Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, Е.А. Черепков

# КОНФИГУРИРОВАНИЕ ЯДРА LINUX

Методические указания к выполнению лабораторной работы

по курсу «Операционные системы»

Калуга – 2018

УДК 004.62
ББК 32.972.1
Б435

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 3 от «24» октября 2018 г.

Зав. кафедрой ИУ4-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол № 3 от «29» октября 2018 г.

Председатель методической
комиссии факультета ИУ-КФ _____ к.т.н., доцент М.Ю. Адкин

- Методической комиссией
КФ МГТУ им.Н.Э. Баумана протокол № 2 от «6» ноября 2018 г.

Председатель методической комиссии
КФ МГТУ им.Н.Э. Баумана
_____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ИУ3-КФ _____ А.В. Финошин

Авторы
к.ф.-м.н., доцент кафедры ИУ4-КФ _____ Ю.С. Белов
ассистент кафедры ИУ4-КФ _____ Е.А. Черепков

Аннотация

Методические указания к выполнению лабораторной работы по курсу «Операционные системы» содержат общие сведения об архитектуре ОС семейства Linux и процессе сборки ядра ОС.

Предназначены для студентов 3-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Операционные системы» на кафедре «Программное обеспечение ЭВМ, информационные технологии» факультета «Информатика и управление» Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержат краткое описание команд для конфигурирования и компиляции ярда ОС Linux.

Методические указания составлены для ознакомления студентов с работой с конфигурированием ядра Linux. Для выполнения лабораторной работы студенту необходимы минимальные навыки программирования и знания об операционной системе Linux.

## ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения лабораторной работы является приобретение практических навыков по конфигурированию и компиляции ядра ОС Linux.

Основными задачами выполнения лабораторной работы являются:

1. Получить навыки конфигурирования ядра Linux.
2. Получить навыки компиляции и запуску ядра Linux.

Результатами работы являются:

1. Демонстрация выполнения команд по конфигурированию и компиляции, и запуска ядра ОС Linux.
2. Подготовленный отчет.

## КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Ядро Linux — ядро операционной системы, соответствующее стандартам POSIX, составляющее основу операционных систем семейства Linux. Разработка кода ядра была начата финским студентом Линусом Торвальдсом в 1991 году, на его имя зарегистрирована Торговая марка «Linux».

Код написан в основном на Си с некоторыми расширениями gcc и на ассемблере (с использованием AT&T-синтаксиса GNU Assembler).

Распространяется как свободное программное обеспечение на условиях GNU General Public License, кроме несвободных элементов, особенно драйверов, которые используют прошивки, распространяемые под различными лицензиями.

Операционную систему можно условно разделить на два уровня, как показано на рисунке 1.

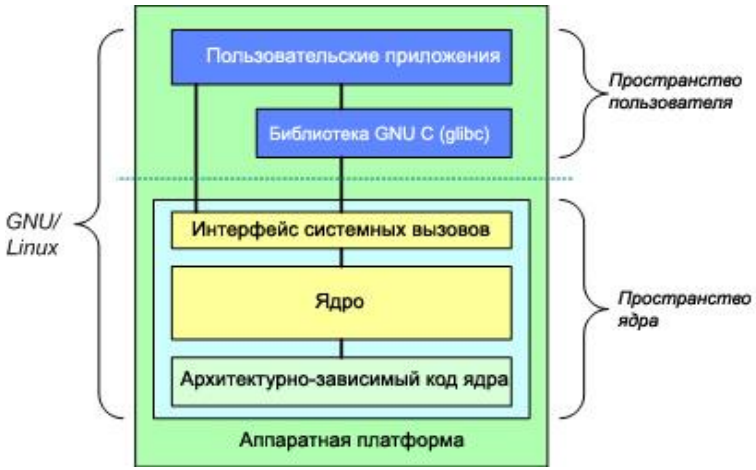


**Рис. 1**. Фундаментальная архитектура операционной системы GNU/Linux

На верхнем уровне находится пользовательское пространство (пространство приложений). Здесь исполняются приложения пользователя. Под пользовательским пространством располагается пространство ядра. Здесь функционирует ядро Linux.

Имеется также библиотека GNU C (glibc). Она предоставляет интерфейс системных вызовов, который обеспечивает связь с ядром и дает механизм для перехода от приложения, работающего в пространстве пользователя, к ядру. Это важно, поскольку ядро и пользовательское приложение располагаются в разных защищенных адресных пространствах. При этом, в то время как каждый процесс в пространстве пользователя имеет свое собственное виртуальное адресное пространство, ядро занимает одно общее адресное пространство.

Ядро Linux можно, в свою очередь, разделить на три больших уровня. Наверху располагается интерфейс системных вызовов, который реализует базовые функции, например, чтение и запись. Ниже интерфейса системных вызовов располагается код ядра, точнее говоря, архитектурно-независимый код ядра. Этот код является общим для всех процессорных архитектур, поддерживаемых Linux. Еще ниже располагается архитектурно-зависимый код, образующий т.н. BSP (Board Support Package - пакет поддержки аппаратной платформы). Этот код зависит от процессора и платформы для конкретной архитектуры.

**Свойства ядра Linux**

В ядре Linux реализован целый ряд важных архитектурных элементов. И на самом общем, и на более детальных уровнях ядро можно подразделить на множество различных подсистем. С другой стороны, Linux можно рассматривать как монолитное целое, поскольку все базовые сервисы собраны в ядре системы. Такой подход отличается от архитектуры с микроядром, когда ядро предоставляет только самые общие сервисы, такие как обмен информацией. ввод/вывод, управление

памятью и процессами, а более конкретные сервисы реализуются в модулях, подключаемых к уровню микроядра.

С течением времени ядро Linux стало более эффективным с точки зрения использования памяти и процессорных ресурсов и приобрело исключительную стабильность. Однако самый интересный аспект Linux, учитывая размер и сложность этой системы - это ее переносимость. Linux можно откомпилировать для огромного количества разных процессоров и платформ, имеющих разные архитектурные ограничения и потребности. Например, Linux может работать на процессоре как с блоком управления памятью (MMU), так и без MMU. Поддержка процессоров без MMU реализована в версии ядра uClinux.

**История**

Начало проекту было положено в 1991 году с публикации сообщения в новостной группе Usenet.

К тому времени GNU уже создал множество составляющих для свободной операционной системы, но её ядро GNU Hurd ещё не было готово. Поэтому пустующее место ядра для свободной операционной системы занял Linux и, несмотря на ограниченную функциональность ранних версий, привлёк к себе множество разработчиков и пользователей.

«Linux» как ядро операционной системы, разработка которого была начата Торвальдсом — лишь небольшая часть многих из использующих его систем, которые обычно тоже называют «Linux». Это иногда приводит к путанице, и те из них, которые используют системные библиотеки (например, GNU C Library) и другие программы проекта GNU, формально называют также «GNU/Linux».

По состоянию на сентябрь 2016 года семейство операционных систем на базе ядра Linux — третье по популярности в мире на рынке настольных компьютеров — 5,8 %. На рынке веб-серверов доля Linux порядка 32 %, остальное — Unix-системы (FreeBSD и др.). По данным

Top500 (июнь 2016 года), Linux используется в качестве операционной системы на 99,4 % самых мощных суперкомпьютеров планеты.

Большинство мобильных устройств, таких, как смартфоны и планшетные компьютеры на базе операционных систем Android, MeeGo, Tizen, а также телевизоры и многие служебные устройства, такие, как внешние сетевые жёсткие диски, маршрутизаторы и модемы, также используют операционные системы на базе ядра Linux.

## СБОРКА ЯДРА

Замечательной чертой Linux-подобных систем является то, что при необходимости можно не только изменить некоторые системные параметры, но и полностью перекомпилировать всю ОС.

Этапы сборки ядра:

1. Приобретение исходных кодов ядра.
2. Подготовка каталогов с исходниками ядра.
3. Конфигурирование ядра.
4. Компиляция ядра и установка модулей.
5. Перемещение ядра.
6. Настройка и запуск lilo.

Для нормальной сборки и компиляции ядра следует выполнять все этапы по порядку. Рассмотрим каждый из этапов подробнее.

**Приобретение исходных кодов ядра:**

Исходные коды ядра можно найти на дистрибутиве с ОС. Но на нем не всегда последняя версия ядра. Исходники последней версии ядра для Linux можно найти на ftp://ftp.kernel.org (или на каком-нибудь зеркале). Там ядра лежат в /pub/linux/. Загрузите ядро и помести его в каталог /usr/src. Там же создайте каталог для файлов и каталогов ядра (обычно создают каталог типа linux-4.X.X, где 4.X.X - версия нового ядра) командой mkdir linux-4.X.X. После этого создайте связь с каталогом linux (ln -s linux-4.X.X linux). Если каталог linux-4.X.X уже существует, то его предварительно надо удалить.

Хронология:

- Апрель 1991 — 21-летний Линус Торвальдс начал работу над некоторыми механизмами операционной системы. Он начал с эмулятора терминала и планировщика задач.
- 25 августа 1991 — Торвальдс поместил сообщение в новостную группу по Minix
- 17 сентября 1991 — Linux версии 0.01 (10 239 строк кода).

- 5 октября 1991 — Linux версии 0.02.
- Декабрь 1991 — Linux версии 0.11. Это была первая версия Linux, на которой можно было собрать Linux из исходных кодов.
- 19 января 1992 — Первое сообщение в группе новостей alt.os.linux.
- 31 марта 1992 — Создана группа новостей comp.os.linux (англ.).
- Апрель 1992 — Linux версии 0.96, на котором стало возможно запустить графический сервер X Window System.
- Весь 1993 и начало 1994 — 15 тестовых выпусков версии 0.99.* (в июле 1993 введено понятие BogoMips).
- 14 марта 1994 — Linux версии 1.0.0 (176 250 строк кода).
- Март 1995 — Linux версии 1.2.0 (310 950 строк кода).
- 9 мая 1996 — Выбран символ Linux — пингвин Tux.
- 9 июня 1996 — Linux версии 2.0.0 (777 956 строк кода).
- 25 января 1999 — Linux версии 2.2.0, изначально довольно недоработанный (1 800 847 строк кода).
- 4 января 2001 — Linux версии 2.4.0 (3 377 902 строк кода).
- 18 декабря 2003 — Linux версии 2.6.0 (5 929 913 строк кода).
- 23 марта 2009 — Linux версии 2.6.29, временный символ Linux — тасманский дьявол Tuz (11 010 647 строк кода).
- 22 июля 2011 — релиз Linux 3.0.
- 24 октября 2011 — релиз Linux 3.1.
- 15 января 2012 — релиз Linux 3.3 преодолел отметку в 15 млн строк кода.
- 23 февраля 2015 — первый релиз-кандидат Linux 4.0. изменения касаются, в основном, исправлений драйверов, обновлений для архитектур nios2, x86.
- Версия 4.1.
- Версия 4.2. — Используется в Ubuntu 15.10
- Версия 4.3.
- Версия 4.4. — Используется в Ubuntu 16.04
- Версия 4.5.
- Версия 4.6.

**Версии**

Торвальдс продолжает выпускать новые версии ядра, объединяя изменения, вносимые другими программистами, и внося свои. Оно обычно называется «ванильным» (vanilla), то есть официальное ядро без каких-либо сторонних изменений. В дополнение к официальным версиям ядра существуют альтернативные ветки, которые могут быть взяты из различных источников. Как правило, разработчики дистрибутивов Linux поддерживают свои собственные версии ядра, например, включая в них драйверы устройств, которые ещё не включены в официальную версию. С 30 мая 2011 изменена политика нумерации версий ядра.

**Нумерация версий до 30 мая 2011**

Номер версии ядра Linux до 30 мая 2011 содержал четыре числа, согласно недавнему изменению в политике именования версий, схема которой долгое время была основана на трёх числах. Для иллюстрации допустим, что номер версии составлен таким образом: A.B.C[.D] (например, 2.2.1, 2.4.13 или 2.6.12.3).

**Число A** обозначает версию ядра. Изначально задумывалось, что оно будет изменяться редко и только тогда, когда вносятся значительные изменения в код и концепцию ядра, первые такие изменения произошли в 1994 году (версия 1.0) и в 1996 году (версия 2.0). Впоследствии правило значительности изменений было нарушено, и дважды очередные версии ядра выходили с обновлённой первой цифрой — 31 мая 2011 года (3.0.0) и 21 апреля 2015 (4.0.0).

**Число B** обозначает старшую версию ревизии ядра. В ядрах до версии 3.0 Чётные числа обозначали стабильные ревизии, то есть те, которые предназначены для продуктивного использования, такие, как 1.2, 2.4 или 2.6, а нечётные — ревизии для разработчиков, предназначенные для того, чтобы тестировать новые улучшения и драйвера до тех пор, пока они не станут достаточно стабильными для того, чтобы включить их в стабильный выпуск.

**Число C** обозначает младшую версию ревизии ядра. В старой трёхчисловой схеме нумерации оно изменялось тогда, когда в ядро

включались заплатки, связанные с безопасностью, исправления ошибок, новые улучшения или драйверы. С новой политикой нумерации, однако, оно изменяется только тогда, когда вносятся новые драйверы или улучшения; небольшие исправления поддерживаются числом D.

**Число D** впервые появилось после случая, когда в коде ядра версии 2.6.8 была обнаружена грубая, требующая незамедлительного исправления ошибка, связанная с NFS. Однако других изменений было недостаточно для того, чтобы это послужило причиной для выпуска новой младшей ревизии (которой должна была стать 2.6.9). Поэтому была выпущена версия 2.6.8.1 с единственным изменением в виде исправления этой ошибки. С ядра 2.6.11 эта нумерация была адаптирована в качестве новой официальной политики версий. Исправления ошибок и заплатки безопасности теперь обозначаются с помощью четвёртого числа, тогда как большие изменения отражаются в увеличении младшей версии ревизии ядра (число C).

**Нумерация версий с 30 мая 2011**

30 мая 2011 Линус Торвальдс выпустил ядро версии 3.0-rc1. Вместе с ним изменена политика нумерации версий ядра. Отменено использование чётных и нечётных номеров для обозначения стабильности ядра, а третье число означает стабильность ядра. Версия 3.0 практически не несёт никаких изменений, кроме изменения политики нумерации ядра. Таким образом, стабильные версии ядра 3.0 будут именоваться 3.0.X, а следующий после этого релиз будет иметь номер 3.1.

## Архитектура

Ядро Linux поддерживает многозадачность, виртуальную память, динамические библиотеки, отложенную загрузку, производительную систему управления памятью и многие сетевые протоколы.

На сегодняшний день Linux — монолитное ядро с поддержкой загружаемых модулей. Драйверы устройств и расширения ядра обычно запускаются в нулевом кольце защиты, с полным доступом к

оборудованию. В отличие от обычных монолитных ядер, драйверы устройств легко собираются в виде модулей и загружаются или выгружаются во время работы системы.

То, что архитектура Linux не является микроядерной, вызвало обширнейшие прения между Торвальдсом и Эндрю Таненбаумом в конференции по Minix в 1992 году.

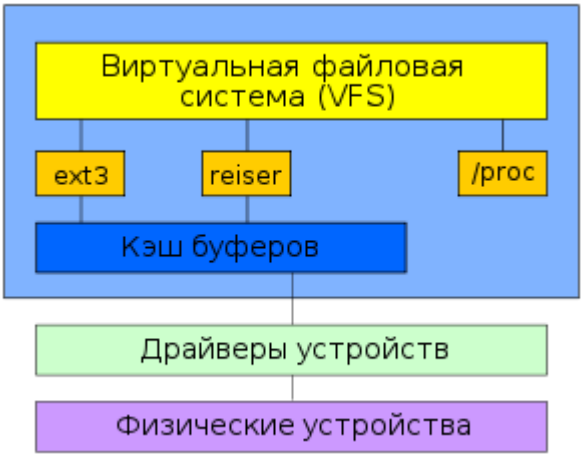


**Рис. 2**. Обобщённая структура файловой системы

**Лицензия**

Linux распространяется на условиях лицензии GNU General Public License, то есть свободно. Эту лицензию выбрал Линус Торвальдс практически сразу после того, как стало понятно, что его хобби начало получать распространение по всему миру. Владельцем торговой марки Linux является Линус, а помогает следить за соблюдением его прав и условий GPL Фонд свободного программного обеспечения.

**Подготовка каталогов с исходными кодами ядра**

После успешно завершенного этапа 1 необходимо подготовить дерево каталогов для файлов исходных кодов ядра.

Синтаксис команды для восстановления дерева зависит от формата скаченного файла. Это могут быть файлы linux-4.X.X.tar.gz и linux-4.X.X.tar.bz2. Для каждого из файлов используется определенный набор команд.

Например, для linux-4.X.X.tar.xz:

```
$ unxz linux-4.X.X.tar.xz
$ tar –xvf linux-4.X.X.tar
```

При выполнении этих команд содержимое файлов развернется в каталог, определенный связью linux. После этого командой cd перейдите в каталог linux. Этот каталог называется каталогом верхнего уровня исходного древа. (linux-X)


**Рис. 3**. Пользователь находится в каталоге linux-X

В нем много каталогов, одним из которых является Documentation, в котором хранится дополнительная информация по ядру.

**Конфигурирование ядра**

На этом этапе придется выбрать опции, которые будут использоваться в новом ядре. Не обязательно вникать в подробности многих опций. В большинстве из них можно воспользоваться настройками по умолчанию.

Существуют три метода создания файла конфигурации, используемого при сборке нового ядра:

1. make config – это наиболее простой пошаговый сценарий

2. make menuconfig – это более удобный метод (использовать если xconfig не запускается)
3. make xconfig – это графическая утилита для настройки ядра. Перед тем, как ей воспользоваться необходимо перейти в среду X Window.

Для каждой из представленных опций есть 3 установочных параметра: y,m,n.

- y(yes) - Включает или встраивает опцию в ядро.
- m(module) - Создает для выбранной опции загружаемый в динамическом режиме модуль (без reboot'a). Существует не для всех опций.
- n(no) - Отключает поддержку опции.

Для использования конфигуратора на базе X в системе должны быть установлены библиотеки TCL/TK.

Находясь в каталоге, где распаковали архив, вводим команду:

$ make menuconfig

Откроется утилита с интерфейсом ncurses.

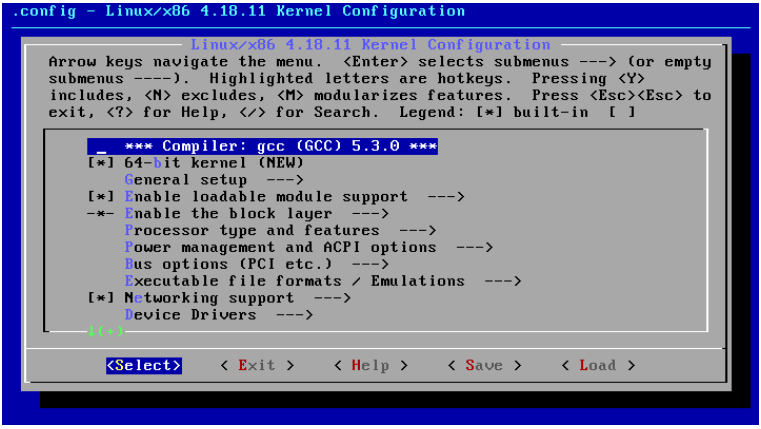


**Рис. 4.** Конфигурация ядра

Как видите, некоторые обязательные опции уже включены, чтобы облегчить вам процесс настройки. Начнем с самых основных настроек. Чтобы включить параметр нажмите y, чтобы включить модулем — m, для перемещения используйте клавиши стрелок и Enter, возвратиться на уровень вверх можно кнопкой Exit Откройте пункт **General Setup**.

Здесь устанавливаем такие параметры:

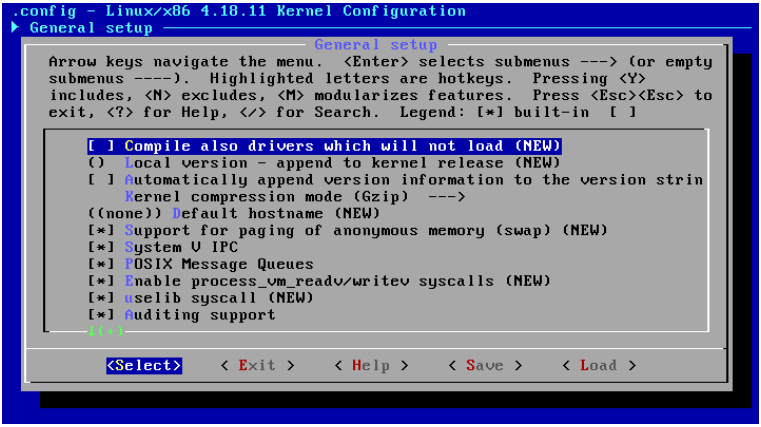


**Рис. 5**. Пункт основных настроек

- Local Version — локальная версия ядра, будет увеличиваться при каждой сборке на единицу, чтобы новые ядра при установке не заменяли собой старые, устанавливаем значение 1.
- Automatically append version information to the version string — добавлять версию в название файла ядра.
- Kernel Compression Mode — режим сжатия образа ядра, самый эффективный lzma.
- Default Hostname — имя компьютера, отображаемое в приглашении ввода
- POSIX Message Queues — поддержка очередей POSTIX
- Support for paging of anonymous memory — включаем поддержку swap

- Control Group support — поддержка механизма распределения ресурсов между группами процессов
- Kernel .config support и Enable access to .config through /proc/config.gz— включаем возможность извлечь конфигурацию ядра через /proc/config.gz

Здесь все, возвращаемся на уровень вверх и включаем Enable loadable module support, эта функция разрешает загрузку внешних модулей, дальше открываем его меню и включаем:

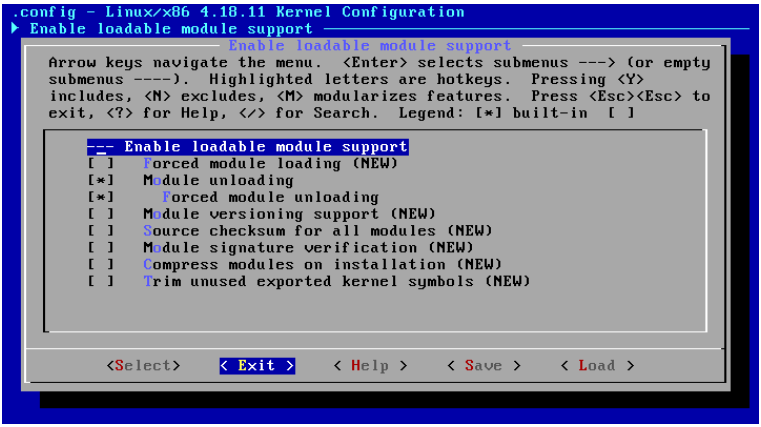


**Рис. 6**. Настройка загрузки внешних модулей

- Module unloading — поддержка отключения модулей
- Forced module unloading — принудительное отключение модулей

Возвращаемся назад и открываем **Processor type and features**:

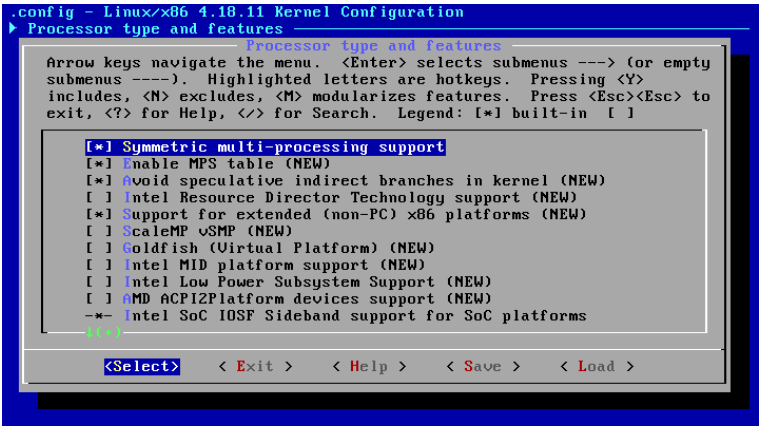


**Рис. 7**. Настройка типа процессора

В пункте Processor family (Opteron/Athlon64/Hammer/K8) — можно выбирать свой тип процессора. Оставим все как есть.

Возвращаемся и переходим в раздел **File systems**, тут установите все нужные галочки.

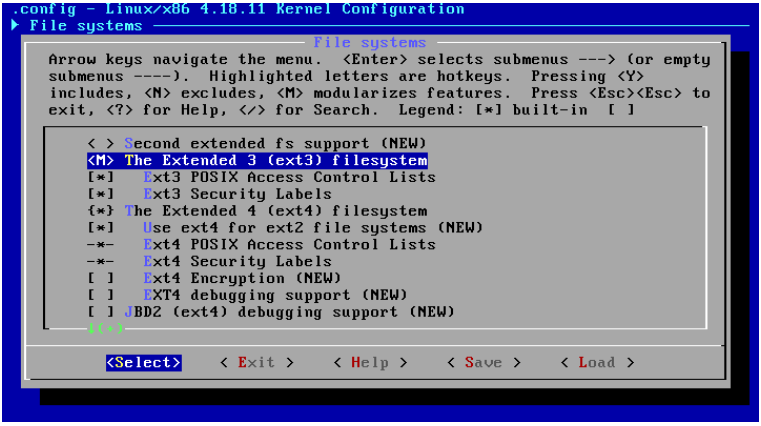


**Рис. 8**. Настройка файловых систем

Обязательно включите The Extended 3 (ext3) filesystem и The Extended 4 (ext4) filesystem — для поддержки стандартных ext3 и ext4 файловых систем.

Возвращаемся и идем в **Kernel hacking**.

Здесь включаем Magic SysRq key — поддержка магических функций SysRq, вещь не первой необходимости, но временами полезная.

Остался еще один пункт, самый сложный, потому что вам его придется пройти самому. Device Drivers — нужно пройтись по разделам и повключать драйвера для своего оборудования. Под оборудованием я подразумеваю нестандартные жесткие диски, мышки, USB устройства, веб-камеры, Bluetooth, WIFI адаптеры, принтеры и т д. Можно оставить все как есть.

Посмотреть какое оборудование подключено к вашей системе можно командой:

```
$ lspci
```

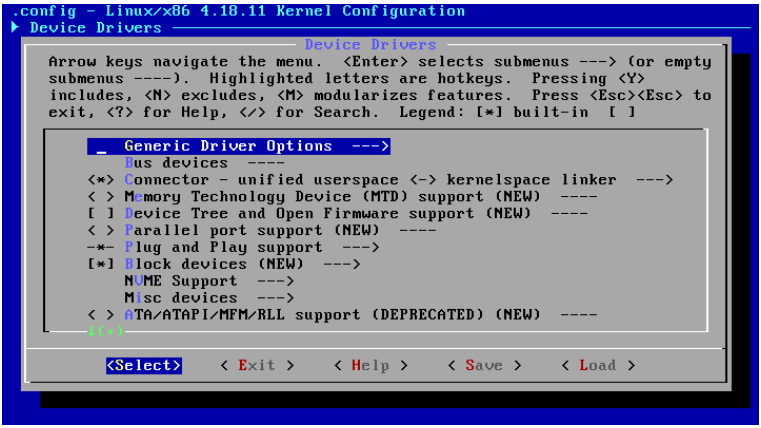


**Рис. 9**. Драйверы устройств

После выполнения всех действий ядро готово к сборке, но вам, скорее всего, предстоит разобраться с очень многим.

Чтобы сохранить настройки переместите указатель с помощью стрелок вправо-влево, з позиции Select в позицию Save и нажмите Enter, потом еще раз подтвердите сохранение:

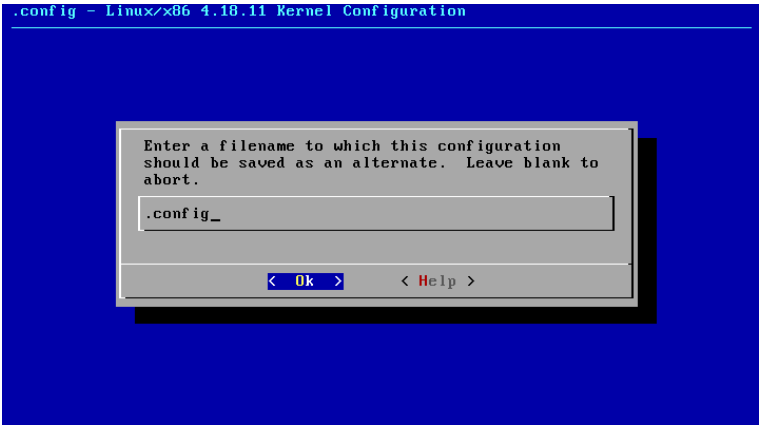


**Рис. 10**. Сохранение настроек

Чтобы выйти нажмите пару раз кнопку Exit. После настройки ядра переходите к 4 пункту.

**Компиляция ядра и установка модулей**

В свою очередь этот этап делится на шаги:

1. Подготовка
   $ make clean
2. Непосредственно сборка ядра
   $ make bzImage|bzdisk|bzlilo (просто ввести make bzImage)
3. Сборка и установка модулей
   $ make modules
   $ make modules_install

Первый из них - make clean - является подготовкой. Команда make clean удалит все лишние (вспомогательные) файлы, созданные от предыдущих процессов компиляции.

Далее идет шаг, при котором необходимо непосредственно собрать ядро. Для сборки ядра придется выбрать одну из 3-х команд: make bzImage, make bzdisk или make bzlilo.

Каждая из команд выполняет фактически одну и ту же операцию, только две последние выполняют одно дополнительное действие.

Рассмотрим подробнее каждую из команд:

**make bzImage** - стандартная операция, при которой будет только скомпилировано ядро. Если все прошло без проблем, то созданное в результате компиляции ядро будет расположено в каталоге /usr/src/linux/arch/i386/boot. В этом случае ядру присваивается имя bzImage. Диспетчер загрузки lilo|grub должен найти это ядро и загрузить его. Для этого достаточно скопировать файл bzImage и выполнить команду lilo для переустановки диспетчера загрузки.

**make bzdisk** - этот метод позволяет выполнить практически ту же задачу, что и bzImage, но после завершения компиляции будет автоматически выполнено копирование нового ядра на дискету. В дальнейшем эту дискету можно будет использовать для загрузки системы.

**make bzlilo** - этот метод формирования и инсталляции нового ядра, требующий предварительной подготовки lilo. При использовании этого метода map-файл ядра не перемещается в другой каталог. Более того новое ядро может быть записано поверх уже существующего, причем записано с ошибками, поэтому его использование не рекомендуется. Этот метод очень похож на bzImage и отличается только наличием дополнительной операцией, которая выполняется после совершения компиляции ядра. После компиляции ядра происходит копирование файлов созданного ядра в каталог / в качестве vmlinuz (при этом сохраняется резервная копия файла vmlinuz), затем выполняется команда lilo, в результате чего происходит переустановка диспетчера загрузки (и распознавание нового ядра).

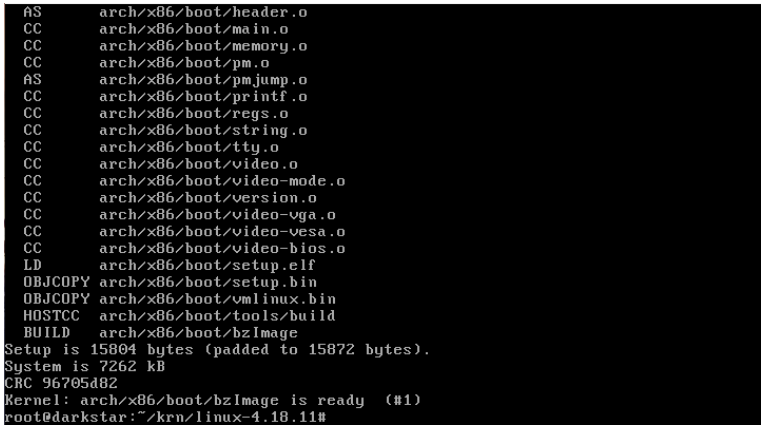


**Рис. 11**. Выполнение команды make bzImage

Третим шагом является сборка и установка модулей ядра.

Этот процесс выполняется с помощью 2-х команд make modules и make modules_install. Название команды make modules говорит само за себя: при выполнении этой команды происходит сборка модулей, которые соответствуют ядру, созданному на предыдущем этапе. Команда make modules_install, в свою очередь, перемещает созданные модули из исходного дерева ядра в каталог /lib/modules/<kernel-version>/kernel/<module-type>. В качестве типа модуля (<module-type>) используется имя категории, к которой относятся созданные модули (Например: block, misk, net, pcmcia, etc...).

**Перемещение ядра**

После завершения процесса сборки необходимо инсталлировать как ядро, так и его map-файл в каталог, где они будут постоянно находиться (/boot). Скопируйте (или переместите) файл System.map в каталог /boot, добавив номер версии ядра в конец строки:

$ cp System.map /boot/System.map-2.X.X

Затем точно так же скопируйте (или переместите) и само ядро:

$ cp arch/i386/boot/bzImage /boot/bzImage-2.X.X

Когда ядро и map-файл будут на своем новом месте останется выполнить только два коротких действия, после которых можно перезагрузиться уже с новым ядром (или со старым, если возникнут проблемы.

**Настройка и запуск lilo**

Диспетчер загрузки lilo дает возможность определить несколько образов загрузки. Вставив новый раздел образа в файл /etc/lilo.conf можно получить дополнительный образ загрузки. Для этого необходимо продублировать последние 4 строки файла и изменить адрес до загружаемого ядра. старый lilo.conf, последние 4 строки)

image = /boot/<Уже стоящее в системе ядро>
lable = linux
root = /dev/hda1
read-only

Модифицированный **lilo.conf**

image = /boot/<Уже стоящее в системе ядро>
lable = linux.orig
root = /dev/hda1
read-only
image = /boot/<Новое ядро>
lable = linux
root = /dev/hda1
read-only
append = "debug=2 noapic nosmp"

Обратите внимание на lable = linux.orig (это метка старого ядра). После конфигурации файла /etc/lilo.conf следует переустановить

диспетчер начальной загрузки системы. Для этого выполни следующие команды:

```
$ lilo
```


**Рис. 12**. Выбор ядра для загрузки

Если все прошло без ошибок, то можно перезагрузить систему. В случае, если новое ядро является причиной некорректной работы системы во время ее загрузки, то ты можешь воспользоваться исходным ядром. Для загрузки системы необходимо ввести имя-метку желаемого образа загрузки. Чтобы вывести на экран перечень доступных меток следует нажать на клавишу <Tab>.

## ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Научиться конфигурировать, компилировать и запускать ядро Limux. Выполнить следующие действия:

1. Подготовка исходного кода ядра. Необходимо заранее подготовить исходники. Скачать ядро Linux можно с официального сайта (https://www.kernel.org)
2. Подготовка каталогов с исходными кодами ядра. Необходимо распаковать архив с исходными кодами ядра. (Команды зависят от типа архива)
3. Конфигурирование ядра. Необходимо указать какие компоненты включить в ядро, а какие убрать (можно оставить стандартные параметры).
4. Компиляция ядра и установка модулей. Необходимо выполнить несколько шагов:
   - Подготовка
   - Непосредственно сборка ядра
   - Сборка и установка модулей
5. Перемещение ядра. Необходимо инсталлировать как ядро, так и его map-файл в каталог, где они будут постоянно находиться.
6. Настройка и запуск lilo. Необходимо настроить загрузчик так, чтобы была возможность выбрать ядро для загрузки (старое и новое).

## КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Раскройте понятие ядра операционной системы.
2. Опишите архитектуру операционной системы Linux.
3. Перечислите основные этапы сборки ядра.
4. Назовите актуальную версию ядра Linux на данный момент.
5. Опишите принцип нумерации версий ядра Linux.
6. Опишите по какой лицензии распространяется ОС Linux.
7. Опишите шаг приобретение исходных кодов ядра.
8. Опишите шаг подготовка каталогов с исходниками ядра.
9. Опишите шаг конфигурирование ядра.
10. Опишите шаг компиляция ядра и установка модулей.
11. Опишите шаг перемещение ядра.
12. Опишите шаг настройка и запуск lilo.
13. Приведите пример команды для конфигурирования ядра
14. Приведите пример команд для компиляции ядра и установки модулей
15. Приведите пример модифицированного файла lilo.conf с возможностью выбора ядра для запуска

## ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 3 занятия (6 академических часов: 5 часов на выполнение и сдачу лабораторной работы и 1 час на подготовку отчета).

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания, ответы на контрольные вопросы, описание процесса выполнения лабораторной работы, выводы.

## ОСНОВНАЯ ЛИТЕРАТУРА

1. Вирт, Н. Разработка операционной системы и компилятора. Проект Оберон [Электронный ресурс] / Н. Вирт, Ю. Гуткнехт ; пер.с англ. Борисов Е.В., Чернышов Л.Н.. — Электрон. дан. — Москва : ДМК Пресс, 2012. — 560 с. — Режим доступа: https://e.lanbook.com/book/39992

## ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

2. Крищенко, В.А. Сервисы Windows [Электронный ресурс] : учебное пособие / В.А. Крищенко, Н.Ю. Рязанова. — Электрон. дан. — Москва : МГТУ им. Н.Э. Баумана, 2011. — 47 с. — Режим доступа: https://e.lanbook.com/book/52416..

3. Войтов, Н.М. Администрирование ОС Red Hat Enterprise Linux. Учебный курс [Электронный ресурс] : учебное пособие / Н.М. Войтов. — Электрон. дан. — Москва : ДМК Пресс, 2011. — 192 с. — Режим доступа: https://e.lanbook.com/book/1081

4. Стащук, П.В. Администрирование и безопасность рабочих станций под управлением Mandriva Linux: лабораторный практикум [Электронный ресурс] : учебно-методическое пособие / П.В. Стащук. — Электрон. дан. — Москва : ФЛИНТА, 2015. — 182 с. — Режим доступа: https://e.lanbook.com/book/70397

**Электронные ресурсы:**

5. Научная электронная библиотека http://eLIBRARY.RU
6. Электронно-библиотечная система http://e.lanbook.com
7. Losst - Linux Open Source Software Technologies https://losst.ru
8. The Linux Kernel Archives https://www.kernel.org