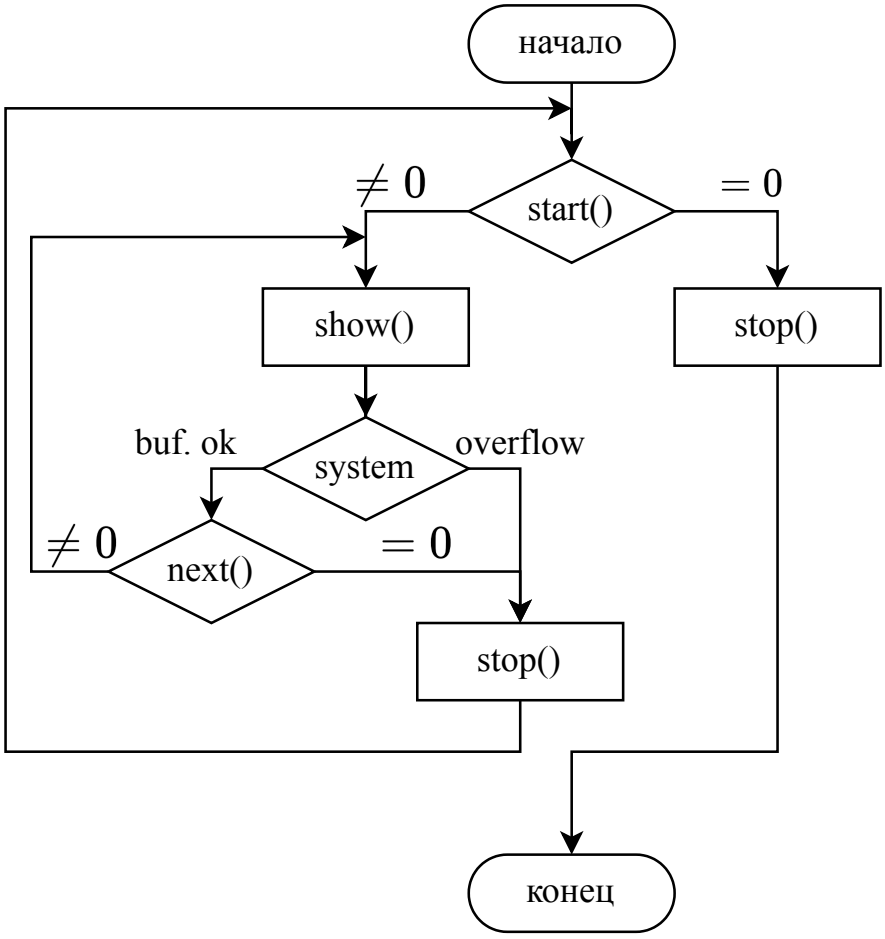

Рисунок 2.3 – Диаграмма вызовов функций, определенных на `seq_file`

Альтернативными способами передачи отпечатка из пространства ядра в пространство пользователя могут быть использование специальных функций ядра, таких как `copy_to_user()` или системного вызова `mmap()`.

В случае разработки драйвера сканера отпечатка пальцев, использование интерфейса файлов последовательностей обосновано тем, что они предоставляют удобный механизм для передачи последовательности данных из ядра в пространство пользователя. Кроме того, они позволяют удобно осуществлять чтение и запись данных с использованием стандартных интерфейсов файловой системы, что упрощает взаимодействие пользовательских программ с драйвером сканера отпечатка пальцев.

3 Технологический раздел

3.1 Выбор языка и среды разработки

Для написания драйвера был выбран язык программирования C.

Для сборки драйвера будет использована утилита Make [9]. В качестве среды разработки был выбран редактор VSCode [10].

Драйвер разрабатывается для ОС Linux версии 6.7.

3.2 Точки входа драйвера

Листинг 3.1 – функция `probe` драйвера

```
1  int fprint_probe(struct usb_interface *intf, const struct usb_device_id
       *id)
2  {
3      printk(KERN_INFO LOG_PREFIX "fingerprint usb interface probed\n");
4
5      struct usb_host_interface *iface = intf->cur_altsetting;
6      printk(KERN_INFO LOG_PREFIX "fingerprint usb interface number: %d\n",
       iface->desc.bInterfaceNumber);
7      if (iface->string != NULL) {
8          printk(KERN_INFO LOG_PREFIX "fingerprint usb interface string:
       %s\n", iface->string);
9      } else {
10         printk(KERN_WARNING LOG_PREFIX "fingerprint usb interface string
       is NULL\n");
11     }
12
13     if (iface->desc.bNumEndpoints != 3) {
14         printk(KERN_INFO LOG_PREFIX "skipping this interface\n");
15         return 0;
16     }
17
18     if (iface->desc.bInterfaceNumber != 0) {
19         printk(KERN_INFO LOG_PREFIX "skipping this interface\n");
20         return 0;
21     }
22
23     struct usb_endpoint_descriptor *endpoint = &iface->endpoint[0].desc;
24     if (endpoint->bEndpointAddress != 0x82) {
25         printk(KERN_WARNING LOG_PREFIX "endpoint[1] must have adress
       0x81\n");
26         return 0;
27     }
28
29     int res = initialize_fprint_endpoint(intf, endpoint);
```

```
30      if (res != 0) {
31          printk(KERN_ERR LOG_PREFIX "failed to initialize endpoint\n");
32      }
33      return res;
34  }
```

Листинг 3.2 – функция `disconnect` драйвера

```
1  void fprint_disconnect(struct usb_interface *intf)
2  {
3      printk(KERN_INFO LOG_PREFIX "fingerprint usb interface
       disconnected\n");
4
5      struct usb_host_interface *iface = intf->cur_altsetting;
6      printk(KERN_INFO LOG_PREFIX "fingerprint usb interface number: %d\n",
       iface->desc.bInterfaceNumber);
7      printk(KERN_INFO LOG_PREFIX "fingerprint usb interface string: %d\n",
       iface->desc.iInterface);
8
9      if (iface->desc.bInterfaceNumber != 0) {
10         printk(KERN_INFO LOG_PREFIX "skipping this interface\n");
11         return;
12     }
13
14     struct usb_endpoint_descriptor *endpoint = &iface->endpoint[0].desc;
15     if (endpoint->bEndpointAddress != 0x82) {
16         printk(KERN_WARNING LOG_PREFIX "endpoint[1] must have adress
       0x81\n");
17         return;
18     }
19
20     struct usb_device* usbdev = interface_to_usbdev(intf);
21     struct fprint_drv_data* drvdata = dev_get_drvdata(&usbdev->dev);
22     if (drvdata != NULL) {
23         kfree(drvdata->transfer_buffer);
24         kfree(drvdata);
25         printk(KERN_INFO LOG_PREFIX "driver data free'd\n");
26     }
27 }
```

3.3 Работа с файловой системой proc

Листинг 3.3 – функции работы с VFS proc

```
1  static atomic_t reader_available = ATOMIC_INIT(1);
2  static DECLARE_WAIT_QUEUE_HEAD(reader_queue);
3  static DECLARE_WAIT_QUEUE_HEAD(poll_wait_queue);
4
5  static const struct seq_operations fprint_seq_ops = {
6      .start = start,
7      .next = next,
8      .stop = stop,
9      .show = show
10 };
11
12 static const struct proc_ops fprint_proc_ops = {
13     .proc_open    = open,
14     .proc_read    = seq_read,
15     .proc_lseek   = seq_lseek,
16     .proc_release = seq_release,
17     .proc_poll    = poll,
18 };
19
20 void* start(struct seq_file *m, loff_t *pos)
21 {
22     seqfile_iterator = 0;
23     return &seqfile_iterator;
24 }
25
26 void stop(struct seq_file *m, void *v)
27 {
28     atomic_set(&reader_available, 1);
29 }
30
31 void* next(struct seq_file *m, void *v, loff_t *pos)
32 {
33     wait_event_interruptible(reader_queue, signature_data_available);
34     signature_data_available = false;
35
36     *pos += signature_data_size;
37     return NULL;
38 }
39
40 int show(struct seq_file *m, void *v)
41 {
42     if (signature_data == NULL) {
43         printk(KERN_WARNING LOG_PREFIX "signature data is NULL\n");
44         return 0;
45     }
```

```
46
47      seq_write(m, signature_data, signature_data_size);
48      return 0;
49  }
50
51  int open(struct inode* inode, struct file* file)
52  {
53      signature_data_available = false;
54      return seq_open(file, &fprint_seq_ops);
55  }
56
57  static unsigned int poll(struct file *file, poll_table *wait)
58  {
59      poll_wait(file, &poll_wait_queue, wait);
60      if (signature_data_available)
61          return POLLIN | POLLRDNORM;
62      return 0;
63  }
```

3.4 Инициализация TLS-соединения

Листинг 3.4 – Инициализация TLS-соединения

```
1  #define PSK_LENGTH 1024
2  static unsigned char global_psk[PSK_LENGTH];
3  static int is_first_attempt = 1;
4
5  struct conn_request {
6      __u8 type;
7  #define CONNECTION_TLS_REQUEST 0x03
8      __u8 data;
9  };
10
11 static int send_connection_request(struct usb_device *dev) {
12     struct urb *urb = usb_alloc_urb(0, GFP_KERNEL);
13     struct conn_request buffer = { .type = CONNECTION_TLS_REQUEST, .data
       = 0 };
14
15     usb_fill_control_urb(urb, dev, usb_sndctrlpipe(dev, 0), &buffer,
       sizeof(struct conn_request), usb_callback_fn, NULL);
16     usb_submit_urb(urb);
17     return usb_wait_fn(urb);
18 }
19
20 static int get_psk(struct usb_device *dev, unsigned char *psk) {
21     struct urb *urb = usb_alloc_urb(0, GFP_KERNEL);
22
```

```
23      usb_fill_bulk_urb(urb, dev, usb_rcvbulkpipe(dev, 1), psk, PSK_LENGTH,
    usb_callback_fn, NULL);
24      usb_submit_urb(urb);
25      return usb_wait_fn(urb);
26  }
27
28  static int initialize_tls_connection(struct usb_device *dev) {
29      int ret;
30      unsigned char psk[PSK_LENGTH];
31
32      ret = send_connection_request(dev);
33      if (ret < 0) {
34          printk(KERN_ERR LOG_PREFIX "Failed to send connection request to
    the fingerprint scanner\n");
35          return ret;
36      }
37
38      ret = get_psk(dev, psk);
39      if (ret < 0) {
40          printk(KERN_ERR LOG_PREFIX "Failed to get PSK from the
    fingerprint scanner\n");
41          return ret;
42      }
43
44      if (is_first_attempt) {
45          memcpy(global_psk, psk, PSK_LENGTH);
46          is_first_attempt = 0;
47      } else {
48          if (memcmp(global_psk, psk, PSK_LENGTH) != 0) {
49              printk(KERN_ERR LOG_PREFIX "Mismatched PSK values,
    terminating TLS connection initialization\n");
50              return -EINVAL;
51          }
52      }
53
54      return 0;
55  }
```

4 Исследовательский раздел

4.1 Постановка исследования

Исследование скорости обработки запроса отпечатка пальца было проведено для значений размера буфера чтения от 16 до 512 байт. Замер времени осуществлялся с использованием функции `clock_gettime` стандартной библиотеки языка C. При этом в качестве идентификатора часов использовался `CLOCK_PROCESS_CPUTIME_ID` для измерения времени, затраченного непосредственно процессом.

4.2 Результаты

На рисунке 4.1 приведен график зависимости времени обработки запроса отпечатка пальца пользовательского приложения от размера буфера чтения.

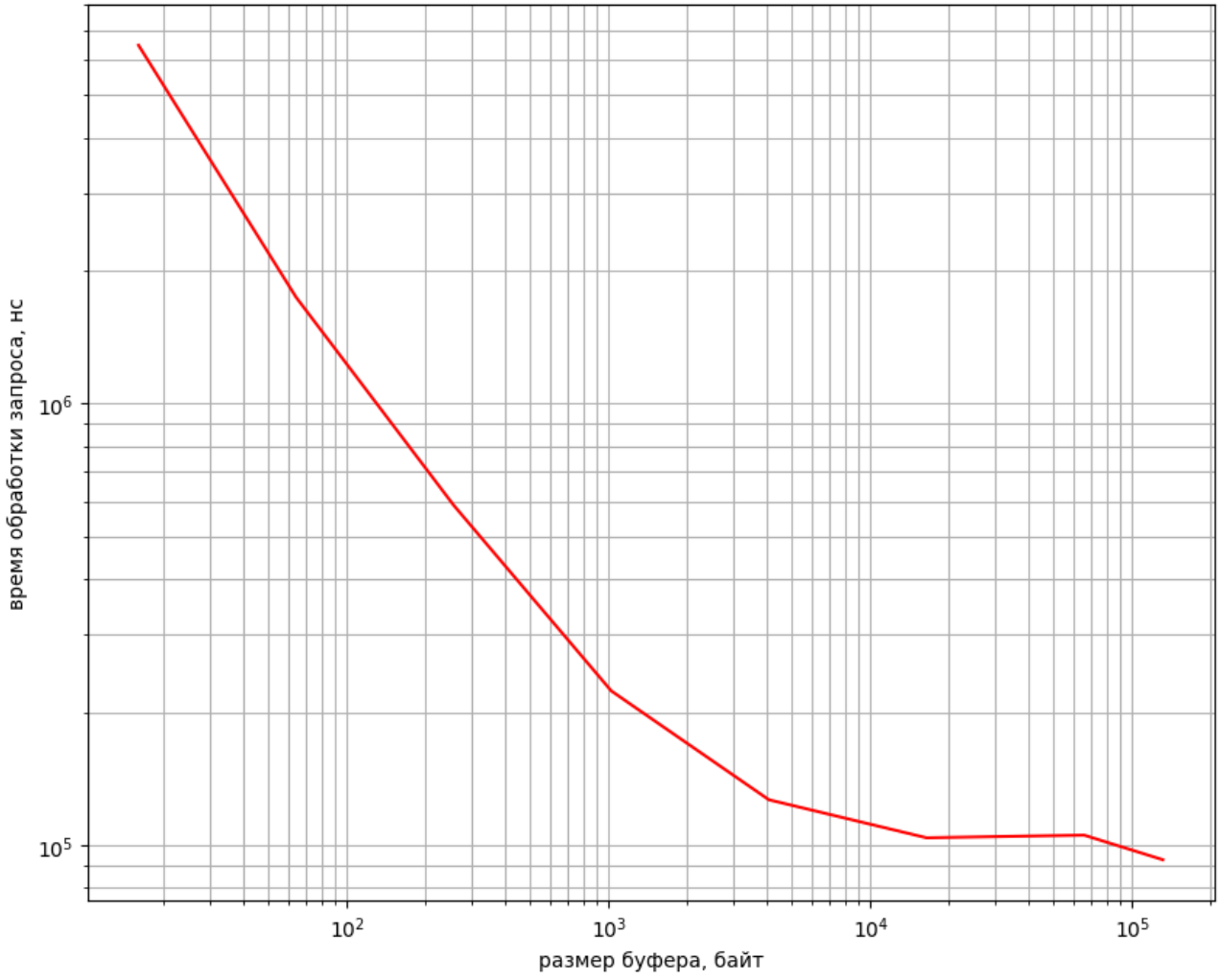

Рисунок 4.1 – Зависимость времени обработки запроса от размера буфера

Выводы

По результатам проведения исследования была установлена зависимость между объемом буфера чтения пользовательского приложения и временем обработки запроса отпечатка пальца.

ЗАКЛЮЧЕНИЕ

В ходе работы была изучена подсистема USB, рассмотрены и решены проблемы связанные с работой сканера отпечатка пальца.

Был разработан драйвер для сканера отпечатка пальцев.

Было проведено исследование зависимости времени обработки запроса отпечатка пальца от пользовательского приложения для различных размеров буферов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. P (VDF-TrFE) thin-film-based transducer for under-display ultrasonic fingerprint sensing applications / C. Peng [и др.] // IEEE Sensors Journal. — 2020. — Т. 20, № 19. — С. 11221—11228.
2. Linux USB Basics. [Электронный ресурс]. URL: https://www.kernel.org/doc/htmldocs/writing_usb_driver/basics.html (Дата обращения: 21.11.2023).
3. *Mochel P.* The sysfs filesystem // Linux Symposium. Т. 1. — The Linux Foundation San Francisco, CA, USA. 2005. — С. 313—326.
4. *Kroah-Hartman G.* Kernel Korner How to Write a Linux USB Device Driver // Linux Journal. — 2001. — Т. 2001, № 90. — С. 4.
5. *Oppliger R.* SSL and TLS: Theory and Practice. — Artech House, 2023.
6. *Bhanot R.*, *Hans R.* A review and comparative analysis of various encryption algorithms // International Journal of Security and Its Applications. — 2015. — Т. 9, № 4. — С. 289—306.
7. The /proc Filesystem – The Linux Kernel documentation. [Электронный ресурс]. URL: https://docs.kernel.org/filesystems/proc.html (Дата обращения: 21.11.2023).
8. On the security of the pre-shared key ciphersuites of TLS / Y. Li [и др.] // Public-Key Cryptography–PKC 2014: 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings 17. — Springer. 2014. — С. 669—684.
9. *Fusco J.* The Linux programmer's toolbox. — Pearson Education, 2007.
10. *Del Sole A.*, *Sole D.* Visual Studio Code Distilled. — Springer, 2019.