《Linux 内核修炼之道》

作者：华清远见

第 2 章 配置与编译内核

本章简介

如果你对 Linux 的兴趣仅是止于上上网、听听歌、看看碟，那么这里的内容或许并不适合你。如果你稍微有那么一点点的好奇心，希望能够体验一下新的特性、新的内核版本，懂得如何配置与编译内核都是必须的。

2.1 配置内核

编译内核的第一步就是配置内核，这是增加或减少对一些内核特性的支持的必要步骤，比如，可以为内核添加对 IPv6 的支持。

2.1.1 几种配置方式

为了完成内核的配置，必须切换到 root 用户，然后转入内核源码目录，比如：

```
# cd /usr/src/linux
```

然后执行下面的命令之一：

```
# make config
# make oldconfig
# make menuconfig
# make xconfig
# make gconfig
# make defconfig
# make allyesconfig
# make allmodconfig
```

这些命令都可以用来配置内核，它们的配置方式各不相同，不过我们比较常用的还是“make oldconfig”和“make menuconfig”。

1．make config

基于文本的最为传统的也是最为枯燥的一种配置方式，但是它可以适应任何情况。这种方式会为每一个内核支持的特性向用户提问。如果用户回答“y”，则把该特性编译进内核；回答“m”，则该特性作为模块进行编译；回答“n”，则表示不对该特性提供支持。

回答每一个问题之前，用户都需要考虑清楚，如果在配置过程中犯了错误给了错误的答案，就只能按“Ctrl+C”强行退出了。采用该方式时的配置界面如图 2.1 所示。

```
*
* Linux Kernel Configuration
*
*
* General setup
*
Prompt for development and/or incomplete code/drivers (EXPERIMENTAL) [Y/n/?]
Local version - append to kernel release (LOCALVERSION) []
Automatically append version information to the version string (LOCALVERSION_AUT
O) [N/y/?]
Support for paging of anonymous memory (swap) (SWAP) [Y/n/?] y
System V IPC (SYSVIPC) [Y/n/?] y
POSIX Message Queues (POSIX_MQUEUE) [Y/n/?] y
BSD Process Accounting (BSD_PROCESS_ACCT) [Y/n/?]
  BSD Process Accounting version 3 file format (BSD_PROCESS_ACCT_V3) [Y/n/?]
Export task/process statistics through netlink (EXPERIMENTAL) (TASKSTATS) [N/y/?
]
User Namespaces (EXPERIMENTAL) (USER_NS) [N/y/?] (NEW)
Auditing support (AUDIT) [Y/n/?]
  Enable system-call auditing support (AUDITSYSCALL) [N/y/?]
Kernel .config support (IKCONFIG) [N/m/y/?] y
  Enable access to .config through /proc/config.gz (IKCONFIG_PROC) [N/y/?] (NEW)

Kernel log buffer size (16 => 64KB, 17 => 128KB) (LOG_BUF_SHIFT) [15]
Cpuset support (CPUSETS) [Y/n/?] _
```

图 2.1　make config 运行时界面

2．make oldconfig

make oldconfig 与 make config 类似，但是它的作用是在现有内核设置文件基础上建立一个新的设置文件，只会向用户提问有关新内核特性的问题。

在内核升级的过程中，make oldconfig 非常有用。用户可以将现有内核的设置文件.config 复制到新内核的源码目录中，执行 make oldconfig，此时，用户只需要回答那些针对新增特性的问题。

3．make menuconfig

基于终端的一种配置方式，提供了文本模式的图形用户界面，用户可以通过移动光标来浏览内核所支持的各种特性。使用这种配置方式时，系统中必须已经安装有 ncurses 库，否则会显示“Unable to find the Ncurses libraries.”的错误提示。make menuconfig 运行时的界面如图 2.2 所示。

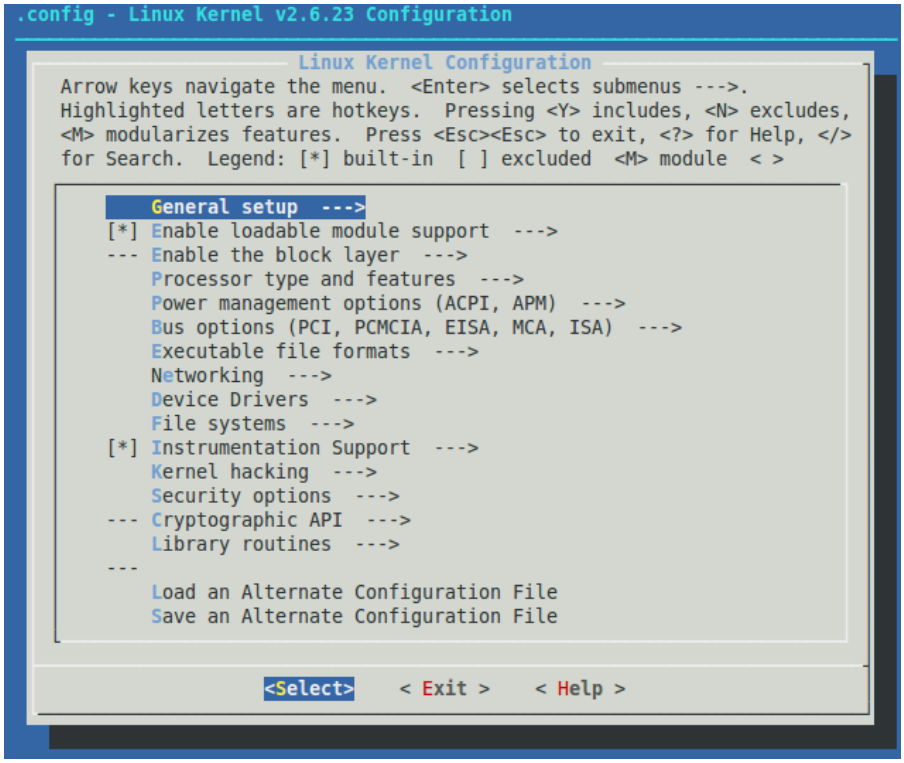

图 2.2　make menuconfig 运行时界面

4．make xconfig

基于 X Windows 的一种配置方式，提供了漂亮的配置窗口，不过只有能够在 X Server 上使用 root 用户运行 X 应用程序时，才可以使用。它依赖于 QT，如果系统中没有安装 QT 库，则会出“Unable to find the QT installation.”的错误提示。图 2.3 所示为 make xconfig 运行时的配置界面。

5．make gconfig

与 make xconfig 类似，不同的是 make gconfig 依赖于 GTK 库。图 2.4 所示为 make gconfig 运行时的配置界面。

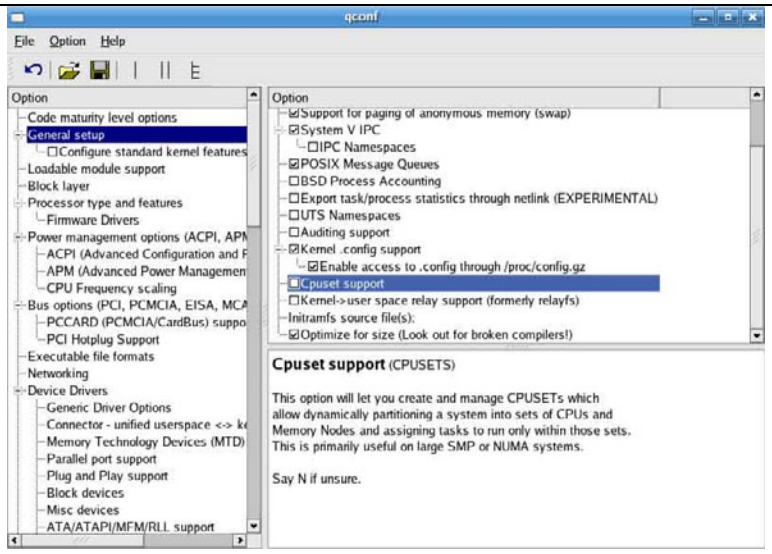

图 2.3　make xconfig 运行时界面

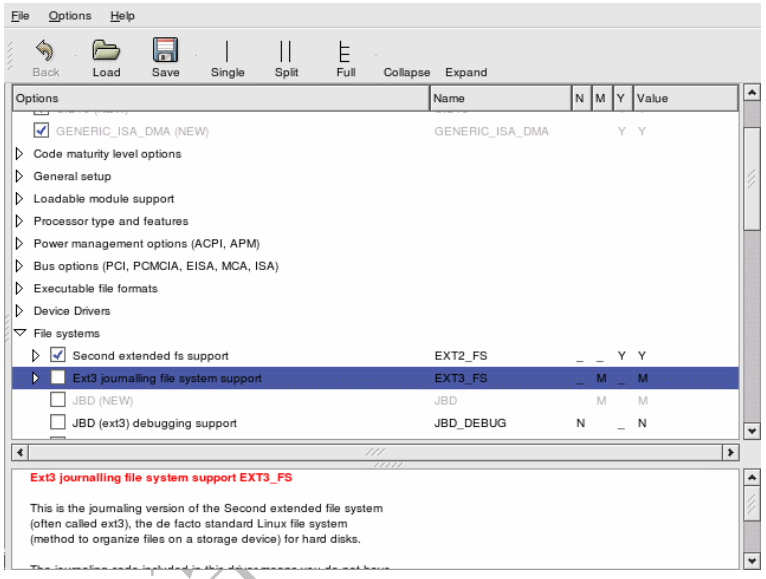

图 2.4　make gconfig 运行时界面

6．make defconfig

按照默认的配置文件 arch/i386/defconfig 对内核进行配置，生成的.config 可以用作初始配置，然后再使用 make menuconfig 进行定制化配置。

7．make allyesconfig

尽可能多地使用“y”设置内核选项值，生成的配置中包括了全部的内核特性。

8．make allmodconfig

尽可能多地使用“m”设置内核选项值来生成配置文件。

2.1.2 .config 文件

不论采用哪种方式配置内核，最终的目的都是为了生成.config 文件，.config 文件位于内核源码树的顶层目录，内容形式如下：

```
#
# SCSI device support
#
# CONFIG_RAID_ATTRS is not set
CONFIG_SCSI=y
CONFIG_SCSI_DMA=y
# CONFIG_SCSI_TGT is not set
CONFIG_SCSI_NETLINK=y
# CONFIG_SCSI_PROC_FS is not set

#
# SCSI support type (disk, tape, CD-ROM)
#
CONFIG_BLK_DEV_SD=y
# CONFIG_CHR_DEV_ST is not set
# CONFIG_CHR_DEV_OSST is not set
CONFIG_BLK_DEV_SR=y
# CONFIG_BLK_DEV_SR_VENDOR is not set
CONFIG_CHR_DEV_SG=y
# CONFIG_CHR_DEV_SCH is not set
```

1．找一个旧的.config 文件作参考

内核配置选项众多，即使使用漂亮的窗口界面去配置，也是相当的麻烦，所以我们最好找一个旧的.config 文件作为参考，在这个.config 文件的基础上执行 make menuconfig 等配置命令进行调整，生成新的.config 文件。

当然，寻找这个作为参考的.config 文件时是需要有的放矢的，一般来说，可以有如下几种途径。

使用 make defconfig 得到参考.config 文件，它其实利用了 arch/i386/defconfig 文件。

利用系统中/lib/modules/<kernel-version>/build 目录下的.config 文件，比如：

```
# cd usr/src/linux-2.6.24
# cp /lib/modules/2.6.23/build/.config .
```

使用当前系统的配置文件，一般位于/boot 目录中。

到网上搜索下载别人配置好的.config 文件。

使用某个发行版提供的.config 文件，比如 slackware 的.config。

2．直接修改.config 文件

如果只是对内核所支持的特性进行微调，则可以直接修改现有的.config 文件而不需使用前面所述的配置方式。.config 文件中每个配置选项的值只能取“y”和“m”两者之一，如果这个特性不再支持，将其对应的选项注释掉即可。

3．备份、重用.config 文件

通常我们需要将生成的.config 文件备份，以备后用。比如当前内核版本为 2.6.23，将它的.config 文件备份为 config-2.6.23。

```
# cd usr/src/linux-2.6.23
# cp .config /root/config-2.6.23
```

如果打算将内核升级到 2.6.24，则可以重用备份的.config 文件，因为这两个内核之间的区别很小。

```
# cd usr/src/linux-2.6.24
# cp /root/config-2.6.23 .config
```

不过，仅仅复制了该文件并不意味着马上就可以编译了。在编译前，我们还必须首先执行 make oldconfig，在执行过程中，如果新内核选项没有出现在复制的.config 文件中，它会停下来等候选择。

2.1.3 配置选项详解

内核的配置选项有很多，通常情况下大部分选项都可以使用默认值，因此并不需要了解它们代表的具体意义。但是某些应用需要将内核裁剪得足够小，这就需要我们知道各个选项所代表的特性，通过去掉不需要的特性，以及将部分特性编译为可加载的模块来减小内核的长度。

另外，即使不考虑内核裁剪的因素，当我们打算深入研究某一个子系统时，也需要首先从该子系统所涵盖的配置选项入手（可以从该子系统代码目录下的各个 Kconfig 文件入手），掌握它的各部分实现代码之间的分布关系，避重就轻地进行代码的分析。

下面的内容仅是对部分比较常用的配置选项进行简单介绍，读者可以参考内核针对各个选项提供的帮助。

1．General setup（常规设置）

Local version - append to kernel release：在内核版本后面加上自定义的版本字符串（小于 64 字符），可以使用“uname –a”命令看到。

Automatically append version information to the version string：自动在版本字符串后面添加版本信息，编译时需要有 perl 以及 git 仓库支持。

Support for paging of anonymous memory (swap)：允许使用交换分区（swap devices）或者交换文件（swap files）来作为虚拟内存。

System V IPC：System V 进程间通信（IPC）支持，许多程序需要这个功能，通常必选。

POSIX Message Queues：POSIX 消息队列，IPC 的一部分。

Export task/process statistics through netlink：通过 netlink 接口向用户空间导出进程的统计信息。netlink 是一种在内核与用户应用之间进行双向数据传输的非常好的方式，用户应用使用标准的 socket API 就可以使用 netlink 提供的强大功能。

Auditing support：审计支持，某些内核模块（例如 SELinux，Security-Enhanced Linux）需要它。

Kernel->user space relay support (formerly relayfs)：在某些文件系统上（比如 debugfs）提供从内核空间向用户空间传递大量数据的接口。

Optimize for size (Look out for broken compilers!)：编译时优化内核尺寸（使用“-Os”而不是“-O2”作为 gcc 参数进行编译）。

2．Loadable module support（可加载模块支持）

Enable loadable module support：打开可加载模块支持，需要执行 make modules_install 将内核模块安装在/lib/modules/<kernel-version>目录下面。

Module unloading：允许卸载已经加载的模块。

Forced module unloading：允许强制卸载正在使用中的模块。

Module versioning support：允许使用其他内核版本的模块。

Automatic kernel module loading：允许内核通过运行 modprobe 自动加载所需的模块。

3．Block layer（块设备层）

Support for Large Block Devices：支持大于 2TB 的块设备。

Support for Large Single Files：支持大于 2TB 的文件。

IO Schedulers：I/O 调度器选项。

Anticipatory I/O scheduler：简称 as 调度器，anticipatory 的中文含义“预料的，预想的”揭示了这个算法的特点，简单地说，在一个个 I/O 发生的时候，如果又有进程请求 I/O 操作，则将产生一个猜测时间，猜测这个 I/O 请求是要干什么的。这会造成比较大的延时，对数据库应用很糟糕，而对于 Web Server 等则会表现的不错。这个算法也可以简单理解为面向低速磁盘的，因为那个“猜测”实际上的目的是为了减少磁头移动时间。

Deadline I/O scheduler：提供了最小的延迟时间和尚佳的吞吐量，特别适合于读取较多的环境（比如数据库）。

CFQ I/O scheduler：为系统内的所有任务分配相同的带宽，提供一个公平的工作环境，它比较适合桌面环境。

No-op：是一个简化的调度程序，它只作最基本的合并与排序。与桌面系统的关系不是很大，主要用在一些特殊的软件与硬件环境下，这些软件与硬件一般都拥有自己的调度机制，对内核支持的要求很小，很适合一些嵌入式系统环境。

4．Processor type and features（CPU 类型及特性）

Symmetric multi-processing support：对称多处理器支持，如果系统中有多个 CPU 或者使用的是多核 CPU 就选上。

Subarchitecture Type：处理器的子架构，如果使用 PC 的话应当选“PC-compatible”。

Processor family：处理器系列，按照实际使用的 CPU 选择。

Generic x86 support：通用 x86 支持。

HPET Timer Support：HPET 是替代 8254 芯片的新一代定时器。

Maximum number of CPUs：支持的最大 CPU 数，每增加一个内核映像（image 文件）将增加大约 8KB。

SMT (Hyperthreading) scheduler support：支持 Intel 的超线程（HT）技术。

Multi-core scheduler support：针对多核 CPU 进行调度策略优化。

Preemption Model：内核抢占模式，有 3 种模式可供选择。“No Forced Preemption (Server)”为适合服务器环境的禁止内核抢占模式；“Voluntary Kernel Preemption (Desktop)”模式支持自愿抢占，但此时内核仍然是不可抢占的，它仅仅是通过增加抢占点缩减了抢占延迟，适用于一些需要较好响应性的环境，如桌面环境；“Preemptible Kernel (Low-Latency Desktop)”模式既包含了自愿抢占，又使内核具有可抢占性，因此具有很好的响应延迟，主要适用于桌面和一些嵌入式系统。

Preempt The Big Kernel Lock：支持抢占大内核锁，适用于桌面环境。

Machine Check Exception：让 CPU 检测到系统故障时通知内核，以便内核采取相应的措施。

check for P4 thermal throttling interrupt：当 P4 的 CPU 过热时显示一条警告消息。

High Memory Support：支持的最大内存，总内存小于等于 1GB 的选“off”，大于 4GB 的选“64G”。

64 bit Memory and IO resources：适用 64 位的内核和 I/O 资源。

Allocate 3rd-level pagetables from highmem：在内存很大的系统上将用户空间的页表放到高端内存区，以节约低端内存。

MTRR (Memory Type Range Register) support：MTRR 规定了读写某段物理内存的策略，用于优化 CPU 数据传送性能。例如，可将 MTTR 设为在显存的地址范围上使用“write-combining”策略，CPU 能够在 PCI/AGP 总线上，将许多次少量的数据写入集合成一次大的数据写入，这样能获得 2.5 倍以上图像传送速度的提升。详细说明可参看内核文档 Documentation/mtrr.txt。

Boot from EFI support：EFI 用于替代传统的 BIOS 计数。

Enable kernel irq balancing：打开中断负载均衡。

Timer frequency：内核时钟频率，桌面环境推荐使用 1000 Hz，服务器环境推荐使用 100 Hz 或 250 Hz。

kexec system call：提供 kexec 系统调用，可以不必重启系统而切换到另一个内核。

5．Power management options（电源管理选项）

Power Management support：电源管理有 APM 和 ACPI 两种标准且不能同时使用。即使关闭该选项，X86 上运行的 Linux 也会在空闲时发出 HLT 指令将 CPU 进入睡眠状态。

Suspend to RAM and standby：支持待机状态，系统将目前的运行状态等数据存放在内存，关闭硬盘、外设等设备，进入等待状态。此时内存仍然需要电力维持其数据，但整机耗电很少。恢复时计算机从内存读出数据，回到挂起前的状态，恢复速度较快。

Hibernation（aka 'suspend to disk'）：支持休眠状态，将目前的运行状态等数据存放在硬盘上某个文件或者某个特定的区域，关闭硬盘、外设等设备，进入关机状态。此时计算机完全关闭，不耗电。恢复时计算机从休眠文件/分区中读出数据，回到休眠前的状态，恢复速度较慢。

ACPI (Advanced Configuration and Power Interface) Support：必须运行 acpid 守护程序 ACPI 才能起作用。ACPI 是为了取代 APM 而设计的，因此应该尽量使用 ACPI 而不是 APM。

APM (Advanced Power Management) BIOS Support：APM 支持。

CPU Frequency scaling：允许动态改变 CPU 主频，达到省电和降温的目的。

6．Bus options (PCI, PCMCIA, EISA, MCA, ISA)

PCI support：PCI 支持，如果使用了 PCI 或 PCI Express 设备就必选。

PCI access mode：PCI 访问模式，即访问 PCI 的方式，可以有 4 个选择，BIOS、MMConfig、Direct、Any。前三个选择分别表示通过 BIOS 去访问、抛开 BIOS 直接（Direct）去访问、通过 MMConfig 去访问。最后一个选择“Any”表示如果拿不准的话可以让内核去选择一种访问方式，当然，内核会按照一定的优先级，首先尝试 MMConfig，然后是 Direct，如果这两种方式都不起效，最后再使用 BIOS。

基于 PCI 总线的特殊地位，BIOS 中专门提供了针对 PCI 总线的操作，这些操作里就包括了总线枚举的整个过程。在系统加电以后自检时，就会完成对 PCI 总线的枚举，之后对 PCI 设备的访问就都是通过 BIOS 调用的形式进行，提供有这些功能和服务的 BIOS 就称之为 PCI BIOS。这也是 PCI 访问模式中的 BIOS 模式所表达的意思。

但是，一些旧的主板上，BIOS 并不支持这么做，还有一些嵌入式系统里甚至于根本就没有 BIOS 的存在，为了适应各种需要，Linux 就自己实现了包括总线枚举在内的一整套 PCI 总线操作，而不再去依赖 BIOS，这就是 Direct 访问模式的由来。当然在 64 位的平台上，是没有什么 PCI BIOS 的，采用的总是 Direct 方式，使用 make menuconfig 配置内核的时候也就根本看不到 PCI access mode 这一项。

至于 MMConfig 模式，是 PCI Express 才用得上的。

Message Signaled Interrupts (MSI and MSI-X)：MSI 是 PCI Spec v2.2 中提出一种全新的中断方式。MSI 通过向一个预定义的内存地址写入一个已经预定义好的 Message 来提出中断请求，这个 Message 到达 PCI 主桥（host bridge）时，主桥会将它转换为具体的中断，发送到处理器。对 PCI 设备来说，这就消除了对中断引脚电路的需要，但是 PCI Spec 里还是要求支持 MSI 的设备最好同时也要具有中断引脚。MSI-X 则是 MSI 的增强。

Interrupts on hypertransport devices：hypertransport 是 AMD 在 1999 年提出的一种总线技术。

ISA support：ISA 设备支持。

MCA support：旧的 IBM 的台式机和笔记本上可能会有这种总线。

PCCARD (PCMCIA/CardBus) support：PCMCIA 卡（主要用于笔记本）支持。
PCI Hotplug Support：PCI 热插拔支持。

7．Executable file formats（可执行文件格式）

Kernel support for ELF binaries：ELF 是 Linux 下最常用的二进制文件格式，必选。
Kernel support for a.out and ECOFF binaries：早期 UNIX 系统的可执行文件格式，目前已经被 ELF 格式取代。
Kernel support for MISC binaries：Linux 是第一个在内核级提供内建 Java 解释器的支持，从而执行 Java 代码的操作系统之一。这在 2.2 内核里已经实现了。2.4 内核又做了改进，将这种支持的方法改为对“Misc”二进制类型的支持。通过使用这种类型的二进制代码类型，用户甚至可以利用 DOSEMU（MS DOS 模拟器）或者 WINE（MS Windows 模拟器）来运行在 DOS/Windows 下的.exe 或.com 的程序。

8．Networking

Networking options：网络选项。
Packet socket：Packet Socket 可以让应用程序（比如 tcpdump、iptables 等）直接与网络设备通信，而不需通过内核中的中间网络协议（intermediate networking protocol）。
TCP/IP networking：TCP/IP 协议支持。
The IPv6 protocol：IPv6 支持。
Network packet filtering (replaces ipchains)：Netfilter 可以对数据包进行过滤和修改，可以作为防火墙、网关、代理或网桥使用。
DCCP Configuration：数据报拥塞控制协议（Datagram Congestion Control Protocol）在 UDP 的基础上增加了流控和拥塞控制机制，适用于流媒体业务的传输。
SCTP Configuration：流控制传输协议（Stream Control Transmission Protocol）是一种新兴的传输层协议。TCP 协议一次只能连接一个 IP 地址，而 SCTP 协议一次可以连接多个 IP 地址，且可以自动平衡网络负载，一旦某一个 IP 地址失效会自动将网络负载转移到其他 IP 地址上。
TIPC Configuration：透明内部进程间通信协议（The Transparent Inter Process Communication Protocol）专门用于内部集群通信（intra cluster communication）。
Asynchronous Transfer Mode (ATM)：异步传输模式（ATM）支持。
Appletalk protocol support：与 MAC 机器通信的协议。
Amateur Radio support：业余无线电支持。
IrDA (infrared) subsystem support：红外线支持，比如无线鼠标。
Bluetooth subsystem support：蓝牙支持。

9．Device Drivers（设备驱动）

Memory Technology Devices (MTD)：MTD（Memory Technology Device）设备支持，比如常用于数码相机或嵌入式系统的闪存卡。
Plug and Play support：即插即用支持。
Block devices：块设备支持。
ATA/ATAPI/MFM/RLL support：通常是指 IDE 硬盘和 ATAPI 光驱。
SCSI device support：SCSI 设备支持。

Serial ATA and Parallel ATA drivers：SATA 和 PATA 设备支持。
Multi-device support (RAID and LVM)：RAID 和 LVM 支持。
Network device support：网络设备支持。
Input device support：输入设备支持。
Character devices：字符设备支持。
I2C support：I^2C 支持。
SPI support：SPI（Serial Peripheral interface）支持。
Multimedia devices：音视频多媒体设备支持。
Graphics support：图形设备以及显卡支持。
Sound：声卡支持（ALSA 和 OSS）。
USB support：USB 支持。
MMC/SD Card support：MMC/SD 卡支持。
Real Time Clock：RTC 支持，RTC 通常与 CMOS 集成在一起，因此 BIOS 可以从中读取当前时间。
DMA Engine support：DMA 将某些传输数据的操作从 CPU 转移到专用硬件，从而可以进行异步传输并减轻 CPU 负载。

10．File systems（文件系统）

Second extended fs support：Ext2 文件系统支持。
Ext3 journalling file system support：Ext3 日志文件系统支持。
Ext4dev/ext4 extended fs support：最新的 Ext4 文件系统支持。
Inotify file change notification support：文件系统事件通知机制 Inotify 支持，用于替代老的 dnotify 机制。
Inotify support for userspace：用户空间 Inotify 支持。
Quota support：磁盘限额支持，限制某个用户或者某组用户的磁盘占用空间。
Filesystem in Userspace support：FUSE 允许在用户空间实现一个文件系统，如果你打算开发一个自己的文件系统或者使用一个基于 FUSE 的文件系统就选吧。
Pseudo filesystems：/proc 等伪文件系统支持。
Native Language Support：本地语言支持，使用 FAT/NTFS 分区时需要选上。

11．Instrumentation Support（工具支持）

Profiling support：打开 Profiling 支持，用于对内核的性能进行分析。
OProfile system profiling：OProfile 是 Linux 的若干种性能分析工具中的一种，能够帮助用户识别诸如循环的展开、高速缓存的使用率过低、低效的类型转换和冗余操作等问题。
Kprobes：使用 printk 收集内核的调试信息是一个众所周知的方法，而使用了 Kprobes，不需要经常重新引导和重新编译内核就可以完成这一任务。

12．Kernel hacking（内核 hack 选项）

Show timing information on printks：在 printk 的输出中包含时间信息，可以用来分析内核启动过程各步骤所用时间。
Magic SysRq key：Magic SysRq 键是一种组合键，在系统无法响应时，可以进行一些基本的维护任务。
Enable unused/obsolete exported symbols：导出无用和废弃的符号，这将使内核不必要地增大。

Run 'make headers_check' when building vmlinux：在编译内核时运行“make headers_check”命令检查内核头文件，当你修改了与用户空间相关的内核头文件后建议启用该选项。

Use 4Kb for kernel stacks instead of 8Kb：如果配置该选项，则内核为进程提供的内核栈缩小为 4KB，为此，每个 CPU 提供了一个 4KB 大小的中断栈专供中断服务程序使用。小的内核栈可以缓解系统的内存压力。

13．Security options（安全选项）

Enable different security models：允许内核选择不同的安全模型，如果未选中则内核将使用默认的安全模型。

Socket and Networking Security Hooks：允许安全模型通过 Security Hook 对 Socket 与 Networking 进行访问控制。

Root Plug Support：一个简单的 Linux 安全模块，在指定的 USB 设备不存在时，它简单地禁止所有 egid 为 0 的进程运行。

14．Cryptographic API（加密 API）

提供核心的加密 API 支持。

2.2 编译内核

2.2.1 准备工作

虽然与配置内核相比，编译内核所做的工作要少得多，但是在正式编译之前，我们仍需要做一些必要的准备。

1．需要了解的基础知识

首先我们需要了解系统中与编译过程有关的目录及文件。

/boot/vmlinuz-<version>：用于启动的压缩内核镜像。

/boot/system.map-<version>：存储内核符号表。

/boot/initrd.img-<version>：一个镜像文件，类似 ramdisk（initrd 的全称就是 initial ramdisk），它将一些驱动程序和命令工具打包到 img 里，比如 sisc_mod、ext3、sd_mod 等模块和 insmod、nash 等命令，然后在开机的时候在内存里开辟一段区域，释放到那里运行。

它的作用是，在没有 mount 根分区（/）以前，系统要执行一些操作，比如挂载 scsi 驱动，此时就把 initrd 释放到内存里，作一个虚拟的/，然后执行其根目录下的脚本，运行 insmod 等命令加载模块。

/boot/grub/menu.lst：GRUB 的配置文件（不同的发行版中它可能位于不同位置）。

/lib/modules/：该目录包含了内核模块（包括系统自带的和自己编译的）及其他文件，不同的子目录由内核版本号来区分。

/lib/modules/<kernel-version>/build/：存放编译新模块所需的文件，包括了 Makefile、.config、module.symVers（模块符号信息）以及内核头文件等。

/lib/modules/<kernel-version>/kernel/：存放模块的 ko 文件。
/lib/modules/<kernel-version>/modules.alias：模块别名定义，模块加载工具使用它来加载相应的模块。
/lib/modules/<kernel-version>/modules.dep：定义了模块间的依赖关系。
/lib/modules/<kernel-version>/modules.symbols：标识符号属于哪个模块。

2．下载内核源码压缩包

需要注意下载 bz2 格式的压缩包时，需要安装 bzip2 工具进行解压。

3．获取相关补丁

如果需要的某些特性并没有被现有内核支持，则需要去获取相关的补丁。比如，为了使内核支持图形化的启动界面，我们可能要用到 bootsplash 工具。bootsplash 项目的网站 http://www.bootsplash.org/上提供了针对很多内核版本的补丁供下载。

4．构建编译环境

编译内核需要用到一系列的工具，在编译之前，我们需要确保它们已经被安装。下面是一些 Debian 和 Ubuntu 发行版上用到的工具包。
modutils：模块工具。
kernel-package：包括了 make-kpkg 等工具。
patch：如果不需要为内核打补丁，可以不安装 patch 工具包。
build-essential：提供了 C/C++的编译环境，包括了 gcc、make 等工具。

5．备份

当修改内核时，我们必须准备一个能够启动的备用内核。实现该目的的一种方式是通过配置 Linux 引导程序（LILO 或 GRUB）以允许用户选择启动的内核映象，其中之一是从未修改过的内核的备份。

2.2.2　如何为内核打补丁

通过打补丁的方法升级内核版本，可以不用下载整个源代码。针对每个内核版本的补丁文件可以在 ftp.kernel.org 上面获得，我们的问题是应该选择哪个补丁文件，一个补丁又到底应该打在哪个版本的内核上。

下面的内容简单介绍了如何应用与卸载补丁，详细的内容也可以查看内核文档 Document/applying-patchs。

1．什么是补丁

一个补丁就是一个文本文档，由 diff 工具创建，它存放了两个不同版本的源代码之间的差异。为了正确地应用一个补丁，我们需要知道这个补丁文件是以哪个版本为基础产生出来的，以及它将把目前的源代码变化到什么新的版本，简单地说，就是需要清楚产生这个补丁文件的两个源码版本的情况。

2．如何打补丁和卸载补丁

patch 工具可以用于打补丁和卸载补丁。内核的补丁是相对于保存内核源码的父目录而生成的，这就意味着，补丁文件中的文件路径包含了内核源码存放目录的名字（比如 linux-2.6.23/，或者像是“a/”和“b/”之类的其他名字）。但是很可能我们本地系统上的内核源码存放目录和补丁中不匹配，为了解决这个问题，我们需要切换到自己的源码目录，并且在执行 patch 命令的时候加上“-p1”参数，这样就会去掉补丁文件中路径的第一个分量。比如：

```
# cd /usr/src/linux
# patch -p1 < ../patch-x.y.z
```

为了卸载一个以前打上的补丁，需要使用“-R”参数。

```
# patch -R -p1 < ../patch-x.y.z
```

3．如何利用补丁升级内核版本

考虑这样的几个场景：将内核从 2.6.23 升级到 2.6.24；将内核从 2.6.23.8 升级到 2.6.24.6；将内核从 2.6.23.6 升级到 2.6.23.8。不管处于哪种场景，打补丁时要谨记的一点是：内核的补丁文件都是以 2.6.x（基础稳定版 basic stable，2.6.x.y 是稳定版 stable）为基础发布的。下面对这 3 种场景的打补丁过程分别进行介绍。

（1）将内核从 2.6.23 升级到 2.6.24。这种情况，可直接使用补丁文件 patch-2.6.24。

```
# patch -p1 < ../patch-2.6.24
```

因为下载得到的补丁文件通常是使用 gzip 或 bzip2 压缩的格式，所以使用前还要将其解压生成 patch-x.y.z 文件。不过，我们也可以不用解压，使用下面的命令形式：

```
# bzcat ../patch-2.6.24.bz2 | patch -p1  //bz2格式
# zcat ../patch-2.6.24.gz | patch -p1  //gz格式
```

（2）将内核从 2.6.23.8 升级到 2.6.24.6。这种情况下，我们需要将升级的过程分解为几个步骤，首先将 2.6.23.8 退回到 2.6.23，然后再升级到 2.6.24，最后升级到 2.6.24.6。

```
# bzcat ../patch-2.6.23.8.bz2 | patch -p1 -R
# bzcat ../patch-2.6.24.bz2 | patch -p1
# bzcat ../patch-2.6.24.6.bz2 | patch -p1
```

（3）将内核从 2.6.23.6 升级到 2.6.23.8。这种情况下，我们同样需要将升级过程分解，首先将 2.6.23.6 退回到 2.6.23，然后再升级到 2.6.23.8。

```
# bzcat ../patch-2.6.23.6.bz2 | patch -p1 -R
# bzcat ../patch-2.6.23.8.bz2 | patch -p1
```

4．patch 的替代工具

除了 patch 之外，也有其他的用来打补丁的工具，比如 interdiff、ketchup 等。

2.2.3　编译步骤

下面是针对 2.6 内核的通用的编译步骤。

（1）下载源码并解压。

虽然我们可以将内核源码存放在任何自己找得到的地方，但通常还是会将内核源码下载到/usr/src 目录并解压（Linus 本人说不要解压到这个目录）。

```
# cd /usr/src
# wget ftp.kernel.org/pub/linux/kernel/v2.6/linux-2.6.23.tar.bz2
```

```
# tar jxvf linux-2.6.23.tar.bz2
```

（2）如果需要的话，下载补丁。

（3）进入刚刚解压的内核源码目录。

```
# cd /usr/src/linux-2.6.23
```

（4）如果需要的话，为内核打补丁。

（5）配置内核。

```
# make menuconfig
```

（6）编译内核。

```
# make
```

（7）安装内核模块。将所有编译得到的内核模块复制到/lib/modules/<kernel-version>/目录下面。

```
# make modules_install
```

（8）安装内核。

```
# make install
```

make install 主要完成了 3 个工作。

复制生成的内核映像到/boot 目录。在内核编译完成后，源码树目录 arch/i386/boot/中会生成一个 bzImage 文件，该文件被复制到/boot 目录并重命名为 vmlinuz-2.6.23。

生成 initrd-<kernel-version>.img 文件。

配置引导程序（GRUB 或 LILO）。

（9）重启进入新内核。

2.2.4 文档的编译

内核源码树的 Documentation/目录下面有大量的文档，它们是内核最好的参考资料，对于我们学习内核有着重要的意义。我们可以使用下面的一些命令生成指定格式的文档。

```
# make htmldocs //生成 HTML 文件
# make pdfdocs  //生成 PDF 文件
# make psdocs   //生成 Postscript 文件
# make mandocs  //为 Kernel API 生成 man 手册
# make installmandocs  //将 Kernel API 手册页安装到 man 程序能够找到的目录中
```

执行 make htmldocs/pdfdocs/psdocs 之后，在 Documentation/DocBook/目录下，会生成一些很重要的文档：

kernel-api：内核开发的 API 手册。

kernel-locking：内核加锁的 HOWTO 文档。

kernel-hacking：内核开发的一些注意事项。

usb：USB Host 端的 API 手册。

gadget：Usb Device 端的 API 手册。

2.2.5 编译小技巧

下面是一些内核编译过程中可以使用的小技巧。

（1）屏蔽编译信息。

```
# make > /dev/null
```

（2）加速编译过程。

可以使用“-j<n>”参数，其中 n = 2 * CPU 的个数，对于一般的单 CPU 系统，通常用是使用“-j2”参数，为编译过程分配两个任务，这样在进行磁盘 I/O 操作时候，CPU 就不会空闲了。

```
# make -j2 > /dev/null
```

（3）使用 verbose 模式，将每一步执行的命令都打印出来，并重定向到一个文件中去，这样以后可以方便地查找模块之间的依赖关系。

```
# make V=1 > ~/bak.txt
```

（4）使用 ccache 提高编译速度。ccache 主页为 http://ccache.samba.org/。使用 ccache 时，需要更改源码树根目录下面的 Makefile 文件，在 CC 和 HOSTCC 变量的定义前添加 ccache。

```
CC      = ccache $(CROSS_COMPILE)gcc
HOSTCC  = ccache gcc
```

2.3 自由软件的编译与安装

在我们使用 Linux 的过程中，经常会遇到需要自己编译软件的情况，可以说自由软件的百家争鸣本身就是 Linux 世界不可分割的一部分。因此，本节简单介绍下自由软件的编译与安装过程。

自由软件和私有软件的区别在于它们对源代码的访问不同。自由软件以源代码文件压缩包的形式发布，用户必须自己编译源代码才能使用。

对于大部分的自由软件，存在已编译版本，用户可以只安装这些预编译的二进制代码。而某些自由软件并不以这种形式发布，或者其早期版本不以二进制代码形式发布，而且，如果你使用的是特殊的操作系统或者特殊的硬件架构，许多软件并没有为此提供编译好的版本。更重要的是，亲自编译软件可以让你只启用自己感兴趣的选项，或者通过对该软件源码的修正以便它能够确实满足自己的需求。

2.3.1 发布时的组织结构

通常，自由软件发布时都具有相同的组织结构。

INSTALL 文件：描述安装步骤。

README 文件：包含有关该软件的一般信息（简短描述、作者、项目 URL、相关文档、有用的链接等）。如果不存在 INSTALL 文件，通常在 README 文件中会包含安装步骤简介。

COPYING 文件：包含许可证或是发布该软件的情形。有时候该文件叫 LICENSE。

CONTRIB 或是 CREDITS 文件：包含该软件相关人员的列表（活动的参与者、相关评论、第三方软件等）。

CHANGES 文件：包含最近改进以及故障修正，有时为较少见的 NEWS 文件。

Makefile 文件：控制该软件的编译（对 make 该文件是必需的）。如果该文件一开始不存在，那么它将由预编译配置过程生成。

经常也有 configure 或 Imakefile 文件以便用户针对特殊系统自定义生成新的 Makefile 文件。

保存源文件以及在编译结束后保存二进制文件的目录，通常为 src。

保存与该软件相关的文档（通常是 man 形式）的目录，通常为 doc。

有时也有保存该软件特定数据（一般是配置文件、数据示例文件或资源文件）的目录。

2.3.2 配置

仅从技术兴趣上看，自由软件作者提供源代码的目的在于使该软件可移植，使其能够在现存的类 UNIX 系统上几乎不加修改地使用，而这需要在编译这些软件之前对其进行配置。

有几种不同的配置方式，但我们需要选择作者指定的那个。

如果在该软件的发布目录中存在一个名为 configure 的文件，则可以使用 AutoConf 进行配置。

如果在该软件的发布目录中存在一个名为 Imakefile 的文件，则可以使用 imake 进行配置。

根据 INSTALL 文件（或 README 文件）的内容提示运行一个 shell 脚本（比如 install.sh）。

1．AutoConf

AutoConf 用于正确配置软件，它创建编译需要的文件（比如 Makefile），且有时会直接对源代码进行修改（比如使用 config.h.in 文件）。AutoConf 的规则十分简单。

该软件的作者了解配置他的软件需要进行哪些测试（比如：“你使用哪一版本的库文件？”），他使用一种精确的语法将这些测试编写进 configure.in 文件。

并且，他通过运行 AutoConf 从 configure.in 文件生成 configure 自动配置脚本，该脚本将在配置软件的时候运行所需的测试。

最终用户执行该脚本，这样 AutoConf 就会为编译进行配置。

下面是一个 AutoConf 的使用示例：

```
# ./configure
loading cache ./config.cache
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for main in -lX11... yes
checking for main in -lXpm... yes
checking for main in -lguile... yes
checking for main in -lm... yes
checking for main in -lncurses... yes
checking how to run the C preprocessor... gcc -E
checking for X... libraries /usr/X11R6/lib, headers /usr/X11R6/include
checking for ANSI C header files... yes
checking for unistd.h... yes
checking for working const... yes
updating cache ./config.cache
creating ./config.status
creating lib/Makefile
creating src/Makefile
creating Makefile
```

为了更好地控制 configure，可以通过命令行或环境变量为其添加选项。比如：

```
# ./configure --with-gcc --prefix=/opt/GNU
```

或者（在 bash 中）：

```
# export CC=`which gcc`
# export CFLAGS=-O2
# ./configure --with-gcc
```

或者：

```
# CC=gcc CFLAGS=-O2 ./configure
```

一般而言，大部分 configure 脚本执行失败时的出错信息都类似于“configure: error: Cannot find library guile”，这表示 configure 脚本找不到某个库文件。configure 脚本在测试时会使用该库文件编译一个小测试程序，如果它不能成功编译该程序，就说明它不能够编译该软件，所以会给出该出错信息。出现该错误后，我们可以采取下面的措施。

可以在 config.log 文件中找到配置过程所执行的每一步骤，从中发现产生这一错误的原因。

检查提示的库文件是否已正确安装。如果没有，在安装之后再运行 configure。检查它是否已安装的有效方式是查找包含提示字符串的库文件，一般是 lib<名称>.so。比如：

```
# find / -name 'libguile*'
```

检查编译器是否能够访问该库文件，它是否在/usr/lib、/lib、/usr/X11R6/lib （或是在由环境变量 LD_LIBRARY_PATH 指出的）目录中。

检查该库文件相应的头文件是否已安装于正确地方（通常是/usr/include 或 /usr/local/include 或 /usr/X11R6/include）。

检查是否拥有足够的磁盘空间（configure 脚本需要一些空间来保存中间文件）。可以通过“dh”命令查看系统中各分区的使用情况。

检查是否某些环境变量设置错了，比如 LD_LIBRARY_PATH。

2．imake

imake 让你能够使用简单的规则生成 Makefile 文件以配置自由软件，这些规则指定需要编译那些文件才能生成所需的二进制文件，而 imake 生成相应的 Makefile。可在 Imakefile 文件中指定这些规则。

使用 imake 最为简单的方法是进入解压后的代码目录，然后运行 xmkmf 脚本，而它又会调用 imake 程序。

3．各种 shell 脚本

阅读 INSTALL 或 README 文件了解进一步信息。通常，需要执行 install.sh 或 configure.sh 文件。该安装脚本或是非交互的（它自己决定需要什么），或者会向询问有关系统的信息（比如是路径）。

另外，某些自由软件在其初期开发阶段，有时会要求用户手动更改某些配置文件。通常，这些文件是 Makefile 文件和 config.h 文件。

2.3.3 编译

既然该软件已正确配置，所剩的就是编译了。这个阶段通常比较简单，并且不会出现什么严重的问题。

编译源代码的各个步骤常存储于 Makefile 或 GNUMakefile 文件中。当执行 make 时，它会从当前目录读取该文件（如果该文件存在的话）。如果它不存在，可以通过 make 的-f 选项指定其他文件。

通常，使用 make 需要遵循如下一些约定。

不加参数执行 make 表示仅编译程序，而不安装。

make install 编译程序（不是一定的），并随后将文件安装到系统的正确位置。某些文件常常无法正确安装（比如 man、info），可能会需要用户自己手动复制。有时候，需要在子目录中再次执行 make install，通常这是由于包含了第三方开发的模块。

make clean 清除编译产生的所有临时文件，大多数情况下还会删除可执行文件。

编译时最为常见的错误有：

（1）main.c:16: decl.h: No such file or directory

编译器不能找到对应的头文件。不过，在软件配置阶段就可能已经发现该错误了。解决方法如下。

检查该头文件确实已经存在于以下某个目录中：/usr/include、/usr/local/include、/usr/X11R6/include 或它们的某个子目录。如果没有，请在整个磁盘上查找它（可以使用 find 或 locate 命令）。如果还是没有，请检查是否已经安装了该头文件所对应的库。

检查该头文件确实可以读取（可以使用 less 命令来测试）。

如果它确实在/usr/local/include 或/usr/X11R6/include 目录中，你可能需要为自己的编译器添加额外的参数，打开编译出错的那个目录中的那个 Makefile 文件，找到出错的那一行，并在调用编译器（gcc，有时是$(CC)）的地方添加字符串-I<路径>，其中<路径>是包含该头文件的路径。如果不清楚要把该选项加到哪里，就把它添加到文件开始处 CFLAGS 或 CC 变量定义后面。

再次运行 make，如果它还是不起作用，请检查前面的那个选项是否被添加于编译过程中出错的地方。

如果还是不起作用，只有向周围的高手或开发社区求助了。

（2）'struct foo' undeclared (first use this function)

结构几乎是所有软件都会使用的一种数据类型，系统在头文件中可能会定义许多。这个错误提示表示该问题可能是由于找不到头文件，或误用头文件所造成。解决该问题的正确步骤如下。

试着检查一下出问题的结构是否已由程序或系统定义，比如用 grep 命令查看该结构是否已经在某个头文件中定义了。比如，进入该软件源代码的根目录中执行：

```
# find . -name '*.h'| xargs grep 'struct foo' | less
```

在屏幕上可能会出现许多行，找到头文件中 grep 指出的那一行，检查它是否就是你所需要的。如果是，则说明该头文件未包含于出错的.c 文件中。有两个解决方法：在出错的.c 文件开始处添加#include "<文件名>.h"；或者将该结构的定义复制到该文件的开始处。

如果没有找到，在系统头文件（通常位于/usr/include、/usr/X11R6/include 或/usr/local/include）中再次查找。不过这一次如果找到的话，就需要在.c 文件中添加#include <<文件名>.h>的语句。

如果该结构还是不存在，请试着找找它是否是在哪个库中定义（请查看 INSTALL 或 README 文件以确定该软件使用了哪些库以及它们的版本）。如果该软件需要的版本不是系统上安装的那一个，就需要更新该库了。

如果依然不起作用，就要检查该程序是否真能在你的架构上运行了（某些程序还没有移植到 Linux 系统上），并检查你是否已经为自己的架构正确配置了该程序（比如在执行 configure 的时候）。

（3）parse error

这个问题解决起来较为复杂，因为编译器常会在真正出错的地方之后报错。有时候，它仅仅是由于某个数据类型未定义。如果碰上如下的错误信息：

```
main.c:1: parse error before `foo_t
main.c:1: warning: data definition has no type or storage class
```

那么很可能是 foo_t 类型未曾定义，解决方式同前一个问题类似。

（4）no space left on device

这个问题解决起来较为简单：磁盘上已经没有足够的空间从源文件生成二进制文件了。你可以通过释放安装目录所在分区的一些空间来解决（删除临时文件或源文件，卸载某些已经不用的程序）。

（5）/usr/bin/ld: cannot open -lglloq: No such file or directory

这表示 ld 程序无法找到某个库。为了包含某个库，ld 将会搜索由-l<库>选项指定的库文件，相应文件为 lib<库>.so。如果 ld 找不到，它将给出一条错误信息。要解决该问题，可以如下操作。

用 locate 命令检查该文件是否在硬盘上。通常，图形库在/usr/X11R6/lib 目录中。比如：

```
# locate libglloq
```

如果上述查找没有结果，就使用 find 命令再次查找（比如 find /usr -name "libglloq.so*"）。如果还是找不到，就需要安装它了。

一旦找到了该库，请检查它是否能被 ld 访问。/etc/ld.so.conf 文件指定了寻找这些库文件的目录，把该库的路径添加到该文件末尾（可能需要重启系统以使改动起作用）。也可以把该目录添加到环境变量 LD_LIBRARY_PATH 中。

如果还是不行，请用 file 命令检查该库文件是否为一个可执行文件。如果它是一个符号链接，请检查该链接完好且没有指向不存在的文件（可用 nm libglloq.so 检查）。而且，该库文件的权限可能是错误的（比如，如果不是 root 用户且该库文件不允许读）。

（6）glloq.c(.text+0x34): undefined reference to `glloq_init'

该问题是由于在编译的最后阶段某个符号找不到。通常，这是因为某个库出问题了。可能的问题如下。

首先，请查找该符号是否应该在某个库文件中。比如，如果该符号以 gtk 开头，它就应该属于 gtk 库。

如果这个库能够很容易地找到，你可以用 nm 命令列出该库中的符号。比如：

```
# nm libglloq.so
0000000000109df0 d glloq_message_func
000000000010a984 b glloq_msg
0000000000008a58 t glloq_nearest_pow
0000000000109dd8 d glloq_free_list
0000000000109cf8 d glloq_mem_chunk
```

使用 nm 时添加-o 选项可以分别在不同行中显示库中的名称，从而使搜索变得更为简单。假定我们要搜索符号 bulgroz_max，可以：

```
# nm /usr/lib/lib*.so | grep bulgroz_max
# nm /usr/X11R6/lib/lib*.so | grep bulgroz_max
# nm /usr/local/lib/lib*.so | grep bulgroz_max
/usr/local/lib/libfrobnicate.so:000000000004d848 T bulgroz_max
```

可以看到该符号 bulgroz_max 定义于 frobnicate 库中（其名称前有一个大写字符 T）。然后，你只需要编辑 Makefile 文件以在编译命令行中添加字符串-lfrobnicate（将其添加于定义 LDFLAGS 或 LFGLAGS 的那一行的末尾，或是在创建相应二进制文件的那一行处）。

编译中使用的库文件不是该软件需要的。请阅读该发布版的 README 或 INSTALL 文件，以查看需要哪个版本。

该发布版的目标文件没有全部正确链接，缺少了定义该函数的文件。请键入 nm -o *.o 以查看应该是哪个文件，然后将相应的.o 文件添加到对应的编译命令行上。

出错的函数或变量可能并不存在，可以尝试删除它：编辑出问题的源文件（它的名字会出现于出错信息的开始处）。这可能会导致程序执行混乱，以及在启动时出现 segfault（段错误）等错误。

（7）Segmentation fault (core dumped)

有时候，编译器立即挂起，并产生该错误信息。除了建议安装一个更新版本的编译器，没有更好的办法了。

（8）no space on /tmp

在不同的阶段，编译器需要临时工作空间，如果它不能申请到这些空间的话，它将出错。因此，你需要清理分区，不过请尽量小心，因为删除某些文件会导致某些正在执行的程序（X 服务器、管道等）挂起。你必需能够清楚知道自己在做什么！如果/tmp 所在的分区并不仅仅包含该目录（比如根目录），请搜索并删除 core 文件。

（9）make/configure 死循环

在你的系统上，这常常只是一个时间上的问题。make 确实需要了解计算机时间和它检查的文件的时间。它比较这两个时间，并根据结果确定某个目标是否已过时。

某些日期问题可能导致 make 不停地编译（或不停地递归编译某个子目录）。在这种情况下，touch（其作用是将有问题的文件的时间设定为当前时间）通常会解决该问题。比如：

```
# touch *
```

或者：

```
# find . | xargs touch
```

2.3.4 安装

既然编译已经完成，接下来就需要将编译后的文件复制到一个合适的位置（通常是在/usr/local 的某个子目录中）。执行 make install 将完成该任务，安装所有需要的文件。

通常，在 INSTALL 或 README 文件中将描述该过程。不过有时候，开发人员会忘了提供这一信息。那时，我们就必须亲自安装所有东西了。

复制可执行文件（程序）到/usr/local/bin 目录。

复制库文件（lib*.so 文件）到/usr/local/lib 目录。

复制头文件（*.h 文件）到/usr/local/include 目录。

复制数据文件通常到/usr/local/share。如果你不知道安装的过程，可以先试着不复制数据文件运行程序，然后就可以按照程序的提示将他们复制到正确的位置了（比如根据某个出错信息：Cannot open /usr/local/share/glloq/data.db）。

文档的安装有一点不同：man 文件通常会被复制到某个/usr/local/man 的子目录中，通常这些文件的格式为 troff（或 groff），且它们的扩展名是某个数字，它们的文件名是某个命令的名称（比如 echo.1），根据其扩展名 n 将其复制到 /usr/local/man/man<n>目录；info 文件复制到/usr/info 或/usr/local/info 目录。

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218