Лабораторная работа №2

Цель работы:

Приобрести навыки установки операционной системы Linux.

Задачи:

1. Закрепить знания о работе с программой VirtualBox.
2. Создать виртуальную машину исходя из предоставленной информации о минимальных аппаратных требованиях, предлагаемых к установке и изучению операционной системы Linux (ОС).
3. Установить ОС Linux на виртуальный компьютер. Разобрать процесс установки ОС на этапы.

Теоретическая часть:

Хронология выхода основных версий ядра Linux:
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.
Апрель 1992 — Linux версии 0.96, на котором стало возможно запустить графический сервер X Window System.
Весь 1993 и начало 1994 — 15 тестовых релизов 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 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 (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.

Linux (Ли́нукс) — семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты. Как и ядро Linux, системы на его основе как правило создаются и распространяются в соответствии с моделью разработки свободного и открытого программного обеспечения. Linux-системы распространяются в основном бесплатно в виде различных дистрибутивов — в форме, готовой для установки и удобной для сопровождения и обновлений, — и имеющих свой набор системных и прикладных компонентов, как свободных, так возможно и собственнических.

Появившись как решения вокруг созданного в начале 1990-х годов ядра, уже с начала 2000-х годов системы Linux являются основными для суперкомпьютеров и серверов, расширяется применение их для встраиваемых систем и мобильных устройств, некоторое распространение системы получили и для персональных компьютеров.

Традиционно системами Linux считаются только те, которые включают в качестве компонентов основные программы проекта GNU, такие как bash, gcc, glibc, coreutils, GNOME и ряд других, в связи с чем часто всё семейство иногда идентифицируется как GNU/Linux, притом существует спор об именовании GNU/Linux. Существует проект стандартизации внутренней структуры Linux-систем — Linux Standard Base, часть из документов которого зарегистрировано в качестве стандартов ISO; но далеко не все системы сертифицируются по нему, и в целом для Linux-систем не существует какой-либо общепризнанной стандартной комплектации или формальных условий включения в семейство. Однако есть ряд систем на базе ядра Linux, но не имеющих в основе зависимости от программ GNU, которые к Linux-семейству традиционно не относят, в частности таковы мобильные системы Android и FirefoxOS.

Linux-системы реализуются на модульных принципах, стандартах и соглашениях, заложенных в Unix в течение 1970-х и 1980-х годов. Такая система использует монолитное ядро, которое управляет процессами, сетевыми функциями, периферией и доступом к файловой системе. Драйверы устройств либо интегрированы непосредственно в ядро, либо добавлены в виде модулей, загружаемых во время работы системы.

Отдельные программы, взаимодействуя с ядром, обеспечивают функции системы более высокого уровня. Например, пользовательские компоненты GNU являются важной частью большинства Линукс-систем, включающей в себя наиболее распространённые

реализации библиотеки языка Си, популярных оболочек операционной системы, и многих других общих инструментов Unix, которые выполняют многие основные задачи операционной системы.

Графический интерфейс пользователя (или GUI) в большинстве систем Linux построен на основе X Window System.

В Linux-системах пользователи работают через интерфейс командной строки (CLI), графический интерфейс пользователя (GUI), или, в случае встраиваемых систем, через элементы управления соответствующих аппаратных средств. Настольные системы, как правило, имеют графический пользовательский интерфейс, в котором командная строка доступна через окно эмулятора терминала или в отдельной виртуальной консоли. Большинство низкоуровневых компонентов Линукс, включая пользовательские компоненты GNU, используют исключительно командную строку. Командная строка особенно хорошо подходит для автоматизации повторяющихся или отложенных задач, а также предоставляет очень простой механизм межпроцессного взаимодействия. Программа графического эмулятора терминала часто используется для доступа к командной строке с рабочего стола Linux.

Дистрибутивы, специально разработанные для серверов, могут использовать командную строку в качестве единственного интерфейса. На настольных системах наибольшей популярностью пользуются пользовательские интерфейсы, основанные на таких средах рабочего стола как KDE Plasma Desktop, GNOME и Xfce, хотя также существует целый ряд других пользовательских интерфейсов. Самые популярные пользовательские интерфейсы основаны на X Window System, которая предоставляет прозрачность сети и позволяет графическим приложениям, работающим на одном компьютере, отображаться на другом компьютере, на котором пользователь может взаимодействовать с ними.

FVWM, Enlightenment и Window Maker — простые менеджеры окон X Window System, которые предоставляют окружение рабочего стола с минимальной функциональностью. Оконный менеджер предоставляет средства для управления размещением и внешним видом отдельных окон приложений, а также взаимодействует с X Window System. Окружение рабочего стола включает в себя оконные менеджеры, как часть стандартной установки: Mutter для GNOME с 2011 года, KWin для KDE с 2000 года, Xfwm для Xfce с 1998 года, хотя пользователь при желании может выбрать другой менеджер окон/

Нумерация версий

Номер версии ядра Linux в настоящее время содержит четыре числа, следуя недавнему изменению в долго используемой до этого политике схемы версий, основанной 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. Оно изменяется менее часто и только тогда, когда вносятся значительные изменения в код и концепцию ядра. Оно изменялось дважды в истории ядра: в 1994 (версия 1.0) и в 1996 (версия 2.0).

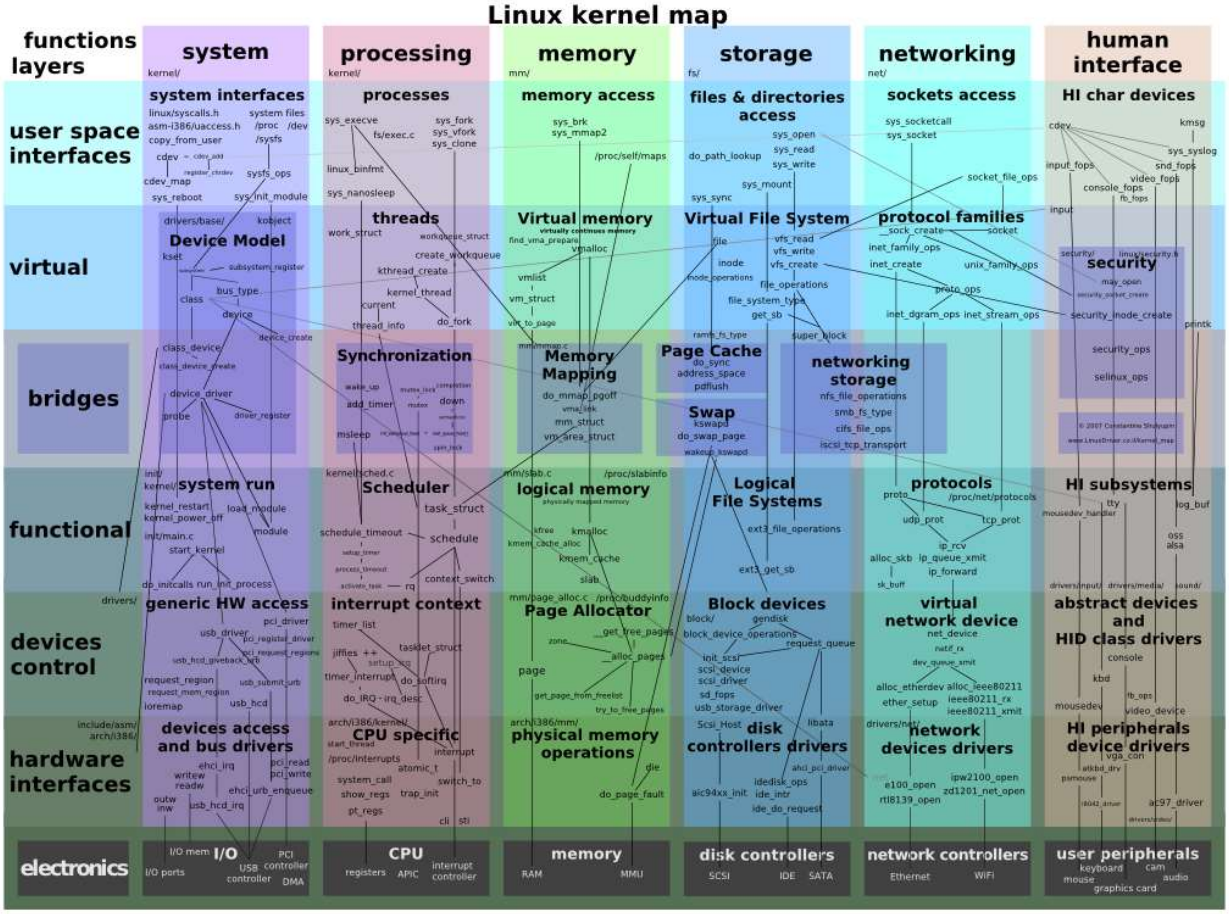
Число B обозначает старшую версию ревизии ядра. Чётные числа обозначают стабильные ревизии, то есть те, которые предназначены для промышленного использования, такие как 1.2, 2.4 или 2.6. Нечётные числа обозначают ревизии для разработчиков, такие как 1.1 или 2.5. Они предназначены для тестирования новых улучшений и драйверов до тех пор, пока они не станут достаточно стабильными для того, чтобы быть включёнными 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, было недостаточно других изменений, 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 исправлением в виде исправления для 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 выполняются в изменениях младшей версии ревизии ядра (число C).

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



Процесс загрузки

При загрузке компьютера происходит последовательная передача управления от системной прошивки компьютера (BIOS или UEFI) к загрузчику, а от него — к ядру. Затем ядро запускает планировщик (для реализации многозадачности) и выполняет программу init (которая настраивает пользовательское окружение и позволяет осуществлять взаимодействие с пользователем и вход в систему), после чего ядро переходит в состояние бездействия до тех пор, пока не получит внешний вызов.

Основные этапы загрузки:

1. Системная прошивка компьютера выполняет первичную проверку и инициализацию аппаратного обеспечения.
2. В случае BIOS прошивка загружает в оперативную память и выполняет загрузочный код с одного из разделов заданного загрузочного устройства, который содержит *фазу 1* загрузчика Linux. Фаза 1 загружает *фазу 2* (значительный по размеру код загрузчика). Некоторые загрузчики могут использовать для этого промежуточный этап (под названием *фаза 1,5*), поскольку современные диски большого объёма могут некорректно считываться без дальнейшего кода. В случае UEFI запускается загрузчик загруженный со служебного раздела (EFS), который

выбирается согласно настройкам приоритета загрузки определенного в энергонезависимой памяти компьютера. При этом возможна загрузка не только специализированного загрузчика, но можно загрузить и непосредственно ядро Linux (для этого ядро должно быть собрано с опцией EFI_STUB).

3. Загрузчик зачастую предлагает пользователю меню с доступными вариантами загрузки. После выбора или после заданного тайм-аута загрузчик загружает ядро.

4. Загруженное ядро распаковывается в памяти, настраивает системные функции, такие как работа необходимого оборудования и управление страницами памяти, после чего делает вызов start_kernel().

5. После этого start_kernel() выполняет основную настройку системы (прерывания, остальные функции управления памятью, инициализацию устройств, драйверов и т. д.), а потом порождает процесс бездействия, диспетчер и отдельно от них — процесс init (выполняющийся в пользовательском пространстве).

6. Планировщик начинает более эффективно управлять системой, в то время как ядро переходит к бездействию.

7. Процесс init выполняет необходимые сценарии, которые настраивают все службы и структуры, не относящиеся к уровню ядра, в результате чего будет создано пользовательское окружение, и пользователю будет предоставлен экран входа в систему.

Когда происходит завершение работы, init вызывается для управляемого закрытия программ пользовательского уровня, тоже согласно сценариям. После этого init закрывается, а ядро производит своё собственное завершение работы.

Фаза загрузчика

При загрузке через BIOS: Фазы загрузчика различаются в зависимости от платформы. Поскольку ранние этапы загрузки не зависят от операционной системы, процесс загрузки обычно начинается следующим образом:

Для x86 или x86-64: код с загрузочного сектора раздела диска выполняется в реальном режиме и загружает первую фазу загрузчика (как правило — часть LILO или GRUB).

С этого момента загрузка продолжается. Первая фаза загружает остальной код загрузчика, который обычно спрашивает, какую операционную систему (или вид её сессии) пользователь хочет запустить. Код загрузчика создаётся на основе конфигурационного файла /etc/lilo.conf (для LILO), в котором определены доступные системы. Этот файл содержит, в частности, информацию о загрузочном разделе и расположении ядра для каждой из таких систем, а также дополнительные параметры загрузки, если они заданы. В

результате выбора соответствующее ядро загружается в ОЗУ, минимальная начальная файловая система настраивается из файла-образа (initrd), а затем, вместе с соответствующими параметрами управление передаётся новой ОС.

LILO и GRUB имеют определённые различия:

LILO не распознаёт файловые системы, поэтому он использует непосредственные (raw) смещения на диске и сведения из BIOS для загрузки данных. Он загружает код меню, а потом, в зависимости от выбора, загружает либо 512-байтные секторы диска для системы, основывающейся на MBR (например, Microsoft Windows), либо образ ядра для Linux.

GRUB, наоборот, распознаёт распространённые файловые системы (например, ext2 и ext3). Так как GRUB хранит свои данные в файле конфигурации, а не в загрузочной записи, и имеет интерфейс командной строки, то зачастую параметры GRUB легче поправить или изменить, если они настроены неправильно или повреждены.

При загрузке через UEFI: В UEFI загрузчик сразу запускается в защищенном режиме (32- или 64-битном) и фактически загружается сразу все фазы загрузчика (с учетом загрузки со служебного раздела для загрузчика нет необходимости разбивать себя на отдельные фазы и размещать их в разных местах). В остальном процесс загрузки и инициализации ядра не отличается от варианта с BIOS.

BIOS:

1. Загрузчик 1-й фазы считывается BIOS из MBR (главной загрузочной записи).
2. Он загружает оставшуюся часть загрузчика (2-ю фазу). Если вторая фаза находится на большом диске, иногда загружается промежуточная фаза 1,5, которая содержит дополнительный код, позволяющий считывать цилиндры с номерами более 1024 (диски LBA). Загрузчик фазы 1,5 хранится (если это необходимо) в MBR или в загрузочном разделе.
3. Выполняется вторая фаза загрузчика и отображает меню запуска GRUB. Оно также позволяет выбрать среду выполнения и просмотреть параметры системы.
4. Когда операционная система выбрана, она загружается и ей передаётся управление.

GRUB поддерживает и прямой, и цепной способ загрузки, а также LBA, ext2, и «истинно командно-ориентированную, дооперационную среду на машинах x86». Он имеет три интерфейса: меню выбора, редактор настроек и командную консоль.

UEFI:

1. Загруженный со служебного раздела EFS GRUB (специальная версия бинарного файла, который умеет загружать UEFI) содержит в себе все необходимые

компоненты для доступа к файловой системе /boot где находятся конфигурация и дополнительные файлы загрузчика.

2. Отображается меню загрузчика и отображает меню запуска GRUB. Оно также позволяет выбрать среду выполнения и просмотреть параметры системы.

3. Когда операционная система выбрана, она загружается и ей передаётся управление.

LILO

LILO старше GRUB и практически аналогичен ему в действии, за исключением того, что не содержит интерфейса командной строки. Поэтому все изменения нужно вносить в его настройки и записывать в MBR, после чего систему перезагружают. Таким образом, ошибка в настройках может сделать диск неспособным к загрузке без использования отдельного загрузочного устройства (дискеты и т. п.), содержащего программу для исправления ошибки. Кроме того, LILO не распознаёт файловые системы; вместо этого, адреса файлов-образов хранятся непосредственно в MBR, а BIOS используется для прямого к ним доступа.

Loadlin

Ещё один способ загрузить Linux — из DOS или Windows 9x, где ядро Linux полностью заменит выполняющуюся копию операционной системы. Это может быть целесообразно, если аппаратное обеспечение должно включаться программно, а соответствующие программы существуют только для DOS, а не для Linux, будучи проприетарным ПО производителя и объектом коммерческой тайны. Этот метод загрузки не особо актуален, так как в Linux есть драйверы для множества аппаратных устройств, хотя в прошлом он был весьма полезен. Другой пример: когда Linux находится на устройстве хранения данных, которое не предназначено для загрузки из BIOS: DOS или Windows могут загрузить соответствующие драйверы, чтобы обойти такое ограничение BIOS, а затем загрузить оттуда Linux.

Фаза ядра

Ядро Linux управляет главными функциями, такими как управление памятью, диспетчер задач, ввод-вывод, межпроцессное взаимодействие и общее управление системой. Загрузка проходит в два этапа: на первом ядро (в виде сжатого файла-образа) загружается в оперативную память и распаковывается, далее настраиваются такие базовые функции как основное управление памятью. Затем управление в последний раз передается основному процессу запуска ядра. Как только ядро становится полностью работоспособным (т. е. загруженным и выполнившим свой код), оно находит и запускает процесс init, который самостоятельно настраивает пользовательское пространство и

процессы, необходимые для функционирования пользовательского окружения и итогового входа в систему. Само ядро переходит в режим бездействия и готовности к вызовам со стороны других процессов.

Этап загрузки ядра

Ядро при загрузке обычно имеет вид файла-образа, сжатого в формат zImage или bzImage с помощью zlib. В нём содержится головная программа, которая проводит минимальную настройку оборудования, распаковывает образ целиком в верхнюю памятьи монтирует RAM-диск, если он предусмотрен. После этого она выполняет запуск ядра посредством ./arch/x86/boot/head и процесса startup_32() (для процессоров семейства x86).

Этап запуска ядра

Функция запуска ядра (также называемая *свопper* или *процесс 0*) организует управление памятью (таблицы страниц и страничную организацию памяти), определяет тип процессора и дополнительные возможности (например, наличие математического сопроцессора), а затем переключается к архитектурно-независимому функционалу ядра Linux путём вызова `start_kernel()`.

`start_kernel()` выполняет множество задач инициализации. Она настраивает обработчики прерываний (IRQ), затем настраивает память, запускает процесс init (первый процесс пользовательского режима), а затем запускает задачу бездействия вызовом `cpu_idle()`. Следует заметить, что процесс запуска ядра также монтирует инициирующий RAM-диск («initrd»), который ранее был загружен в роли временной корневой файловой системы в фазе загрузки. Это позволяет загружать модули драйверов, не опираясь на другие физические устройства и драйверы, и поддерживать небольшой размер ядра. Корневая файловая система впоследствии подменяется с помощью вызова `pivot_root()`, который размонтирует временную и заменяет её настоящей корневой ФС, как только последняя станет доступна. Использованная временной системой память затем освобождается.

Таким образом, ядро инициализирует устройства, монтирует указанную загрузчиком файловую систему в режиме «только чтение» и запускает процесс init (`/sbin/init`), который обозначается как первый процесс, запущенный системой (с идентификатором процесса PID = 1). Соответствующие сообщения выводит ядро (при монтировании файловой системы) и init (при запуске одноимённого процесса). Ядро также может выполнить initrd для обработки настроек и инициализации устройств до монтирования корневой файловой системы.

По заявлению компании «Red Hat», детали процесса загрузки на этом этапе можно подытожить так: «Когда загружается ядро, оно сразу же инициализирует и конфигурирует

память компьютера и настраивает различное подключённое к системе оборудование, включая все процессоры, подсистемы ввода-вывода и устройства хранения данных. Затем оно ищет сжатый образ initrd в заранее определённом участке памяти, распаковывает его, монтирует и загружает все необходимые драйверы. Затем оно инициализирует виртуальные устройства, связанные с файловой системой, например LVM или программные RAID-массивы, прежде чем демонтировать образ диска initrd и освободить всю память, ранее занимаемую образом. Потом ядро создает корневое устройство, монтирует корневой раздел только для чтения и освобождает всю неиспользованную память. К этому времени ядро загружено в память и работоспособно. Тем не менее, поскольку нет пользовательских программ для осуществления осмысленного ввода данных в систему, с ней мало что можно делать.»

Теперь, когда включены прерывания, диспетчер может принять общее управление системой, чтобы обеспечить вытесняющую многозадачность, а процесс init остается продолжать загрузку пользовательского окружения в пространстве пользователя.

Процесс init (только типа UNIX System V)

Init является родителем всех процессов. Его главная задача — создавать процессы по сценарию из файла `/etc/inittab`. В этом файле обычно содержатся записи, указывающие init породить getty для каждой линии, по которой пользователи могут входить в систему. Он также контролирует автономные процессы, требуемые какой-либо системе. Уровень выполнения — программная конфигурация системы, которая позволяет существовать только заданной группе процессов. Процессы, порождаемые init на каждом из таких уровней выполнения, определяются в файле `/etc/inittab`.

По сути init организует и поддерживает всё пользовательское пространство, что включает в себя также проверку и монтирование файловых систем, запуск нужных пользовательских служб и, переключение в пользовательскую среду, когда запуск системы завершится. Он похож на процессы init в Unix и BSD, от которых произошёл, но в некоторых случаях он изменён или переделан. В обычной системе Linux init имеет параметр, известный как уровень выполнения, принимающий значения от 1 до 6 и определяющий, какие подсистемы следует включить. Для каждого уровня выполнения есть собственные сценарии, которые регламентируют различные процессы, участвующие в установлении или снятии данного уровня, и именно эти сценарии считаются необходимыми для процесса загрузки. Сценарии init обычно хранятся в каталогах с именами вида `/etc/rc…`. Главный файл конфигурации уровней для init — `/etc/inittab`.

Во время загрузки системы он проверяет, описан ли уровень по умолчанию в `/etc/inittab`, а если же нет — запрашивает его через системную консоль. Затем он продолжает выполнять все соответствующие сценарии загрузки для этого уровня, включая загрузку модулей, проверку целостности файловой системы (которая монтировалась только для чтения), перемонтирование её для чтения-записи и настройку сети.[1]

В частности, по сообщению Red Hat, процесс init следует такой схеме:[2]

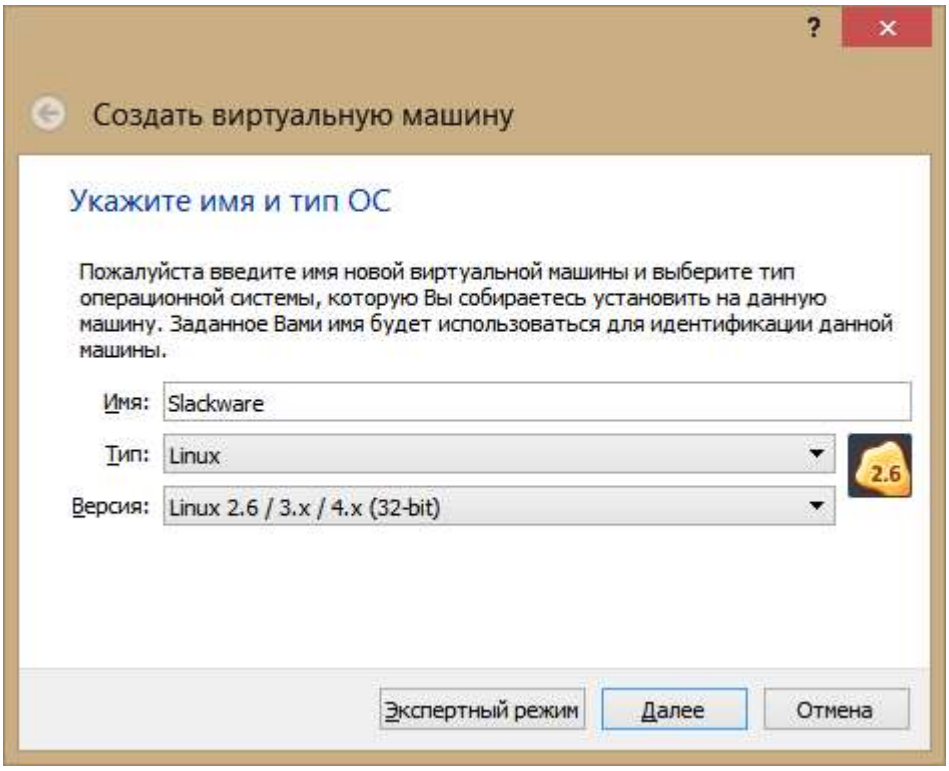
1. Он просматривает сценарий `sysinit`, который "устанавливает путь к среде, запускает `swap`, проверяет файловые системы и делает всё, что необходимо для инициализации системы. Это, в частности, системные и аппаратные часы, специальные процессы для последовательного порта и т. п.
2. Затем init просматривает конфигурацию, указанную для заданного уровня выполнения.
3. После этого init устанавливает исходную библиотеку функций для системы. Это определяет, как следует запустить или снять программу и как определить её PID.
4. Затем он запускает все предусмотренные процессы и создает сессию входа пользователя в систему.

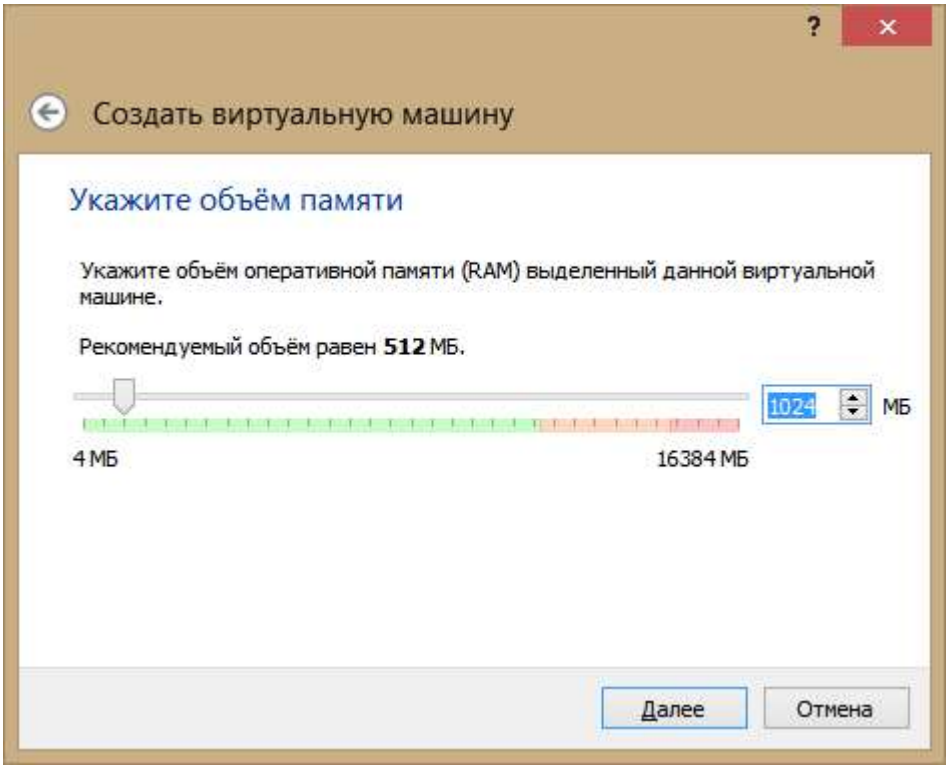
После того, как он породил все заданные процессы, init переходит в режим ожидания и ждет одного из трёх событий:

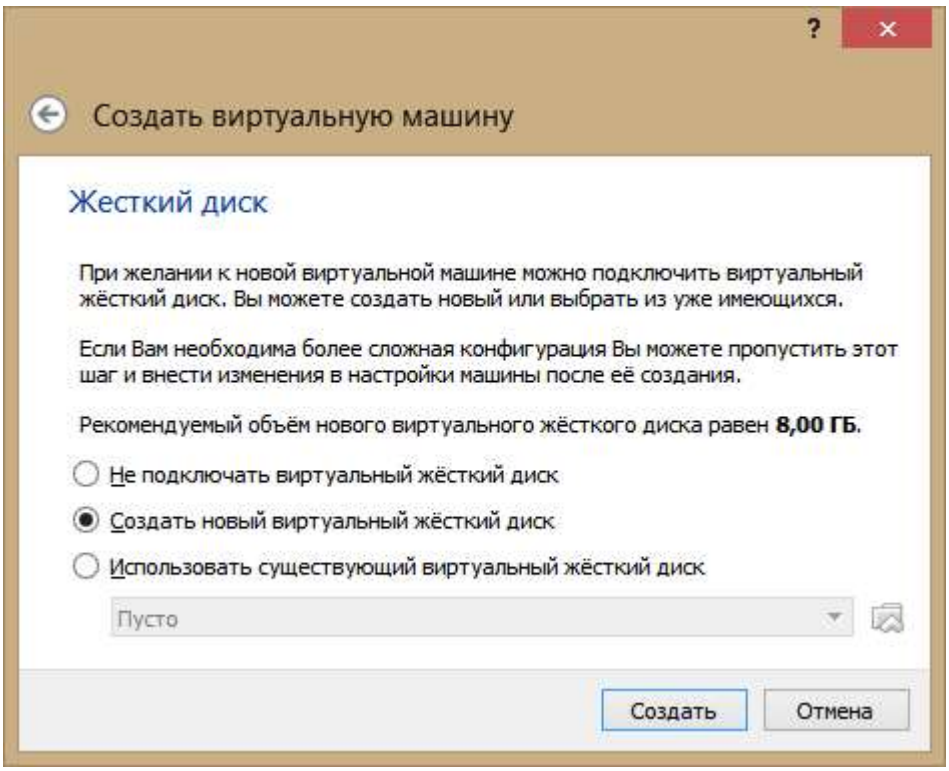
1. Нормального или аварийного завершения порождённых процессов.
2. Сигнала аварии питания.
3. Запроса от `/sbin/telinit` на изменение уровня выполнения.

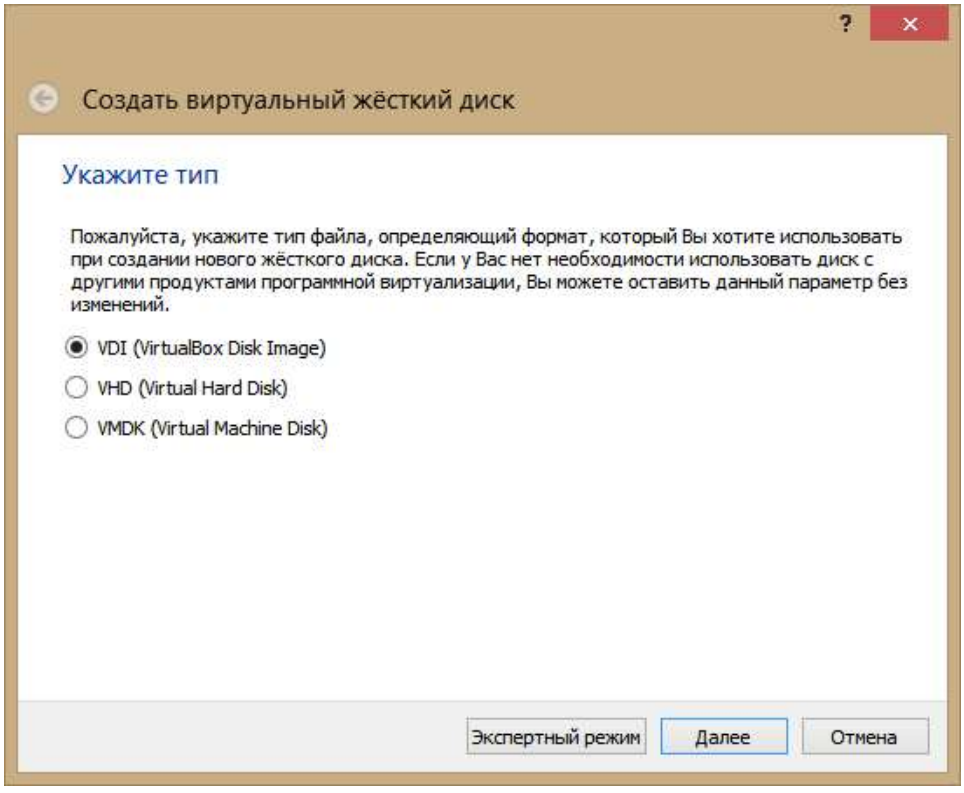
Это относится к программе init в стиле UNIX System V. Другие программы init могут вести себя иначе.

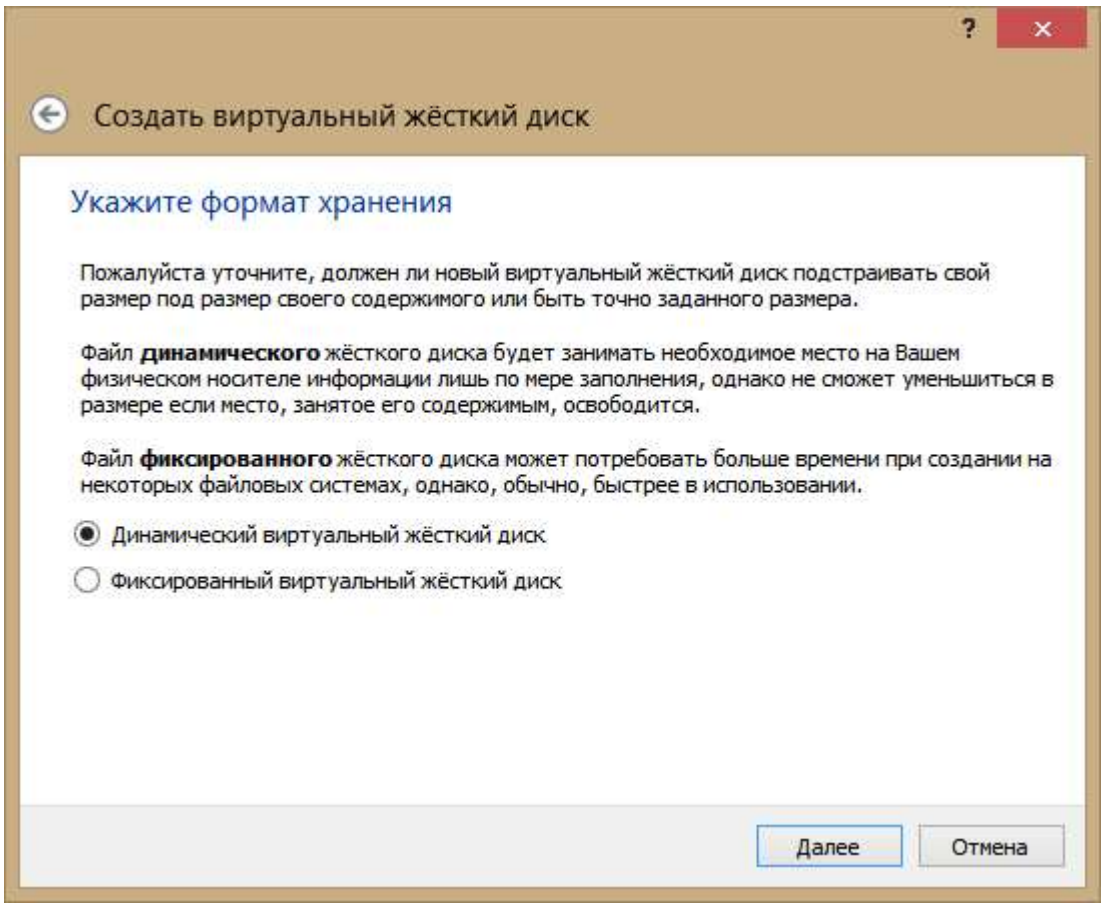
Практическая часть

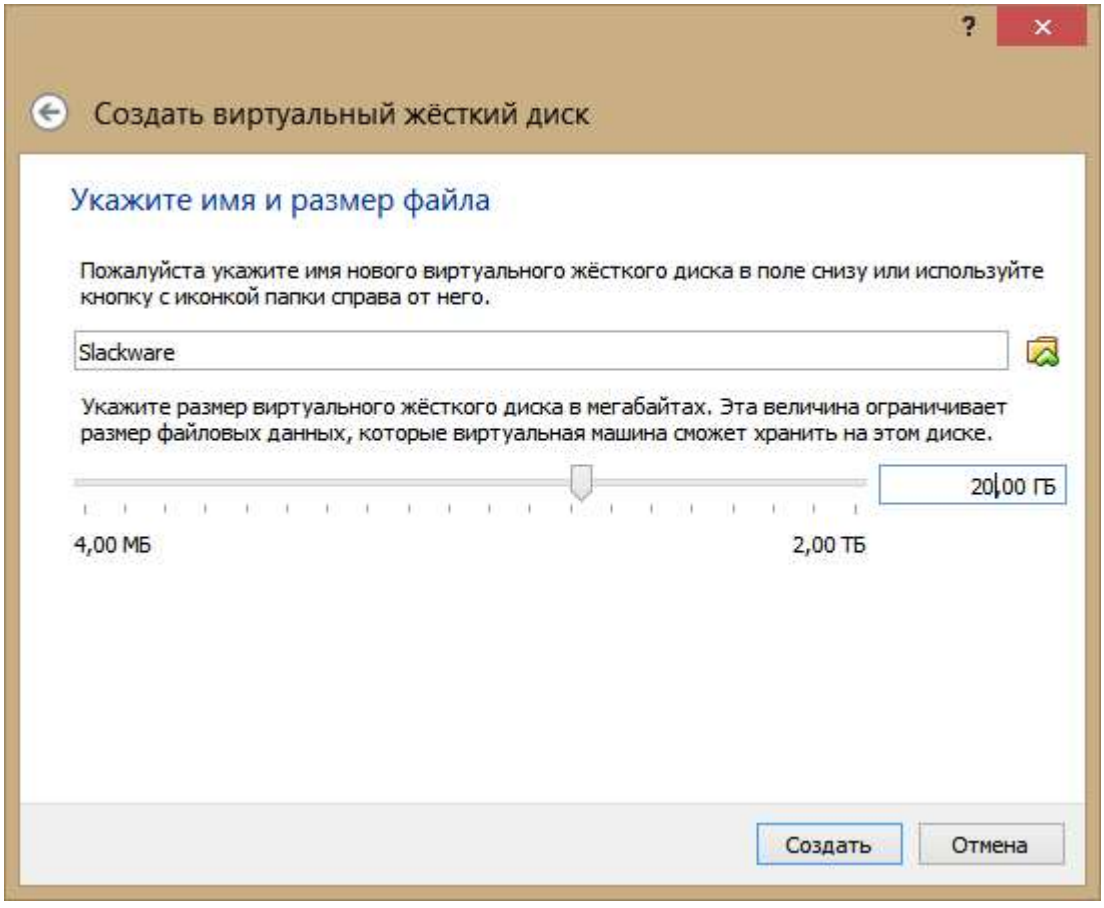


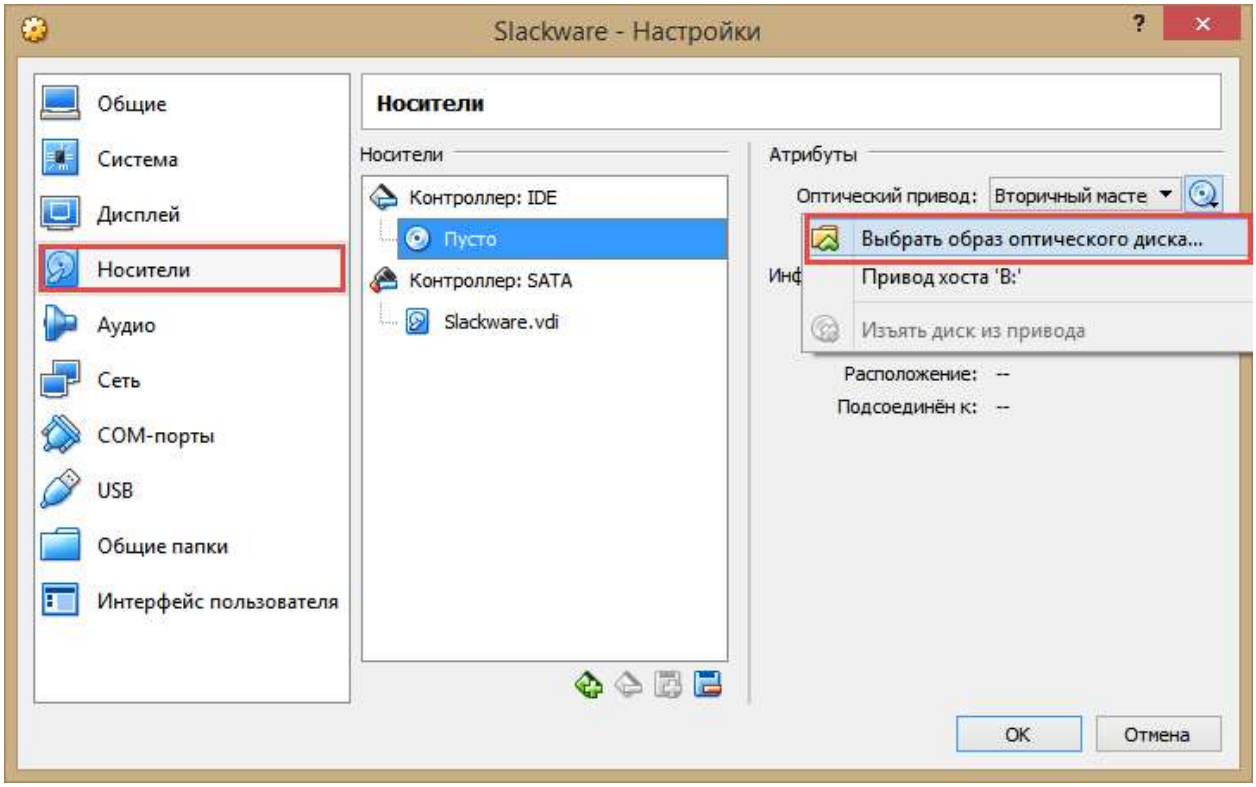



Создать виртуальную машину
Жесткий диск
При желании к новой виртуальной машине можно подключить виртуальный жёсткий диск. Вы можете создать новый или выбрать из уже имеющихся.
Если Вам необходима более сложная конфигурация Вы можете пропустить этот шаг и внести изменения в настройки машины после её создания.
Рекомендуемый объём нового виртуального жёсткого диска равен 8,00 ГБ.
Не подключать виртуальный жёсткий диск
Создать новый виртуальный жёсткий диск
Использовать существующий виртуальный жёсткий диск
Пусто
Создать
Отмена


Создать виртуальный жёсткий диск
Укажите тип
Пожалуйста, укажите тип файла, определяющий формат, который Вы хотите использовать при создании нового жёсткого диска. Если у Вас нет необходимости использовать диск с другими продуктами программной виртуализации, Вы можете оставить данный параметр без изменений.
VDI (VirtualBox Disk Image)
VHD (Virtual Hard Disk)
VMDK (Virtual Machine Disk)
Экспертный режим
Далее
Отмена

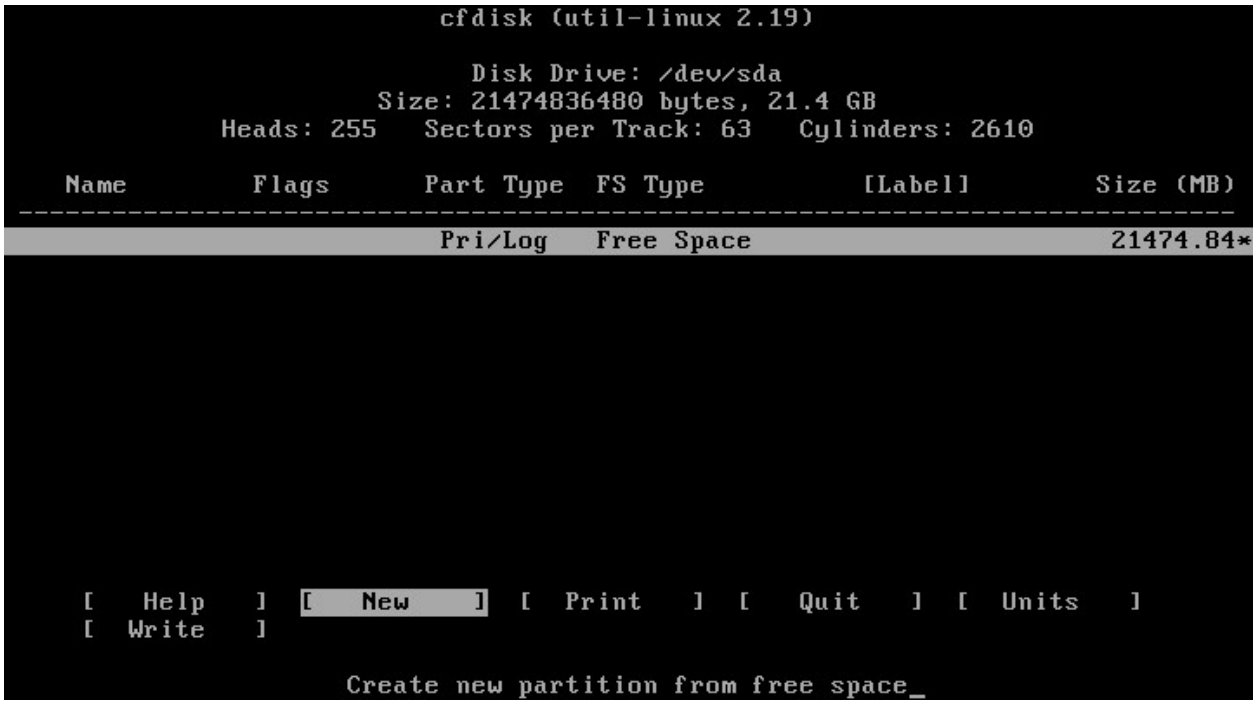

Создать виртуальный жёсткий диск
Укажите формат хранения
Пожалуйста уточните, должен ли новый виртуальный жёсткий диск подстраивать свой размер под размер своего содержимого или быть точно заданного размера.
Файл **динамического** жёсткого диска будет занимать необходимое место на Вашем физическом носителе информации лишь по мере заполнения, однако не сможет уменьшиться в размере если место, занятое его содержимым, освободится.
Файл **фиксированного** жёсткого диска может потребовать больше времени при создании на некоторых файловых системах, однако, обычно, быстрее в использовании.
Динамический виртуальный жёсткий диск
Фиксированный виртуальный жёсткий диск
Далее
Отмена


Создать виртуальный жёсткий диск
Укажите имя и размер файла
Пожалуйста укажите имя нового виртуального жёсткого диска в поле снизу или используйте кнопку с иконкой папки справа от него.
Slackware
Укажите размер виртуального жёсткого диска в мегабайтах. Эта величина ограничивает размер файловых данных, которые виртуальная машина сможет хранить на этом диске.
20,00 ГБ
4,00 МБ
2,00 ТБ
Создать
Отмена



Запускаем виртуальную машину.

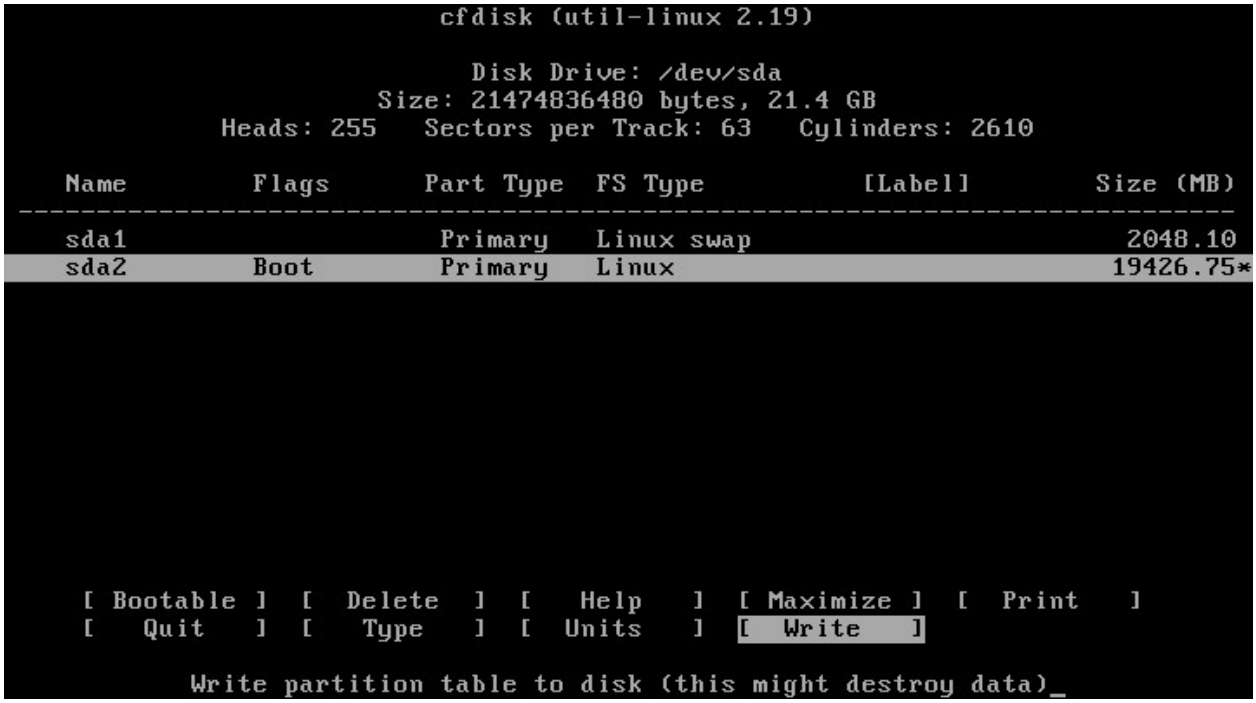
Для начала нужно разметить диск, набрав команду cfdisk



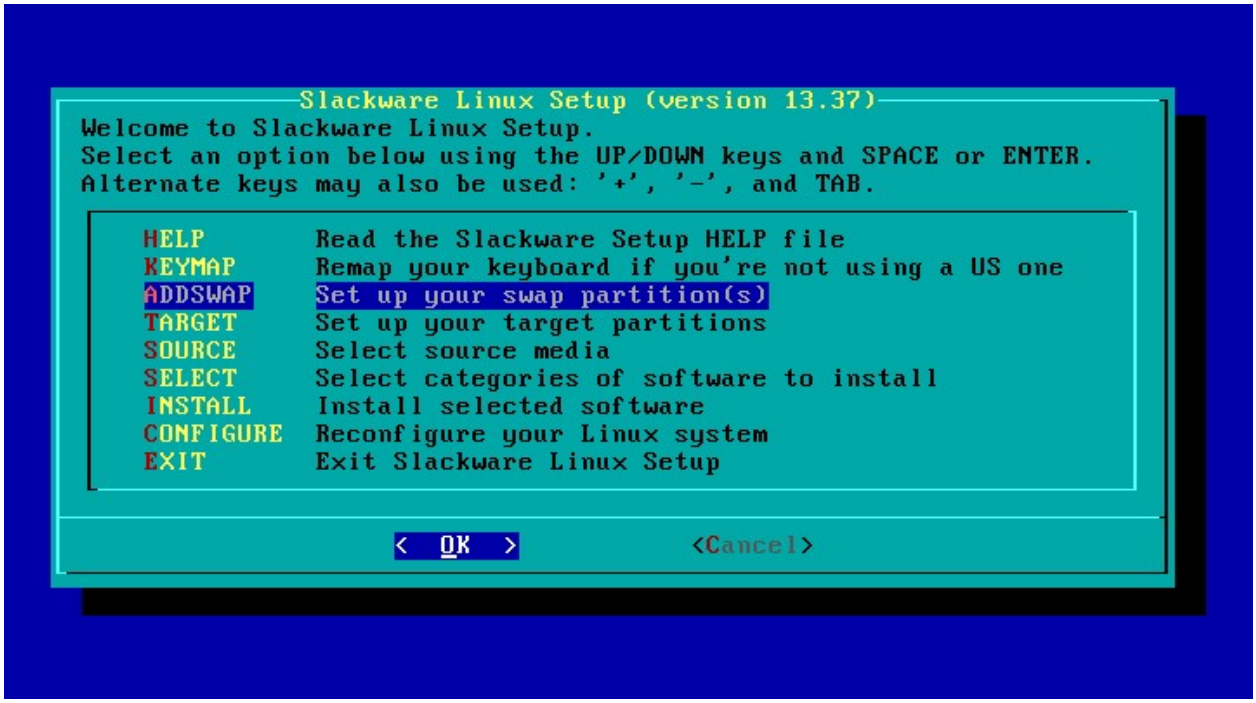
Необходимо создать два раздела:

- Раздел подкачки (Linux Swap – 82)
- Основной раздел (Linux – 83)

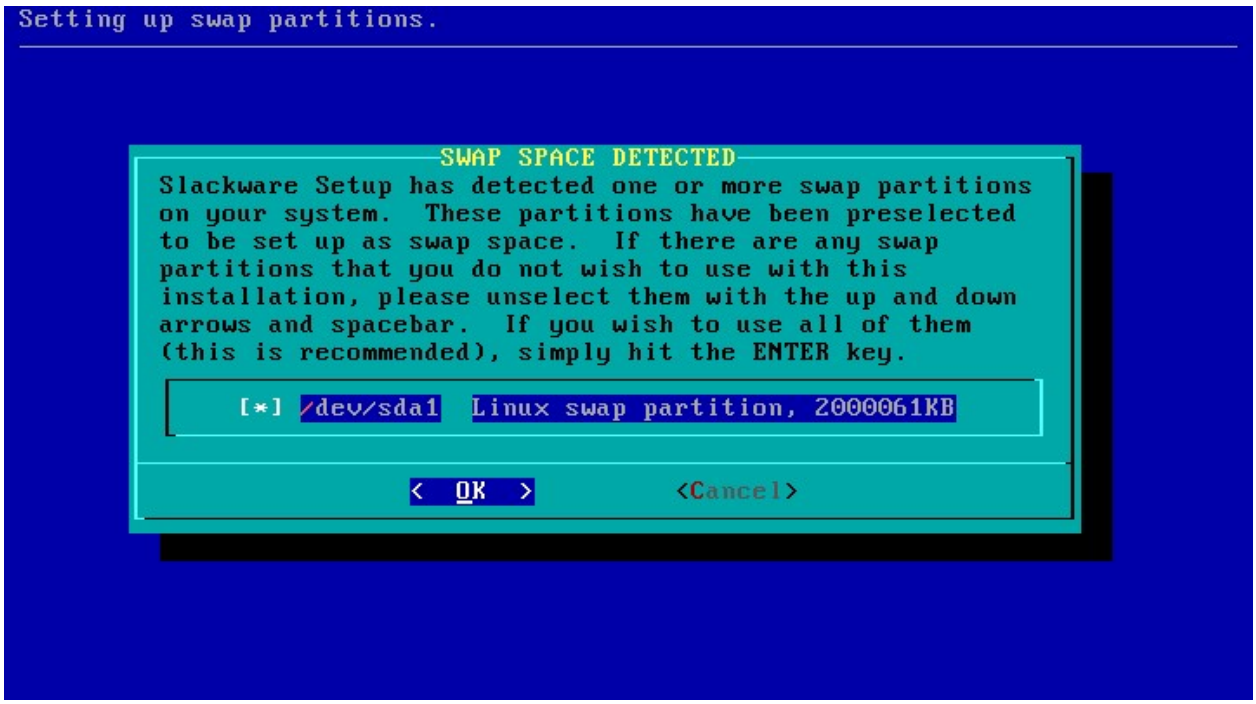
Раздел подкачки рекомендуется делать размером в два раза превышающим размер физической памяти. Под основной раздел забираем весь оставшийся объём. Не забыть поставить флаг **bootable** на основной раздел



После записи геометрии вводим команду **setup**



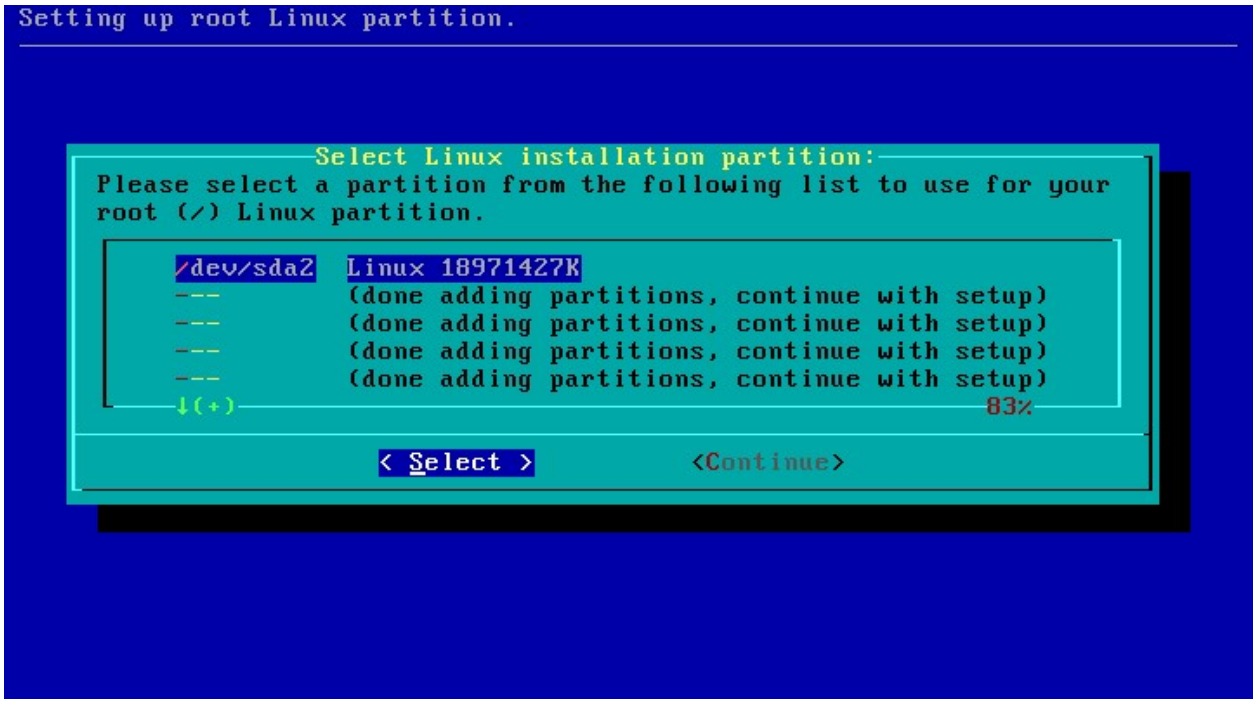
Сначала говорим системе, что у нас есть раздел подкачки и мы хотим его использовать – выбираем **ADDSWAP**. Система сама найдёт нужный раздел (Linux Swap).



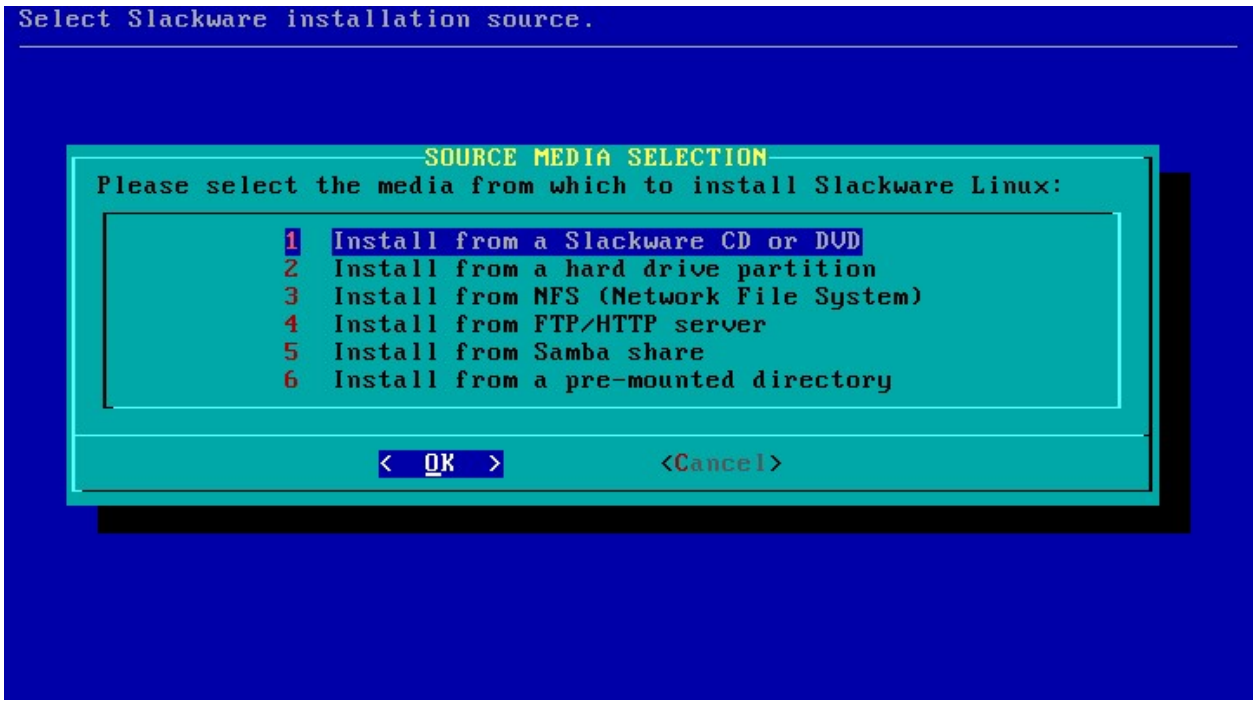
Далее система предложит провести проверку раздела на bad-блоки.

После выполнения команды **swapon** (кстати, можно было запустить вручную сразу после создания разделов) видим сообщение об успешном выполнении операции.

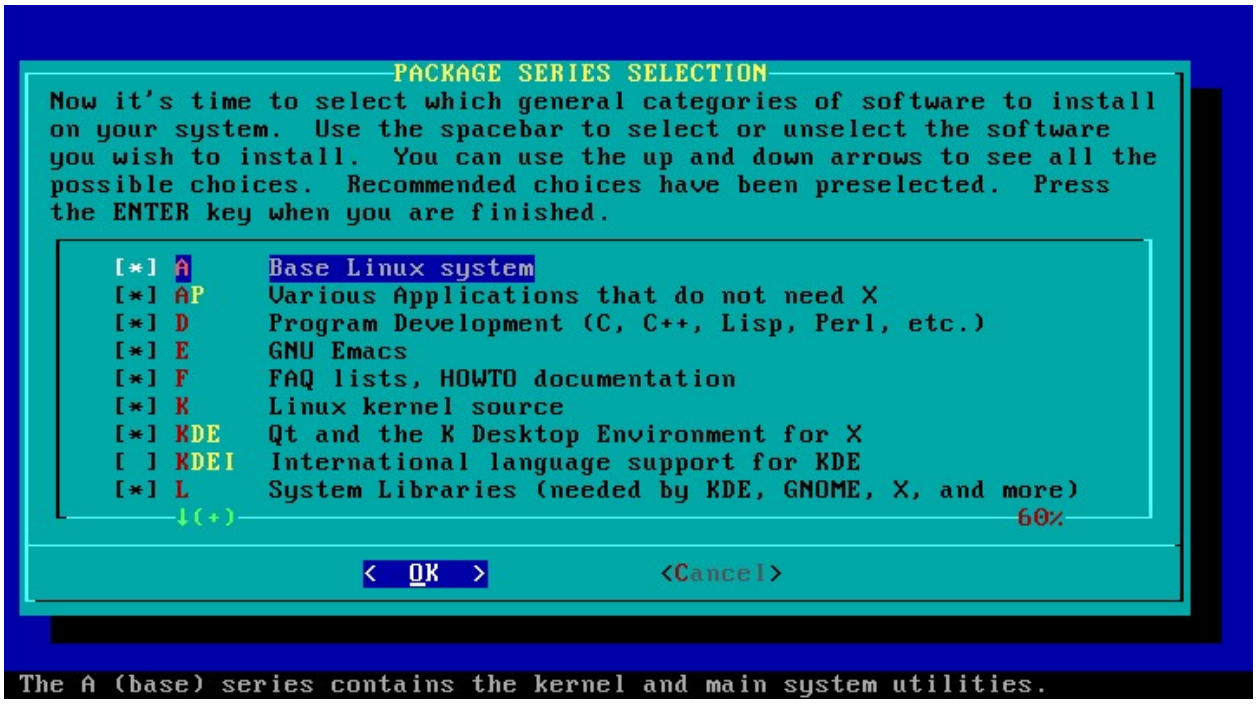
Далее система просит указать основной раздел, он у нас один, поэтому проблем с выбором быть не должно)



Форматируем раздел в файловую систему **ext4.** После выполнения видим сообщение об успешном выполнении операции. Далее необходимо выбрать источник установки. В нашем случае DVD.



После того как система найдёт привод и дистрибутив в нём, нам предложат выбрать устанавливаемые пакеты.

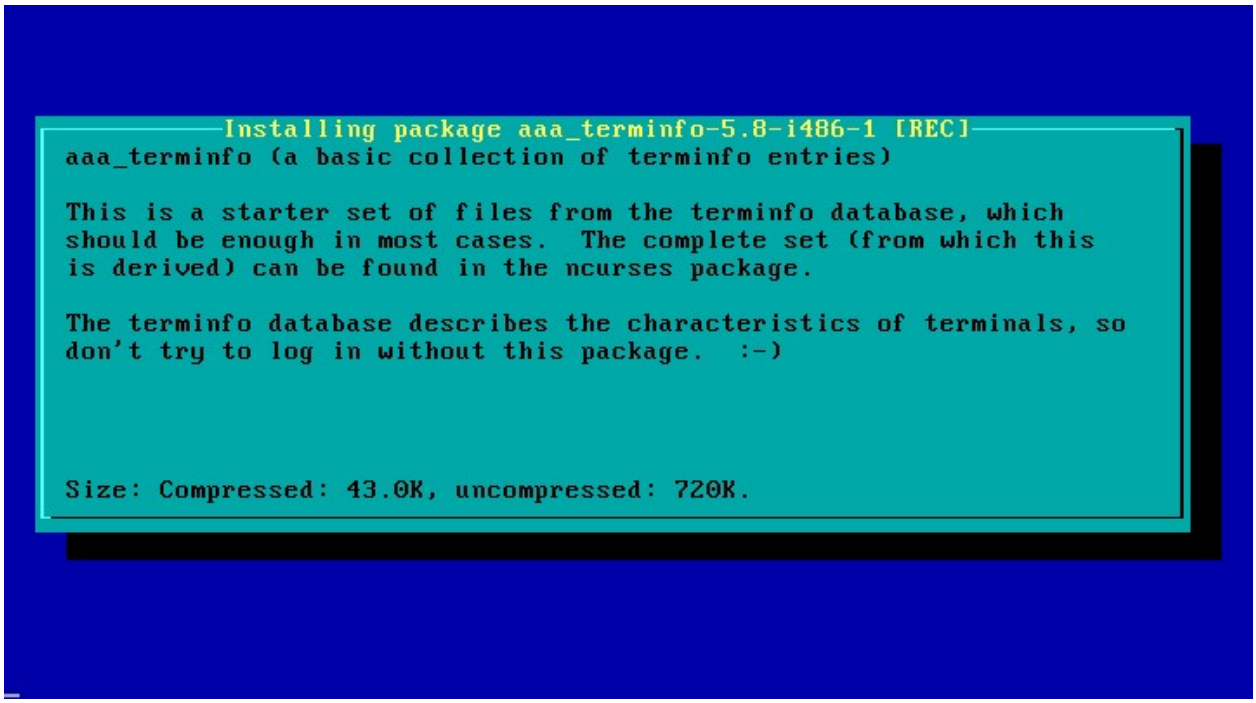


Выбираем ВСЁ!

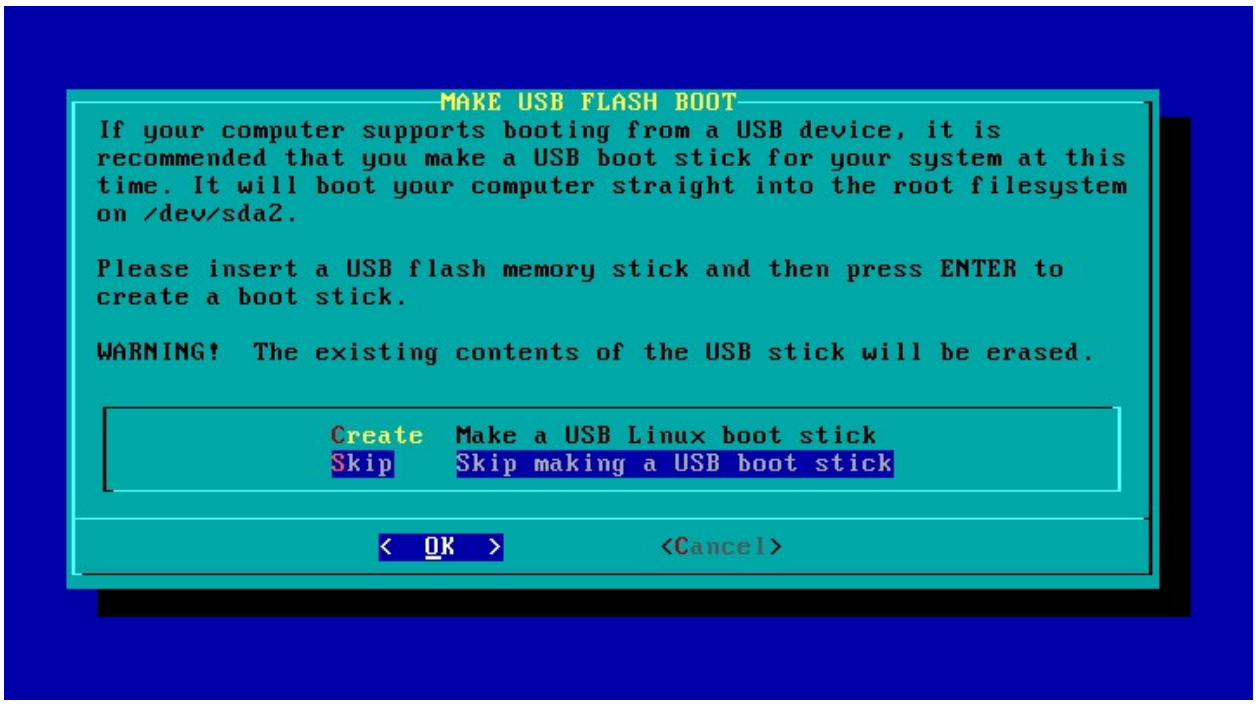
А также полное описание устанавливаемого ПО, которое будет появляться во время установки.

После нажатия OK начнётся процесс установки.

Пример описания устанавливаемого пакета

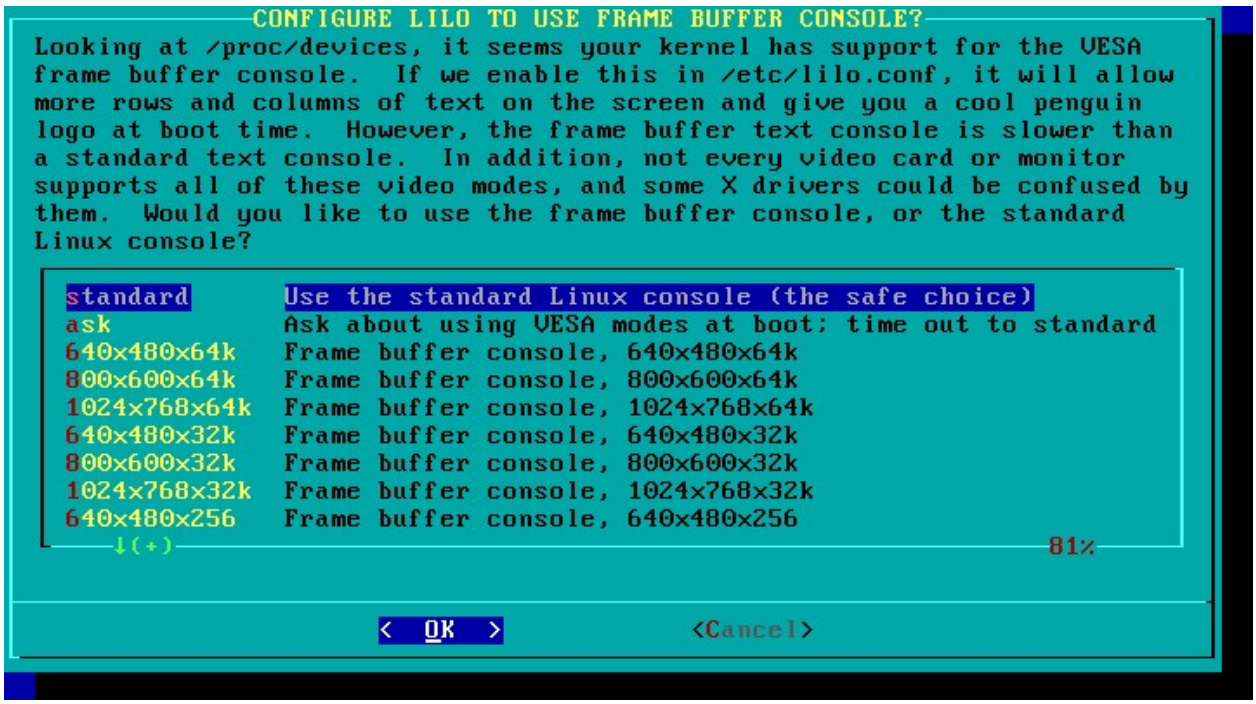


После завершения установки пакетов нам предложат создать загрузочную флэшку

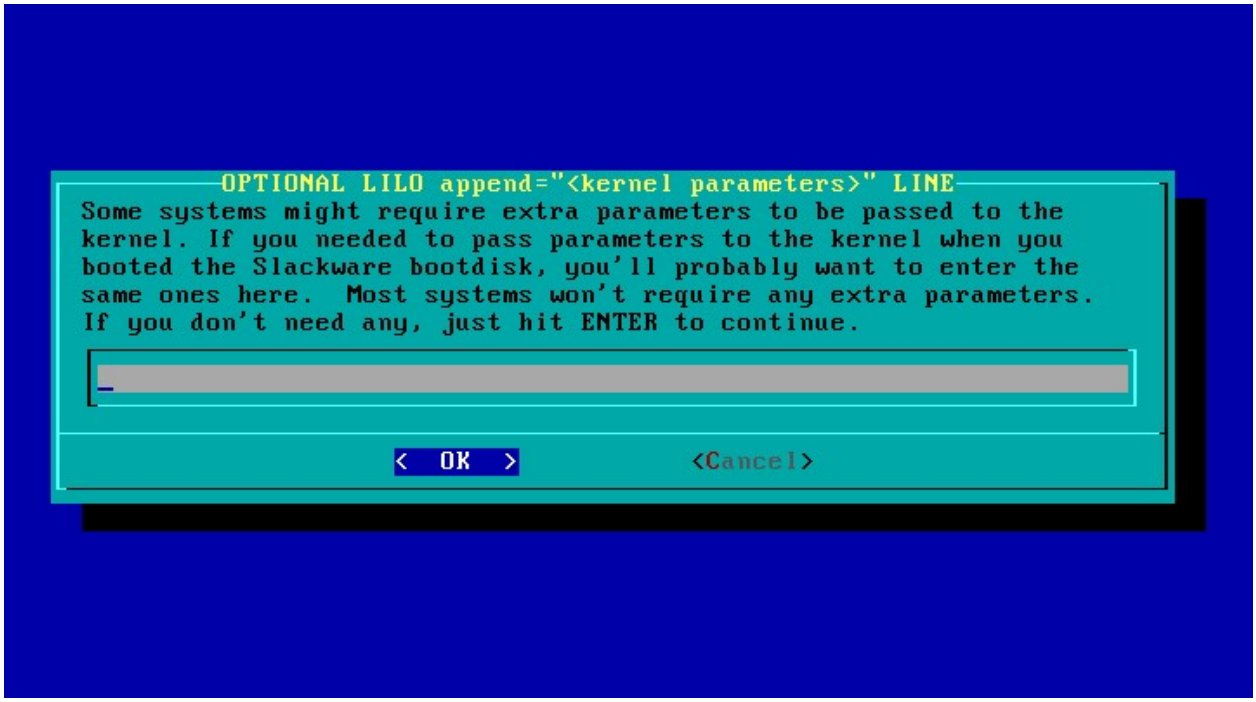


Далее устанавливаем загрузчик (LILO)

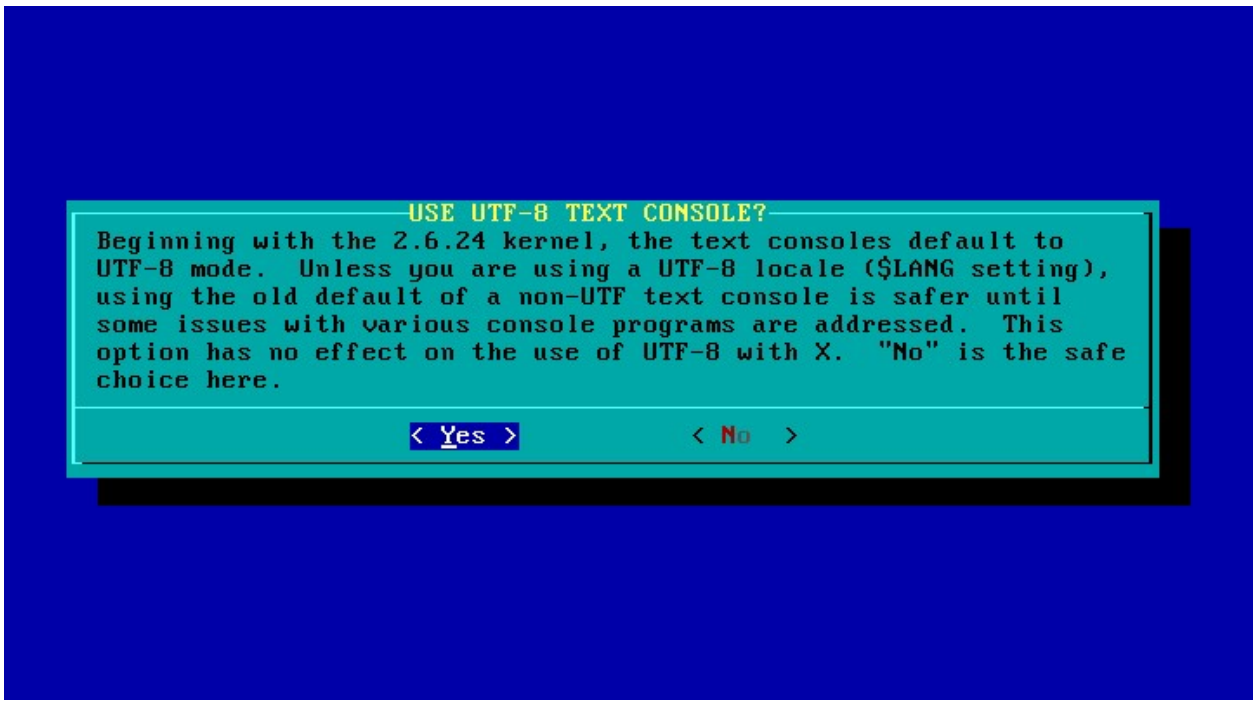
Конфигурируем LILO для использования фрэйм-буфера



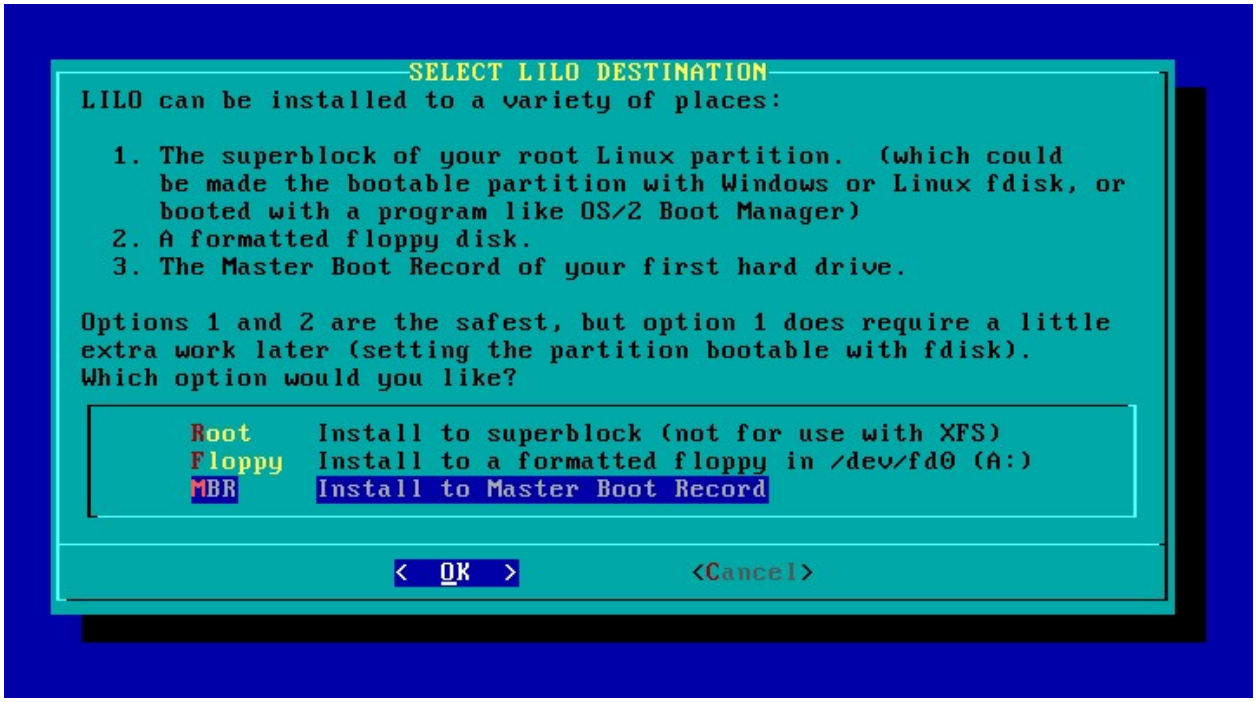
Дополнительные параметры загрузчика (оставляем пустым)



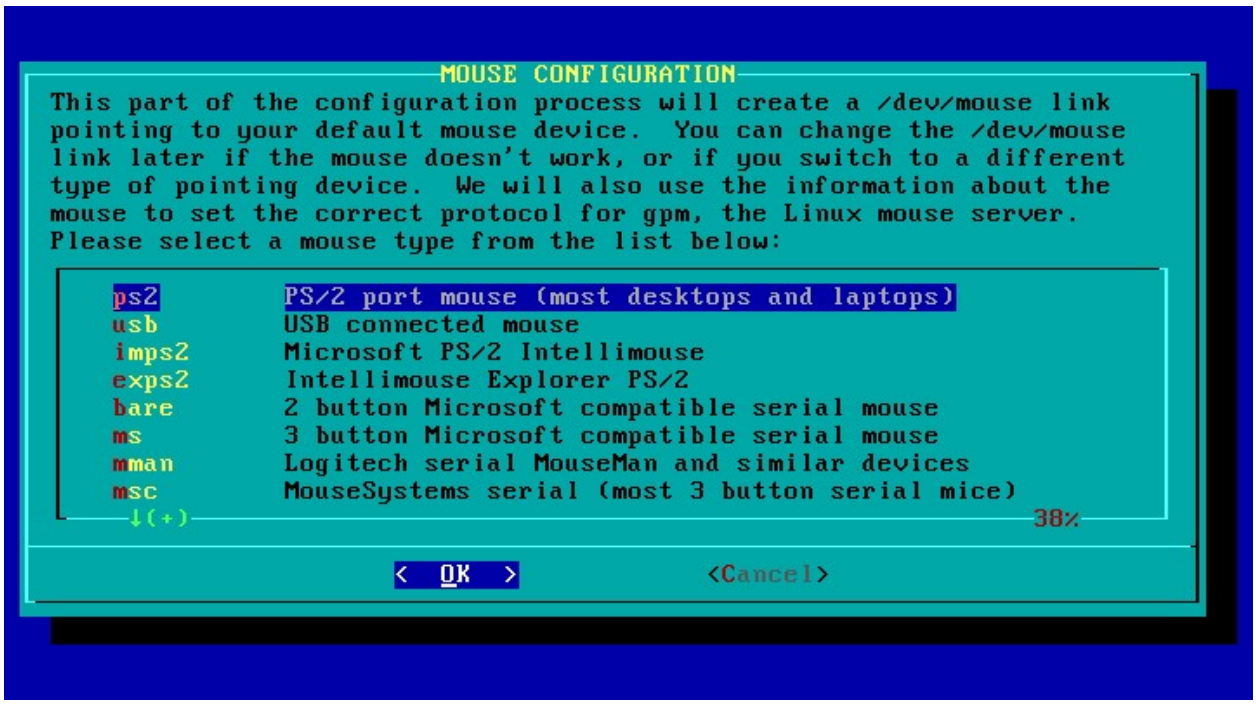
Включаем поддержку UTF-8



Выбираем место расположения загрузчика. У нас одна ОС поэтому размещаем в MBR.



Далее конфигурируем мышку. В настройках виртуальный машины мы выбрали мышь PS/2

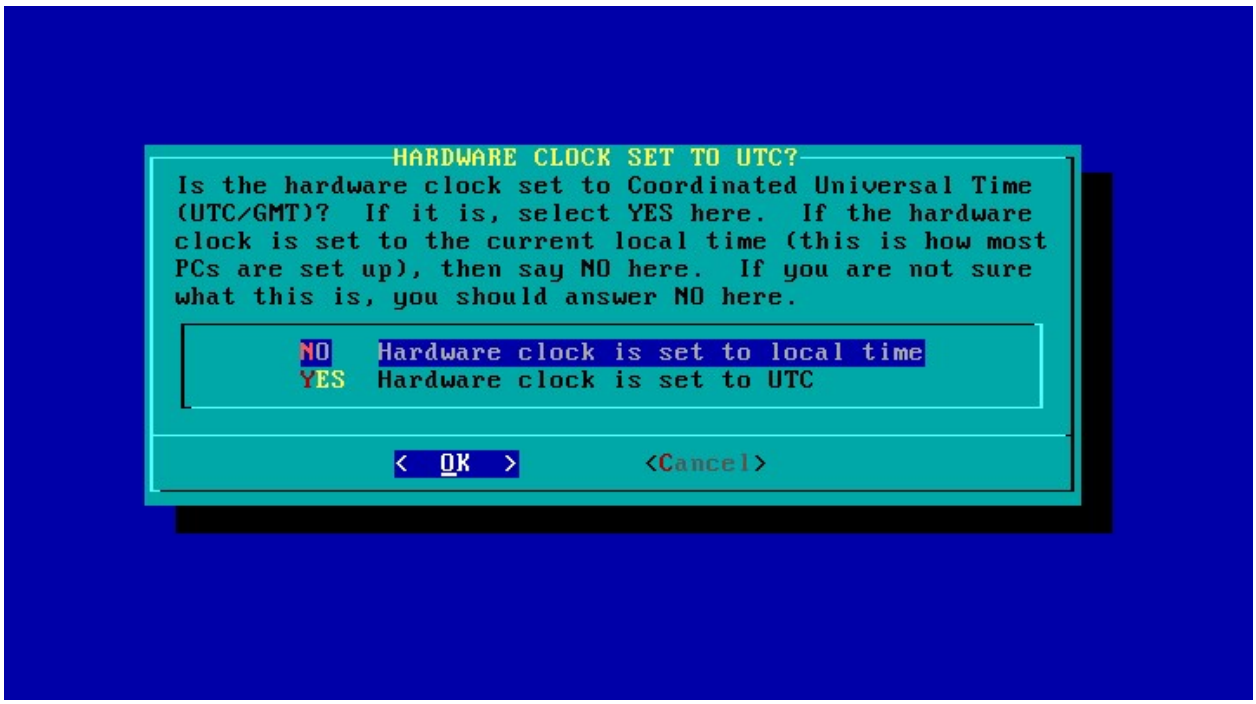


Конфигурацию сети пропускаем.

Конфигурируем сервисы, которые будут запускаться при загрузке. Оставим все которые есть, добавим лишь **rc.samba**

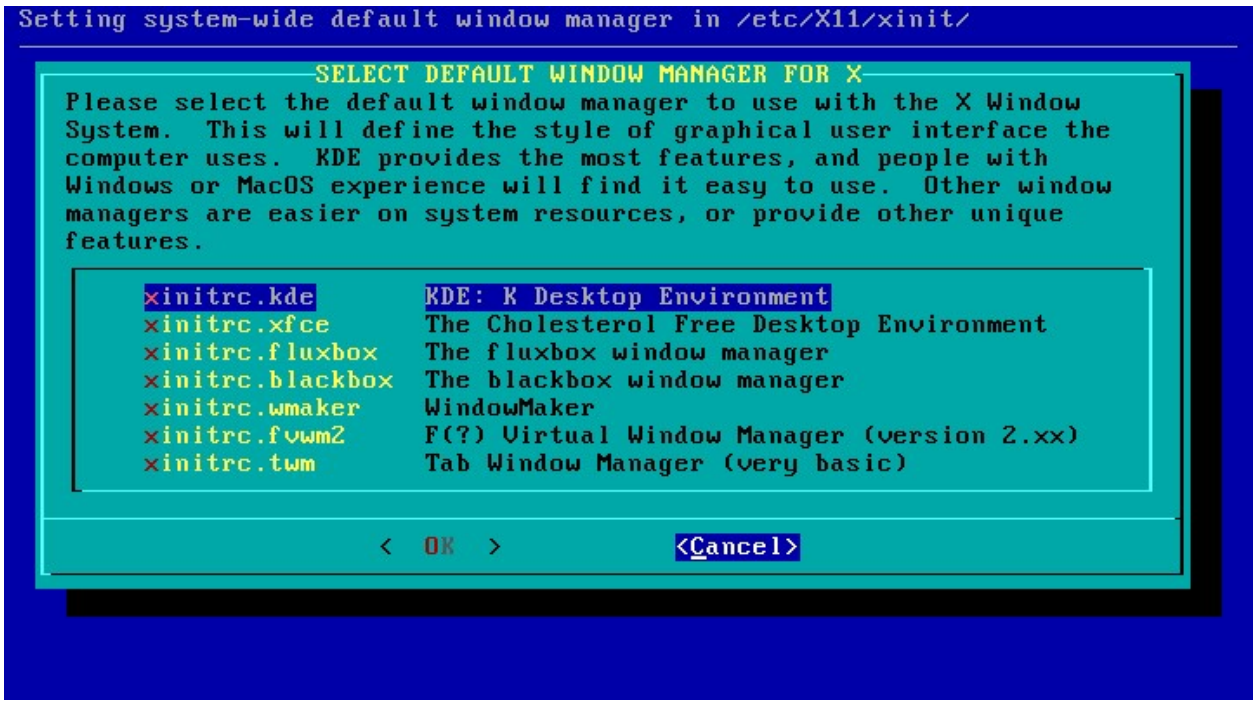
Настройку экранных шрифтов пропускаем

Настраиваем часы

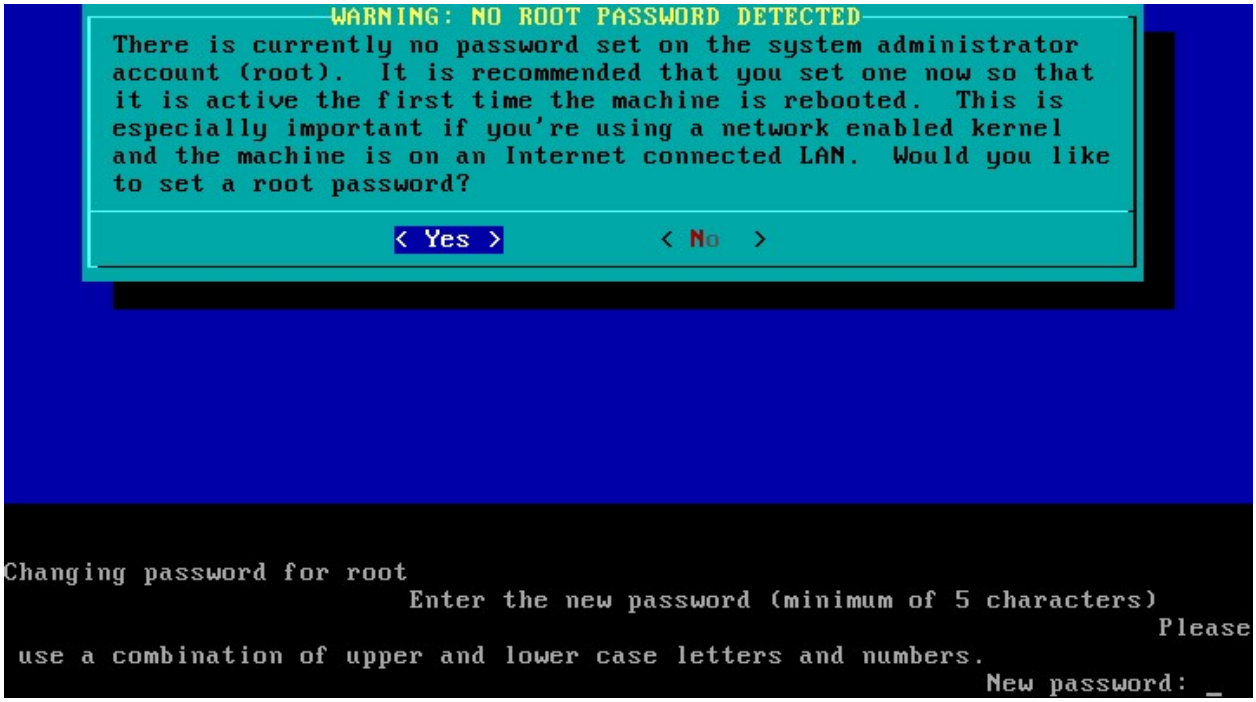


Выбираем Europe/Moscow

Настраиваем графическую среду (оболочка по умолчанию)



Устанавливаем пароль для root



После установки пароля система информирует о завершении процесса установки

Выбираем EXIT, перегружаем систему. Не забыть выставить загрузку с диска.

Содержание отчета:

1. Цель работы
2. Задачи
3. Этапы выполнения лабораторной работы с описанием и скриншотами
4. Ответы на контрольные вопросы
5. Вывод

Контрольные вопросы:

1. Укажите, как реализуется модель Linux-системы
2. Раскройте способ нумерации версий ядра Linux. Укажите, за что отвечает каждая часть
3. Перечислите и опишите основные этапы процесса загрузки Linux
4. Приведите примеры загрузчиков операционных систем
5. Опишите основные функции загрузчиков операционной системы
6. Раскройте сущность фазы загрузчика при загрузке через BIOS
7. Раскройте сущность фазы загрузчика при загрузке через UEFI
8. Укажите основные возможности загрузчика операционной системы G RUB
9. Раскройте сущность фазы ядра
10. Раскройте сущность этапов загрузки ядра и запуска ядра

11. Охарактеризуйте процесс init, укажите основную задачу процесса