Конфигурационная сборка варианта ядра Linux для прикладных систем[1]

С.В. Козин <kozyyy@yandex.ru>

Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.

Национальный исследовательский университет Высшая школа экономики, 101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Операционная система Linux – это современная открытая операционная система, содержащая более 10 000 конфигурационных переменных и множество функциональных системных элементов. Ставится задача создания некоторого варианта ОС для класса прикладных систем (медицины, биологии и др.). Эта задача решается путем анализа базовых функций ядра ОС и выбора из множества элементов наиболее подходящих для оперативного управления прикладными функциями. На их основе создается модель вариабельности из базовых характеристик ОС и модель варианта ОС, включающая основные функциональные элементы ядра ОС. Эти модели тестируются на предмет правильности их идентификации и связей с другими элементами. Затем по этим моделям проводится конфигурирование варианта ОС в виде конфигурационного файла. Этот файл верифицируется, и проходит комплексное тестирование на наборе тестов, проверяющих правильность функционирования операционной среды и процессов обработки заданий прикладных систем. В данной работе рассматривается способ сборки готового варианта ядра операционной системы. Будут затронуты необходимые пакеты, патчи для них и способы их установки. Затем представляется способ конфигурирования собранного варианта системы и настройки ядра для запуска.

Ключевые слова: система Linux; модель характеристик; модель системы; верификация; тестирование; вариант ядра ОС; конфигурационная сборка; верификация исходного файла; тестирование выходного файла.

DOI:10.15514/ISPRAS-2018-30(6)-9

Для цитирования: Козин С.В. Конфигурационная сборка варианта ядра Linux для прикладных систем. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 161-170. DOI: 10.15514/ISPRAS-2018-30(6)-9

1. Введение

Операционная система Linux – это современная открытая операционная система, содержащая более 10 000 конфигурационных переменных и множество функциональных системных элементов. Ставится задача создания некоторого варианта ОС для класса прикладных систем (медицины, биологии и др.) [1-3]. Эта задача решается путем анализа базовых функций ядра ОС и выбора из множества элементов наиболее подходящих для оперативного управления прикладными функциями.

[1] Работа поддержана грантом РФФИ №16-01-00352

На их основе создается модель вариабельности из базовых характеристик ОС и модель варианта ОС, включающая основные функциональные элементы ядра ОС [4-5]. Эти модели тестируются на предмет правильности их идентификации и связей с другими элементами.

Затем по этим моделям проводится конфигурирование варианта ОС в виде конфигурационного файла. Этот файл верифицируется, и проходит комплексное тестирование на наборе тестов, проверяющих правильность функционирования операционной среды и процессов обработки заданий прикладных систем [6-8].

В данной работе рассматривается способ сборки готового варианта ядра операционной системы.

К настоящему моменту сформировались стандарты, определяющие интерфейсы, облегчающие сборку вариантов разных систем, в том числе и ОС. К ним относятся стандарты:

- POSIX.1-2008;
- Filesystem Hierarchy Standard (FHS) Version 3.0;
- Linux Standard Base (LSB) Version 5.0 (2015).

Используемые пакеты, необходимые для удовлетворения требований LSB:

- **LSB Core**: Bash, Bc, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib;
- **LSB Runtime Languages**: Perl.

Вариант ОС строится с использованием уже установленного дистрибутива Linux (такого как Debian, Open Mandriva, Fedora или openSUSE). Система Linux (хост) используется в качестве отправной точки для предоставления необходимых программ, включая компилятор, компоновщик и оболочку, для создания нового варианта ОС.

При этом в хост-систему необходимо установить пакеты:

Bash-3.2, Binutils-2.25, Bison-2.7, Bzip2-1.0.4, Coreutils-6.9, Diffutils-2.8.1, Findutils-4.2.31, Gawk-4.0.1, GCC-4.9, включая компилятор C++, g++ , Glibc-2.11, Grep-2.5.1a, Gzip-1.3.12, Linux Kernel-3.2, M4-1.4.10, Make-4.0, Patch-2.5.4, Perl-5.8.8, Sed-4.1.5, Tar-1.22, Texinfo-4.7, Xz-5.0.0.

Затем создается новый раздел на диске и файловая система, пригодная для Linux, выбираются и скачиваются необходимые пакеты и патчи для создания варианта системы и сохранения его в новой файловой системе и в новом разделе. Устанавливается ряд пакетов, которые будут формировать базовый пакет разработки (первичный вариант системы) для использования в реальной

системе. Некоторые из этих пакетов используются при сборке компилятора. После этого создается и конфигурируется финальный вариант системы.

2. Подготовка к созданию нового варианта ОС

Предлагается подход к созданию системы, который основан на использовании свободного пустого раздела для размещения на нем системы Linux (около 6 гигабайт), хранения всех исходных архивов и компиляции пакетов [1-4].
Кроме того, требуется свободное временное хранилище достаточно большого размера и дисковое пространство для компиляции пакетов. Например, рекомендуется использовать небольшой раздел диска для обеспечения пространства подкачки – swap.
Первоначально диск разбивается с помощью специальной утилиты, например, cfdisk или fdisk (жесткий диск) для создания нового раздела, например, /dev/sda для основного диска. После создания этого раздела создается файловая система, например, ext3 или ext4.

3. Подготовка инструментария (toolchain)

Пакет ОС Binutils устанавливается первым и как Glibc выполняет различные функциональные тесты на ассемблере и компоновщике, чтобы определить, какие программные функции включены или отключены. Неправильная конфигурация GCC или Glibc может привести к неочевидной поломке инструментальной цепочки, которая может не выявиться до конца сборки всего дистрибутива. Для предотвращения неправильных конфигураций выполняется тестовый набор, который позволяет обнаружить ошибки, прежде чем будет выполнена слишком большая работа по установке некоторых функций ОС.
Пакет Binutils устанавливает свой компилятор, компоновщик и заголовки APILinux. Это позволяет стандартной библиотеке C (Glibc) взаимодействовать с функциями, которые предоставляет ядро Linux.
Следующий пакет – Glibc. Для построения Glibc используются компилятор, бинарные инструменты и заголовки ядра. При этом в Glibc применяется компилятор, который был задан параметром -host, переданным скрипту configure (например, компилятор может быть i686-lfs-linux-gnu-gcc).
Бинарные инструменты и заголовки ядра более сложны в установке. После запуска configure проверяется содержимое файла config.make в каталоге glibc-build.

4. Создание версии ОС

Когда ядро загружает систему, требуется наличие нескольких узлов устройств, в частности консольных. Узлы устройств должны быть созданы на жестком диске, чтобы они были доступны до запуска udevd, а также при запуске Linux с init=/bin/bash. Рекомендуется подключение виртуальной файловой системы (например, tmpfs) в каталоге /dev и возможность динамического создания

устройств в этой виртуальной файловой системе по мере их обнаружения или доступа. Создание устройства обычно выполняется во время загрузки udev.
Linux поддерживает список смонтированных файловых систем в файле /etc/mtab. Современные ядра хранят этот список внутри себя и предоставляют его пользователю через файловую систему /proc. Чтобы работали утилиты, ожидающие наличия /etc/mtab, нужно создать соответствующую символическую ссылку.
В соответствии с документацией необходимо установить следующие пакеты: Linux-4.18.5 API Headers, Man-pages-4.16, Glibc-2.28, Zlib-1.2.11, File-5.34, Readline-7.0, M4-1.4.18, Bc-1.07.1, Binutils-2.31.1, GMP-6.1.2, MPFR-4.0.1, MPC-1.1.0, Shadow-4.6, GCC-8.2.0, Bzip2-1.0.6, Pkg-config-0.29.2, Ncurses-6.1, Attr-2.4.48, Acl-2.2.53, Libcap-2.25, Sed-4.5, Psmisc-23.1, Iana-Etc-2.30, Bison-3.0.5, Flex-2.6.4, Grep-3.1, Bash-4.4.18, Libtool-2.4.6, GDBM-1.17, Gperf-3.1, Expat-2.2.6, netutils-1.9.4, Perl-5.28.0, XML::Parser-2.44, Intltool-0.51.0, Autoconf-2.69, Automake-1.16.1, Xz-5.2.4, Kmod-25, Gettext-0.19.8.1, Libelf 0.173, Libffi-3.2.1, OpenSSL-1.1.0i, Python-3.7.0, Ninja-1.8.2, Meson-0.47.1, Procps-ng-3.3.15, E2fsprogs-1.44.3, Coreutils-8.30, Check-0.12.0, Diffutils-3.6, Gawk-4.2.1, Findutils-4.6.0, Groff-1.22.3, GRUB-2.02, Less-530, Gzip-1.9, IPRoute2-4.18.0, Kbd-2.0.4, Libpipeline-1.5.0, Make-4.2.1, Patch-2.7.6, Sysklogd-1.5.1, Sysvinit-2.90, Eudev-3.2.5, Util-linux-2.32.1, Man-DB-2.8.4, Tar-1.30, Texinfo-6.5, Vim-8.1.
Чтобы начать сборку и установку окончательного варианта системы LFS, необходимо войти в среду chroot.

5. Конфигурирование варианта ОС

Для конфигурации варианта системы вы сможете выбрать параметры, включенные в ядро. В зависимости от архитектуры будут доступны разные наборы параметров, а также их содержимое. Однако некоторые параметры будут доступны независимо от варианта выбора встроенной архитектуры. Ниже приведен список основных параметров меню, доступных для всех встроенных архитектур Linux [4-8]:

- общая настройка;
- поддержка загружаемых модулей;
- блоки;
- сетевые настройки;
- драйверы устройств;
- файловые системы;
- параметры безопасности;
- криптографические параметры;
- библиотечные процедуры.

Ядро может включать в себя множество специальных опций безопасности в составе стека SELinux, реализованного Национальным агентством безопасности США (NSA). Однако только для нескольмких встроенных устройств предпочтительно использование такой обширной

функциональности.

Для начала проведения конфигурирования варианта системы ОС требуется проверить файл /etc/udev/rules.d/70-persistent-net.rules, чтобы определить имя, присвоенное сетевому устройству, и настройки именования через правила udev. Описание файла начинается с блока комментариев, за которым следуют две строки для каждого сетевого адаптера. Первая строка сетевой карты – это описание с комментариями, показывающее идентификаторы оборудования (например, поставщик PCI и идентификаторы устройств в случае карты PCI) и драйверы в круглых скобках.

Ни идентификатор аппаратного обеспечения, ни драйвер не используются для определения имени для предоставления интерфейса; эта информация предназначена только для справки. Вторая строка – это правило udev, которое соответствует сетевой карте и фактически присваивает имя.

Некоторое ПО, которое может быть установлено позже (например, различные медиаплееры), ожидает, что существуют символические ссылки /dev/cdrom и /dev/dvd, указывающие на устройство CD-ROM или DVD-ROM. Кроме того, может быть удобно помещать ссылки на эти символические ссылки в /etc/fstab. udev поставляется со сценарием, который будет генерировать файлы правил для создания этих символических ссылок в зависимости от возможностей каждого устройства. Во-первых, сценарий может работать в режиме «by-path» (используется по умолчанию для устройств USB и FireWire), где создаваемые правила зависят от физического пути к устройству CD или DVD. Во-вторых, он может работать в режиме «по-id» (по умолчанию для устройств IDE и SCSI), где создаваемые ими правила зависят от идентификационных строк, хранящихся на самом устройстве CD или DVD. Путь определяется сценарием path_id в udev, а строки идентификации считываются из аппаратного обеспечения с помощью своих программ ata_id или scsi_id, в зависимости от того, какой тип устройства у вас есть.

Если ожидается, что физический путь к устройству (то есть порты и / или слоты, в которые он подключается) изменится, например, при потребности переместить диск на другой порт IDE или другой разъем USB, то необходимо использовать режим «by-id». С другой стороны, в случае изменения идентификации устройства или его уничтожения, его необходимо заменить другим устройством с теми же возможностями, подключая к одному и тому же разъему -path.

Для каждого устройства с проблемой дублирования находится соответствующий каталог в разделе /sys/class или /sys/block. Для видеоустройств это может быть /sys/class/video4linux/videoX. При этом анализируются атрибуты, которые идентифицируют устройство однозначно, а затем создаются символьные ссылки.

То, какие интерфейсы связаны с сетевым сценарием, обычно зависит от файлов в /etc/sysconfig/. Этот каталог должен содержать файл для каждого настраиваемого интерфейса, например, ifconfig.xyz, где «xyz» должно описывать сетевую карту. Обычно используется имя интерфейса (например,

eth0). Внутри этого файла есть атрибуты этого интерфейса, такие как его IP-адрес, маски подсети и т. д.

Во время инициализации ядра первая запущенная программа указывается в командной строке, либо по умолчанию ей является init, которая считывает файл инициализации /etc/inittab.

Для ОС выполняется команда rc – для запуска всех скриптов, начинающихся с S в каталоге /etc/rc.d/rcS.d, а также скриптов, начинающихся с S в файле /etc/rc.d/rc?.d, где знак вопроса задается значением initdefault.

rc-скрипт читает библиотеку функций в /lib/lsb/init-functions. Эта библиотека также читает дополнительный файл конфигурации /etc/sysconfig/rc.site. Любые параметры файла конфигурации системы могут быть альтернативно размещены в этом файле, что позволяет консолидировать в нем все системные параметры.

Файл inputrc используется для конфигурирования библиотеки Readline, предназначенной для редактирования командных строк пользователя, вводимых с терминала. Readline используется Bash и большинством других оболочек, а также многими другими приложениями.

6. Подготовка к запуску варианта ОС

Файл /etc/fstab используется некоторыми программами для определения того, где должны быть установлены файловые системы по умолчанию, в каком порядке и какие из них необходимо проверить (для контроля целостности) перед установкой.

Файловым системам с MS-DOS или Windows-источником (т.е. Vfat, ntfs, smbfs, cifs, iso9660, udf) требуется специальная опция utf8, чтобы символы в именах файлов, не входящие в набор ASCII, были правильно интерпретированы. Для локалей, отличных от UTF-8, значение iocharset должно быть таким же, как и набор символов локали. Соответствующее определение набора символов (найденное в Файловых системах -> Поддержка родного языка при настройке ядра) должно быть скомпилировано в ядро или построено как модуль. Для файловых систем vfat и smbfs также необходима опция «codepage».

Построение ядра включает в себя несколько этапов – настройку, компиляцию и установку.

В большинстве случаев модули Linux загружаются автоматически, но иногда требуется определить другой порядок. Для этой цели нужно использовать modprobe или insmod, порядок для которых задается в файле /etc/modprobe.d/usb.conf. Этот файл необходимо создать так, чтобы, если USB-драйверы (ehci_hcd, ohci_hcd и uhci_hcd) были созданы как модули, они были загружены в правильном порядке; ehci_hcd необходимо загрузить до ohci_hcd и uhci_hcd, чтобы избежать появления предупреждения во время загрузки.

Расположение загрузочного раздела GRUB – это выбор пользователя, который влияет на конфигурацию. Одна рекомендация состоит в том, чтобы иметь отдельный небольшой (рекомендуемый размер – 100 МБ) раздел только для

загрузки информации. Таким образом, каждая сборка может обращаться к одним и тем же загрузочным файлам, и доступ может быть сделан из любой загруженной системы.

Используя приведенную выше информацию, можно определить соответствующий указатель для корневого раздела (или отдельного загрузочного раздела, если он используется), установить файлы GRUB и настроить загрузочный трек, сгенерировать /boot/grub/grub.cfg. После этого новая система готова к загрузке.

7. Заключение

В процессе разработки варианта ОС на базе ядра Linux был сконфигурирован некоторый вариант системы для конечного пользования. Данная система имеет все стандартные функции ядра Linux, функцию сжатия данных, файловую систему, интерфейс командной строки, функции арифметического вычисления для рациональных чисел (включая числа с плавающей точкой), функции чтения аудио-файлов, компилятор языка C и C++, систему защиты паролей, интерфейс установки пакетов pkg, список контроля доступа, текстовый редактор, возможность работы с сетью, поиск по системе, библиотеку функций баз данных, генератор хэш функций, XML-парсер, openSSL и некоторые другие вспомогательные функции.

Такая система может использоваться во многих областях. Она обладает необходимым набором инструментов и для домашнего пользования, если пользователь хорошо знаком с семейством операционных систем Linux.

Список литературы

[1] Лаврищева Е. М., Коваль Г.И., Слабоспицкая О.О., Колесник А.Л. Особенности процессов управления при создании семейств программных систем. Проблемы программирования, no. 3, 2009 г., стр. 40-49.

[2] Лаврищева Е. М., Коваль Г.И., Слабоспицкая О.О., Колесник А.Л. Теоретические аспекты управления вариабельностью в семействах программных систем. Вестник КНУ, серия физ.-мат. наук, no. 1, 2011 г., стр.151-158

[3] Лаврищева Е.М. Программная инженерия и технология программирования сложных систем. Учебник. 2-издание. Москва, Юрайт, 2018, 431 стр.

[4] Е.М. Лаврищева, В.С. Мутили, А.Г. Рыжов. Аспекты моделирования вариабельных программных и операционных систем. Сб. трудов XIX Всероссийский научной конференции «Научный сервис в сети Интернет», 2017, стр. 327-341.

[5] Лаврищева Е.М. Петренко А.К. Моделирование систем и их семейств. Труды ИСП РАН, 2016 г., том 28, вып. 6, с. 49-65. DOI: 10.15514/ISPRAS-2016-28(6)-4.

[6] И.С. Захаров, М.У. Мандрыкин, В.С. Мутилин, Е.М. Новиков, А.К. Петренко, А.В. Хорошилов. Конфигурируемая система статической верификации модулей ядра операционных систем. Труды ИСП РАН, том 26, вып. 2, 2014 г., стр. 5-42. DOI: 10.15514/ISPRAS-2014-26(2)-1.

[7] Kozin S.V., Mutilin V.S. Static Verification of Linux Kernel Configurations. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 217-230. DOI: 10.15514/ISPRAS-2017-29(4)-14.

[8] Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды ИСП РАН, 2016, том 28, вып. 3, стр. 189-208. DOI: 10.15514/ISPRAS-2016-28(3)-12.

Linux kernel configuration build for application systems

S.V. Kozin <kozyyy@yandex.ru>

Ivannikov Institute of System Programming, RAS,
25, A. Solzhenitsyn str., Moscow, 109004, Russia
National Research University Higher School of Economics,
20, Myasnitskaya str, Moscow., 101000, Russia

Abstract. The Linux operating system is a modern open operating system containing more than 10,000 configuration variables and a large variety of functional system elements that handle the processing of various kinds of tasks. The task is to create some version of the OS for a class of applied systems (medicine, biology, etc.). This task is solved by analyzing the basic functions of the OS kernel and choosing from a variety of elements the most suitable for the operational management of application functions. Based on them, a model of variability is created from the basic characteristics of the OS and the model of the OS variant, including the main functional elements of the OS kernel. These models are tested for the correctness of their identification and relationships with other elements. Then, using these models, the OS version is configured as a configuration file. This file is verified and undergoes comprehensive testing on a set of tests that verify the correct functioning of the operating environment and the processing of tasks of applied systems. This paper discusses how to build a ready-made version of the operating system kernel from source. The preparations, the necessary packages, the patches for them and the ways of their installation will be affected. Then it presents a method for configuring a system version assembled from source and configuring the kernel to run.

Keywords: Linux system; characteristics model; system model; verification; testing; OS kernel version; configuration building; source file verification; output file testing

DOI: 10.15514/ISPRAS-2018-30(6)-9

For citation: Kozin S.V. Linux kernel configuration build for application systems. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 161-170 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-9

References

[1] Lavrischeva E.M., Koval' G.I., Slabospitskaya O.O., Kolesnik A.L. Product Line Development Management Specifics. Problemy programmiovaniya [Problems of Software Developement], no. 3, 2009., pp. 40-49 (in Ukrainian).

[2] Lavrischeva E.M., Slabospitskaya O.O., Koval' G.I., Kolesnik A.L. Theoretical Aspects of Variability Management in Product Lines. Vesnik KNU seria fiz.-mat. nauk [Notes of KNU, series on maths and physics], no. 1, 2011, pp.151-158 (in Ukrainian).

[3] Lavrischeva E.M. Software engineering and programming technology for complex systems. Textbook. 2nd edition. Moscow, Yuright, 2018, 431 p. (in Russian).

[4] Lavrischeva K.M., Mutilin V.S., Ryzhov A.G. Aspects of Modeling of Variable Software and Operating Systems. In Proc. of the XIX All-Russian conference on Scientific Services in the Internet, 2017, pp. 327-341 (in Russian).

[5] Lavrischeva K.M., Petrenko A.K. Software Product Lines Modeling. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 6, 2016, pp. 49-64 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-4 (in Russian).

[6] Zakharov I.S., Mandrykin M.U., Mutilin V.S., Novikov E.M., Petrenko A.K., Khoroshilov A.V. Configurable Toolset for Static Verification of Operating Systems Kernel Modules. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 2, 2014, pp. 5-42. DOI: 10.15514/ISPRAS-2014-26(2)-1 (in Russian).

[7] Kozin S.V., Mutilin V.S. Static Verification of Linux Kernel Configurations. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 217-230. DOI: 10.15514/ISPRAS-2017-29(4)-14.

[8] Kuliamin V.V., Lavrischeva E.M., Mutilin V.S., Petrenko A.K. Verification and analysis of variable operating systems. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 3, 2016, pp. 189-208 (in Russian). DOI: 10.15514/ISPRAS-2016-1(2)-12/