Компилиране на Linux-ядрото за начинаещи и не толкова напреднали, част I: Въведение

Н. Антонов,

pcradio@netbg.com

Такива текстове има много. Мрежата изобилства от тях и ако не ви се чете, не го правете. Особено ако сте по-напреднал потребител и се интересувате от нещо задълбочено и професионално. Ако обаче и понятие си нямате от компилацията на Linux-ядрото или не ви се полагат повече усилия в четеното на чуждоезични текстове, можете да продължите.

Малко общи приказки

Ядрото (kernel)

Linux-ядрото е сърцето на операционната система. Всъщност, то е собствено самата операционна система. Нарича се още kernel. Зарежда се със стартирането на компютъра и се зазижда в паметта до изключването на машината. Има две основни задачи:

обслужва хардуера на ниско ниво (напр. - прекъсванията);

създава средата, необходима за работата на процесите (неслучайно се нарича още "платформа");

Linux-ядрото се характеризира като монолитно - един голям изпълним файл (обикновено между 850k - 1Mb), съдържащ множество логически разделени компоненти.

Както знаете, ядрото се разпространява с отворен изходен текст. Не код, а текст, защото е написан на човешки език, макар и разбираем само от програмистите. Текстът ще стане код след компилацията, защото ще придобие вид, разбираем само за процесора. Можете да го изтеглите от ftp://ftp.kernel.org/pub/linux/kernel. Там са публикувани всички версии на ядрото. Интересува ни серията 2.4.x. Серията 2.2.x все още се използува, но ако искате да опитате по-нов хардуер или например невероятната файлова система ReiserFS, ще ви трябва ядро от по-новата серия 2.4.x и дори по възможност последна версия. За ReiserFS например пише, че до версията 2.4.16 ядрата имат сериозни бъгове и не е желателно да се използуват с тази файлова система. ßдро от серията 2.4.x ще ви трябва и ако искате да използувате последните новости в областта на мрежовата защита, осигурявана с помощта на iptables. За 2.2.x ядрата има ipchains.

Версиите с нечетно число след точката са нестабилни, "хакерски", предназначени за тестване и не се препоръчват за системи, които искат стабиност и сигурност. Такава е серията 2.5.x, която се разработва активно в момента.

ßдрото се разпространява под формата на bzip2 или gzip архив. Именувано е така: linux-x.y.z.tar.bz2. Този вид компресия е по-добра и дава по-малки по обем файлове. Под x.y.z разбирайте номера на версията. Изтеглете последната версия с четен номер след точката.

След като изтеглите файла, можете да го разархивирате на произволно място, но е добре да създадете символичен линк към неговата директория в /usr/src под името linux. Може и да ви потрябва след време, а може и да не ви потрябва. Обикновено разни пачове търсят по подразбиране изходния текст (сорс) на ядрото в тази директория. Ако нямате опит при работата с такива архиви, можете да използувате mc (Midnight Commander), който ги отваря и манипулира с тях като със стандартни файлове и директории.

Модулите

Тук е мястото да кажем какво представляват модулите. Те са голяма магия. Дават възможност на ядрото да придобива временни пълномощия, каквито не са му вродени при компилацията. ßдрото може да зарежда динамично допълнителни парчета код, за да изпълни конкретна задача и след това да изчисти модула от паметта. Иначе силата на модулите е неограничена. Дават възможност на ядрото да работи с най-различни файлови системи и специфични хардуерни устройства, но след като си свърши работата с тях и не се нуждае вече от допълнителни възможности, модулът просто може да се изпари от паметта, за да освободи място за по-важни неща.

Ако искате да видите модулите, които работят в паметта или просто изчакват да бъдат извикани, използувайте командата lsmod. С комнадите modprobe и insmod можете да зареждате конкретни модули, стига да знаете тяхното име. Командата modprobe е за предпочитане за тази цел. С rmmod можете да изчиствате модулите от паметта, но само ако не са заети с някоя задача.

Има някои драйвери или функции, които не могат да работят като модули. Те трябва или да се вградят в ядрото, или напълно да се изключат. Практиката показва, че оптималният резултат се постига от златната среда - повечето неща да се копилират като модули, но някои, от които зависи пряко бързодействието на системата, задължително да присъстват. Всъщност, конфигураторът сам ще ви подскаже донякъде какво е необходимо да се вгради и какво може да остане като модул.

Защо да компилираме собствено ядро?

Всяка дистрибуция идва с набор от ядра “по подразбиране”. Макар и компилирани за различни конкретни нужди (многопроцесорна поддръжка, завишено ниво на сигурност, 64-битова система и т.н.), те все пак са замислени да поддържат широк набор от процесорни семейства и да работят с разнообразни видове хардуерни устройства (SCSI, USB и т.н.). Вероятно вашата система не използува голяма част от функциите, които ядрото по подразбиране поддържа. Аз например нямам SCSI-устройства, RAID не ми трябва, а процесорът ми е Celeron. Това са само малки примери. В действителност, нещата са много по-сложни. Ако ограничите ядрото да включва в себе си само хардуера, който ползувате, ще получите по-малък файл, оптимизиран за конкретната система. Излишно е да обяснявам, че това ще ускори и подобри като цяло работата на вашия Linux, особено ако имате по-екзотичен хардуер.

Понякога компилирането на собствено ядро се налага и поради публикуването на нови кръпки за подобряване на сигурността или заради излизането на подобрения във файловите системи, както и на нови версии на драйверите за отделни устройства.

Конфигуриране на ядрото

Обикновено това е най-интересната част, защото ви засяга пряко. Няма как да участвате в самата компилация, нали:-)

И така, 'cd /usr/src/linux'. За да конфигурирате ядрото, можете да използвате три режима:

make config - пита за всичко наред и нямате възможност да се върнете назад, ако нещо объркате (работи изцяло в текстов режим и е предназначен за мазохисти:));

make menuconfig - удобно и подредено меню в текстов режим (предпочитано от мнозина); можете да разлеждате и задавате опциите в произволен ред;

make xconfig - графичен конфигуратор, аналог на make menuconfig, но предназначен за работа с X (с малко объркваща подредба);

Какъв е принципът?

ßдрото може да съдържа зададената от вас функция или драйвер за устройство като 1) *вграден вътрешен код (y),* като 2) *външен модул (m),* който да се зарежда в паметта при поискване или 3) *изобщо да не се компилира (n),* ако например съвсем не ви трябва.

И така, да започваме!

Ние ще конфигурираме едно примерно ядро за стандратна настолна система, така че не очаквайте да бъдем изчерпателни. Важното е да схванете принципа.

make menuconfig

Стартира се конфигурационният скрипт. Появява се едно обширно меню. Опции, опции, опции...

Предлагам ви един примерен подход към конфигурирането на ядрото. Разбира се, че всичко зависи от съответния хардуер и именно затова не мога да предложа никакво универсално решение. Конкретната задача изисква конкретно решение. Ето защо, аз също подхождам конкретно, а означенията *[y/m/n]* са просто препоръки и по никакъв начин не гарнатират правилната и оптималната работа на системата, ако се съобразите с тях. Те са просто едно от възможните решения.

Компилация на Linux-ядрото, част II: Първите стъпки

Code maturity level options

Prompt for development and/or incomplete code/drivers [y]

Включете го! Така ще имате възможност да добавите необходимата ви поддръжката на някои допълнителни компоненти като звуковата платка или фреймбуфера, нуждаещи се от драйвери, които са все още в процес на разработка или не са част от официалния сорс на ядрото.

Loadable module support

Enable loadable module support [y]

Също е необходимо, ако ще компилирате модули, а това е почти винаги необходимо. Поне в нашия случай. Имате само две възможности - до го изключите или да го добавите. Второто е наложително.

Set version information on all module symbolls [y]

Не е препоръчително да го изключавте, ако ще компилирате модули.

Kernel module loader [y]

Добре е да включите и тази функция. При всяко зареждане ядрото ще се опитва да зареди необходимите модули от само себе си, изпълнявайки програмата modprobe.

Processor type and features

Тук избирате семейството, към което спада вашият процесор. ßдра, предназначени за *Pentium Pro* и по-високи поколения процесори, няма да работят на машини, оборудвани с 486 и 386 чипове. Но ядро за 386 процесор, ще работи на всички. Избирането на конкретна процесорна архитектура принципно олекотява обема на ядрото и е доста полезна функция.

Math emulation (CONFIG_MATH_EMULATION) представлява емулация на копроцесор. Функция, която ви е необходима, единствено ако процесорът ви е от рода на 486SX или 386, които все още нямат копроцесор за изчисленията с плаваща запетая. От 486DX нагоре вече всички чипове са снабдени с копроцесор и софтуерната емулация на липсващото FPU не ви е необходима.

Ако системата ви е еднопроцесорна, няма да ви трябва нищо повече от разрешаването на *MTRR support,* в случай че процесорът ви е поколение Pentium Pro и по-нагоре. Подобрява контрола на достъп до паметта и върши особено добра работа, ако имате PCI или AGP видеоконтролер. Използува се и от приложения като X. Ако не го включите, ще забележите в логовете съобщения за грешка, свързана с mtrr.

Тук е и мястото, където избирате дали ядрото трябва да поддържа многопроцесорна система (CONFIG_SMP) - Symmetric Multi Processing. Ако го включите, ще работи и на еднопроцесорна машина, но ще добави излишен код в ядрото и освен това няма да имате пълноценно управление на захранването. Linux не страда от илюзията, като някои други операционни системи, че управлението на захранването при повече процесори е проста работа.

General setup

Важно място. Може да се каже, че това е командният център на ядрото.

Networking support [y]

Задължително е да разрешите тази функция, независимо дали компютърът ви ще работи в мрежа или не. За Linux поддръжката на мрежа е задължителна дори за правилната работа на повечето привидно “немрежови” програми. Не забравяйте, че дори X се нуждае от мрежа, за да работи правилно.

PCI Support [y]

Ако имате PCI шина на дънната палтка, включете го. Впрочем, кое дъно няма PCI шина?

PCI Access mode [Any]

Оставете го на “Any” за най-сигурно. Би трябвало вашият BIOS сам да разпознае и конфигурира PCI устройствата.

EISA support [n]

Една функция, която едва ли ще ви потрябва. Все пак, ако дъното ви е старо и е базирано на EISA шина, ще трябва да включите това в ядрото. Иначе, можете спокойно да го пренебрегнете.

MCA support [n]

MCA е стандарт, подобен на ISA и PCI. Ще ви трябва, единствено ако дъното ви работи с него. Иначе, вж. по-горе :-)

System V IPC [y]

Необходимо е за работата на някои приложения. Понятие, добре познато на програмистите. Не е пепоръчително да го изключвате, освен ако не знаете точно какво правите.

BSD Process Accounting [y]

Също познато на програмистите. Някои програми се нуждаят от тази функция. По-добре оставете това в ядрото.

Sysctl support [y]

Добавя около 8K код в ядрото, но е добре да се включи, освен ако не се стремите към минимален размер на ядрото, което е наложително, когато е предназначено да работи на една единствена дискета например. Променя динамично някои параметри на ядрото, без да се налага прекомпилиране или дори рестартиране.

Kernel core (/proc/kcore) format [ELF]

Това е подразбиращата се и препоръчителната настройка за всяка съвременна Linux-система. Директорията /proc представлява виртуална файлова система, в която са разположени всички процеси, включително и самото ядро, а ELF (Executable and Linkable Format) е стандартният формат за изпалнимите файлове и библиотеките в Unix.

Kernel support for a.out binaries [y/m]

Не се знае дали ще ви потрябва, но има и малка вероятност да се сблъскате с по-стария формат библиотеки и изпълними файлове за Unix. Компилирайте го за всеки случай, но като модул, за да се зареди, само когато потрябва.

Kernel support for ELF binaries [y]

Включването му ще добави 13K код към ядрото, но е наложително да го имате. Това е стандартният формат за библиотеките и изпълнимите файлове на различни архитектури и операционни системи към момента.

Kernel support fot MISC binaries [y/m]

Отново се отнася до поредния стандарт формати за библиотеки и изпълними файлове. Добре е да го компилирате като модул, за всеки случай.

Power managment support [y]

Управлението на захранването е важно за всяка система. Ще ви дам един пример как можете да го конфигурирате, за да работи добре, без да добавя излишно количество код, което може да се получи, ако за всеки случай включите всички функции.

ACPI support [n]

Отново се отнася до управлението на захранването, но е все още в етап на разработка. Ако го включите, ще добави към 120K код в ядрото. Освен това няма логика да включвате с него и APM. От двата стандарта ще бъде използуван само този, който се зареди първи.

Advanced Power Managment BIOS support [y]

Ignore USER SUSPEND [n]

Enable PM at boot time [y]

Make CPU Idle calls when idle [y]

Enable console blanking using APM [n]

RTC stores time in GMT [y]

Allow interrupts during APM BIOS calls [n]

Use real mode APM BIOS calls to power off [n]

Компилация на Linux-ядрото, част III: Parallel port, IDE...

Parallel port support

Нищо сложно. Компилирайте като модули поддръжката на паралелния порт, *PC-style hardware* и *Use FIFO/DMA if available.* Ако непременно държите, можете да го включите и направо в ядрото. Аз например установих, че старият ми принтер печата много по-бързо, когато lp (модулът за самия принтер), parport и parport_pc са вградени в ядрото. Последната функция ще ускори трансфера между компютъра и устройствата, закачени за парарелния порт, използувайки известния алгоритъм FIFO и включвайки директния достъп до паметта, ако сте го позволили в BIOS.

Plug and play configuration [y]

Непременно ви трябва, включете го в ядрото.

Block devices [y]

Normal PC floppy disk support [y/m]

Досещате се за какво се отнася. Ще ви трябва, ако имате флопи-дисково устройство. А кой няма?

Останалите функции, ако щете ми вярвайте, може и да не ви потрябват, освен ако не са ви нужни за поддръжката на конкретно устройство или на античните 8-битови твърди дискове :)

Някои от параметрите в този раздел обаче са важни и ще им обърнем подобаващо внимание.

RAM disk support [m/n]

Използува се, когато компилирате ядро, предназначено да работи на една дискета. Заделя се известно количество RAM (посочвате го ръчно, обикновено е 4096K), която се използува като виртуално дисково устройство. Ако ядрото е предназначено да се зарежда от харддиск и да работи на стандартен компютър, не ви трябва *RAM disk support.*

Loop device support [m/n]

Параметър, който ще ви трябва, ако искате да използувате един файл, който да се третира като блоково устройство. Колкото и странно да звучи, за Linux всяко хардуерно устройство е просто файл. Възможно е например да създавате флопи-имиджи, преди да сте ги записали върху физическа дискета. Този параметър, както и scsi-емулацията, за която ще говорим по-нататък, ще са ви необходими и в случай, че записвате CD-та и искате да проверите състоянието на ISO 9660 файловата система, преди да я запишете върху физическия носител.

Network Block Device support [m/n]

Този параметър позволява на компютъра да работи като клиент към мрежово блоково устройство. С една дума, представете си, че една и съща машина работи едновременно като сървър и клиент посредством loopback блоково устройство.

ATA/IDE/MFM/RLL support

Също много важен раздел. Ако знаете как да настроите IDE-устройствата си, ще можете да се радвате на сериозно подобрение на дисковата производителност, което влияе на цялата система доста благоприятно. Непременно разрешете този параметър и разгледайте неговия подрзадел.

Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support [y]

Наименованието на този подраздел говори повече от ясно за какво се отнася. От него зависи дали чипсетът ви ще поддържа различни стандарти на комуникация с дисковите устройства. Тъй като опциите са много, ще се спрем само на най-важните от тях.

Include IDE/ATA-2 DISK support [y]

Ако дискът ви не е много стар и разбира се е IDE, а не SCSI, добре ще е да включите този параметър. Отнася се до чипсети, които поддържат стандартите ATA-2 и нагоре.

Use multi-mode by default (y)

Едно важно допълнение към горната опция. Включете го по принцип и най-вече ако често пъти при по-големи трансфери виждате следната грешка:

"*DriveReady SeekComplete Error"*

Include IDE/ATAPI CDROM support [y]

SCSI emulation support [y/m/n]

Как ще отговорите тук, зависи изцяло от вашите конкретни нужди. Ако искате да използувате вашия Linux като платформа за пържене на дискове, ще трябва да отговорите с [y], но да изключите *Include IDE/ATAPI CDROM support*. Така вашите cdrom-устройства ще използуват scsi-емулация, необходима на повечето програми за прогаряне на дискове. Вместо да ги откривате като /dev/hdd и т.н., ще ги намерите като /dev/scd0 и т.н. Т.е. те ще се възприемат от операционната система като scsi-устройства. Важно! За да включите тази опция, ще трябва да отговорите с [y] и на *SCSI generic support* в раздела *SCSI support*.

Следващите опции се отнасят до поддръжката на конкретни чипсети. Ако разпознаете някъде там чипсета на вашия компютър, включете каквото има за него. Аз, например, съм със стария класически Intel 440 BX/ZX чипсет и съм отговорил с [y] само на *Intel PIIXn chipsets support*, както и на *PIIXn tuning supprt*. Но освен за конкретни нужди, тук има и опции, които са важни за всеки.

Generic PCI IDE chipset support [y]

Най-вероятно вашият чипсет е базиран на PCI шина и включването на тази опция е просто задължително.

Generic PCI bus-master DMA support [y]

Също е желателно да го имате. Използуването на DMA (direct memory access) подобрява чувствително работата на дисковите устройства, стига те да го поддържат. По този начин се освобождава централният процесор от натоварването с транфера от твърдия диск към паметта, тъй като дискът получава директен достъп до нея.

Use PCI DMA by default when available [y]

ATA Work(s) In Progress /EXPERIMENTAL/ [n]

Все още в експериментален етап. Ако го включите, можете да опитате функции, които все още се разработват. Ако държите на стабилността и не желаете да рискувате, засега го изключете.

Good-Bad DMA Model-Firmware [n]

Параметър, който не е задължителен и дори се препоръчва да не го включвате. Съществуват "бъгави" фърмуери (малкото парче софтуер, което управлява чипсета на най-ниско ниво) и вашето ядро ще провери дали нямате проблем с поддръжката на DMA, т.е. дали има бъг във фърмуера, който твърди, че дискът поддържа DMA, но всъщност това не отговаря на истината. Интересна опцийка :)

Networking options

Тази секция от конфигурацията на кернела е една от най-големите. Оттук се настройват мрежовите възможности на вашата система. Ако сте начинаещ потребител, най-добре е да оставите настройките по подразбиране - те са определени така, че да нямате проблеми с използването на най-масовите мрежови протоколи. Ако сте напреднал потребител, тогава сигурно много от нещата описани тук са ви известни и добре познати. Това описание е предназначено за не-толкова-напредналите и любопитни потребители, които искат да се запознаят с мрежовите възможности на линукс кернела. Ако искате да използвате реално някои от тези възможности, добре е да потърсите допълнителна информация и софтуер в Мрежата.

Packet socket

Оттук може да разрешите на някои програми да комуникират директно с мрежовите ви устройства, без да използват мрежови протоколи.Това е полезно ако използвате "снифери" - програми, които следят мрежовия трафик на най-ниско ниво.

Packet socket:mmapped IO

Тази опция ускорява директната комуникация с мрежовите устройства

Netlink device emulation

Създава устройства в директорията */dev* за комуникация с мрежовия слой на кернела. Опцията скоро ще бъде премахната, новите кернели използват *netlink* сокети за тази комуникация.

Network packet filtering

Тази опция разрешава филтрирането на пакети. Ще ви трябва, ако искате да използвате системата си за firewall, proxy, gateway т.е. ако ще работи като рутер. Не ви е необходима за обикновен настолен компютър, освен ако не сте параноичен или просто искате да разучите как работи. В някои ситуации малко допълнителна сигурност не е излишна :). Ако я разрешите, трябва допълнително да конфигурирате начина на филтриране в под-секцията "*IP: Netfilter Configuration*"

Network packet filtering debugging

Разрешава изкарването на допълнителни съобщения, следящи работата на пакетния филтър. Не ви е необходима, освен ако не сте програмист и искате да разберете как работи този филтър.

Socket filtering

Позволява филтрирането на отделни сокети. Ако опцията е разрешена, в кернела се добавят две нови *ioctl* системни извиквания, чрез които на даден сокет може да се закачи/откачи допълнителен код, който да следи и контролира данните, преминаващи през сокета. За съжаление засега не се поддържат TCP сокети.

Unix domain sockets

Едни от най-използваните сокети в линукс. Ще ви трябват, дори и да нямате мрежа. Използват се масово за комуникация между процеси, вървящи на една машина.

TCP/IP networking

Това е Интернет! И не само - използва се от много програми, дори и да нямате мрежа. Това е задължителна опция. Ще увеличи кернела с 144К, но си струва :)

IP: milticasting

Позволява на кернела да работи с *IP multicast* пакети. Това са пакети, които имат специални IP адреси, чрез които може да се адресират едновременно група от компютри. Полезна опция, но за съжаление все още малко рутери могат да рутират *multicast* трафик. Ако вашият компютър е в локална мрежа можете да я разрешите - ще увеличи кернела с около 2К, но има някои програми за видео и TV разпръскване, които използват *multicast*. Ако използвате системата за рутер, също ще ви трябва - много протколи за динамично рутиране използват *multicast* пакети за обмен на рутираща информация.

IP: advanced routrer

Ако ще използвате системата си за рутер е добре да разреште тази опция - тя ще ви даде достъп до опциите за настройка на професионалните рутиращи възможности на линукс.

IP: policy routing

Обикновенно рутерите вземат решения за рутиране на пакети на база адреса на получателя на пакета. Чрез тази опция може да се разреши използването и на други критерии при рутирането - адрес на изпращача, полето TOS в IP хедъра и др.

IP: use netfilter MARK value as routing key

Тази опция ви позволява да използвате специална MARK стойност при рутиране. Такава стойност може да се присвои на някои пакети чрез използване на *iptables* филтри

IP: fast network address translation

Позволява на кернела да променя адреса на получателя и изпращача на пакетите, които рутира. Правилата, по които става това могат да се зададат чрез *iptables.*

IP: equal cost multipath

Тази опция разрешава в рутиращата таблица към един запис да се присвоят няколко пътя. Така ако пакет трябва да се рутира на базата на този запис, кернела избира по случаен начин един от тези пътища.

IP: use TOS value as routing key

Позволява при рутиране да се използва стойността на полето TOS в IP хедъра на пакета. Това поле определя "типа на услугата" на пакета - така данни с по-голям приоритет могат да се рутират през по-бързи пътища.

IP: verbose routing monitoring

Разрешава изкарването на допълнителна информация за рутиращия процес. Полезна опция ако сте параноичен и искате да имате информация за необичайните пакети, преминали през рутера ви :)

IP: large routing tables

Ако във вашата рутираща таблица ще има повече от 64 записа, добре е да разрешите тази опция. Това ще ускори работата с големи рутиращи таблици.

IP: kernel level autoconfiguration

Позволява на кернела да използва специални протоколи за автоматично конфигуриране на IP адреса. Тази опция е полезна предимно за бездискови машини, които се конфигурират през мрежа. Друго полезно приложение е за динамично разпределяне на наличните IP адреси в мрежата.

IP: DHCP support

Широко използван протокол за автоматично конфигуриране на IP адреси. Позволява обмен и на допълнителна конфигурационна информация, както и присвояване на IP адрес за определено време

IP: BOOTP support

Подобен на DHCP протокол, позволява обмен на допълнтелна конфигурационна информация. Тези два протокла работят на Layer 3 - използват broadcast UDP пакети.

IP: RARP support

Протокол за автоматично конфигуриране на IP адреси, работещ на Layer 2.

IP: tunneling

Позволява капсулирането на IP в IP пакети. На пръв поглед - безмислено :). Полезна опция ако искате дадена машина да се мести от мрежа в мрежа, без да си сменя IP адреса, като "виртуално" остава свързана към една мрежа. Намира приложение при свързването на мобилните компютри към интернет.

IP:GRE tunnels over IP

Позволява капсулирането на произволен протокол в IP. Намира практическо приложение при капсулацията на IPv6 в IPv4 пакети - така може да се изгради IPv6 мрежа, използвайки съществуваща IPv4 мрежа. Предимството на този вид капсулация е че позволява разпространението на multicast пакети през тунела.

IP: broadcast GRE over IP

Позволява разпространеието на broadcast пакети през тунела. Така с GRE тунели може да се изгради виртуална WAN мрежа, която да се държи като ethernet, но физически да е разпокъсана и да използва интернет за преносна среда.

IP: multicast routing

Позволява рутирането на *multicast* пакети. Това са пакети, които имат специалени IP адреси, чрез които могат да се адресират едновременно група от компютри.

IP: PIM-SM version 1 support

Разрешава поддръжката на версия 1 на PIM. Това е един от най-използваните протокли за динамично *multicast* рутиране. За да използвате този протокол, се нуждаете и от допълнителен софтуер извън кернела.

IP: PIM-SM version 2 support

Разрешава поддръжката на версия 2 на PIM протокола.

IP: ARP daemon support

Разрешава използването на допълнителен, външен ARP демон, който да се грижи за ипзращането/получаването на ARP заявки за адреси, лидпсващи в ARP кеша на кернела. Ако тази опция е включена, в ARP кеша ще се съхраняват до 256 адреса. Препоръчително е тази опцията да не се използва.

IP: TCP explicit congestion notification support

Позволява на рутера да съобщава на клиентите за претоварвания по мрежата, като вдига ECN флага в някои от пакетите си. Има рутери, които не работят добре с този флаг, затова по добре не го разрешавайте.

IP: TCP syncookie support

Осигурява някаква защита по време на "*SYN flood*" атака. Когато машината е под такава атака, TCP стека може да използва специален "*SYN cookie*" протокол за легитимиране на потрбителите, които могат да осъществяват връзка. Ако разрешите опцията, тази функционалност остава забранена по подразбиране - трябва да я разрешите чрез /proc :

echo 1 > /proc/sys/net/ipv4/tcp_syncookies

след стартирането на машината.

The IPv6 protocol support

Включва поддръжка за версия 6 на IP протокола. За да използвате реално IPv6, трябва да се снабдите и с последните версии на всички мрежови програми.

Kernel httpd acceleration

Разрешава вградения в кернела HTTP сървер. Това е ограничен web сървер, който може да изпълнява http заявки, но не може да работи с cgi скриптове. Полезна опция, ако използвате машината за web сървер и искате да експериментирате - това значително ще подобри производителността на сървера. Работата на вградения сървер може да се синхронизира с външен сървер, към които да се пренасочват заявките, които не могат да се обслужат.

Asynchronous Transfer Mode (ATM)

Разрешава поддръжката за ATM мрежи. За да използвате линукс в такива мрежи, ви трябват допълнителни програми, и разбира се ATM устройства.

Classical IP over ATM

Позволява капсулация на IP в ATM пакети и поддръжка на ARP протокола за ATM мрежи.

Do NOT send ICMP if no neighbor

Указва на кернела да не изпраща ICMP пакети за събитието "*host unreachable*", когато адреса на получателя не е намерен в ATMARP таблицата.

LAN emulation support (LANE)

Включва код за емулация на *ethernet LAN* мрежа през ATM мрежа.

Multi-Protocol over ATM support (MPOA)

Позволява изграждането на виртуални връзки между ATM гранични устройства. Тази опция намира практическо приложение при свързването на два (или повече) рутери през ATM мрежа, като рутирания трафик минава през виртуалната връзка - така се избягва рутирането през ATM рутери.

802.1 Q VLAN support

Включва поддръжката за този вид виртуални *Layer2 ethernet* мрежи. Чрез специални тагове преди *ethernet* хедъра, една физическа *ethernet* мрежа можа да се раздели на няколко логически мрежи.

The IPX protocol

Включва поддръжка за IPX протолкола на Novell. Това е протокол, който изпълнява аналогични функции на IP. Използва се и в някои windows мрежи.

IPX: Full internal IPX network

Позволява кернела да емулира "вътрешна" IPX мрежа. Така ако машината ви е свързана към повече от една IPX мрежа, за другите машини вашия линукс ще представлява отделна мрежа. Полезна опция, ако

компютъра ви е някакъв сървер и искате да се вижда по един и същи начин от другите мрежи.

Appletalk protokol support

Включва поддръжката за фамилията мрежови протоколи на Apple. Това са протоколи, използвани предимно при Macintosh машините. Поддържат се EtherTalk и LocalTalk протоколите, но за да ги използвате се нуждаете от допълнителен софтуер.

DECnet support

Включва поддръжка за мрежовия протокол на Digital. Не е широко разпространен.

DECnet: SIOCGIFCONF support

Променя *SIOCGIFCONF* системното извикване, като добавя поддръжка за DECnet мрежи - това може да наруши работата на някои програми, използващи това системно извикване, така че внимавайте.

DECnet: router support

Позволява на машината ви да работи като DECnet рутер.

DECnet: use FWMARK value as routing key

Разрешава използването на *FWMARK* при рутирането. Такава стойност може да се присвои на някои пакети чрез *iptables*

802.1d Ethernet Bridging

Добавя поддръжка за *ethernet bridge* : прехвърляне на layer 2 пакети между различни мрежови сегменти. Това ви позволява да изградите голяма *ethernet* мрежа, състояща се от няколко физически отделни сегмента.

CCITT X.25 packet layer

Подръжка за X25 мрежи. Тази опция включва имплементацията на PLP (Packet Layer Protocol), протокол от високо ниво използван в тези мрежи.

LAPB Data link driver

Включва поддръжка за LAPB, протокол от ниско ниво използван в X25 мрежите. Повечето X25 карти имат вградена хардуерна поддръжка за този протокол, затова тази опция ви е необходима само ако вашата X25 карта няма такава.

802.2 LLC

Този протокол позволява да изградите X25 мрежа, използвайки *ethernet* или *token ring* за преносна среда.

Frame diverter

Позволява ви да контролирате целия трафик по мрежата на ниско ниво - дори и пакетите, които не са предназначени за вашата машина. Полезна опция, ако искате да направите *transperant proxy* (има и други начини за това), ограничение на трафика, и др. - могат да се намерят много полезни приложения :)

Acorn Econet/AUN protocols

Малко използван, стар и бавен протокол. Може да работи със специални *Econet* карти, или през *Ethernet* . Използван е при *Acorn* компютрите за достъп до файлови и печатни сървери.

AUN over UDP

Позволява използването на *AUN* протокола през *ethernet*, чрез капсулиране в *UDP/IP* пакети.

Native Econet

Позволява използването на *AUN* протокола през оригинални *econet* мрежови карти.

WAN router

Дава възможност да направите *WAN* рутер. Ако имате WAN мрежа, изградена чрез *X.25, frame relay* или друга подобна преносна среда и не искате да купувате някой скъп *WAN* рутер, може да си свършите работата с една линукс машина и с тази опция .

Fast switching

Позволява директен трансфер на данни между мрежовите карти, като се заобикаля голяма част от мрежовия слой на кернела - това води до значително увеличаване на скоростa, но за сметка на някои екстри като *Packet filtering* , които няма да може да използвате.

Forwarding between high speed interfaces

Позволява директен трансфер между мрежовите карти по време на големи мрежови натоварвания. Това позволява рутера ви да остане работещ, дори когато мрежовия трафик е особенно голям и не може да се обработи от машината ви. Тази опция зависи от хардуера - трябава да се поддържа от мрежовите ви карти.

IP:Netfilter Configuration

В тази подескция се настройват параметрите на пакетния филтър за IPv4 пакети. За да имате достъп до нея, трябва да сте разрешили опцията "*Network packet filtering* " в секция "*Networking Options*".

Connection tracking

Позволява на кернела да следи зависимостите между пакетите, които преминават през рутера - така могат да се филтрират вички пакети, които принадлежат на дадена връзка. Това е задължителна опция, ако ще използвате някаква форма на NAT(Network Address Translation) - тук влиза и Masquerading, той също е вид NAT. Друго полезно приложение е филтриране на база състояние на връзката - например могат да се отрежат всички tcp пакети, за които няма ESTABLISHED връзка.

FTP protocol support

Проследяването на FTP връзки е малко по-особенно, затова се нуждае от допълнителен код. Ако ще използвате NAT и искате да имате FTP - разрешете тази опция

IRC protocol support

Включва поддръжка за DCC протокола на IRC, за клиенти които се намират зад NAT рутер. Това е протокол, който се използва за директна комуникация между два IRC клиента, без трафика да преминава през IRC сървер. Ако ще използвате NAT и IRC - включете опцията.

Userspace queueing via NETLINK

Позволява на кернела да пренасочва пакети към външни програми

IP tables support

Това е интерфейс към пакетния филтър, използван в кернелите от серия 2.4. Щом сте стигнали до тук, този интерфейс ще ви трябва - позволява ви да конфигурирате динамично работата на филтъра. Не ви е необходим ако искате за съвместимост да използвате някои от старите интерфейси (*ipchains* или *ipfwadm*).

limit match support

Позволява ви да контролирате скоростта, с която се прилага даден филтър. Полезна опция за защита от DoS (Denial of Service) атаки - например може да ограничите по време пакетите от непознати IP адреси. Друго полезно приложение е за ограничаване на логовете - ако не искате да логвате целия трафик, може да укажете да се логват само определен брой пакети в минута.

MAC address match support

Позволява ви да филтрирате пакети на базата на техния ethernet адрес.

netfilter MARK match support

Позволява филтриране на базата на специална MARK стойност, която може да се присвои на някои пакети.

Multiple port match support

Позволява филтрирането на TCP и UDP пакети на базата на група портове. Ако опцията не е включена, ще можете да филтрирате само за един порт.

TOS match support

Позволява филтриране на базата на полето TOS (Type of Service) в IP хедъра на пакета.

AH/ESP match support

Позволява филтриране на база SPI (Security Parameter Index) - това се специални полета, които осигуряват криптиране и защита на данните, пренасяни с IPSec пакети. IPSec (IP Security protocol) е допълнителен хедър, прикачен към IP хедъра, който може да съдържа AH (Authentication Header) и/или ESP (Encapsulating Security Payload), които се грижат за защитата и криптирането на пакетите. Този механизъм за защита се използва при изграждането на VPN мрежи, затова ако през вашия рутер ще минава такъв трафик, и искате да го филтрирате - разрешете опцията.

LENGTH match support

Позволява филтриране на база дължината на пакета.

TTL match support

Позволява филтриране на база полето TTL (Time To Live) в IP хедъра.

tcpmss match support

Позволява филтриране на базата на стойноста MSS (Maximum Segment Size) - това е стойност, която се договаря при откриването на една TCP връзка и определя максималната големина на пакетите за тази връзка.

Connection state match support

Позволява филтриране на пакети на база състояние на връзката - например могат да се филтрират всички TCP пакети с данни, за които няма установена връзка.

Unclean match support

Позволява откриването на "странни" пакети - за критерии се използват определени полета в хедърите на IP, TCP, UDP и ICMP пакетите.

Owner match support

Позволява филтрирането на локално генерирани пакети на базата на потребителя, групата, процеса или сесията на източника на пакета

Packet filtering

Включва в кернела FILTER таблицата, където се съхраняват всички филтри, които могат само да филтрират пакети - нямат право да ги променят. Тази таблица работи с LOCAL_INPUT, LOCAL_OUTPUT и FORWARD веригите.

REJECT target support

Позволява на даден филтър да укаже изпращането на ICMP съобщение при отхвърлянето на филтриран пакет.

MIRROR target support

Позволява на даден филтър да укаже филтрираните пакети да се връщат обратно на изпращача.

Full NAT

Включва в кернела NAT (Network Address Translation) таблицата, където се съхраняват филтрите които могат да променят адресите на изпращача или получателя на пакетите. Таблицата работи с PRE_ROUTING, POST_ROUTING и LOCAL_OUTPUT веригите.

MASQUERADE target support

Това е вид source NAT - позволява на даден филтър да укаже промяна на адреса на изпращача на филтрираните пакети. Използва се предимно при *dialup* връзки - когато новия адрес на пакета зависи от текущата *dilaup* сесия. При прекъсване на връзката, работата на филтъра се спира.

REDIRECT target support

Това е вид destination NAT - позволява на даден филтър на укаже промяна на адреса на получатля на филтрираните пакети. Новият адрес е един от локалните адреси на рутера - така пакетите се насочват към рутера, вместо да се рутират навън. Тази опция е полезна ако искате да направите *transparent proxy*.

Basic SNMP-ALG support

Включва поддръжка за ALG (Application Layer Gateway) за SNMP пакети на устройства, намиращи се зад NAT рутер. Това позволява на рутера да променя както IP адресите на изпращача/получателя, така и IP адресите в самия SNMP пакет. Така една SNMP станция може да работи с устройства, скрити зад NAT рутер, които имат еднакви IP адреси.

Packet mangling

Включва в кернела MANGLE таблицата. В нея се съхраняват филтрите, които могат да променят самите пакети. Таблицата работи с PRE_ROUTING и LOCAL_OUTPUT веригите.

TOS target support

Позволява на даден филтър да променя TOS стойноста в IP хедъра на филтрираните пакети.

MARK target support

Позволява на даден филтър да присвоява специална MARK стойност към филтрираните пакети. Тази стойност може да се използва при рутирането на пакета.

LOG target support

Позволява да се правят филтри, които записват хедъра на филтрираните пакети в системния лог.

ULOG target support

Позволява да се правят филтри, които копират филтрираните пакети към външна програма посредством *netlink* сокети.

TCPMSS target support

Позволява да се правят филтри, които променят MSS (Maximum Segment Size) стойноста за TCP връзка.

ipchains (2.2 - style) support

Включва в кернела *ipchains* интерфеса към пакетния филтър, използван в кернелите от серия 2.2

ipfwadm (2.0 - style) support

Включва в кернела *ipfwadm* интерфеса към пакетния филтър, използван в кернелите от серия 2.0. Тези две опции ви трябват само за съвместимост - ако имате програми, писани за старите версии на кернела, и искате да ги използвате.

IPv6: Netfilter Configuration

В тази подсекция се настройват параматрите на пакетния филтър за IPv6 пакети. Те имат същото значение, както и при IPv4 пакетите, затова за повече подробности погледнете подсекцията "*IP:Netfilter Configuration"*.

QoS and/or fair queueing

В тази подсекция се настройват параметрите на така наречения "*scheduler*", или разпределител. Когато пакетите вече са рутирани от рутера, т.е. - определен е техния изходен интерфейс и мрежовия адрес на следващия рутер, те се нареждат в една опашка и чакат да бъдат изпратени. Всеки интерфейс си има такава опашка. Ако интерфейса е към бърза мрежа и трафика през него не е голям, пакетите почти не се задържат в опашката - веднага се изпращат. Ако обаче трафика е голям и мрежата е бавна, в опашката могат да се съберат доста пакети. Работата на този разпределител е да реши кои пакети от опашката ще бъдат изпратени първи. Оттук се настройват критериите, по които работи разпределителя.

QoS and/or fair queueing

Ако забраните тази опция, разпределителя ще използва стандартната логика при работата си: първия пакет, дошъл в опашката ще се изпрати първи към мрежата - FIFO (First In First Out). Ако разрешите опцията, ще може да изберете алтернативни алгоритми на работа. Полезна опция, ако искате да контролирате трафика през даден интерфейс, или да използвате QoS (Quality of Service) - пакети с по-висок приоритет да се обсужват с предимство. Обърнете внимание, че може да използвате комбинации от тези алгоритми, което прави настройката на разпределителя изключително гъвкава.

CBQ packet scheduler

Разрешава използването на алгоритъма CBQ (Class-Based Queueing). Този алгоритъм разпределя чакащите пакети в дървовидна йерархия, спрямо определени класове. Позволява използването на други алгоритми за управление на различните клонове от дървото. Той е един от най-използваните и гъвкави алгоритми.

CSZ packet scheduler

Разрешава използването на алгоритъма CSZ (Clark-Shenker-Zhang). Не е толкова гъвкав като CBQ, но е единствения алгоритъм който може да гарантира обсужването на важен трафик в реално време. Той заделя предварително части от пропускателната лента на мрежата за видовете трафик, които трябва да гарантира. Останалата част се разпределя между по-маловажния трафик.

ATM pseudo-scheduler

Позволява използването на пакетния разпределител за пакети, които преминават през ATM мрежа чрез виртуални връзки.

The simplest PRIO pseudo-scheduler

Позволява работата на PRIO алгоритъма. Той е подобен на стандартния FIFO алгоритъм, разликата е че дава възможност трафика да се раздели в няколко вътрешни опашки. Разделянето става посредством потребителски филтри. Полезен алгоритъм, ако искате да приоритизирате трафика чрез ваши критерии, независими от полето TOS на IP хедъра.

RED queue

Позволява използването на RED (Random Early Detection, понякога наричан и Random Early Drop)

алгоритъм. Той се грижи опашката никога да не достигне абсолютния си максимум, т.е. да не се напълни. Алгоритъма предсказва кога може да се стигне до препълване и отхърля по случаен начин част от пакетите, за да го предотврати.

SFQ queue

Позволява използването на SFQ (Stochastic Fairness Queueing) алгоритъм. Този алгоритъм разаделя трафика на отделни потоци, като в повечето случаи един поток се асоциира с една UDP или TCP сесия. SFQ изпраща равномерно пакети от всеки един поток - така се избягва заемането на целия трафик от една TCP сесия. Има най-малко изисквания към системните ресурси.

TEQL queue

Позволява използването на TEQL (True Link Equalizer) алгоритъм. Чрез него може да направите "*load balancing*" на трафика си. Той ви позволява да обедините няколко физически интерфейса в един логически, като се грижи за рапределянето на трафика между физическите интерфейси.

TBF queue

Позволява използването на TBF (Token Bucket Filter) алгоритъм. Чрез него може да направите изкуствено ограничение да трафика в опашката. Може да зададете максимална лимит на трафика, като алгоритъма ще отхвърля всички пакети, когато този лимит се достигне. TBF може да обслужва и кратковременни пикове които надвишават лимита, ако преди тях е имало моменти с малък трафик.

GRED queue

Позволява използването на GRED (Generic Random Early Detection) алгоритъм. Подобен е на RED алгоритъма, разликата е че предварително разделя трафика в няколко вътрешни опашки на база полето DiffServ от IP хедъра (това поле заема мястото на полето TOS).

Diffserv field marker

Позволява изплзването на полето DiffServ от IP хедъра при определяне приоритета на пакетите. Това поле заема мястото на полето TOS.

Ingress Qdisc

Позволява контролирането на влизащия в опашката трфик - може да зададете лимит, над който пакетите няма да се приемат в опашката, а ще бъдат отхвърляни.

QoS support

Тази опция позволява използването на RSVP (Resource Reservation Protocol) и други подобни протоколи, чрез които може да се резервира част от пропускателната лента на мрежата за определен трафик.

Rate estimator

Позволява на керенла да определя какъв е моментния трафик на мрежовите устройства. Тази информация се използва от QoS алгоритмите.

Packet classifier API

Позволява ви да изберете критериите спрямо които може да разделяте пакетите в опашката на различни класове. Това разделяне на класове се използва от CBQ алгоритъма при построяването на дървовидната йерархия.

TC index classifier

Класифицира пакетите спрямо DiffServ полето в IP хедъра. Когато пакета пристигне в рутера, кернела създава една вътрешна структура, sk_buff. В едно от нейните полета - tc_index, се копира стойноста на DiffServ от IP хедъра. При обработката на пакета, кернела може да промени стойноста на tc_index - т.е. този класификатор работи на база променената стойност, а не оригиналния DiffServ на пакета.

Routing table based classifier

Пакетите се класифицират спрямо записа в рутиращата таблица, който е бил използван при рутирането им.

Firewall based classifier

Класифицира пакетите спрямо специална MARK стойност, присвоена от пакетния филтър.

U32 classifier

Класифицира пакетите спрямо адреса на получателя на пакета.

Special RSVP classifier

Класифицира IPv4 пакети спрямо ресурса, заделен за тях чрез RSVP (Resource Reservation Protokol)

протокола. Чрез този протокол може да се заделят минимална и максимална част от пропускателната лента на мрежата за пакети от определена връзка.

Special RSVP classifier for IPv6

Класифицира IPv6 пакети спрямо ресурса, заделен за тях чрез RSVP протокола.

Trafic policing

Позволява ви да ограничавате трафика в дадени клонове от CBQ дървото.

Network device support

В тази секция се настройват мрежовите устройства, поддържани от линукс кернела. Тук трябва да изберете подходящия драйвер за вашата мрежова карта, оттук може да конфигурирате поддръжката за някои серийни протоколи като PPP. Ако не намерите подходящ драйвер, не се отчайвайте - потърсете в Мрежата. Ако вашето устройство е широко използвано, почти сигурно е че някой вече е написал линукс драйвер за него.

Network device support

Оттук се разрешава на линукс да използва мрежови устройства. Ако ще свързвате вашия комютър с други машини (в днешно време това е направо задължително :)), разрешете опцията.

Dummy net driver support

С тази опция разрешавате на кернела да използва "фалшиви" интерфейси - това са изцяло софтуерни интерфейси, които емулират работата на физическите. Информацията, изпратена към такъв интерфейс се губи - реално не отива никъде. Опцията е полезна предимно за тестове. Ако искате да имате повече от един "фалшив" интерфейс, трябва да компилирате тази поддръжка като модул.

Bonding driver support

Тази опция ви позволява да обедините няколко физически *ethernet* интерфейса към друга машина в един логически. Така може да правите "load balancing" на трафика към другата машина. Разбира се, другия край също трябва да поддържа тази опция.

EQL (serial line load balancing) support

Тази опция ви позволява да обедините няколко физически сериини линии в една логческа линия. Полезна опция, ако имате повече от една PPP или SLIP връзка към друга машина и искате да правите "load balancing" на трафика по тези линии. Опцията трябва да се поддържа и от другия край на линиите.

Universal TUN/TAP device driver support

Включва поддръжката за TUN/TAP виртуални тунелни интерфейси. Тези интерфейси могат да емулират работата на физически интерфейс, като вместо да получават/изпращат пакети към мрежа, работят с потребителски програми. TUN интерфейса работи с IP пакети, използва се за емулиране на PPP интерфейс. TAP интерфейса работи с *ethernet* пакети - емулира *ethernet* интерфейс. Тази опция се използва за *tunneling* - капсулиране на произволен протокол в IP или *ethernet* протокол чрез използване на потребителски програми.

Ethertap network tap

Разрешава използването на виртуален TAP интерфейс, чрез които се емулира *ethernet* интерфейс. Тази опция скоро ще бъде премахната, вместо нея трябва да се използва опцията *Universal TUN/TAP device driver.*

General Instruments Surfboard 1000

Ако имате Surfboard 1000 вътрешен кабелен модем, това е драйвера за него.

FDDI driver support

Разрешава използването на FDDI (Fiber Distributed Data Interface) устройства. Това е висикоскоростен интерфейс, използван в някои локални мрежи. Поддържат се следните карти:

Digital DEFEA and DEFPA adapter support

Sys Konnekt FDDI PCI support

HIPPI driver support

Разрешава използването на HIPPI (HIgh Performance Parallel Interface) устройства. Чрез този интерфейс може да се достигне скорост от 1.6 Gbit/sec, използва се за изграждане на клъстери и свързване на суперкомпютри. Поддържа се следната карта:

Essemtial RoadRunner HIPPI PCI adapeter support

Use large TX/RX rings

Разрешава на HIPPI драйвера да задели допълнителна памет (до 2 MB) за ускоряване на мрежовите операции.

PLIP (parallel port) support

Включва поддръжка за PLIP (Parallel Line Internet Protocol) проткола. Използва се за свързване на компютри чрез паралелния порт.

PPP (point-to-point protocol) support

Включва поддръжка за PPP (Point to Point Protocol) протокола. Използва се за свързване на компютри чрез серийния порт. Това е един от най-използваните протоколи при *dial-up* връзки, затова ако искате да имате интернет през модем и телефонна линия, разрешете опцията.

PPP multilink support

Тази опция ви позволява да обедините няколко физически PPP връзки в една логическа връзка. За да я използвате, трябва да се поддържа и от другия край на връзката.

PPP filtering

Позволява да се филтрират пакети, преминаващи през PPP интерфейса. Това филтриране се използва за определяне на критерия за "свободна" линия - част от пакетите с данни могат да се филтрират като "маловажни".

PPP support for async serial ports

Разрешава използването на PPP през асинхронни серийни портове - това са стандартните COM портове, използвани за връзка с модем.

PPP support for sync tty ports

Разрешава използването на PPP през синхронни серийни портове - това са висикоскоростни портове, използвани при ISDN връзки или наети линии.

PPP Deflate compression

Включва поддръжката за *Deflate* компресия на PPP пакетите.

PPP BSD-Compress compression

Включва поддръжката за *BSD* компресия на PPP пакетите. Използва се *LZW* алгоритъм за компресия.

PPP over Ethernet

Включва поддръжката за енкапсулация на PPP пакети в *Ethernet* пакети. Това позволява изграждането на PPP връзки през *ethernet* мержи.

PPP over ATM

Включва поддръжката за енкапсулация на PPP пакети в ATM пакети. Това позволява изграждането на PPP връзки през ATM мержи.

SLIP (serial line) support

Включва поддръжка за SLIP (Serial Line Internet Protocol) протокола. Не е широко разпространен, вместо него се използва PPP.

CSLIP compressed headers

Разрешава използването на CSIP (Comress SLIP) - компесират се TCP/IP хедърите на пакетите.

Keepalive linefill

Добавя към SLIP драйвера допълнителна функционалност за следене на линията.

Six bit SLIP encapsulation

Указва на SLIP драйвера да кодира всички данни чрез печатни ASCII символи (без да се използват контролни ASCII символи). Тази функционалност е необходима ако пакетите преминават през серийни мрежи, които не пропускат всички ASCII символи.

Fibre Channel driver support

Разрешава използването на *Fiber Channel* устройства. Това са устройства за вискоскоростни трансфери, работещи с *Fiber Channel* протокола - използва се за връзка между компютъра и външни запомнящи устройства с голям капацитет. Поддържа се следното устройство:

Interphase 5526 Tachyon chipset based adapter support

Red Creek Hardware VPN

Включва драйвера за *Red Greek* устройство, използвано за изграждане на VPN (Virtual Private Network) мрежи.

Trafic Shaper

Разрешава използването на *Trafic Shaper* - това е виртуално мрежово устройство. Чрез него може да се ограничи трафика, преминаващ през физически интерфейс. Изпълнява аналогични финкции на CBQ алгоритъма от QoS секцията. За да го използвате се нуждаете от допълнителен софтуер.

Подсекции

В подсекциите на тази секция са описани драйверите за всички мрежови устройства, поддържани от линукс кернела. Ако не намерите драйвер за вашето устройство - потърсете в Мрежата. Драйвера може да не е включен в кернела поради ред причини - ако не е под GPL лиценз, ако е в ранен етап на разработка, или просто автора предпочита да го разпостранява отделно. Обърнете внимание, че за линукс е важен не производителя на устройството, а производителя на "чипсета" - това е чипа, който върши основната работа. В повечето случаи фирмите производители са различни. За да разберете кой е направил чипсета погледнете документацията на устройството, или направо погледнете платката - вижте какво пише върху най-големия чип от нея. След като разберете производителя и модела на вашия чипсет, можете да изберете подходящия драйвер за него.

ARCnet devices

Тук са драйверите за *ARCnet* устройствата. Използват се за изграждане на *ARCnet* мрежи - подобни по фунционалност на *Ethernet*. Могат да работят на 2.5 Mbits или 100 Mbits.

Appletalk devices

Тук са драйверите за *AppleTalk* устройства. Слабо разпространени устройства на Apple за изграждане на локални мрежи.

Ethernet (10 or 100 Mbit)

Тук са драйверите за обикновенни *Ethernet* устройства. Това са най-разпространените мрежови карти, използват се масово за изграждане на локални мрежи - най-вероятно и вашата карта е такава.

Ethernet (1000 Mbit)

Тук са драйверите за бързи *Ethernet* устройства. Не са толкова разпространени като обикновенните *Ethernet* карти, главно поради по-високата им цена. Използват се за изграждане на бързи локални мрежи.

Wireless LAN (non-hamradio)

Тук са драйверите за безжични LAN устройства. Безжичните мрежи са една популярна алтернатива на традиционните мрежи. В последно време стават все по-разпространени, цената на тези устройства пада, а скоростта им се увеличава.

Token Ring devices

Тук са драйверите за *Token Ring* устройства. Слабо разпространени устройства на IBM за изграждане на локални мрежи.

Wan interface

Тук са драйверите за WAN (Wide Area Networks) устройства. Използват се за изгражадне на бързи мрежи за големи разстояния, състоящи се от по-бавни LAN мрежи.

PCMCIA network device support

Тук са драйверите за *PCMCIA* и *CardBus* мрежови устройства. Използват се предимно при преносимите компютри.

ATM devices

Тук са драйверите за ATM (Asynchronous Transfer Mode) устройства. Използват се за изграждане на високоскоростни LAN и WAN мрежи

Следконфигурационна настройка, самата компалиация, lilo

Следконфигурационна настройка

Звучи малко тафталогично. Нали преди малко точно това правехме - настройвахме? Все пак, след като сме приключили с 'make menuconfig' и сме записали избраните от нас опции в един скрит файл '.config',

който можете да разгледате съвсем безпроблемно, може да искаме да пипнем тук и там, преди да извикаме компилатора да си свърши работата.

Един пример. От един и същи сорс можем да си направим безбройно количество ядра. Всеки път, когато компилираме ново ядро, то ще се именува по един и същи начин и неговите модули ще се инсталират в една и съща директория (/lib/modules/x.x.x). Така, модулите на различните ядра ще се припокриват и ще настане пълна каша. В крайна сметка, единственото работещо ядро ще е само това, което сме компилирали последно. Как да накараме модулите да отиват в отделни директории? Като преименуваме ядрото. А как да го преименуваме? Като редактираме файла 'Makefile' в директорията '/usr/src/linux' преди компилацията.

В първите три реда на този файл, който задава базовите настройки за изработването на новото ядро, е описана именно версията на ядрото. Третият ред - 'EXTRAVERSION' - е празен. В него разработчиците на дистрибуции обикновено вписват номера на кръпката, която правят на ядрото. Вече казахме, че те често пъти добавят нови драйвери или правят собствени корекции в изходния текст. Това не означава, че и ние не можем да го използуваме, разбира се. Именно в него можем да добавим нещо, с което да отличим нашето ядро от останалите. Например:

EXTRAVERSION = -1MyLinux

Така, след компилацията, нашето ядро ще се означава като '2.4.18-1MyLinux', неговите модули ще отидат в директорията '/lib/modules/2.4.18-1MyLinux' и няма да припокрият модулите на другите ядра, които ползуваме. Ето как можем съвсем безопасно да експериментираме свободно с компилацията на ядрото, без да се страхуваме, че нещо ще повредим в системата, ако случайно новото ядро не проработи.

Забележка! *Името на ядрото не е тъждествено с файла, който представлява самото ядро и който можем да наречем както си поискаме. Името на ядрото е записано в неговите хедъри, благодарение на които програмите разпознават неговата версия.*

Компилация на ядрото и модулите, инсталиране на модулите

Дотук беше нашата работа. Сега следват няколко магически команди и с малко търпение ще имаме ново Linux-ядро.

make dep

Проверява за зависимости.

make bzImage

Компилира самото ядро като bzip имидж. Ще го намерите като файл bzImage в директорията 'arch/i386/boot'.

make modules

Компилира модулите.

make modules_install

Инсталира модулите.

Можем да ги дадем и наведнъж:

make dep bzImage modules modules_install

Времето, което отнема изпълнението на тези команди, е добър тест за производителността на вашия компютър. При самата компилация в конзолата ще се появи цяла орда от съобщения, от които можете да научите докъде е стигнала процедурата (например, кой модул се компилира в момента) и дали всичко преминава по план. Фатални грешки излизат рядко и обикновено говорят за неправилно конфигуриран или повреден сорс, възможно е да се дължат и на хардуерни проблеми (грешки в модулите на паметта, форсиран процесор). По-чести обаче са предупрежденията (warnings), които не трябва да ви притесняват.

Съществува и друг начин за автоматизирано инсталиране на ядрото - с командата **make install.** Специален скрипт ще се погрижи да копира файла bzImage в директорията /boot, ще го преименува по подходящ начин и ще редактира файла /etc/lilo.conf така, че да можете да зареждате вече новото ядро. Е, може да не ви хареса начинът, по който ще го именува, но поне ви спестява ръчната настройка, за която ще стане дума по-долу. Имайте предвид, че ако използувате някоя по-екзотична дистрибуция, тази команда може да не се окаже удачното решение. И не забравяйте, че освен ядрото, трябва да инсталирате и модулите, така че **make modules_install** си остава.

Настройка на lilo за зареждане на новото ядро

Ядрото е готово, модулите са инсталирани. Идва време да кажем на lilo, че имаме ново ядро и трябва да го зареди. Независимо от дистрибуцията, която ползувате, ядрото обикновено се намира в '/boot' под име vmlinuz. Всъщност vmlinuz понякога е символичен линк към самия файл на ядрото. Така, само чрез

редактиране на линка можете да стартирате новото ядро, без да се налага до променяте настройките на lilo. В linuxconf също има модул за настройка на lilo, който обаче като че ли отнема повече време, отколкото ако просто отворите /etc/lilo.conf и напишете две-три думи.

Ето ви последователността от команди, с които можете да подготвите новото ядро за зареждане:

1. Вариант с припокриване на оригиналното ядро (не е препоръчителен)

cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz

cp /usr/src/linux/System.map /boot

При това положение /etc/lilo.conf не би трябвало да се нуждае от редактиране, защото той по подразбиране зарежда vmlinuz като оригинално ядро. Преди това се уверете, че наистина е така, разбира се, като проверите дали в /etc/lilo.conf е написано 'image = /boot/vmlinuz'.

Тогава трябва само да стартирате:

lilo

2. Вариант без припокриване на предишното ядро (препоръчителен)

Знаем, че по начало lilo търси и зарежда /boot/vmlinuz. Можем да преименуваме старото ядро:

mv /boot/vmlinuz /boot/vmlinuz.old

mv /boot/System.map /boot/System.map.old

Да копираме новото:

cp /usr/src/linux/arch/i386/boot/vmlinuz.new /boot

cp /usr/src/linux/System.map /boot/System.map.new

Да създадем символичен линк към него под името vmlinuz:

ln -s /boot/vmlinuz.new vmlinuz

ln -s /boot/System.map.new System.map

Да добавим в /etc/lilo.conf опция за зареждане на старото ядро по избор, в случай че не тръгне новото:

image=/boot/vmlinuz.old # Името на файла-ядро

label=OldLinux # Произволно наименование на ядрото

root = /dev/hda1 # root-дяла на вашата Linux-инсталация

И да изпълним командата:

lilo

Готово, вашето ядро е подготвено за зареждане. Сега рестартирайте компютъра и опитайте дали ще се зареди правилно. Ако нещо се объркa, рестартирайте с вълшебната клавишна комбинация и изберете за зареждане старото ядро. Разбира се, ако дистрибуцията ви е по-нестандартна, ще трябва сами да проверите къде е разположено ядрото за зареждане, като разгледате щателно файла /etc/lilo.conf и се запознаете с man lilo.

При зареждането на новото ядро проследете внимателно съобщенията - дали разпознава и конфигуира правилно хардуера, както и на init-системата - всичко ли е [OK]?.