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

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Управление мышью при помощи драйвера для игрового манипулятора Xbox One Controller

Студент ИУ7-73Б _____ И.М. Ильясов
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ К.Л. Тассов
(Подпись, дата) (И.О.Фамилия)

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

Москва, 2020 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 ___ИУ7__
(Индекс)
_____ __И.В.Рудаков___
(И.О.Фамилия)
« _____ » _____ 20_ __ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _Операционные системы_____

Студент группы __ИУ7-73Б____

_____Ильясов Идрис Магомет-Салиевич_____
(Фамилия, имя, отчество)

Тема курсового проекта _Управление мышью при помощи драйвера для игрового манипулятора Xbox One Controller.

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_учебный_____
Источник тематики (кафедра, предприятие, НИР) ___кафедра_____

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание __Разработать загружаемый модуль ядра для игрового манипулятора, который в пространстве ядра будет считывать нажатые на нем клавиши и управлять указателем мыши.
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _20-25__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
К защите должны быть подготовлены презентация и доклад, отражающие суть выполненной работы, содержание и методы решения основных задач, а также полученные результаты.

_____

Дата выдачи задания « _ _ » _____ 20__ г.

Руководитель курсового проекта _____ ___К.Л. Тассов_____
(Подпись, дата) (И.О.Фамилия)

Студент _____ ___И.М. Ильясов____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

СОДЕРЖАНИЕ

РЕФЕРАТ

Данная работа посвящена разработке загружаемого модуля ядра, позволяющего управлять указателем мыши на ОС Linux при помощи игрового контроллера Xbox One Controller.

Расчетно-пояснительная записка содержит 32 стр., 5 рис., 7 листингов, 9 источников.

ВВЕДЕНИЕ

С каждым годом компьютеры все сильнее проникают в обычную человеческую жизнь и используются практически во всех сферах деятельности: в производстве, в офисах, в развлекательной сфере, в научной сфере для проведения экспериментов, в коммерческой. Взаимодействовать пользователям с компьютером помогают специальные устройства взаимодействия с человеком (HID – Human Interface Device). К HID-устройствам относят компьютерную клавиатуру, мышь, игровые контроллер, джойстики, манипуляторы, руль, педали, и многие другие. Существует множество коммерчески успешных устройств, цель которых заключается в выполнении задач, предназначенных обычно для других устройств.

Одним из самых популярных игровых контроллеров является контроллер Xbox One Controller от компании Microsoft Corp. Xbox One Controller – это основной игровой контроллер для домашней игровой консоли Xbox от Microsoft. Впервые данная модель была представлена в 2013 году. Игровой контроллер Xbox One Controller в виду своей популярности по умолчанию поддерживается ОС Linux.

Linux – это операционная система с монолитным ядром. Все службы данной операционной системы существуют и выполняются в адресном пространстве ядра. Для избегания полной перекомпиляции ядра при добавлении нового функционала используются загружаемые модули ядра.

Целью данной курсовой работы является разработка загружаемого модуля ядра для ОС Linux, который позволит считывать нажатые на игровом манипуляторе Xbox One Controller клавиши и управлять указателем мыши. Для достижения поставленной цели необходимо решить следующие задачи:

- провести сравнительный анализ существующих решений перехвата ввода в ядре;
- изучить подходы обработки ввода с игрового контроллера в ядре;

- разработать программное обеспечение, позволяющее осуществить перехват ввода с игрового контроллера и последующую передачу команды указателю мыши в соответствии с нажатой на контроллере клавишей.

1 Аналитический раздел

В данном разделе производится постановка задачи, проводится анализ методов решения поставленной задачи.

1.1 Постановка задачи

В соответствии с техническим заданием на курсовой проект необходимо разработать загружаемый модуль ядра, который позволит осуществить управление указателем мыши в соответствии с нажатой на игровом контроллере Xbox One Controller кнопкой. На вход поступает событие, связанное с нажатием клавиши на контроллере. В загружаемом модуле ядра считанная передается мыши. На выходе производится управление указателем мыши. Информация о нажатиях клавиш на игровом выводится в буфер ядра.

1.2 Драйверы устройств

Драйвером устройства называют программу или кусок программного кода, который предназначен для управления этим устройством. В операционной системе драйвера нужны для того, чтобы каждое устройство воспринимало только свой строго фиксированный набор специализированных команд, которыми можно управлять данным устройством. Каждое отдельное устройство, будь то дисковод, клавиатура или принтер, должно иметь свой программный драйвер, выполняющий роль транслятора или связующего звена между аппаратной частью устройства и программными приложениями, использующими это устройство [1].

В ОС Linux драйверы устройств делятся на три типа.

1.2.1 Драйверы первого типа

Драйверы первого типа являются частью программного кода ядра. Соответствующие устройства обнаруживаются системой и становятся доступны для системы. Таким образом, обеспечивается поддержка устройств необходимых для монтирования корневой файловой системы и запуска системы [1].

1.2.2 Драйверы второго типа

Драйверы второго типа в ОС Linux представлены модулями ядра. Они оформлены в виде отдельных файлов и для их подключения необходимо выполнить специальную команду, после чего будет обеспечено управление соответствующим устройством. При отсутствии надобности использования устройства, модуль можно отключить [1].

1.2.3 Драйверы третьего типа

Программный код драйверов третьего типа поделен между ядром и специальной утилитой, предназначенной для управления данным устройством. Примерами драйверов третьего типа являются драйверы модемов и драйверы видеоадаптера.

Во всех трех случаях непосредственное взаимодействие с устройством осуществляет ядро или какой-то модуль ядра. А пользовательские программы взаимодействуют с драйверами устройств через специальные файлы, рас положенные в каталоге /dev и его подкаталогах. На рисунке 1.1 приведена схема взаимодействия прикладных программ с аппаратной частью компьютера в ОС Linux [1].

Рисунок 1.1 – взаимодействие прикладной программы с аппаратной частью компьютера.

1.3 Загружаемые модули ядра

Одним из несомненных достоинств ОС Linux является использование загружаемых модулей ядра. Они позволяют расширить функциональность ядра системы, не перекомпилируя его. Часть исполняемого кода, которая может быть добавлена в ядро во время работы, называется модулем. Ядро Linux предлагает поддержку довольно большого числа типов модулей, включая, но не ограничиваясь, драйверами устройств. Каждый модуль является подготовленным объектным кодом, который может быть динамически подключен в работающее ядро командой insmod и отключен командой rmmod. При помощи команды modinfo производится извлечение информации из модулей ядра (лицензия, автор, описание и др.) [2]. При создании загружаемого модуля ядра стоит учитывать то, что должны обязательно присутствовать макросы module_init() и module_exit(). Макрос module_init() служит для регистрации функции инициализации модуля. Макрос принимает на вход в качестве фактического параметра имя функции. В результате указанная функция будет вызываться при загрузке модуля в ядро. При успешном завершении выполнения функции инициализации, возвращается ноль, а в случае ошибки – ненулевой значение. Макрос module_exit() служит для регистрации функции,

которая вызывается при удалении модуля из ядра. Обычно эта функция выполняет задачу освобождения ресурсов. После завершения функции модуль выгружается. Функция, передаваемая при инициализации, должна соответствовать прототипу int func_init(void), а функция, передаваемая при завершении, – прототипу void func_exit(void) [3].

1.4 Контроллер Xbox One Controller

Используемый в рамках данного курсового проекта игровой контроллер Xbox One Controller приведен на рисунке 1.2.

Рисунок 1.2 – контроллер Xbox One Controller.

У данного контроллера всего 15 элементов взаимодействия:

- цифровая крестовина, называемая D-Pad;
- 2 аналоговых триггера (LT, RT);
- 2 кнопки рядом с аналоговыми триггерами (LB, RB);
- 2 аналоговых стика с возможностью нажатия (left stick click, right stick click);
- 7 цифровых кнопок (Y, A, X, B, Menu, View, Xbox);
- кнопка активации беспроводного подключения.

Подключение контроллера к компьютеру производится с использованием порта USB.

1.5 Обработка ввода с контроллера в ядре

Для обработки ввода с устройства могут использоваться прерывания или подсистема ввода/вывода. Подсистема ввода/вывода (Input/Output Subsystem) производит буферизацию данных и взаимодействует непосредственно с драйверами устройств. Основными функциями данной подсистемы являются:

- обеспечение удобного логического интерфейса между устройствами и остальной частью системы;
- динамическая загрузка и выгрузка драйверов без дополнительных действий с операционной системой;
- поддержка синхронных и асинхронных операций ввода-вывода;
- организация параллельной работы устройств ввода-вывода и процессора;
- поддержка нескольких различных файловых систем и др.

При обработке ввода с контроллера задействуется подсистема ввода, схема которой приведена на рисунке 1.3.

Подсистема ввода – это часть ядра Linux, которая управляет различными устройствами ввода (такими как клавиатуры, мыши, джойстики, планшеты и др.), которые пользователь использует для взаимодействия с ядром, командной строкой и графическим пользовательским интерфейсом. Эта подсистема является частью ядра, поскольку доступ к периферийным устройствам обычно осуществляется через специальные аппаратные интерфейсы (такие как последовательные порты, порты PS/2 и др.), которые защищены и управляются ядром [4].

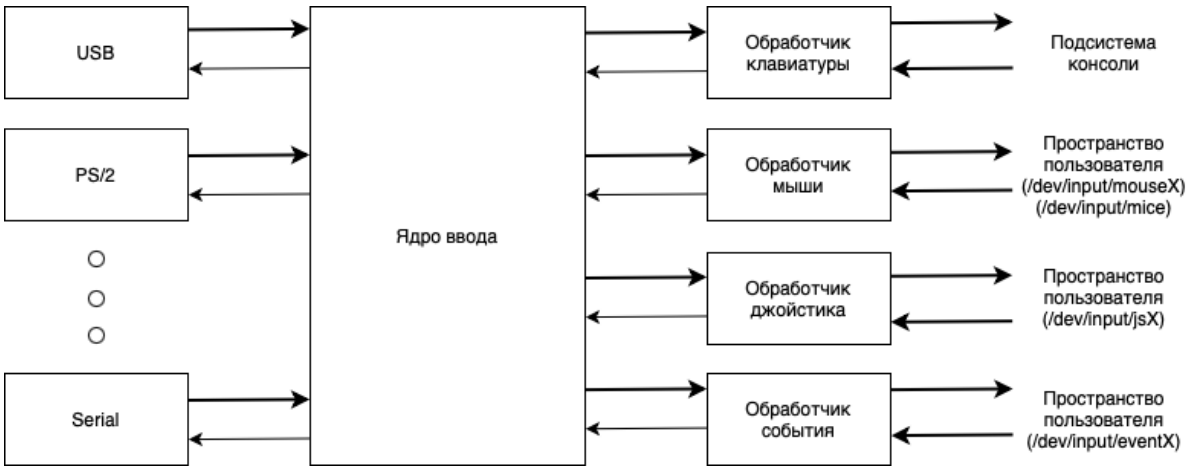

Рисунок 1.3 – подсистема ввода.

Тремя главными элементами подсистемы ввода являются ядро ввода, драйверы и обработчики событий. Стоит отметить, что связь между данными элементами двухсторонняя. При прямом пути от драйверов к обработчикам событий производится ввод, а при обратном, например, установка светодиодов на клавиатуре или вибрация игрового контроллера.

Данная подсистема ввода позволяет нам получить информацию о нажатых кнопках на игровом контроллере и смещенных стиках и передать соответствующую команду мыши [4].

1.6 Вывод

В данном разделе были рассмотрены существующие типы драйверов, изучены принципы работы загружаемых модулей ядра и подсистемы ввода для дальнейшего считывания нажатий на контроллере с последующей передачей команды мыши.

2 Конструкторский раздел

В данном разделе рассматривается процесс проектирования структуры программного обеспечения.

2.1 Состав программного обеспечения

Программное обеспечение состоит из загружаемого модуля ядра, который в пространстве ядра считывает нажатые на контроллере клавиши и управляет указателем мыши.

2.2 Обработка ввода с контроллера и передача команд мыши

Для управления мышью при помощи контроллера Xbox One Controller необходимо создать и зарегистрировать два устройства – сам контроллер и мышь, которой будут передаваться команды с данного контроллера.

Перед регистрацией нового устройства производится выделение памяти под него при помощи вызова функции input_allocate_device(). В случае успешного выделения памяти возвращается указатель на структуру struct input_dev – одну из важных структур данных, которая представляет собой устройство ввода. Структура input_dev приведена в листинге 2.1 [4].

Листинг 2.1 – структура struct input_dev

```
struct input_dev
{
  const char * name;
  ...
  unsigned long evbit[BITS_TO_LONGS(EV_CNT)];
  unsigned long keybit[BITS_TO_LONGS(KEY_CNT)];
  unsigned long relbit[BITS_TO_LONGS(REL_CNT)];
  ...
};
```

В данном листинге приведены основные требуемые в рамках данной задачи поля структуры [5]. Рассмотрим их подробнее:

- поле name – имя устройства;
- поле evbit – это битовая карта типов событий, которые поддерживаются созданным устройством;
- поле keybit – это битовая карта клавиш и кнопок, которые есть у созданного устройства;
- поле relbit – это битовая карта относительных осей созданного устройства.

Для передачи информации от устройства использовалась структура input_event, которая представлена в листинге 2.2.

Листинг 2.2 – структура struct input_event

```
struct input_event
{
  unsigned int __usec;
  __u16 type;
  __u16 code;
  __s32 value;
};
```

Данная структура содержит в себе информацию о времени события, о типе события, о коде события и о его значении.

Поле type показывает общий тип сообщаемого события, например, нажатие клавиши или кнопки, относительное движение (перемещение мыши) или абсолютное движение (перемещение стика на контроллере). В поле code хранится информация, какая из различных кнопок или осей используется, а в поле value – произведено ли соответствующее событие.

Поле type может принимать одно из следующих значений [6]:

- **EV_SYN** - используется как маркер для разделения событий. События могут быть разделены во времени или в пространстве;

- **EV_KEY** – используется для описания изменений состояния клавиатуры, кнопок или других устройств, похожих на клавиши;
- **EV_REL** – используется для описания относительных изменений значений оси, например перемещение мыши;
- **EV_ABS** – используется для описания изменений абсолютного значения оси, например описание координат касания на сенсорном экране;
- **EV_MSC** – используется для описания различных входных данных, которые не подходят для других типов;
- **EV_SW** – используется для описания входных переключателей двоичного состояния;
- **EV_LED -** используется для включения и выключения светодиодов на устройствах;
- **EV_SND** – используется для вывода звука на устройства;
- **EV_REP** – используется для автоповторяющихся устройств;
- **EV_FF** – используется для отправки команд обратной связи по усилию на устройство ввода;
- **EV_PWR** – особый тип для кнопки;
- **EV_FF_STATUS** – используется для получения статуса устройства обратной связи по усилию.

Возможных значений для поля code несколько сотен, поэтому перечислим необходимые в рамках решаемой задачи [6]:

- **BTN_A** (304 или 0x130) – нажатие и отжатие кнопки A на контроллере;
- **BTN_B** (305 или 0x131) – нажатие и отжатие кнопки B на контроллере;
- **ABS_X** (0 или 0x00) – перемещение стика на контроллере по оси OX;
- **ABS_Y** (1 или 0x01) – перемещение стика на контроллере по оси OY;
- **ABS_HAT0X** (16 или 0x10) – нажатие на крестовину (dpad) на контроллере влево и вправо;

- **ABS_HAT0Y** (17 или 0x11) – нажатие на крестовину (dpad) на контроллере вверх и вниз.

Для подключения, отключения и обработки всех событий на устройстве используется структура input_handler, приведенная в листинге 2.3.

Листинг 2.3 – структура struct input_handler [6].

```
struct input_handler
{
  ...
  void (*event)(struct input_handle *handle, unsigned
                              int type, unsigned int code,
                              int value);
  ...
  int (*connect)(struct input_handler *handler, struct
                              input_dev *dev, const struct
                              input_device_id *id);
  void (*disconnect)(struct input_handle *handle);
  ...
  const char *name;
  const struct input_device_id *id_table;
  ...
};
```

Рассмотрим поля данной структуры:

- event – обработчик события – метод вызывается ядром ввода с отключенными прерываниями;
- connect – вызывается при подключении (присоединении) обработчика к устройству ввода;
- disconnect – отсоединяет обработчик от устройства ввода;
- name – имя обработчика (будет отображено в /proc/bus/input/handlers)
- id_table – указатель на таблицу input_device_ids, которую может обрабатывать данный драйвер.

На приведенных ниже рисунках продемонстрирована схема алгоритма переназначения кнопок контроллера для управления мышью.

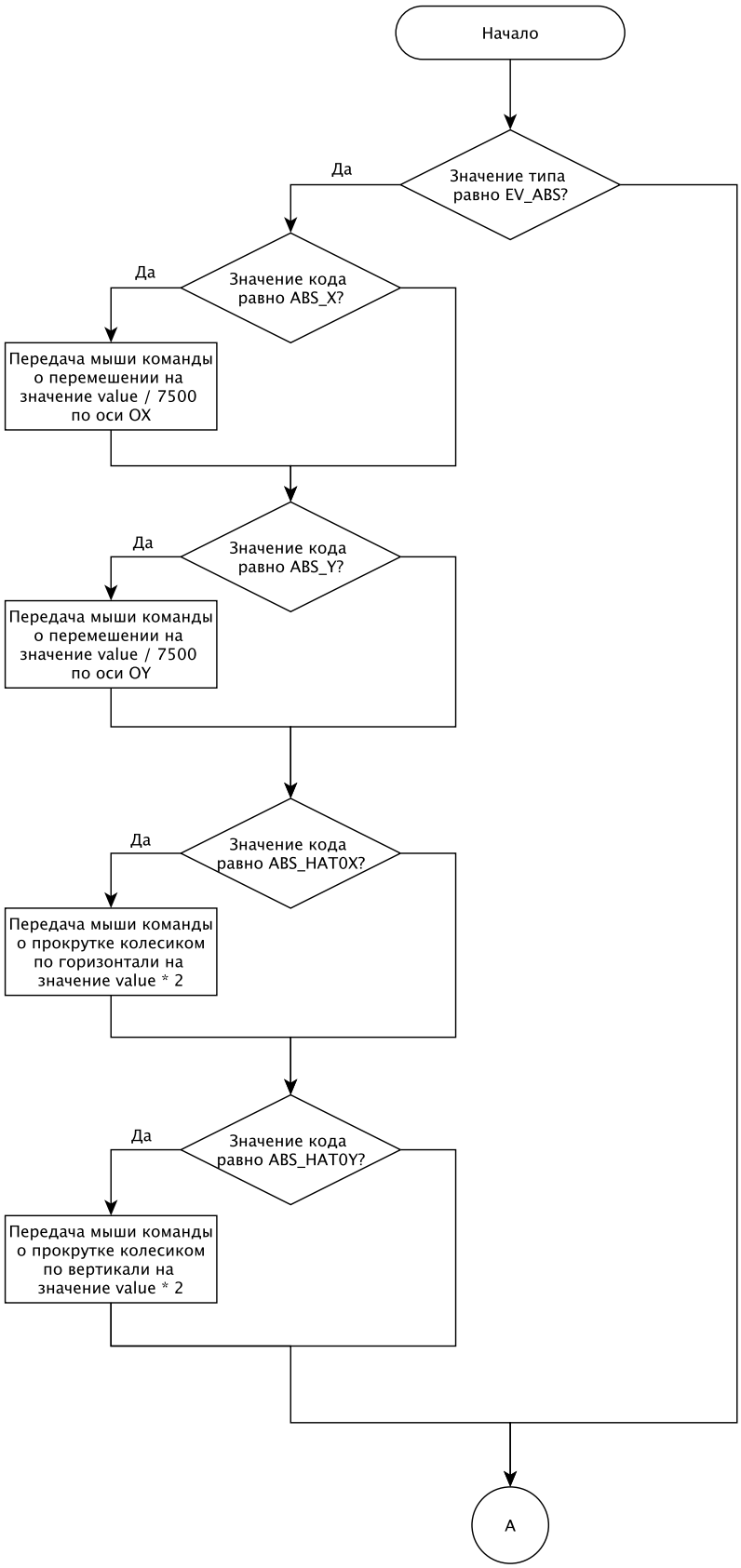

Рисунок 2.1 – схема алгоритма управления мышью при помощи контроллера.

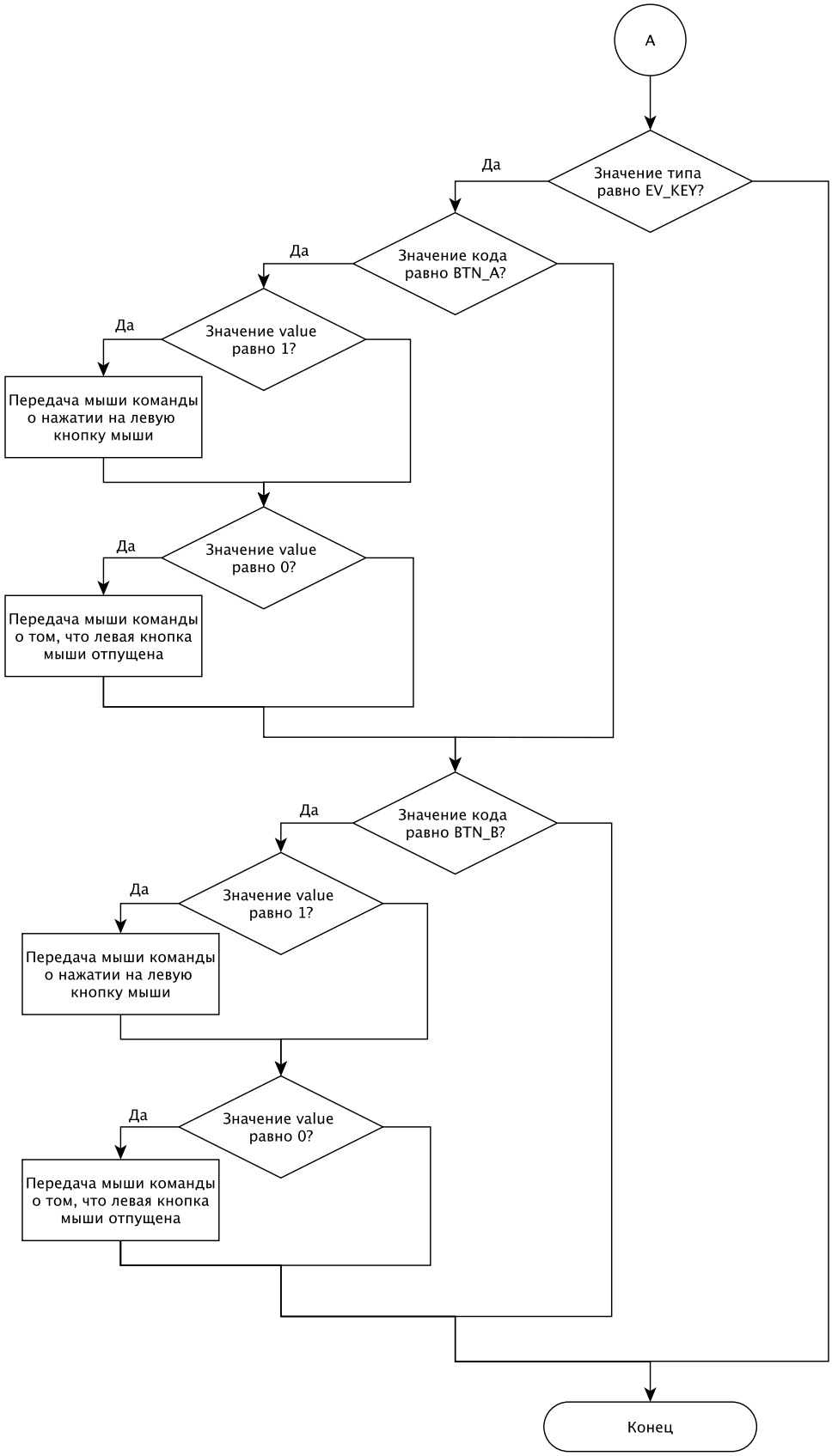

Рисунок 2.2 – схема алгоритма управления мышью при помощи контроллера.

2.3 Вывод

В данном разделе был рассмотрен процесс проектирования структуры программного обеспечения, были выбраны структуры, системные вызовы и значения событий необходимых для управления мышью с контроллера.

3 Технологический раздел

В данном разделе выбирается язык программирования, на котором будет реализована поставленная задача, производится выбор среды разработки и рассматриваются некоторые моменты реализации загружаемого модуля ядра, позволяющего управлять мышью при помощи контроллера Xbox One Controller.

3.1 Выбор языка программирования

В качестве языка программирования для реализации данного курсового проекта был выбран язык C. При помощи этого языка реализованы все модули ядра и драйверы в ОС Linux. Язык C позволяет эффективно использовать возможности современных вычислительных машин. В качестве компилятора использовался компилятор gcc.

3.2 Выбор среды разработки

Выбранной средой разработки является редактор исходного кода Atom. Достоинствами данного решения являются:

- встроенный менеджер пакетов;
- легковесность;
- возможность бесплатного использования;
- встроенная поддержка системы контроля версий Git.

В листинге 3.1 приведено содержимое Makefile – файла, содержащего набор инструкций, используемых утилитой make в инструментарии автоматизации сборки.

Листинг 3.1 – содержимое Makefile.

```
ifneq ($(KERNELRELEASE),)
      obj-m := mouse_gamepad.o
else
      CURRENT = $(shell uname -r)
      KDIR = /lib/modules/$(CURRENT)/build
      PWD = $(shell pwd)

default:
      $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
      rm -rf .tmp_versions
      rm *.ko
      rm *.o
      rm *.mod.c
      rm *.symvers
      rm *.order

endif
```

3.3 Описание некоторых моментов реализации

Для корректной работы загружаемого модуля ядра в контексте считывания нажатий на кнопки контроллера в правильном порядке была использована очередь работ. Произошедшие события записывались в очередь и затем поочередно из нее извлекались.

В приведенном ниже листинге 3.2 производится объявление устройств – мыши и игрового контроллера.

Листинг 3.2 – объявление устройств.

```
struct input_dev *mouse;
gamepad_t *gamepad;
```

В листинге 3.3 приведена функция добавления события в очередь работ после нажатия кнопки на контроллере.

Листинг 3.3 – добавление события в очередь работ.

```
static void gamepad_event(struct input_handle *handle,
                                    unsigned int type,
                                    unsigned int code,
                                            int value)
{
    struct event_work *work = kzalloc(sizeof(struct
event_work), GFP_ATOMIC);
    work->event.code = code;
    work->event.type = type;
    work->event.value = value;

    INIT_WORK(&work->work, gamepad_mouse_events);
    queue_work(queue, &work->work);
}
```

В листинге 3.4 продемонстрирована структура gamepad. Данная структура содержит в себе указатель на устройство ввода, обработчик ввода и связующее звено между устройством и обработчиком ввода.

Листинг 3.4 – структура gamepad.

```
struct gamepad
{
    struct input_dev *dev;
    struct input_handle *handle;
    struct input_handler *handler;
};
```

Весь исходный код загружаемого модуля ядра приведен в приложении А.

3.4 Пример работы загружаемого модуля ядра

Ниже на рисунке 3.1 приведен вывод буфера сообщений ядра при помощи команды dmesg.

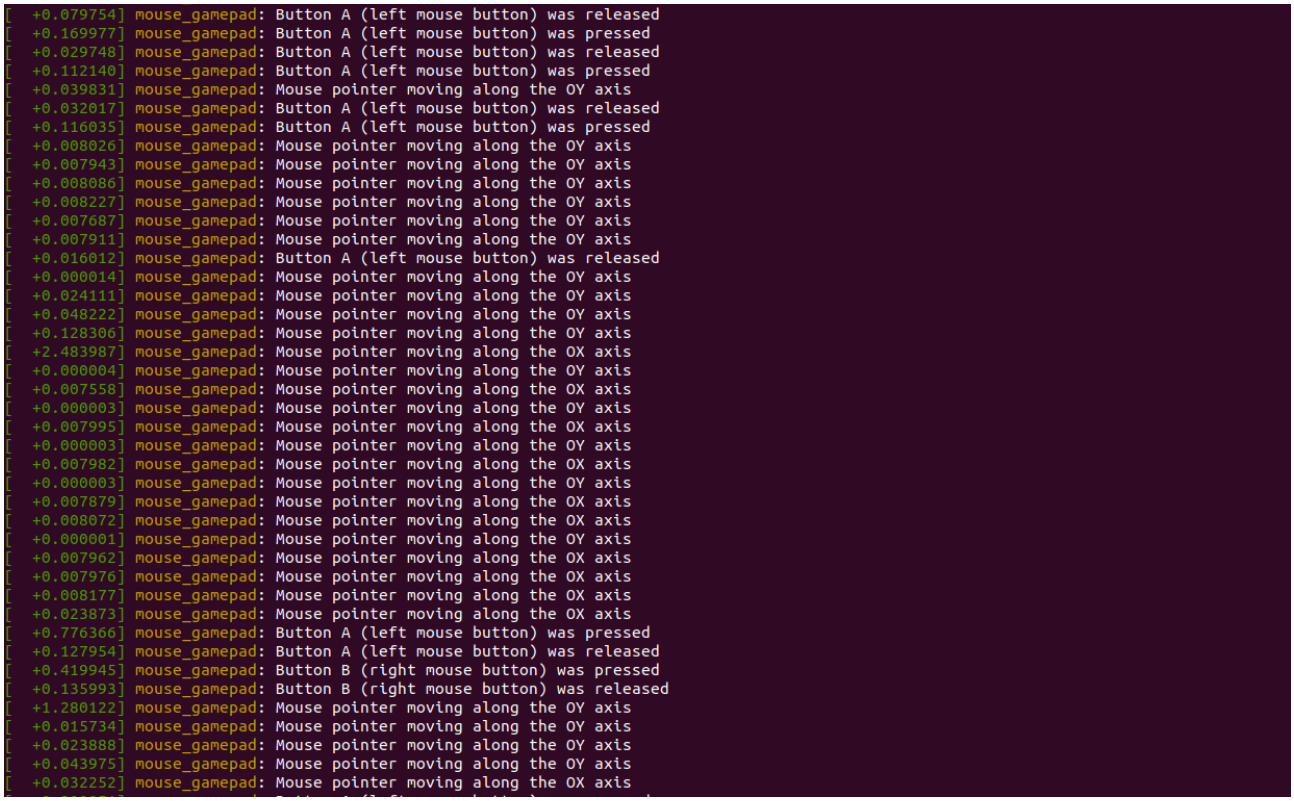

Рисунок 3.1 – вывод буфера сообщений ядра.

3.5 Вывод

В данном разделе был выбран язык C в качестве языка программирования, на котором была реализована поставленная задача, был выбран редактор исходного кода Visual Studio Code в качестве среды разработки, были рассмотрены некоторые моменты реализации загружаемого модуля ядра.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы были достигнуты следующие задачи:

- был проведен сравнительный анализ существующих решений перехвата ввода в ядре;
- были изучены подходы обработки ввода с игрового контроллера в ядре;
- было разработано программное обеспечение, позволяющее осуществить перехват ввода с игрового контроллера и последующую передачу команды указателю мыши в соответствии с нажатой на контроллере клавишей.

Список использованных источников

1. Драйверы устройств. «Linux для пользователя». [Электронный ресурс] – Режим доступа: http://www.uhlib.ru/kompyutery_i_internet/linux_dlja_polzovatelja/p10.php (дата обращения: 06.12.2020).
2. Цирюлик О.И. Модули ядра Linux. Внутренние механизмы ядра. [Электронный ресурс] – Режим доступа: http://rus-linux.net/ MyLDP/BOOKS/Moduli-yadra-Linux/kern-mod-index.html (дата обращения: 03.12.2020).
3. Рязанова Н.Ю., Лекции по Операционным Системам, 2019-2020.
4. The Linux USB Input Subsystem, Part I – Linux Journal. [Электронный ресурс] – Режим доступа: https://www.linuxjournal.com/article/6396 (дата обращения: 14.12.2020).
5. Input Subsystem – The Linux Kernel Documentation [Электронный ресурс] – Режим доступа: https://www.kernel.org/doc/html/v4.17/driver-api/input.html (дата обращения: 02.12.2020).
6. Исходный код ядра Linux. [Электронный ресурс] – Режим доступа: https://elixir.bootlin.com/linux/latest/source (дата обращения: 4.12.2020).

ПРИЛОЖЕНИЕ А

```
#include <linux/module.h>
#include <linux/input.h>
#include <linux/time.h>
#include <linux/ftrace.h>
#include <linux/string.h>
#include <linux/slab.h>
#include <linux/workqueue.h>
#include <linux/string.h>


MODULE_LICENSE("GPL");
MODULE_AUTHOR("Ilyasov Idris");
MODULE_DESCRIPTION("Control mouse pointer by gamepad");


struct gamepad
{
    struct input_dev *dev;
    struct input_handle *handle;
    struct input_handler *handler;
};

typedef struct gamepad gamepad_t;


struct event_work
{
    struct input_event event;
    struct work_struct work;
};


struct workqueue_struct *queue;
struct input_dev *mouse;
gamepad_t *gamepad;


void gamepad_mouse_events(struct work_struct *work)
{
    struct event_work *event = container_of(work, struct
                                    event_work, work);

    if (event->event.type == EV_ABS)
    {
    if (event->event.code == ABS_X)
     {            input_report_rel(mouse, REL_X, (event-
>event.value) / 7500);
            printk(KERN_INFO "mouse_gamepad: Mouse pointer moving
along the OX axis");
     }
```

```
     if (event->event.code == ABS_Y)
     {
             input_report_rel(mouse, REL_Y, (event->event.value) /
7500);
             printk(KERN_INFO "mouse_gamepad: Mouse pointer moving
along the OY axis");
     }

     if (event->event.code == ABS_HAT0X)
     {
             input_report_rel(mouse, REL_HWHEEL, (event-
>event.value) * 2);
             printk(KERN_INFO "mouse_gamepad: Horizontal scrolling
by dpad");
     }

     if (event->event.code == ABS_HAT0Y)
     {
             input_report_rel(mouse, REL_WHEEL, -(event-
>event.value));
             printk(KERN_INFO "mouse_gamepad: Vertical scrolling
by dpad");
     }

      input_sync(mouse);
    }

    else if (event->event.type == EV_KEY)
    {
        if (event->event.code == BTN_A)
        {
            if (event->event.value == 1)
            {
                input_report_key(mouse, BTN_LEFT, 1);
                printk(KERN_INFO "mouse_gamepad: Button A (left
mouse button) was pressed");
            }
            else if (event->event.value == 0)
            {
                input_report_key(mouse, BTN_LEFT, 0);
                printk(KERN_INFO "mouse_gamepad: Button A (left
mouse button) was released");
            }
        }

        if (event->event.code == BTN_B)
        {
            if (event->event.value == 1)
            {
                input_report_key(mouse, BTN_RIGHT, 1);
                printk(KERN_INFO "mouse_gamepad: Button B (right
mouse button) was pressed");
```

```
          }
            else if (event->event.value == 0)
            {
                 input_report_key(mouse, BTN_RIGHT, 0);
                 printk(KERN_INFO "mouse_gamepad: Button B (right
mouse button) was released");
            }
        }

        input_sync(mouse);
    }

    kfree(event);
}


static void gamepad_event(struct input_handle *handle, unsigned
int type, unsigned int code, int value)
{
    struct event_work *work = kzalloc(sizeof(struct event_work),
GFP_ATOMIC);
    work->event.code = code;
    work->event.type = type;
    work->event.value = value;

    INIT_WORK(&work->work, gamepad_mouse_events);
    queue_work(queue, &work->work);
}


static int gamepad_connect(struct input_handler *handler, struct
input_dev *dev, const struct input_device_id *id)
{
    struct input_handle *handle;
    int result;

    if (strcmp(dev->name, "Microsoft X-Box One S pad"))
    {
        return 0;
    }

    handle = kzalloc(sizeof(struct input_handle), GFP_KERNEL);

    if (!handle)
    {
        return -ENOMEM;
    }

    gamepad->handle = handle;
    handle->dev = dev;
    handle->handler = handler;

    result = input_register_handle(handle);
```

```
    if (result)
    {
        kfree(handle);
        return result;
    }

    result = input_open_device(handle);
    if (result)
    {
        input_unregister_handle(handle);
        kfree(handle);
        return result;
    }

    printk(KERN_INFO "%s", dev->name);
    return 0;
}


static void gamepad_disconnect(struct input_handle *handle)
{
    input_close_device(handle);
    input_unregister_handle(handle);
    kfree(handle);
}


static const struct input_device_id gamepad_ids[] =
{
    {
    .flags = INPUT_DEVICE_ID_MATCH_EVBIT,
    .evbit = { BIT_MASK(EV_KEY) },
    },

    {
    .flags = INPUT_DEVICE_ID_MATCH_EVBIT,
    .evbit = { BIT_MASK(EV_REL) },
    },

    {
    .flags = INPUT_DEVICE_ID_MATCH_EVBIT,
    .evbit = { BIT_MASK(EV_ABS) },
    },

    {
    .flags = INPUT_DEVICE_ID_MATCH_EVBIT,
    .evbit = { BIT_MASK(EV_MSC) },
    },
};


static struct input_handler gamepad_handler =
{
```

```
    .event = gamepad_event,
    .connect = gamepad_connect,
    .disconnect = gamepad_disconnect,
    .name = "gamepad",
    .id_table = gamepad_ids,
};



int __init gamepad_init(void)
{
    int result;
    queue = create_workqueue("mouse_gamepad");
    if (!queue)
    {
        printk(KERN_ERR "mouse_gamepad: workqueue wasn't
allocated\n");
        return -1;
    }

    mouse = input_allocate_device();
    mouse->name = "virtual_mouse";


    gamepad = kzalloc(sizeof(struct gamepad), GFP_KERNEL);
    gamepad->handler = &gamepad_handler;


    result = input_register_handler(&gamepad_handler);
    if (result)
    {
        return result;
    }

    gamepad->dev = input_allocate_device();

    if (!gamepad->dev)
    {
        printk(KERN_ALERT "mouse_gamepad: Bad
input_allocate_device()\n");
        return -1;
    }

    set_bit(EV_REL, mouse->evbit);
    set_bit(REL_X, mouse->relbit);
    set_bit(REL_Y, mouse->relbit);
    set_bit(REL_WHEEL, mouse->relbit);
    set_bit(REL_HWHEEL, mouse->relbit);

    set_bit(EV_KEY, mouse->evbit);
    set_bit(BTN_LEFT, mouse->keybit);
    set_bit(BTN_RIGHT, mouse->keybit);
```

```
    input_register_device(gamepad->dev);
    input_register_device(mouse);

    printk("mouse_gamepad: Module was loaded\n");
    return 0;
}


void __exit gamepad_exit(void)
{
    if (gamepad->dev)
    {
        input_unregister_device(gamepad->dev);
    }

    if (mouse)
    {
        input_unregister_device(mouse);
    }

    input_unregister_handler(gamepad->handler);
    flush_workqueue(queue);
    destroy_workqueue(queue);
    kfree(gamepad);
    printk("mouse_gamepad: Module was unloaded\n");
}


module_init(gamepad_init);
module_exit(gamepad_exit);
```