**Лабораторная работа**
**Ядро Linux**

**Цели работы:**

- изучить архитектуру и основные компоненты ядра Linux, включая механизм системных вызовов
- изучить возможности утилиты **strace** для отладки и анализа работы программ на уровне системных вызовов
- приобрести практический навык создания и загрузки модуля в ядро Linux

## Теоретические сведения

Ядро операционной системы является центральной частью операционной системы, которая обеспечивает приложениям координированный доступ к ресурсам компьютера. Ядро управляет доступом к процессору, памяти, устройствам ввода-вывода, жесткими дискам, сетевым интерфейсам и другим аппаратным ресурсам; работает в привилегированном режиме и может выполнять операции, недоступные для процессов пользовательского режима.

Ядро Linux было разработано Линусом Торвальдсом в 1991 году и соответствует стандартам POSIX. Оно составляет основу операционных систем семейства Linux и активно развивается сообществом разработчиков. Ядро Linux распространяется под лицензией GNU General Public License версии 2 (GPLv2), это означает, что исходный код ядра является открытым и доступным для каждого. Любой желающий может использовать, изменять и распространять его в соответствии с условиями лицензии.

## Компоненты ядра Linux

- Планировщик процессов **Process Scheduler** (SCHED) отвечает за контроль над доступом процессов к CPU. Он определяет, какие процессы должны быть запущены, когда они должны быть запущены и какой приоритет должен быть назначен каждому процессу. Поддерживает несколько различных алгоритмов планирования.
- Менеджер памяти **Memory Manager** (MM) обеспечивает различным процессам безопасный доступ к основной памяти системы, включая выделение и освобождение памяти, управление виртуальной памятью и кеширование.
- Виртуальная файловая система **Virtual File System** (VFS) создает абстрактный слой, скрывая детали оборудования, предоставляя общий файловый интерфейс для всех устройств.
- Сетевые интерфейсы **Network Interface** (NET) обеспечивают управление сетевыми соединениями в системе, включая управление протоколами сетевого уровня и сетевыми настройками.
- Подсистема межпроцессного взаимодействия **Inter-Process Communication** (IPC) реализует механизмы обмена данными между процессами, такие как сигналы, сокеты, каналы и разделяемая память. Кроме этого, она предоставляет средства синхронизации (мьютексы, семафоры, события, барьеры, условные переменные) для координации работы одновременно выполняющихся потоков.

Ядро Linux обладает множеством дополнительных функций, которые делают его одним из наиболее гибких и мощных ядер операционных систем в мире. Эти функции включают поддержку управления безопасностью, энергопотреблением, виртуализацией и многоядерных процессоров.

С точки зрения архитектуры ядро Linux представляет собой монолитное ядро с возможностью загрузки и выгрузки модулей во время работы системы. Это дает возможность добавлять новую функциональность без перекомпиляции ядра или перезагрузки системы. Примером использования загружаемых модулей является добавление драйверов для поддержки новых устройств.

Таким образом, ядро Linux сочетает в себе преимущества как монолитных, так и микроядерных систем. Оно быстро обрабатывает системные вызовы и взаимодействует с аппаратным обеспечением, а также позволяет динамически добавлять и удалять функциональность ядра, обеспечивая гибкость и расширяемость системы.

## Системные вызовы Linux

Системные вызовы - это механизм, который операционная система предоставляет приложениям для взаимодействия с ядром. Приложения обращаются к ядру с помощью системных вызовов для выполнения операций, запрещенных в пользовательском режиме. Ядро обрабатывает запрос и возвращает результат приложению.

Для переключения из пользовательского в привилегированный режим используется механизм программных прерываний. Ядро операционной системы предоставляет API системных вызовов, который позволяет процессам получать доступ к различным системным функциям, таким как создание нового процесса, выполнение операций ввода-вывода, создание конвейеров для взаимодействия между процессами и др..

## Утилита strace

Утилита **strace** - инструмент, позволяющий отслеживать системные вызовы и сигналы, генерируемые процессом. С помощью **strace** можно просмотреть список системных вызовов, которые выполняются в процессе, переданные им аргументы и возвращаемые значения. Утилита присутствует во многих дистрибутивах Linux по умолчанию, и может использоваться в диагностических, учебных или отладочных целях.

В простейшем случае **strace** запускает переданную команду (или программу) с её аргументами и выводит в стандартный поток ошибок или в файл (с опцией -o) все системные вызовы, которые возникают при выполнении команды, а также информацию о полученных сигналах.

Синтаксис: **strace <опции> <команда> <аргументы>**.

Полный набор опций, доступных для команды **strace,** содержатся в ее руководстве (man-странице). Чтобы просмотреть руководство, введите в терминале команду ***man strace***. Вывод команды: имя_системного_вызова(параметр1,параметр2)= результат сообщение.

Имя системного вызова указывает, какой именно вызов использовала программа. Передаваемые параметры приводятся в круглых скобках. После

знака равенства выводится результат выполнения. Если вызов выполнен успешно, возвращается ноль или положительное число. Отрицательное значение означает, что произошла ошибка. В этом случае выводится сообщение.

## Модули ядра

**Модули ядра,** загружаемые модули ядра (loadable kernel module, LKM) - динамически подключаемые объектные файлы, которые можно загружать и выгружать из ядра во время работы системы. Они используются для добавления новой функциональности, такой как поддержка новых устройств или файловых систем, а также для расширения возможностей системных вызовов. Модули ядра обычно хранятся в директории /lib/modules в файлах с расширением .ko. При необходимости модуль может быть выгружен.

В Linux имеются утилиты, предназначенные для работы с модулями ядра:

**modprobe** - загружает, выгружает и настраивает модули ядра с автоматической установкой зависимостей модуля и их загрузкой в случае, если они не были загружены ранее;

**insmod -** загружает модуль ядра в систему, не проверяет зависимости модуля, что может привести к ошибкам;

**rmmod -** выгружает модуль ядра из системы;

**lsmod -** отображает список загруженных модулей ядра;

**modinfo -** выводит информацию о модуле, такую как имя, версия, автор, описание и зависимости;

**depmod -** генерирует файлы зависимостей модулей для ядра, использующиеся при загрузке модулей и гарантирующие, что все зависимости будут загружены.

## Автозагрузка и автосборка модулей

В Linux модули ядра могут быть автоматически загружены при старте системы и автоматически собраны при обновлении ядра. Для этого существуют специальные механизмы автозагрузки и автосборки модулей.

## Автозагрузка модулей

В дистрибутивах Linux используются различные методы настройки автоматической загрузки модулей ядра при запуске системы, в зависимости от используемой системы инициализации.

Система инициализации **systemd,** которая присутствует во многих современных дистрибутивах Linux, поддерживает два механизма автоматической загрузки модулей ядра. Первый механизм использует файл **/etc/modules,** в котором перечисляются модули, которые должны быть загружены при старте системы. Этот механизм работает не только для **systemd**, но и для других систем инициализации, например, **sysvinit**.

Второй механизм - это использование файлов конфигурации в директории **/etc/modules-load.d/**. В них перечислены модули, которые должны быть загружены при старте системы, только в случае, если используется **systemd**. Этот механизм является более гибким, так как позволяет загружать разные модули в зависимости от конфигурации системы.

Некоторые дистрибутивы Linux могут использовать различные системы инициализации, такие как **systemd**, **sysvinit**, **Upstart** или **OpenRC**. Например, в Ubuntu начиная с версии 15.10 по умолчанию используется система инициализации **systemd**, однако, для совместимости со старыми версиями и пакетами, также поддерживаются **sysvinit** и **Upstart**.

### Автосборка модулей

При обновлении ядра некоторые его модули могут перестать работать, так как они скомпилированы под старую версию ядра. Для решения этой проблемы используется механизм автосборки модулей.

**DKMS** (Dynamic Kernel Module Support) - фреймворк для автоматической сборки и установки модулей ядра в Linux. **DKMS** устанавливает модуль ядра в систему, используя Makefile. Кроме того, **DKMS** позволяет автоматически обновлять и удалять модули ядра, если они больше не нужны или если ядро было обновлено.

### Команды для работы с DKMS

1. Установка DKMS: ***sudo apt-get install dkms***
2. Просмотр списка установленных модулей ядра: ***dkms status***
3. Добавление нового модуля в DKMS:
   ***sudo dkms add -m <module-name> -v <module-version>***
4. Сборка и установка модуля:
   ***sudo dkms build -m <module-name> -v <module-version>***
   ***sudo dkms install -m <module-name> -v <module-version>***
5. Обновление модуля:
   ***sudo dkms upgrade -m <module-name> -v <module-version>***
6. Удаление модуля из DKMS:
   ***sudo dkms remove -m <module-name> -v <module-version> --all***

Более подробная информация о командах и использовании **DKMS** содержится в официальной документации **DKMS**.

### Порядок выполнения работы

#### Задание 1. Анализ системных вызовов с помощью утилиты strace

1. Убедитесь, что **strace** установлена, запустив ее с параметром -V: ***strace -V***. Если утилита отсутствует, установите ее: ***sudo apt install strace***.
2. Ознакомьтесь со справкой об использовании утилиты **strace**: ***man strace***.
3. Запустите strace для команды из таблицы 1. Обратите внимание, что в некоторых случаях потребуется задать аргументы. На основании полученных результатов заполните таблицу 2 для 5-7 различных системных вызовов.

Таблица 1. Варианты команд

| № вар | Команда | № вар | Команда | № вар | Команда | № вар | Команда | № вар | Команда |
|---|---|---|---|---|---|---|---|---|---|
| 1 | ls | 7 | clear | 13 | hostname | 19 | head | 25 | sleep |
| 2 | echo | 8 | whoami | 14 | du -V | 20 | tail | 26 | file |
| 3 | man | 9 | uname | 15 | strace | 21 | id | 27 | getconf |
| 4 | pwd | 10 | free | 16 | lastb | 22 | lscpu | 28 | getfacl |
| 5 | file | 11 | find | 17 | lsattr | 23 | lslocks | 29 | grep |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 6 | date | 12 | cat | 18 | nproc | 24 | lsusb | 30 | last |

Документация по системным вызовам приведена на странице руководства man syscalls(2): man 2 <syscall-NAME> .

Таблица 2. Результаты анализа

| № | Системный вызов | Описание вызова | Входные параметры | Время выполнения, мкс | Возвращаемое значение |
|---|---|---|---|---|---|
| | | | | | |

4. Перенаправьте вывод **strace** в файл log в вашей домашней директории.
5. Получите статистику выполненных системных вызовов.
6. Выполните трассировку системных вызовов для произвольного работающего процесса, подключившись к нему по PID.

**Задание 2. Сборка и загрузка модуля в ядро Linux**

1. Установите необходимые пакеты:

***apt-get install gcc make linux-headers-$(uname -r)***

2. Создайте файл модуля:

***mkdir kmod-hello_world***
***cd kmod-hello_world/***
***touch ./mhello.c***

с содержимым:

```
#define MODULE
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
MODULE_LICENSE("GPLv3");
int init_module(void){
   printk("<1> Hello,World\n");
   return 0;
}
void cleanup_module(void){
   printk("<1> Goodbye.\n");
}
```

3. Создайте Makefile:

***touch ./Makefile***

с содержимым:

```
obj-m += mhello.o
hello-objs := mhello.c
all:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

Перед командой "make" необходимо использовать табуляцию для создания отступа, а не пробелы.

4. Соберите модуль

***make all***

и установите его с помощью insmod:

***sudo insmod mhello.ko***

После этого модуль появится в списке установленных модулей:

***lsmod***

При помощи команды dmesg выведите буфер сообщений от ядра:

***sudo dmesg***

В конце вывода должно отобразиться сообщение от установленного модуля «Hello, World”.

В отчете по работе приведите снимки экрана установки модуля, результатов выполнения **dmesg** и **lsmod**.

5. Выгрузите модуль с помощью команды **rmmod** и включите снимок экрана вывода в отчет. Убедитесь, что модуль выгружен с помощью **dmesg** и **lsmod.**

**Дополнительное задание**

Выполните разбор системных вызовов для собственной программы (или скрипта). Программа должна выполняться не менее 1 минуты. Заполните таблицу 2 из задания 1. Определите, какой процент от времени выполнения всей программы составляет время выполнения системных вызовов.