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»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Загружаемый модуль ядра, предоставляющий информацию о сокетах

Студент _____ИУ7-75Б_____ _____ А.А. Лаврова
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ Н.Ю. Рязанова
(Подпись, дата) (И.О.Фамилия)

2020 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Ядро операционной системы Linux — это сложный, портативный, модульный программный комплекс, широко используемый в серверных и встраиваемых системах. Более половины всех ЭВМ мира использует Linux в качестве базовой системы. Конфигурации таких машин сильно варьируются, существует великое множество устройств обработки, ввода, вывода, хранения и передачи информации, отличающихся в физической и программной реализации.

В конкурсе на лучшую компьютерную идею всех времен и народов сокеты, без сомнения, могли бы рассчитывать на призовое место. Как и другие средства межпроцессного взаимодействия, сокеты впервые были реализованы именно на платформе Unix, однако концепция сокетов, как универсального средства обмена данными между процессами, оказалась настолько удачна, что все современные системы поддерживают, по крайней мере, некоторое подмножество сокетов. Причины успеха сокетов заключаются в их простоте и универсальности. Программы, обменивающиеся данными с помощью сокетов, могут работать в одной системе и в разных, используя для обмена данными как специальные объекты системы, так и сетевой стек.

Целью данной курсовой работы является разработка загружаемого модуля ядра для ОС Linux, который позволит выводить информацию о сокетах в виртуальную файловую систему /proc. Для достижения поставленной цели необходимо решить следующие задачи:

- изучить устройство модуля ядра;
- рассмотреть существующие типы сокетов;
- разработать программное обеспечение, позволяющее осуществить считывание и вывод информации о сокетах.

1. Аналитический раздел

В данном разделе производится постановка задачи и анализ методов решения поставленной задачи.

1.1 Формализация задачи

В соответствии с техническим заданием на курсовой проект необходимо разработать загружаемый модуль ядра, который позволит просмотреть следующую информацию о сокетах TCP и UDP:

- размер очередей приёма;
- размер очередей получения;
- локальный IP-адрес и номер используемого порта;
- удаленный IP-адрес и номер используемого порта;
- состояние TCP-соединения;
- параметры сокетов.

1.2 Загружаемый модуль ядра

Ядро Linux относится к категории так называемых монолитных – это означает, что большая часть функциональности операционной системы называется ядром и запускается в привилегированном режиме. Этот подход отличен от подхода микроядра, когда в режиме ядра выполняется только основная функциональность (взаимодействие между процессами [inter-process communication, IPC], диспетчеризация, базовый ввод-вывод [I/O], управление памятью), а остальная функциональность вытесняется за пределы привилегированной зоны (драйверы, сетевой стек, файловые системы). Можно было бы подумать, что ядро Linux очень статично, но на самом деле все как раз наоборот. Ядро Linux динамически изменяемое – это означает, что вы можете загружать в ядро дополнительную функциональность, выгружать функции из ядра и даже добавлять новые модули, использующие другие модули ядра. Преимущество загружаемых модулей заключается в

возможности сократить расход памяти для ядра, загружая только необходимые модули (это может оказаться важным для встроенных систем).

Linux – это не единственное (и не первое) динамически изменяемое монолитное ядро. Загружаемые модули поддерживаются в BSD-системах, Sun Solaris, в ядрах более старых операционных систем, таких как OpenVMS, а также в других популярных ОС, таких как Microsoft Windows и Apple Mac OS X.

1.2.1 Устройство модуля ядра

Загружаемые модули ядра имеют ряд фундаментальных отличий от элементов, интегрированных непосредственно в ядро, а также от обычных программ. Обычная программа содержит главную процедуру (main) в отличие от загружаемого модуля, содержащего функции входа и выхода. Функция входа вызывается, когда модуль загружается в ядро, а функция выхода – соответственно при выгрузке из ядра. Поскольку функции входа и выхода являются пользовательскими, для указания назначения этих функций используются макросы module_init и module_exit. Загружаемый модуль содержит также набор обязательных и дополнительных макросов [1]. Они определяют тип лицензии, автора и описание модуля, а также другие параметры. Пример очень простого загружаемого модуля приведен на рисунке 1.1.

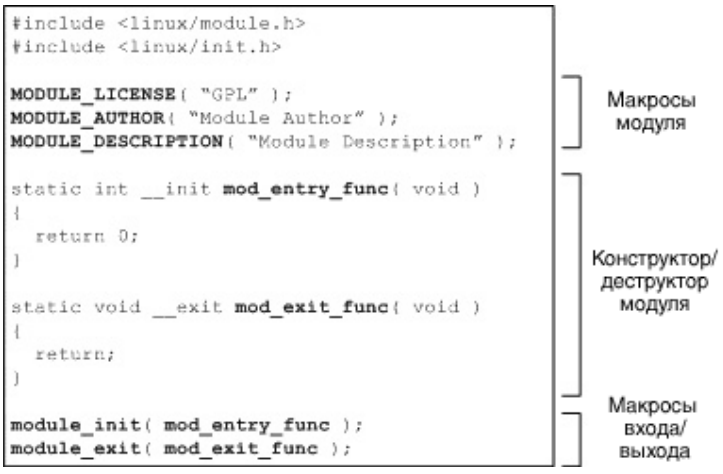

Рисунок 1.1 - Пример загружаемого модуля с разделами ELF

1.3 Сокеты

Сокет – это абстракция конечной точки взаимодействия. Абстракция сокетов была введена в 4.2 BSD (Berkley Software Distribution) UNIX и были созданы для организации взаимодействия процессов, причем безразлично, где эти процессы выполняются: на одной машине или на нескольких машинах. Другими словами, сокеты являются универсальным средством межпроцессного взаимодействия в том смысле, что они могут использоваться как для взаимодействия процессов на отдельно стоящей машине, так и для взаимодействия процессов в сети (рисунок 1.2).

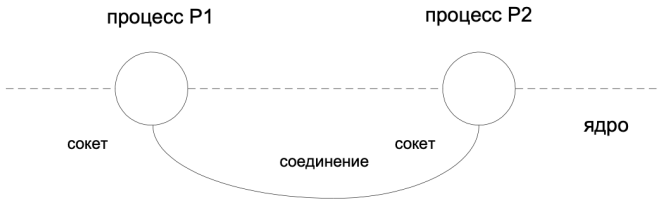

Рисунок 1.2 - Взаимодействия процессов в сети

Сокеты находятся в областях связи (доменах). Домен сокета — это абстракция, которая определяет структуру адресации и набор протоколов. Сокеты могут соединяться только с сокетами в том же домене. Всего выделено 23 класса сокетов (см. файл <sys/socket.h>), из которых обычно используются только UNIX-сокеты и Интернет-сокеты. Сокеты могут использоваться для установки связи между процессами на отдельной системе подобно другим формам IPC [2].

Класс сокетов UNIX обеспечивает их адресное пространство для отдельной вычислительной системы. Сокеты области UNIX называются именами файлов UNIX. Сокеты также можно использовать, чтобы организовать связь между процессами на различных системах. Адресное пространство сокетов между связанными системами называют доменом Интернета. Коммуникации домена Интернета используют стек протоколов TCP/IP.

Типы сокетов определяют особенности связи, доступные приложению. Процессы взаимодействуют только через сокеты одного и того же типа. Основные типы сокетов:

- поточный – обеспечивает двухсторонний, последовательный, надежный, и недублированный поток данных без определенных границ. Тип сокета – SOCK_STREAM, в домене Интернета он использует протокол TCP;
- дейтаграммный – поддерживает двухсторонний поток сообщений. Приложение, использующее такие сокеты, может получать сообщения в порядке, отличном от последовательности, в которой эти сообщения посылались. Тип сокета – SOCK_DGRAM, в домене Интернета он использует протокол UDP;
- сокет последовательных пакетов – обеспечивает двухсторонний, последовательный, надежный обмен дейтаграммами фиксированной максимальной длины. Тип сокета – SOCK_SEQPACKET. Для этого типа сокета не существует специального протокола;
- простой сокет – обеспечивает доступ к основным протоколам связи.

Протоколы для взаимодействия с использованием сокетов выбираются на основе трех параметров:

- семейство или домен (family);
- тип сокета (type);

Домен определяет семейство протоколов, которое будет использоваться для связи. Эти семейства определены в <sys/socket.h>. Наиболее часто используются: AF_UNIX, AF_INET для сетевого протокола IPv4, PF_INET6 для IPv6, PF_UNIX для локальных сокетов (используя файл).

Параметр тип определяет семантику соединения:

- SOCK_STREAM – обеспечивает последовательное, надежное, двустороннее соединение (надёжная потокоориентированная служба (TCP) (сервис) или потоковый сокет);

- SOCK_DGRAM – поддерживает дейтаграммы (UDP) или дейтаграммные сокеты (без установления соединения, ненадежная передача сообщений; сообщения фиксированной максимальной длины);
- SOCK_SEQPACKET – обеспечивает последовательное, надежное двустороннее соединение для дейтаграмм фиксированной длины (максимальной); потребитель должен прочитать весь пакет с каждым входным системным вызовом;
- SOCK_RAW — «сырой» (нижнего уровня) протокол поверх сетевого уровня.

Параметр протокол определяет конкретный протокол, который будет использоваться с сокетом. Обычно существует только один протокол для поддержки определенного типа сокета в данном семействе протоколов, и в этом случае протокол может быть указан как 0. Однако возможно, что существует много протоколов, и в этом случае конкретный протокол должен быть указан непосредственно. Используемый номер протокола зависит от «домена связи», в котором должна осуществляться связь.

1.3.1 Функции сокетов

В пространстве пользователя сокеты представляются как дескрипторы файлов. Эти файловые дескрипторы используются для выполнения операций чтения и записи. Однако, создание коммуникационных отношений существенно отличается от открытия файла. Поэтому на сокетах определены специальные системные вызовы. Взаимодействие процессов через сокеты выполняется по модели клиент-сервер. Роли сервера и клиентов различаются с точки зрения поддержки коммуникационных отношений: клиент активно устанавливает соединения с сервером, сервер сначала пассивно ожидает поступления входящих запросов на установку соединения, а при поступлении запроса сервер его фиксирует и начинает обрабатывать; обработав запрос, сервер посылает ответ клиенту. Таким образом, на стороне клиента и на

стороне сервера в интерфейсе сокетов выполняется разная последовательность системных вызовов (рисунок 1.3).

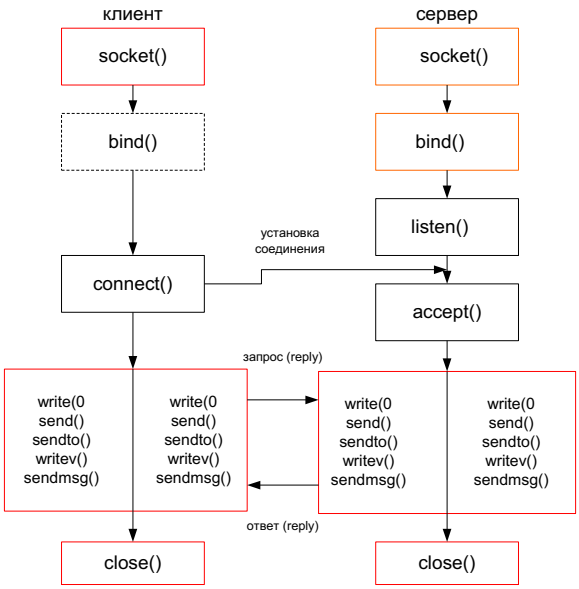

Рисунок 1.3 - Последовательность системных вызовов

1.3.2 Протокол TCP

Протокол управления передачей. TCP (Transmission Control Protocol) является протоколом, ориентированным на установление соединения и предоставляющим надежный двусторонний байтовый поток использующим его приложениям. Сокеты TCP — типичный пример потоковых сокетов (stream sockets). TCP обеспечивает отправку и прием подтверждений, обработку тайм-аутов, повторную передачу и тому подобные возможности. Большинство прикладных программ в Интернете используют TCP. Заметим, что TCP может использовать как IPv4, так и IPv6.

TCP также обеспечивает надежность (reliability). Когда TCP отправляет данные на другой конец соединения, он требует, чтобы ему было выслано подтверждение получения. Если подтверждение не приходит, TCP автоматически передает данные повторно и ждет в течение большего количества времени. После некоторого числа повторных передач TCP оставляет эти попытки. В среднем суммарное время попыток отправки данных занимает от 4 до 10 минут (в зависимости от реализации). Однако TCP не

гарантирует получение данных адресатом, поскольку это в принципе невозможно. Если доставка оказывается невозможной, TCP уведомляет об этом пользователя, прекращая повторную передачу и разрывая соединение. Следовательно, TCP нельзя считать протоколом, надежным на 100%: он обеспечивает надежную доставку данных или надежное уведомление о неудаче.

TCP обеспечивает управление потоком (flow control). TCP всегда сообщает своему собеседнику, сколько именно байтов он хочет получить от него. Это называется объявлением окна (window). В любой момент времени окно соответствует свободному пространству в буфере получателя. Управление потоком гарантирует, что отправитель не переполнит этот буфер. Окно изменяется динамически с течением времени: по мере того как приходят данные от отправителя, размер окна уменьшается, но по мере считывания принимающим приложением данных из буфера окно увеличивается [3].

Процесс установления связи в TCP-сеансе представлен на рисунке 1.4 [4].

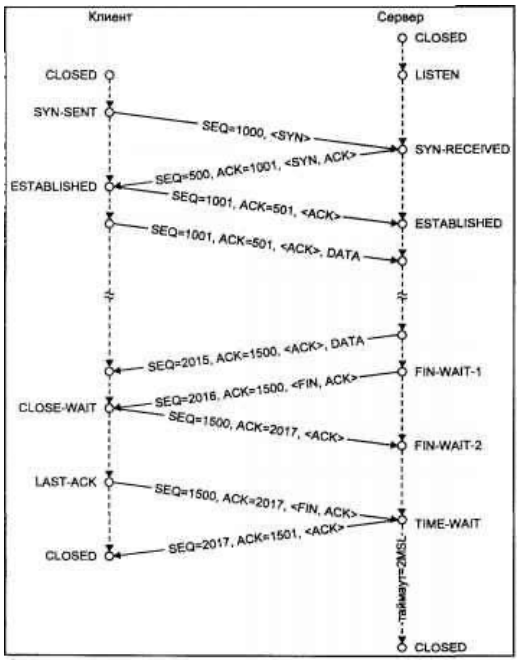

Рисунок 1.4 - процесс установления связи

Расшифровка состояний приведена в таблице 1.1 [4].

Таблица 1.1

Состояние	Описание
LISTEN	Готовность узла к получению запроса на соединение от любого удаленного узла.
SYN-SENT	Ожидание ответного запроса на соединение.
SYN-RECEIVED	Ожидание подтверждения получения ответного запроса на соединение.
ESTABLISHED	Состояние канала, при котором возможен дуплексный обмен данными между клиентом и сервером.
CLOSE-WAIT	Ожидание запроса на окончание связи от локального процесса, использующего данный коммуникационный узел.
LAST-ACK	Ожидание подтверждения запроса на окончание связи, отправленного удаленному узлу. Предварительно от удаленного узла уже был получен запрос на окончание связи и канал стал симплексным.
FIN-WAIT-1	Ожидание подтверждения запроса на окончание связи, отправленного удаленному узлу (инициирующий запрос, канал переходит в симплексный режим).
FIN-WAIT-2	Ожидание запроса на окончание связи от удаленного узла.
CLOSING	Ожидание подтверждения от удаленного узла на запрос окончания связи.
TIME-WAIT	Таймаут перед окончательным разрушением канала, достаточный для того, чтобы удаленный узел получил подтверждение своего запроса окончания связи. Величина тайм-аута составляет 2 MSL (Maximum Segment Lifetime).
CLOSED	Фиктивное состояние, при котором коммуникационный узел и канал фактически не существуют.

1.3.3 Протокол UDP

Протокол пользовательских дейтаграмм. UDP (User Datagram Protocol) — это протокол, не ориентированный на установление соединения. Сокеты UDP служат примером дейтаграммных сокетов (datagram sockets). В отличие от TCP, который является надежным протоколом, в данном случае отнюдь не гарантируется, что дейтаграммы UDP когда-нибудь достигнут заданного места назначения. Как и в случае TCP, протокол UDP может использовать как IPv4, так и IPv6.

Протокол UDP не обеспечивает надежности. UDP сам по себе не имеет ничего похожего на описанные подтверждения передачи, порядковые номера, определение RTT, тайм-ауты или повторные передачи. Если дейтаграмма UDP дублируется в сети, на принимающий узел могут быть доставлены два экземпляра. Также, если клиент UDP отправляет две дейтаграммы в одно и то же место назначения, их порядок может быть изменен сетью, и они будут доставлены с нарушением исходного порядка [3].

1.4 Вывод

В данном разделе были изучены принципы работы загружаемых моделей ядра, рассмотрены существующие типы сокетов.

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 данные о сокетах и выводит в отдельный файл, создаваемый в виртуальной файловой системе /proc.

2.2 Вывод информации о сокетах

Для вывода информации о сокетах создаются виртуальные файлы в виртуальной файловой системе /proc.

Начиная с версии 2.6 ядро содержит набор функций, который призван упростить разработчикам виртуальных файлов правильную работу. Интерфейс seq_file доступен через <linux/seq_file.h>. У seq_file есть три аспекта: интерфейс итератора, который позволяет реализации виртуального файла выполнять пошаговые инструкции по отображаемым объектам, некоторые служебные функции для форматирования объектов для вывода, а также набор стандартных file_operations, которые реализуют большинство операций с виртуальным файлом [5].

Листинг 2.1 – структура seq_file

```
struct seq_file {
	...
	const struct file *file;
	void *private;
};
```

- file – указатель на виртуальный файл в VFS /proc;
- private – используется функциями структуры seq_operations для доступа одних методов к другим.

2.3 Доступ к информации о сокетах

Для того, чтобы корректно обработать информацию о сокетах, использующих протокол TCP, необходимо знать, в каком состоянии сеанса они находятся. Эту информацию можно получить из структуры tcp_iter_state [5].

Листинг 2.3 – структура tcp_iter_state

```
struct tcp_iter_state {
     ...
     enum tcp_seq_states    state;
     ...
};
```

- state – состояние TCP-сеанса (TCP_SEQ_STATE_LISTENING, TCP_SEQ_STATE_ESTABLISHED).

Для того, чтобы получить информацию о домене, необходимо воспользоваться структурой tcp_seq_afinfo или udp_seq_afinfo (в зависимости от типа протокола) [5].

Листинг 2.4 – структура tcp_seq_afinfo

```
struct tcp_seq_afinfo {
     sa_family_t            family;
};
```

- family – AF_INET or AF_INET6.

Листинг 2.5 – структура udp_seq_afinfo

```
struct udp_seq_afinfo {
     sa_family_t            family;
};
```

- family – AF_INET or AF_INET6.

За представление сокетов на сетевом уровне отвечает структура sock. Структура sock — это имплементация в ядре для AF_INET сокетов, которая может использоваться как ядром, так и пространством пользователей [5].

Листинг 2.6 – структура sock

```
struct sock {
…
#define sk_state          __sk_common.skc_state
#define sk_v6_daddr       __sk_common.skc_v6_daddr
#define sk_v6_rcv_saddr   __sk_common.skc_v6_rcv_saddr
#define sk_reuse          __sk_common.skc_reuse
#define sk_reuseport      __sk_common.skc_reuseport
...
u32                sk_ack_backlog;
...
};
```

- sk_ack_backlog – текущая очередь прослушивания.

Также, за минимальное представление сокетов на сетевом уровне отвечает структура sock_common [5].

Листинг 2.7 – структура sock_common

```
struct sock_common {
...
volatile unsigned char    skc_state;
...
struct in6_addr           skc_v6_daddr;
struct in6_addr           skc_v6_rcv_saddr;
...
unsigned char             skc_reuse:4;
unsigned char             skc_reuseport:1;

union {
              __addrpair   skc_addrpair;
              struct {
                     __be32 skc_daddr;
                     __be32 skc_rcv_saddr;
              };
       };

union {
              __portpair   skc_portpair;
              struct {
                     __be16 skc_dport;
                     __u16  skc_num;
              };
       };


};
```

- skc_state – состояние подключения;
- skc_v6_daddr – адрес назначения IPV6;
- skc_v6_rcv_saddr – адрес источника IPV6;
- skc_reuse – параметр SO_REUSEADDR;
- skc_reuseport – параметр SO_REUSEPORT;
- skc_daddr – внешний IPv4 адрес;
- skc_rcv_saddr – связанный местный IPv4 адрес;
- skc_dport – «заполнитель» для inet_dport/tw_dport;
- skc_num – «заполнитель» для inet_num/tw_num.

Структура inet_timewait_sock позволяет избежать проблем с потреблением памяти сокетами на сильно загруженных серверах, но без нарушения спецификации протокола. В ней содержатся данные о адресах и портах клиента и сервера, которые используют сокеты с протоколом TCP [5].

Листинг 2.8 – структура inet_timewait_sock

```
struct inet_timewait_sock {

#define tw_v6_daddr       __tw_common.skc_v6_daddr
#define tw_v6_rcv_saddr   __tw_common.skc_v6_rcv_saddr
#define tw_daddr          __tw_common.skc_daddr
#define tw_rcv_saddr      __tw_common.skc_rcv_saddr
#define tw_dport          __tw_common.skc_dport

__be16              tw_sport;
volatile unsigned char    tw_substate;

};
```

- tw_sport – сравнение демультиплексирования сокетов для входящих пакетов;
- tw_substate – содержит состояние (либо TCP_TIME_WAIT, либо TCP_FIN_WAIT2).

Структура tcp_sock отвечает за представление TCP сокета на сетевом уровне [5].

Листинг 2.9 – структура tcp_sock

```
struct tcp_sock {
...
u32    rcv_nxt;
u32    copied_seq;
u32    write_seq;
u32    snd_una;
...
unsigned int        keepalive_time;
unsigned int        keepalive_intvl;
...
u8     keepalive_probes;
...

static inline struct tcp_sock *tcp_sk(const struct sock *sk)
{
       return (struct tcp_sock *)sk;
}

...
u8     nonagle     : 4,
       ...;

...
};
```

- rcv_nxt – что мы хотим получить в следующий раз;
- copied_seq – заголовок еще непрочитанных данных;
- write_seq –данные в буфере отправки TCP;
- snd_una – первый байт, для которого нужно подтверждение;
- keepalive_time – время до того, как keep alive займет место;
- keepalive_intvl – временной интервал между проверками активности;
- keepalive_probes – количество разрешенных проверок активности;
- nonagle – выключение алгоритма Нагла.

Структура inet_sock отвечает представление сокетов INET [5].

Листинг 2.10 – структура inet_sock

```
struct inet_sock {

#define inet_daddr         sk.__sk_common.skc_daddr
#define inet_rcv_saddr     sk.__sk_common.skc_rcv_saddr
#define inet_dport         sk.__sk_common.skc_dport
```

```
...

__be16                  inet_sport;

...

static inline struct inet_sock *inet_sk(const struct sock *sk)
{
        return (struct inet_sock *)sk;
}
};
```

- inet_sport – исходный порт.

TCP Fast Open является расширением для ускорения открытия последовательных протоколов управления передачей (TCP) соединения между двумя конечными точками [5].

Листинг 2.11 – структура struct fastopen_queue

```
struct fastopen_queue {
        int             max_qlen;
};
```

- max_qlen – максимальное количество запросов TFO, разрешенное до отключения TFO. (!= 0 if TFO is currently enabled).

На приведенных ниже рисунках продемонстрирована схема алгоритма получения информации о сокетах.

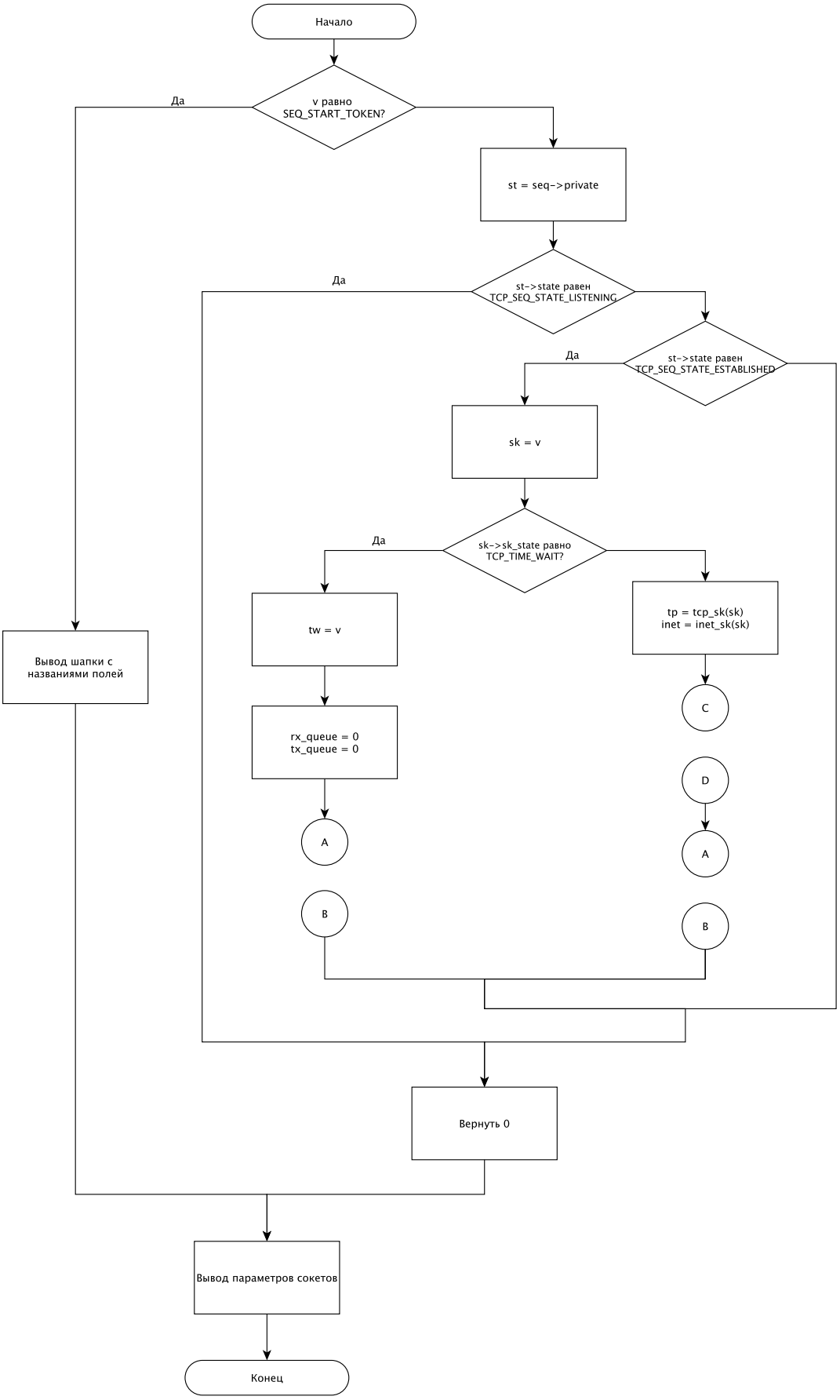

Рисунок 2.1 - схема алгоритма получения информации о сокетах

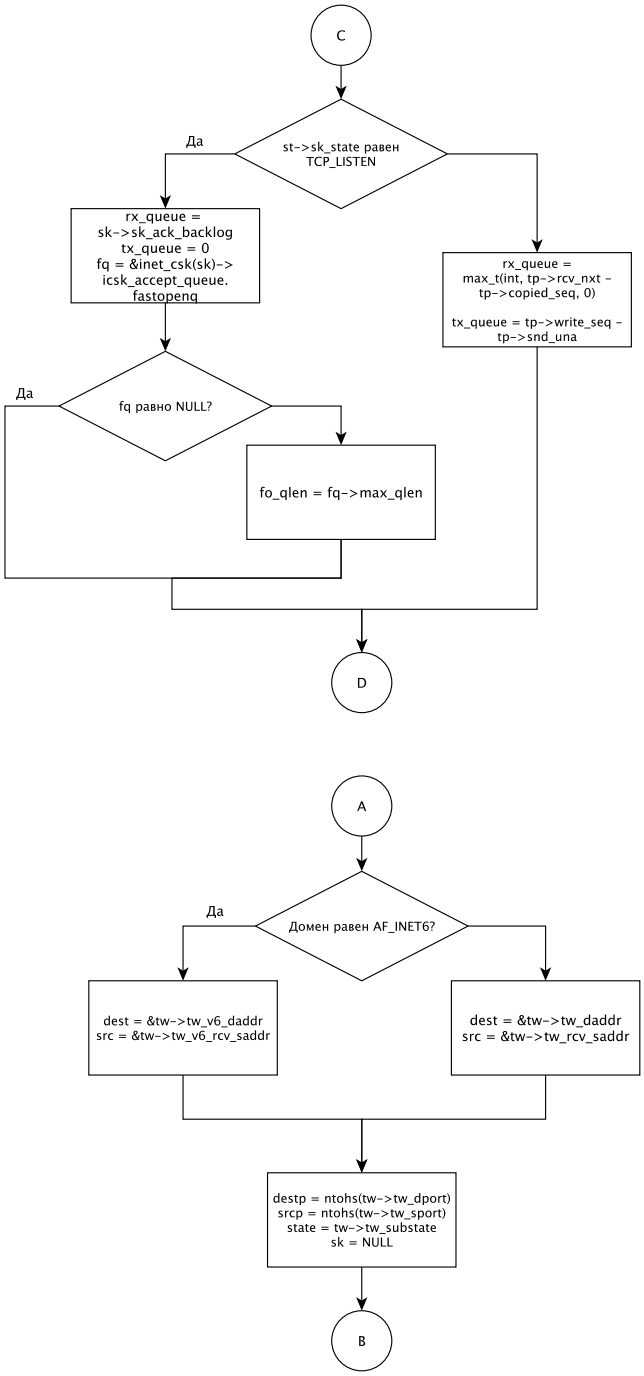

Рисунок 2.2 - схема алгоритма получения информации о сокетах

2.4 Вывод

В данном разделе был рассмотрен процесс проектирования структуры программного обеспечения, были выбраны структуры, необходимые для получения и вывода информации о сокетах.

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.

3.1 Выбор языка программирования

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

3.2 Выбор среды разработки

В качестве среды разработки был выбран стандартный текстовый редактор ОС Linux.

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

Листинг 3.1 – содержимое Makefile

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

KBUILD_EXTRA_SYMBOLS := $(KSRC)/Module.symvers

obj-m += knetstat.o

all:
     make -C $(KSRC) M=$(PWD) modules

clean:
     make -C $(KSRC) M=$(PWD) clean
```

3.3 Описание некоторых моментов реализации

В листинге 3.2 производится инициализация модуля ядра с помощью макросов входа/выхода, в листинге 3.3 – макросов модуля, в листинге 3.4 и 3.5 – конструктора/деструктора модуля.

Листинг 3.2 – макросы входа/выхода

```
module_init(knetstat_init)
module_exit(knetstat_exit)
```

Листинг 3.3 – макросы модуля

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Anastasia Lavrova");
MODULE_DESCRIPTION("Support for /proc/net/tcpstat,
/proc/net/tcp6stat, /proc/net/udpstat, /proc/net/udp6stat");
```

Листинг 3.4 – конструктор модуля

```
static int __net_init knetstat_net_init(struct net *net) {
     if (!proc_create_net_data("tcpstat", 0444, net->proc_net,
&tcpstat_seq_ops,
          sizeof(struct tcp_iter_state), &tcpstat_seq_afinfo))
     return -ENOMEM;

     if (!proc_create_net_data("tcp6stat", 0444, net->proc_net,
&tcp6stat_seq_ops,
          sizeof(struct tcp_iter_state), &tcp6stat_seq_afinfo))
     remove_proc_entry("udpstat", net->proc_net);

     if (!proc_create_net_data("udpstat", 0444, net->proc_net,
&udpstat_seq_ops,
          sizeof(struct udp_iter_state), &udpstat_seq_afinfo))
     remove_proc_entry("tcp6stat", net->proc_net);

     if (!proc_create_net_data("udp6stat", 0444, net->proc_net,
&udp6stat_seq_ops,
          sizeof(struct udp_iter_state), &udp6stat_seq_afinfo))
     remove_proc_entry("tcpstat", net->proc_net);

     return 0;
```

Листинг 3.5 – деструктор модуля

```
static void __net_exit knetstat_net_exit(struct net *net) {
     remove_proc_entry("tcpstat", net->proc_net);
     remove_proc_entry("tcp6stat", net->proc_net);
     remove_proc_entry("udpstat", net->proc_net);
     remove_proc_entry("udp6stat", net->proc_net);
}
```

Для работы с seq_file были использованы стандартные функции start, next и stop, а функция show переопределена собственной функцией tcp_seq_show.

Листинг 3.6 – структура seq_operations tcpstat_seq_ops

```
static const struct seq_operations tcpstat_seq_ops = {
     .show          = tcp_seq_show,
     .start         = tcp_seq_start,
     .next          = tcp_seq_next,
     .stop          = tcp_seq_stop,
};
```

С помощью функции sock_common_options_show выводится в соответствующий файл-последовательности параметры сокетов.

Листинг 3.7 – функция sock_common_options_show

```
static void sock_common_options_show(struct seq_file *seq, struct
sock *sk) {
     if (sk->sk_userlocks & SOCK_RCVBUF_LOCK) {
     seq_printf(seq, ",SO_RCVBUF=%d", sk->sk_rcvbuf / 2);
     }
     if (sk->sk_userlocks & SOCK_SNDBUF_LOCK) {
     seq_printf(seq, ",SO_SNDBUF=%d", sk->sk_sndbuf / 2);
     }

     if (sk->sk_rcvtimeo != MAX_SCHEDULE_TIMEOUT) {
     seq_printf(seq, ",SO_RCVTIMEO=%ldms", sk->sk_rcvtimeo*1000/HZ);
     }
     if (sk->sk_sndtimeo != MAX_SCHEDULE_TIMEOUT) {
     seq_printf(seq, ",SO_SNDTIMEO=%ldms", sk->sk_sndtimeo*1000/HZ);
     }

     if (sock_flag(sk, SOCK_LINGER)) {
     seq_printf(seq, ",SO_LINGER=%lds", sk->sk_lingertime / HZ);
     }
}
```

Для вывода информации об адресе и порте сокета используется функция addr_port_show, которая получает на вход файл-последовательности для вывода, значение домена, адрес и порт сокета.

Листинг 3.8 – функция addr_port_show

```
static void addr_port_show(struct seq_file *seq, sa_family_t family,
const void* addr, __u16 port) {
     seq_setwidth(seq, 23);
     seq_printf(seq, family == AF_INET6 ? "%pI6c" : "%pI4", addr);
     if (port == 0) {
     seq_puts(seq, ":*");
```

```
      } else {
      seq_printf(seq, ":%d", port);
      }
      seq_pad(seq, ' ');
}
```

Функция для обработки информации о TCP сокете представлена в листинге 3.9 [6].

```
static int tcp_seq_show(struct seq_file *seq, void *v) {
      if (v == SEQ_START_TOKEN) {
      seq_printf(seq, "Recv-Q Send-Q Local Address           Foreign
Address         Stat Options\n");
      } else {
      struct tcp_iter_state *st = seq->private;

      struct tcp_seq_afinfo *afinfo = PDE_DATA(file_inode(seq-
>file));
      sa_family_t family = afinfo->family;

      ...

      switch (st->state) {
            case TCP_SEQ_STATE_LISTENING:
            case TCP_SEQ_STATE_ESTABLISHED: {
                  sk = v;
                  if (sk->sk_state == TCP_TIME_WAIT) {
                        ...
                  } else {
                        ...
                        switch (sk->sk_state) {
                              case TCP_LISTEN:
                                    ...
                              default:
                                    ...
                        }
                        if (family == AF_INET6) {
                              ...
                        } else {
                              ...
                        }
                        ...
                  }
                  break;
            }
            default:
                  return 0;
      }
      ...
      seq_printf(seq, "%6d %6d ", rx_queue, tx_queue);
      addr_port_show(seq, family, src, srcp);
      addr_port_show(seq, family, dest, destp);

      seq_printf(seq, "%s ", tcp_state_names[state]);
      if (sk != NULL) {
```

```
            seq_printf(seq,
"SO_REUSEADDR=%d,SO_REUSEPORT=%d,SO_KEEPALIVE=%d", sk->sk_reuse, sk-
>sk_reuseport, sock_flag(sk, SOCK_KEEPOPEN));
                            if (tcp_sk(sk)->keepalive_time > 0) {
                                    ...
                            }
                            if (tcp_sk(sk)->keepalive_probes > 0) {
                                    ...
                            }
                            if (tcp_sk(sk)->keepalive_intvl > 0) {
                                    ...
                            }
                ...

        }
        seq_printf(seq, "\n");
        }
        return 0;
}
```

Функция для обработки информации о UDP сокете представлена в листинге 3.10.

```
static int udp_seq_show(struct seq_file *seq, void *v) {
        if (v == SEQ_START_TOKEN) {
        seq_printf(seq, "Recv-Q Send-Q Local Address           Foreign
Address         Options\n");
        }
        else {
        ...
        if (family == AF_INET6) {
                dest = &sk->sk_v6_daddr;
                src = &sk->sk_v6_rcv_saddr;
        } else {
                dest = &inet->inet_daddr;
                src = &inet->inet_rcv_saddr;
        }
        destp = ntohs(inet->inet_dport);
        srcp = ntohs(inet->inet_sport);

        seq_printf(seq, "%6d %6d ", rx_queue, tx_queue);
        addr_port_show(seq, family, src, srcp);
        addr_port_show(seq, family, dest, destp);

        seq_printf(seq, "SO_REUSEADDR=%d,SO_REUSEPORT=%d", sk-
>sk_reuse, sk->sk_reuseport);

        sock_common_options_show(seq, sk);

        seq_printf(seq, ",SO_BROADCAST=%d", sock_flag(sk,
SOCK_BROADCAST));
        seq_printf(seq, "\n");
        }
        return 0;
}
```

3.4 Пример работы загружаемого модуля ядра

Ниже на рисунке 3.1 приведен вывод буфера сообщений ядра при помощи команды cat /proc/net/<имя файла> (например, tcpstat, tcp6stat, udpstat, udp6stat).

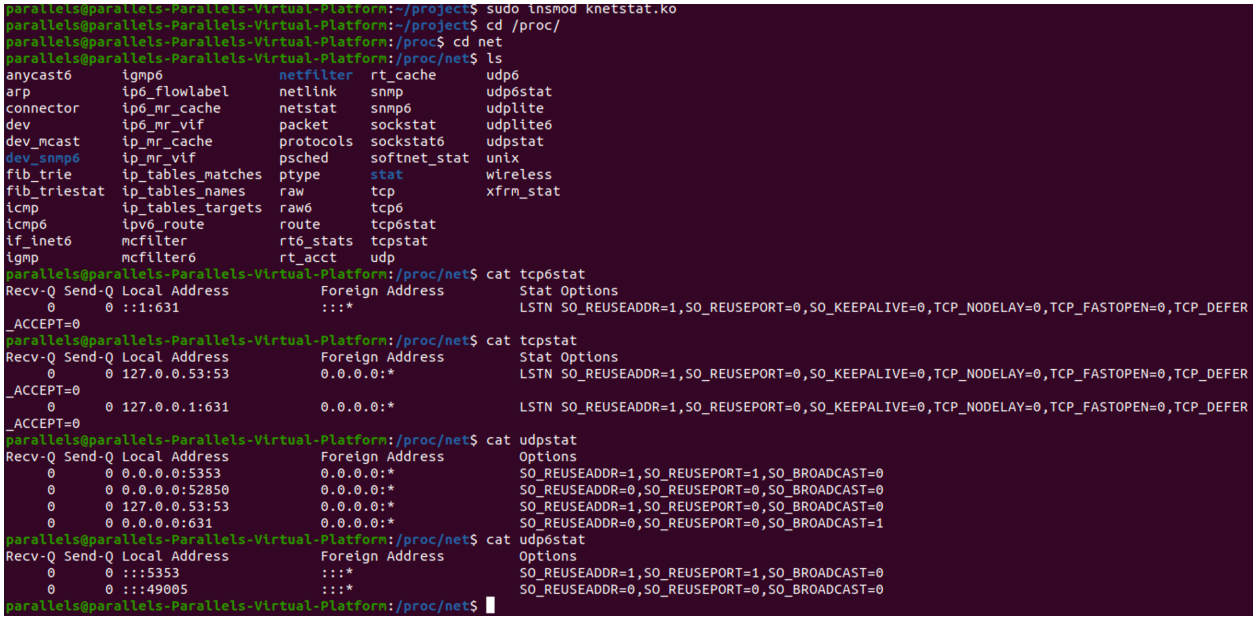

Рисунок 3.1 – пример работы загружаемого модуля ядра

3.5 Вывод

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

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы были выполнены следующие задачи:

- было изучено устройство модуля ядра;
- рассмотрены существующие типы сокетов;
- разработано программное обеспечение, позволяющее осуществить считывание и вывод информации о сокетах.

Список используемых источников

1. Анатомия загружаемых модулей ядра Linux. [Электронный ресурс] – Режим доступа: https://www.ibm.com/developerworks/ru/library/l-lkm/index.html (дата обращения: 05.12.2020).
2. Программирование для Linux. Сокеты. [Электронный ресурс] – Режим доступа: http://citforum.ru/programming/unix/sockets/ (дата обращения: 08.12.2020).
3. У.Р. Стивенс, Б. Феннер, Э.М. Рудофф «UNIX: разработка сетевых приложений». – 3-е изд. - СПб.: Питер, 2007.
4. Стесик О., Немнюгин С., Робачевский А. «Операционная система UNIX». – 2-е изд. - СПб: БХВ-Петербург, 2010.
5. Исходный код ядра Linux. [Электронный ресурс] – Режим доступа: https://elixir.bootlin.com/linux/latest/source (дата обращения: 13.12.2020).
6. Состояния сеанса TCP [Colobridge Wiki]. [Электронный ресурс] – Режим доступа: https://wiki.colobridge.net/сети/состояния_сеанса_tcp (дата обращения: 15.12.2020).

ПРИЛОЖЕНИЕ А

```
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <linux/types.h>
#include <net/tcp.h>
#include <net/tcp_states.h>
#include <net/udp.h>

#include <net/net_namespace.h>


static const char *const tcp_state_names[] = {
     "NONE",
     "ESTB",
     "SYNS",
     "SYNR",
     "FNW1",
     "FNW2",
     "TIMW",
     "CLSD",
     "CLSW",
     "LACK",
     "LSTN",
     "CLSG",
     "SYNR"
};

static void sock_common_options_show(struct seq_file *seq,
struct sock *sk) {
     if (sk->sk_userlocks & SOCK_RCVBUF_LOCK) {
     seq_printf(seq, ",SO_RCVBUF=%d", sk->sk_rcvbuf / 2);
     }
     if (sk->sk_userlocks & SOCK_SNDBUF_LOCK) {
     seq_printf(seq, ",SO_SNDBUF=%d", sk->sk_sndbuf / 2);
     }

     if (sk->sk_rcvtimeo != MAX_SCHEDULE_TIMEOUT) {
     seq_printf(seq, ",SO_RCVTIMEO=%ldms", sk-
>sk_rcvtimeo*1000/HZ);
     }
     if (sk->sk_sndtimeo != MAX_SCHEDULE_TIMEOUT) {
```

```
    seq_printf(seq, ",SO_SNDTIMEO=%ldms", sk-
>sk_sndtimeo*1000/HZ);
    }

    if (sock_flag(sk, SOCK_LINGER)) {
    seq_printf(seq, ",SO_LINGER=%lds", sk->sk_lingertime
/ HZ);
    }
}

static void addr_port_show(struct seq_file *seq,
sa_family_t family, const void* addr, __u16 port) {
    seq_setwidth(seq, 23);
    seq_printf(seq, family == AF_INET6 ? "%pI6c" :
"%pI4", addr);
    if (port == 0) {
    seq_puts(seq, ":*");
    } else {
    seq_printf(seq, ":%d", port);
    }
    seq_pad(seq, ' ');
}

static int tcp_seq_show(struct seq_file *seq, void *v) {
    if (v == SEQ_START_TOKEN) {
    seq_printf(seq, "Recv-Q Send-Q Local Address
Foreign Address         Stat Options\n");
    } else {
    struct tcp_iter_state *st = seq->private;

    struct tcp_seq_afinfo *afinfo =
PDE_DATA(file_inode(seq->file));
    sa_family_t family = afinfo->family;

    int rx_queue;
    int tx_queue;
    const void *dest;
    const void *src;
    __u16 destp;
    __u16 srcp;
    int state;
    struct sock *sk;
    int fo_qlen = 0;
    u8 defer = 0;
```

```
    switch (st->state) {
        case TCP_SEQ_STATE_LISTENING:
        case TCP_SEQ_STATE_ESTABLISHED: {
            sk = v;
            if (sk->sk_state == TCP_TIME_WAIT) {
                const struct inet_timewait_sock *tw =
v;

                rx_queue = 0;
                tx_queue = 0;
                if (family == AF_INET6) {
                    dest = &tw->tw_v6_daddr;
                    src = &tw->tw_v6_rcv_saddr;
                } else {
                    dest = &tw->tw_daddr;
                    src = &tw->tw_rcv_saddr;
                }
                destp = ntohs(tw->tw_dport);
                srcp = ntohs(tw->tw_sport);
                state = tw->tw_substate;
                sk = NULL;
            } else {
                const struct tcp_sock *tp;
                const struct inet_sock *inet;
                const struct fastopen_queue *fq;

                tp = tcp_sk(sk);
                inet = inet_sk(sk);
                defer = inet_csk(sk)-
>icsk_accept_queue.rskq_defer_accept;

                switch (sk->sk_state) {
                    case TCP_LISTEN:
                        rx_queue = sk-
>sk_ack_backlog;
                        tx_queue = 0;
                        fq = &inet_csk(sk)-
>icsk_accept_queue.fastopenq;
                        if (fq != NULL) {
                            fo_qlen = fq->max_qlen;
                        }
                        break;
                    default:
                        rx_queue = max_t(int, tp-
>rcv_nxt - tp->copied_seq, 0);
```

```
                            tx_queue = tp->write_seq -
tp->snd_una;
                    }
                    if (family == AF_INET6) {
                        dest = &sk->sk_v6_daddr;
                        src = &sk->sk_v6_rcv_saddr;
                    } else {
                        dest = &inet->inet_daddr;
                        src = &inet->inet_rcv_saddr;
                    }
                    destp = ntohs(inet->inet_dport);
                    srcp = ntohs(inet->inet_sport);
                    state = sk->sk_state;
                }
                break;
            }
            default:
                return 0;
        }

        if (state < 0 || state >= TCP_MAX_STATES) {
            state = 0;
        }

        seq_printf(seq, "%6d %6d ", rx_queue, tx_queue);
        addr_port_show(seq, family, src, srcp);
        addr_port_show(seq, family, dest, destp);

        seq_printf(seq, "%s ", tcp_state_names[state]);
        if (sk != NULL) {
            seq_printf(seq,
"SO_REUSEADDR=%d,SO_REUSEPORT=%d,SO_KEEPALIVE=%d", sk-
>sk_reuse, sk->sk_reuseport, sock_flag(sk,
SOCK_KEEPOPEN));
                            if (tcp_sk(sk)->keepalive_time
> 0) {
                                seq_printf(seq,
",TCP_KEEPIDLE=%u", tcp_sk(sk)->keepalive_time/HZ);
                            }
                            if (tcp_sk(sk)-
>keepalive_probes > 0) {
                                seq_printf(seq,
",TCP_KEEPCNT=%u", tcp_sk(sk)->keepalive_probes);
                            }
```

```
                        if (tcp_sk(sk)-
>keepalive_intvl > 0) {
                                seq_printf(seq,
",TCP_KEEPINTVL=%u", tcp_sk(sk)->keepalive_intvl/HZ);
                        }

        sock_common_options_show(seq, sk);

        seq_printf(seq, ",TCP_NODELAY=%d",
!!(tcp_sk(sk)->nonagle&TCP_NAGLE_OFF));

        if (state == TCP_LISTEN) {
            seq_printf(seq, ",TCP_FASTOPEN=%d",
fo_qlen);
        }

        seq_printf(seq, ",TCP_DEFER_ACCEPT=%d", defer);

    }
    seq_printf(seq, "\n");
    }
    return 0;
}

static const struct seq_operations tcpstat_seq_ops = {
    .show         = tcp_seq_show,
    .start        = tcp_seq_start,
    .next         = tcp_seq_next,
    .stop         = tcp_seq_stop,
};

static struct tcp_seq_afinfo tcpstat_seq_afinfo = {
    .family       = AF_INET,
};

static const struct seq_operations tcp6stat_seq_ops = {
    .show         = tcp_seq_show,
    .start        = tcp_seq_start,
    .next         = tcp_seq_next,
    .stop         = tcp_seq_stop,
};

static struct tcp_seq_afinfo tcp6stat_seq_afinfo = {
    .family       = AF_INET6,
};
```

```
static int udp_seq_show(struct seq_file *seq, void *v) {
    if (v == SEQ_START_TOKEN) {
    seq_printf(seq, "Recv-Q Send-Q Local Address
Foreign Address         Options\n");
    } else {
    struct udp_seq_afinfo *afinfo =
PDE_DATA(file_inode(seq->file));
    sa_family_t family = afinfo->family;
    struct sock *sk = v;
    int tx_queue = sk_wmem_alloc_get(sk);
    int rx_queue = sk_rmem_alloc_get(sk);
    struct inet_sock *inet = inet_sk(sk);
    const void *dest;
    const void *src;
    __u16 destp;
    __u16 srcp;

    if (family == AF_INET6) {
        dest = &sk->sk_v6_daddr;
        src = &sk->sk_v6_rcv_saddr;
    } else {
        dest = &inet->inet_daddr;
        src = &inet->inet_rcv_saddr;
    }
    destp = ntohs(inet->inet_dport);
    srcp = ntohs(inet->inet_sport);

    seq_printf(seq, "%6d %6d ", rx_queue, tx_queue);
    addr_port_show(seq, family, src, srcp);
    addr_port_show(seq, family, dest, destp);

    seq_printf(seq, "SO_REUSEADDR=%d,SO_REUSEPORT=%d", sk-
>sk_reuse, sk->sk_reuseport);

    sock_common_options_show(seq, sk);

    seq_printf(seq, ",SO_BROADCAST=%d", sock_flag(sk,
SOCK_BROADCAST));

    seq_printf(seq, "\n");
    }
    return 0;
}
```

```
static const struct seq_operations udpstat_seq_ops = {
     .start       = udp_seq_start,
     .next        = udp_seq_next,
     .stop        = udp_seq_stop,
     .show        = udp_seq_show,
};

static struct udp_seq_afinfo udpstat_seq_afinfo = {
     .family      = AF_INET,
     .udp_table   = &udp_table,
};

static const struct seq_operations udp6stat_seq_ops = {
     .start       = udp_seq_start,
     .next        = udp_seq_next,
     .stop        = udp_seq_stop,
     .show        = udp_seq_show,
};

static struct udp_seq_afinfo udp6stat_seq_afinfo = {
     .family      = AF_INET6,
     .udp_table   = &udp_table,
};

static int __net_init knetstat_net_init(struct net *net) {
    if (!proc_create_net_data("tcpstat", 0444, net-
>proc_net, &tcpstat_seq_ops,
         sizeof(struct tcp_iter_state),
&tcpstat_seq_afinfo))
        return -ENOMEM;

    if (!proc_create_net_data("tcp6stat", 0444, net-
>proc_net, &tcp6stat_seq_ops,
         sizeof(struct tcp_iter_state),
&tcp6stat_seq_afinfo))
        remove_proc_entry("udpstat", net->proc_net);

    if (!proc_create_net_data("udpstat", 0444, net-
>proc_net, &udpstat_seq_ops,
         sizeof(struct udp_iter_state),
&udpstat_seq_afinfo))
        remove_proc_entry("tcp6stat", net->proc_net);

    if (!proc_create_net_data("udp6stat", 0444, net-
>proc_net, &udp6stat_seq_ops,
```

```
         sizeof(struct udp_iter_state),
&udp6stat_seq_afinfo))
        remove_proc_entry("tcpstat", net->proc_net);

    return 0;
}

static void __net_exit knetstat_net_exit(struct net *net) {
    remove_proc_entry("tcpstat", net->proc_net);
    remove_proc_entry("tcp6stat", net->proc_net);
    remove_proc_entry("udpstat", net->proc_net);
    remove_proc_entry("udp6stat", net->proc_net);
}

static struct pernet_operations knetstat_net_ops = { .init =
knetstat_net_init,
    .exit = knetstat_net_exit, };

static int knetstat_init(void) {
    int err;

    err = register_pernet_subsys(&knetstat_net_ops);
    if (err < 0)
    return err;

    return 0;
}

static void knetstat_exit(void) {
    unregister_pernet_subsys(&knetstat_net_ops);
}

module_init(knetstat_init)
module_exit(knetstat_exit)

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Anastasia Lavrova");
MODULE_DESCRIPTION("Support       for       /proc/net/tcpstat,
/proc/net/tcp6stat, /proc/net/udpstat, /proc/net/udp6stat");
```