

实验 5 内核定制与编程

一、实验目的

- 1、熟悉 Linux 内核源代码组织形式。
- 2、掌握 Linux 内核编译方法。

二、预备知识

1、编译内核

- (1) Linux 内核源文件缺省位置：/usr/src/linux。
- (2) make mrproper 命令确保源代码目录下没有不正确的.o 文件以及文件的互相依赖。
- (3) make config、make menuconfig, 或者 make xconfig 配置内核。选择相应的配置时, 有三种选择, 它们分别代表的含义如下:

Y – 将该功能编译进内核。

N – 不将该功能编译进内核。

M – 将该功能编译成可以在需要时动态插入到内核中的模块。

- (4) make olddefconfig 使用系统默认配置内核。

- (5) make clean 清除一些现有文件。

- (6) make bzImage 编译内核, 花费时间较长。

- (7) make -j \$(nproc) 并行编译内核, 速度较快。

- (8) 可选如下方式安装模块:

#make modules 生成相应的模块;

#make modules_install 把模块拷贝到需要的目录中;

#depmod -a 生成模块间的依赖关系。

注意: 在进行配置的过程中, 只有回答 Enable loadable module support (CONFIG_MODULES)时选了"Yes", 该选项才是必要的。

2、创建 Linux 系统调用

可以通过编辑/kernel 目录下的 Makefile 文件来添加一个包含自定义函数的文件, 或在源代码树中添加一个文件加入自定义的函数代码。

三、实验内容

- 1、在 Linux 官网 (<https://www.kernel.org/>) 下载一个较新的稳定的内核版本。

- 2、编译准备

注意: gcc 版本问题可能造成某些内核版本的编译过程报错, 如图 E5-0 所示。

```
[root@localhost linux-2.6.0]# make config
HOSTCC scripts/fixdep
scripts/fixdep.c: 在函数'traps'中:
scripts/fixdep.c:359: 警告: 提领类型双关的指针将破坏强重叠规则
scripts/fixdep.c:361: 警告: 提领类型双关的指针将破坏强重叠规则
SHIPPED scripts/kconfig/zconf.tab.h
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/mconf.o
scripts/kconfig/mconf.c:91: 错误: 对'current_menu'的静态声明出现在非静态声明之后
scripts/kconfig/lkc.h:63: 附注: 'current_menu'的上一个声明在此
make[1]: *** [scripts/kconfig/mconf.o] 错误 1
make: *** [config] 错误 2
[root@localhost linux-2.6.0]#
```

图 E5-0 编译内核

所以建议读者使用较新的 Linux 内核。本实验将以 Ubuntu 系统为例演示实验过程, 操作步骤如下:

- (1) 使用命令查看当前系统的内核版本, 操作如图 E5-1 所示, 本机内核版本为 4.15.0。

```
File Edit View Search Terminal Help
jsj2018@donghua:~$ uname -a
Linux donghua 4.15.0-29-generic #31~16.04.1-Ubuntu SMP Wed Jul 18 08:54:04 UTC 2
018 x86_64 x86_64 x86_64 GNU/Linux
jsj2018@donghua:~$
```

图 E5-1 查看内核版本

(2) 更新&准备编译环境，操作如图 E5-2 所示。

apt update: 检查已安装的软件包是否有可用的更新。

apt upgrade: 更新已安装的软件包。

apt install: 安装软件包。

-y: 安装时，所有询问默认 y。

build-essential: 所需要安装的软件包，包含一些 Linux 下开发的必要工具。

libncurses5-dev: 所需要安装的软件包，是一个在 Linux 下广泛应用的图形函数库。

gcc: 编译器。

libssl-dev: OpenSSL 通用库，用于在 Internet 上进行安全通信。

grub2: 来自 GNU 项目的启动引导程序。

bc: 一个任意精度计算器语言。

bison: 一个语法分析器生成器。

flex: 一个词法分析器生成器。

libelf-dev: 所需要安装的软件包，提供共享库。

注: apt 适用于较高版本，若系统版本较低改为 apt-get。

```
File Edit View Search Terminal Help
jsj2018@donghua:~$ sudo apt update && sudo apt upgrade && sudo apt install -y bu
ild-essential libncurses5-dev gcc libssl-dev grub2 bc bison flex libelf-dev
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [107 kB]
Hit:2 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/main amd64 DEP-11 Metada
ta [67.7 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 64x64 Icons
[68.0 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 DEP-11 Me
tadata [109 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/universe DEP-11 64x64 Ic
ons [149 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 DEP-11 Metad
ata [320 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu xenial-updates/main DEP-11 64x64 Icon
s [225 kB]
Get:11 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 DEP-11
Metadata [248 kB]
Get:12 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe DEP-11 64x64
Icons [337 kB]
Get:13 http://us.archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 DEP-1
1 Metadata [5,964 B]
```

图 E5-2 更新&准备编译环境

(3) 下载并解压源码，操作如图 E5-3 所示。

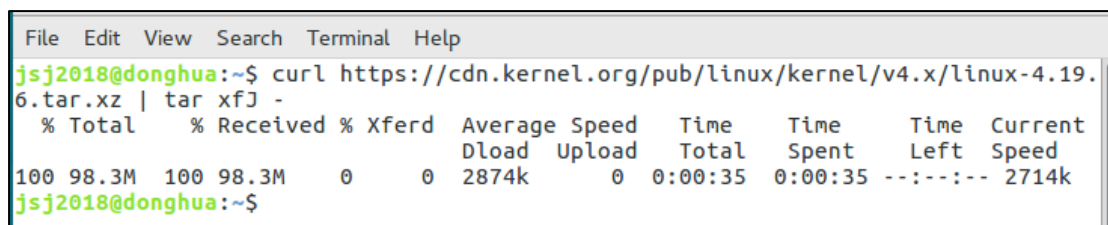
#curl: 利用 url 规则在命令行下工作的文件传输工具，此处用于从 <https://cdn.kernel.org/pub/linux/kernel/v4.x> 网站下载 4.19.6 版本的内核源码。

#tar xfJ -: 解压缩当前文件。

-x: 解压文件。

-c: 压缩文件。

- f: 指定文件名。
- J: 压缩文件的存档名为*.xz。



```
File Edit View Search Terminal Help
jsj2018@donghua:~$ curl https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.6.tar.xz | tar xfJ -
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 98.3M  100 98.3M    0     0  2874k      0  0:00:35  0:00:35 --:--:-- 2714k
jsj2018@donghua:~$
```

图 E5-3 下载并解压源码

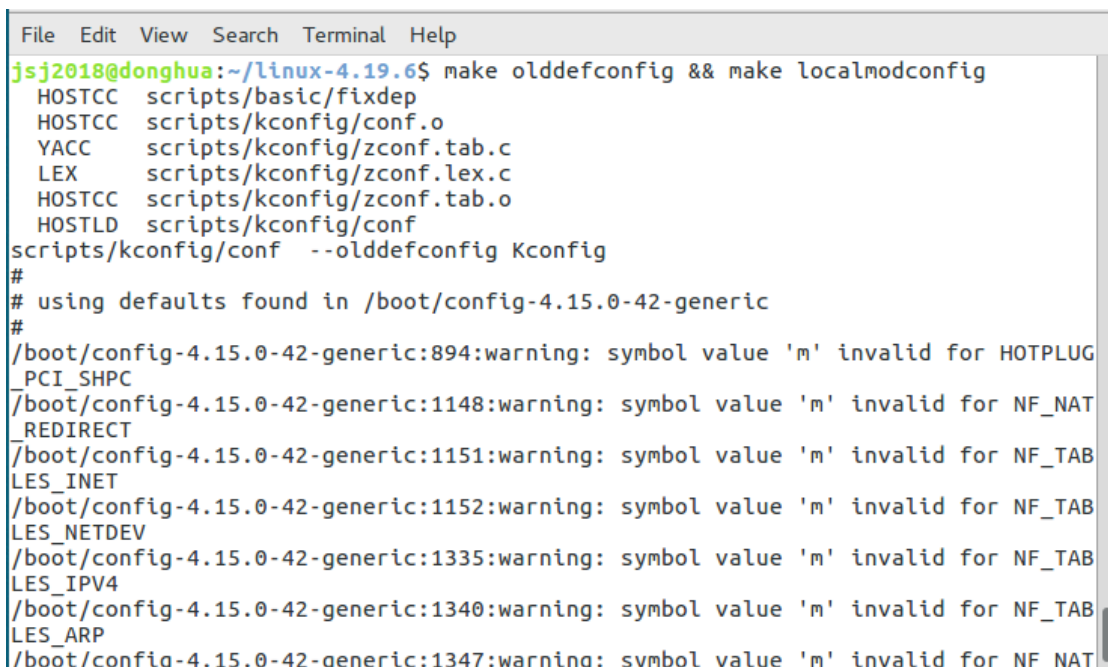
3、 内核编译过程

(1) 配置内核，操作如图 E5-4 所示。

#make olddefconfig 使用系统默认配置，推荐使用。

#make config 需要输入一些内核的配置信息，可以一路回车下去。

make localmodconfig 只编译用户需要的模块，所以会提高编译效率。



```
File Edit View Search Terminal Help
jsj2018@donghua:~/linux-4.19.6$ make olddefconfig && make localmodconfig
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/kconfig/conf.o
YACC    scripts/kconfig/zconf.tab.c
LEX     scripts/kconfig/zconf.lex.c
HOSTCC  scripts/kconfig/zconf.tab.o
HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf  --olddefconfig Kconfig
#
# using defaults found in /boot/config-4.15.0-42-generic
#
/boot/config-4.15.0-42-generic:894:warning: symbol value 'm' invalid for HOTPLUG
_PCI_SHPC
/boot/config-4.15.0-42-generic:1148:warning: symbol value 'm' invalid for NF_NAT
_REDIRECT
/boot/config-4.15.0-42-generic:1151:warning: symbol value 'm' invalid for NF_TAB
LES_INET
/boot/config-4.15.0-42-generic:1152:warning: symbol value 'm' invalid for NF_TAB
LES_NETDEV
/boot/config-4.15.0-42-generic:1335:warning: symbol value 'm' invalid for NF_TAB
LES_IPV4
/boot/config-4.15.0-42-generic:1340:warning: symbol value 'm' invalid for NF_TAB
LES_ARP
/boot/config-4.15.0-42-generic:1347:warning: symbol value 'm' invalid for NF_NAT
```

图 E5-4 配置内核

(2) 编译内核，操作如图 E5-5 所示。

-j: 并行编译，表示在同一时间可以进行并行编译的任务数。

nproc: 操作系统级别对每个用户创建的进程数的限制。

```
File Edit View Search Terminal Help
jsj2018@donghua:~/linux-4.19.6$ make -j $(nproc)
SYSTBL arch/x86/include/generated/asm/syscalls_32.h
SYSHDR arch/x86/include/generated/asm/unistd_32_ia32.h
SYSHDR arch/x86/include/generated/asm/unistd_64_x32.h
SYSTBL arch/x86/include/generated/asm/syscalls_64.h
HYPERCALLS arch/x86/include/generated/asm/xen-hypercalls.h
WRAP arch/x86/include/generated/uapi/asm/bpf_perf_event.h
WRAP arch/x86/include/generated/uapi/asm/poll.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_32.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_64.h
UPD include/generated/uapi/linux/version.h
SYSHDR arch/x86/include/generated/uapi/asm/unistd_x32.h
DESCEND objtool
UPD include/config/kernel.release
HOSTCC /home/jsj2018/linux-4.19.6/tools/objtool/fixdep.o
HOSTLD /home/jsj2018/linux-4.19.6/tools/objtool/fixdep-in.o
LINK /home/jsj2018/linux-4.19.6/tools/objtool/fixdep
CC /home/jsj2018/linux-4.19.6/tools/objtool/exec-cmd.o
CC /home/jsj2018/linux-4.19.6/tools/objtool/help.o
CC /home/jsj2018/linux-4.19.6/tools/objtool/pager.o
CC /home/jsj2018/linux-4.19.6/tools/objtool/parse-options.o
CC /home/jsj2018/linux-4.19.6/tools/objtool/run-command.o
CC /home/jsj2018/linux-4.19.6/tools/objtool/sigchain.o
CC /home/jsj2018/linux-4.19.6/tools/objtool/subcmd-config.o
```

图E5-5 编译内核

这样会在/arch/ x86/boot/目录下生成bzImage文件。

(3) 安装内核模块，操作如图E5-6所示。

```
File Edit View Search Terminal Help
jsj2018@donghua:~/linux-4.19.6$ sudo make modules_install
[sudo] password for jsj2018:
INSTALL arch/x86/crypto/aes-x86_64.ko
INSTALL arch/x86/crypto/aesni-intel.ko
INSTALL arch/x86/crypto/blowfish-x86_64.ko
INSTALL arch/x86/crypto/camellia-aesni-avx-x86_64.ko
INSTALL arch/x86/crypto/camellia-aesni-avx2.ko
INSTALL arch/x86/crypto/camellia-x86_64.ko
INSTALL arch/x86/crypto/cast5-avx-x86_64.ko
INSTALL arch/x86/crypto/cast6-avx-x86_64.ko
INSTALL arch/x86/crypto/chacha20-x86_64.ko
INSTALL arch/x86/crypto/crc32-pclmul.ko
INSTALL arch/x86/crypto/crct10dif-pclmul.ko
INSTALL arch/x86/crypto/des3_edc-x86_64.ko
INSTALL arch/x86/crypto/ghash-clmulni-intel.ko
INSTALL arch/x86/crypto/glue_helper.ko
INSTALL arch/x86/crypto/poly1305-x86_64.ko
INSTALL arch/x86/crypto/serpent-avx-x86_64.ko
INSTALL arch/x86/crypto/serpent-avx2.ko
INSTALL arch/x86/crypto/serpent-sse2-x86_64.ko
INSTALL arch/x86/crypto/sha1-mb/sha1-mb.ko
INSTALL arch/x86/crypto/sha1-ssse3.ko
INSTALL arch/x86/crypto/sha256-mb/sha256-mb.ko
INSTALL arch/x86/crypto/sha256-ssse3.ko
```

图E5-6 安装内核模块

(4) 安装内核，操作如图E5-7所示。

```
File Edit View Search Terminal Help
jsj2018@donghua:~/linux-4.19.6$ sudo make install
sh ./arch/x86/boot/install.sh 4.19.6 arch/x86/boot/bzImage \
    System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.19.6 /boot/vmlinuz-4.19.6
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.19.6 /boot/vmlinuz-4.19.6
update-initramfs: Generating /boot/initrd.img-4.19.6
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.19.6 /boot/vmlinuz-4.19.6
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.19.6 /boot/vmlinuz-4.19.6
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.19.6 /boot/vmlinuz-4.19.6
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is set is no longer supported.
Found linux image: /boot/vmlinuz-4.19.6
Found initrd image: /boot/initrd.img-4.19.6
Found linux image: /boot/vmlinuz-4.19.6.old
Found initrd image: /boot/initrd.img-4.19.6
Found linux image: /boot/vmlinuz-4.15.0-42-generic
Found initrd image: /boot/initrd.img-4.15.0-42-generic
Found linux image: /boot/vmlinuz-4.15.0-29-generic
```

图E5-7 安装内核模块

(5) 查看安装之后的内核版本。重启系统，选择新的内核，通过 `uname -a` 命令可以查看目前正在运行的系统内核版本，操作如图E5-8所示。

```
File Edit View Search Terminal Help
jsj2018@donghua:~$ uname -a
Linux donghua 4.19.6 #1 SMP Tue Dec 4 19:45:26 PST 2018 x86_64 x86_64 x86_64 GNU/Linux
jsj2018@donghua:~$ █
```

图E5-8 查看安装之后的内核版本

4、 在新内核中创建一个系统调用 `ourcall`，要求调用时能够输出自己的姓名和学号，并写出过程和测试结果。

(1) 编辑源文件，操作如图E5-9、E5-10所示。

文件中编辑自定义的系统调用函数，输出自己的姓名和学号。

```
#include <linux/kernel.h>    //头文件，包含内核打印函数printk()等
#include <linux/syscalls.h>    //头文件，用于系统调用
```

```
SYSCALL_DEFINE0(ourcall){
    printk("stu_id\n");
    printk("stu_name\n");
    return 0;
}
```

`SYSCALL_DEFINE0(name)`: Linux的系统调用

`printk()`: 将内核信息输出到内核信息缓冲区。由于内核代码中无法调用 `printf()`函数，此处使用`printk()`函数。


```
File Edit View Search Terminal Help
else
    ${warning "Cannot use CONFIG_STACK_VALIDATION=y, please install libelf-dev
, libelf-devel or elfutils-libelf-devel")}
endif
SKIP_STACK_VALIDATION := 1
export SKIP_STACK_VALIDATION
endif
endif

ifeq ($(KBUILD_EXTMOD),)
core-y      += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ ourcall/

vmlinux-dirs := $(patsubst %/,%, $(filter %/, $(init-y) $(init-m) \
$(core-y) $(core-m) $(drivers-y) $(drivers-m) \
$(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%, $(filter %/, \
$(init-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))

init-y      := $(patsubst %/, %/built-in.a, $(init-y))
core-y      := $(patsubst %/, %/built-in.a, $(core-y))
drivers-y   := $(patsubst %/, %/built-in.a, $(drivers-y))
-- INSERT --                                     969,73-81      55%
```

图E5-13 添加ourcall目录

(4) 修改syscalls头文件，操作如图E5-14、E5-15所示。

添加函数声明asmlinkage long sys_ourcall(void);

asmlinkage：通过堆栈传递参数

```
File Edit View Search Terminal Help
jsj2018@donghua:~/linux-4.19.6$ vim include/linux/syscalls.h
```

图E5-14 修改syscalls头文件

```
File Edit View Search Terminal Help
    return do_sys_open(AT_FDCWD, filename, flags, mode);
}
extern long do_sys_truncate(const char __user *pathname, loff_t length);
static inline long ksys_truncate(const char __user *pathname, loff_t length)
{
    return do_sys_truncate(pathname, length);
}
static inline unsigned int ksys_personality(unsigned int personality)
{
    unsigned int old = current->personality;

    if (personality != 0xffffffff)
        set_personality(personality);

    return old;
}
asmlinkage long sys_ourcall(void);
#endif
"include/linux/syscalls.h" 1298L, 50598C written      1297,0-1      Bot
```

图E5-15 文件内容

(5) 添加系统调用号

在系统调用表syscall_64.tbl中添加ourcall系统调用号（系统调用号选用未被占用的，此处使用335）。

335 common ourcall __x64_sys_ourcall

若系统为32位则系统调用表为syscall_32.tbl，操作如图E5-16、E5-17所示。

```
File Edit View Search Terminal Help
jsj2018@donghua:~/linux-4.19.6$ vim arch/x86/entry/syscalls/syscall_64.tbl
```

图E5-16 添加系统调用号

```
File Edit View Search Terminal Help
325 common mlock2 __x64_sys_mlock2
326 common copy_file_range __x64_sys_copy_file_range
327 64 preadv2 __x64_sys_preadv2
328 64 pwritev2 __x64_sys_pwritev2
329 common pkey_mprotect __x64_sys_pkey_mprotect
330 common pkey_alloc __x64_sys_pkey_alloc
331 common pkey_free __x64_sys_pkey_free
332 common statx __x64_sys_statx
333 common io_pgetevents __x64_sys_io_pgetevents
334 common rseq __x64_sys_rseq
335 common ourcall __x64_sys_ourcall
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys_*() compatibility system calls if X86_X32
# is defined.
#
512 x32 rt_sigaction __x32_compat_sys_rt_sigaction
513 x32 rt_sigreturn sys32_x32_rt_sigreturn
514 x32 ioctl __x32_compat_sys_ioctl
515 x32 readv __x32_compat_sys_readv
516 x32 writev __x32_compat_sys_writev
"arch/x86/entry/syscalls/syscall_64.tbl" 389L, 15698C written 347,0-1 91%
```

图E5-17 系统调用表

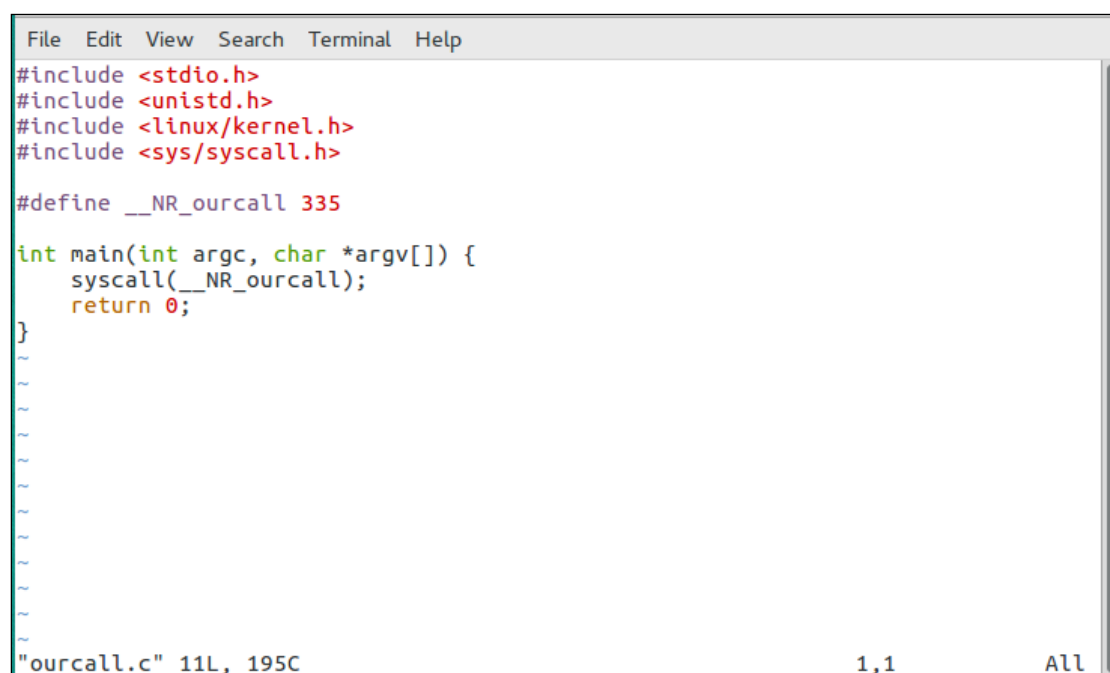
(6) 重新编译内核，操作如图E5-18所示。

```
File Edit View Search Terminal Help
jsj2018@donghua:~/linux-4.19.6$ make bzImage -j $(nproc)
DESCEND objtool
CALL scripts/checksyscalls.sh
CHK include/generated/compile.h
CC ourcall/ourcall.o
AR ourcall/built-in.a
GEN .version
CHK include/generated/compile.h
UPD include/generated/compile.h
CC init/version.o
AR init/built-in.a
AR built-in.a
LD vmlinux.o
MODPOST vmlinux.o
KSYM .tmp_kallsyms1.o
KSYM .tmp_kallsyms2.o
LD vmlinux
SORTEX vmlinux
SYSMAP System.map
CC arch/x86/boot/version.o
VOFFSET arch/x86/boot/compressed/./voffset.h
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
RELOCS arch/x86/boot/compressed/vmlinux.relocs
CC arch/x86/boot/compressed/kaslr.o
```


图E5-18 重新编译内核

编译后需要再次执行make install安装内核。

(7) 验证系统调用，通过syscall()函数发起系统调用，参数为自定义的系统调用号操作如图E5-19所示。



```
File Edit View Search Terminal Help
#include <stdio.h>
#include <unistd.h>
#include <linux/kernel.h>
#include <sys/syscall.h>

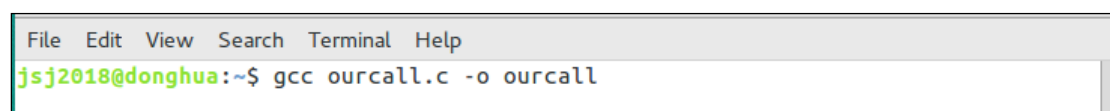
#define __NR_ourcall 335

int main(int argc, char *argv[]) {
    syscall(__NR_ourcall);
    return 0;
}
~
~
~
~
~
~
~
~
~
~
"ourcall.c" 11L, 195C 1,1 All
```

图E5-19验证系统调用

(8) 编译，操作如图E5-20所示。

gcc -o: 编译ourcall.c生成目标程序ourcall



```
File Edit View Search Terminal Help
jsj2018@donghua:~$ gcc ourcall.c -o ourcall
```

图E5-20 编译

(9) 运行，操作如图E5-21所示。

编译成功后执行命令 ./ourcall。



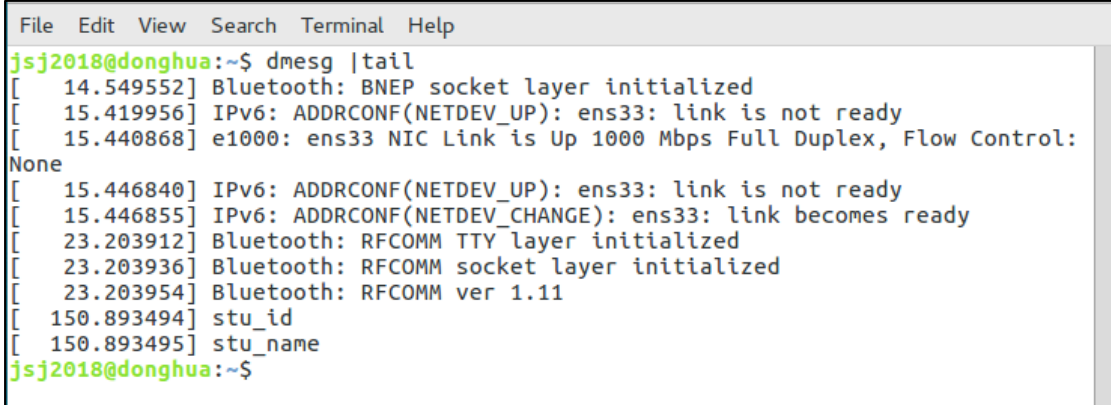
```
File Edit View Search Terminal Help
jsj2018@donghua:~$ ./ourcall
jsj2018@donghua:~$
```

图E5-21 运行

(10) 查看系统日志，操作如图E5-22所示。

dmesg: 显示内核日志

tail: 查看日志尾部的信息（查看系统调用ourcall在系统空间的运行情况）

A terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar. The prompt is 'jsj2018@donghua:~\$'. The command 'dmesg |tail' has been executed, showing the last lines of the system log. The output includes timestamps in brackets followed by log messages: Bluetooth BNEP socket layer initialized, IPv6 ADDRCONF(NETDEV_UP) for ens33, ens33 NIC link up, IPv6 ADDRCONF(NETDEV_UP) and (NETDEV_CHANGE) for ens33, Bluetooth RFCOMM TTY and socket layers initialized, Bluetooth RFCOMM version 1.1, and 'stu_id' and 'stu_name' messages.

```
File Edit View Search Terminal Help
jsj2018@donghua:~$ dmesg |tail
[ 14.549552] Bluetooth: BNEP socket layer initialized
[ 15.419956] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 15.440868] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
None
[ 15.446840] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 15.446855] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 23.203912] Bluetooth: RFCOMM TTY layer initialized
[ 23.203936] Bluetooth: RFCOMM socket layer initialized
[ 23.203954] Bluetooth: RFCOMM ver 1.11
[ 150.893494] stu_id
[ 150.893495] stu_name
jsj2018@donghua:~$
```

图E5-22 查看系统日志

四、拓展

1、深入分析 Linux4.19.6 版本的内核源代码。

在 Linux 内核 4.19.6 版本的源代码中既包括对中断机制、进程调度、内存管理、进程间通信、虚拟文件系统、设备驱动程序及网络子系统的代码实现，也包括 Linux 的启动过程的代码实现及 Linux 模块之间的联系。其中，最基础的代码是关于进程管理、内存管理及文件管理三个方面的内容。

2、分析 Android 操作系统与 Linux 之间的关系。

Android 建立在 Linux 内核基础之上，Linux 为 Android 提供核心服务,如安全,内存管理,进程管理,网络和驱动模型等。Android 按移动设备需求对 Linux 内核进行了修改，并添加了驱动相关新功能，但 Android 无法并入 Linux 主开发树。