HZ BOOKS

Linux
与自由软件资源丛书

Linux

Linux Core Kernel Commentary

（美）Scott Maxwell 著
冯 锐 刑 飞
刘隆国 陆丽娜 译

内核源代码分析

附 CD-ROM 赠

机械工业出版社
China Machine Press
CORIOLIS

Linux与自由软件资源丛书

Linux内核源代码分析

（美） Scott Maxwell 著

冯锐 邢飞 刘隆国 陆丽娜 译

机械工业出版社
China Machine Press

Linux 拥有现代操作系统所有的功能，如真正的抢先式多任务处理、支持多用户，内存保护，虚拟内存，支持SMP、UP，符合POSIX标准，联网、图形用户接口和桌面环境。具有快速性、稳定性等特点。本书通过分析Linux的内核源代码，充分揭示了Linux作为操作系统的内核是如何完成保证系统正常运行、协调多个并发进程、管理内存等工作的。

现实中，能让人自由获取的系统源代码并不多，通过本书的学习，将大大有助于读者编写自己的新程序。本书附赠光盘，有关光盘内容请见附录C。

Scott Maxwell:Linux Core Kernel Commentary.

Original English language edition published by The Coriolis Group LLC, 14455 N.Hayden Drive, Suite 220, Scottsdale, Arizona 85260 USA, telephone(602) 483-0192, fax(602) 483-0193.

Copyright © 2000 by The Coriolis Group. All rights reserved.

Simplified Chinese language edition copyright © 2000 by China Machine Press. All rights reserved.

本书中文版由美国Coriolis公司授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书版权登记号：图字：01-2000-1157

图书在版编目（CIP）数据

Linux 内核源代码分析/（美）马克斯韦尔（Maxwel 1，S.）著；冯锐等译. －北京：机械工业出版社，2000.6

（Linux 与自由软件资源丛书）

书名原文：Linux Core Kernel Commentary

ISBN 7-111-08092-0

I. L… II. ①马… ②冯… III. Linux 操作系统－程序分析 IV.TP316.89

中国版本图书馆CIP数据核字（2000）第32493号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：刘立卿

北京市密云县印刷厂印刷·新华书店北京发行所发行

2000年6月第1版·2000年10月第2次印刷

787mm × 1092mm 1/16·40.25印张

印数：7 001-10 000册

定价：69.00元（附光盘）

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

译 者 序

UNIX是人们普遍关注的一种操作系统，很多计算机的研究、开发和应用工作，大学教育与培训工作都是基于UNIX操作系统。越来越多的科研人员、大学生、研究生，不仅需要学习和使用UNIX，而且迫切要求深入了解UNIX系统的内部构造。

Linux是UNIX在个人计算机领域最有影响的操作系统之一，它已经成为商业教育及个人所用的操作系统。有了它，就可以在个人计算机上运行各种UNIX命令，使用各种UNIX软件，享有从Internet上获取的免费的各种为UNIX编写的软件和工具。其次，Linux是免费可得的，它本身就是一个可以从Internet上获取的完整的操作系统。

一般市场上出售的UNIX的完整的实现，除了价格昂贵外，还不提供其核心程序的源代码。因此，若想了解UNIX的核心或在核心程序上做一些改进等就很困难，更谈不上作为操作系统的教学和科研的平台了。而Linux则提供了从核心到上层所有的软件的全部源程序代码。此外Linux提供了很全面的计算机网络服务软件，对于从事计算机科学教学与科研的人员来说，Linux具有更重要的意义。

本书由两部分组成，第一部分是经过筛选的Linux内核源代码的一个子集。第二部分是对源代码部分进行注释，注释和代码是一一对应的。本书提供了一个最新的、完整的服务器版本的源代码（写这本书时发布的最新版本2.2.5版）；提供了对每个子系统功能的一般性概述；研究了各个子系统主要的函数和数据结构；并对开发者应怎样通过修改源代码来改善和扩展内核提出建议。通过学习将知道Linux内核是怎样构造与工作的，是如何实现系统功能的，进而也能够编写自己的代码以实现所需要的功能。本书是一本很好的深入学习Linux的参考书，我们将它翻译出来奉献给读者，希望它对操作系统的教学与科研有所帮助，也希望对广大计算机爱好者、Internet用户、网络维护人员、广大程序人员带来收益，让你们少走弯路，尽快步入Linux的奇妙世界。

全书共分11章，由冯锐、邢飞、刘隆国、陆丽娜译校。由于译者水平有限，翻译中不当之处在所难免，希望广大读者提出宝贵意见。

西安交通大学电信学院计算机系
译 者
2000.4

前　言

本书旨在给程序员和学生提供比以前更详细和更易理解的Linux内核源代码分析。书中分析了核心代码，并对重要的函数、系统调用和数据结构提供了大量的注释。

本书介绍了流行的功能强大的Linux操作系统的结构和函数实现的内幕。写作的主要目的是:

1) 提供一个最新的、完整的服务器版本的完整源代码（本书分析的版本是2.2.5版,也是写这本书时发布的最新版本）。

2) 提供一个对每个子系统功能的一般性概述。

3) 研究各个子系统主要的函数和数据结构。

4) 对开发者应怎样通过修改源代码来改进和扩展内核提出了建议。

本书的“定制”是学习内核代码最具吸引力的内容。通过理解内核是怎样工作的，能够编写自己的代码用以在操作系统中实现所需要的功能。如果允许其他人共享你的改进，你的代码甚至会在官方发行的内核代码中出现，被全世界数百万计的人们所使用。

开放源代码是指让开发者研究源代码并实现功能性扩展。Linux是全世界成长最快的操作系统,开放源代码是其快速发展的主要原因之一。从玩游戏到网上冲浪，到为大大小小的ISP们提供稳定的Web服务器平台及解决最庞杂的科学难题，Linux能胜任全部工作。它之所以能如此强大，是因为有像你一样的开发者在研究、学习和扩充这个系统。

你能从本书中学到什么

这本书集中解释了Linux内核源代码的核心中专用代码行是如何运行的。读者将学习到内核最内部的子系统是怎样构造和这种构造能够实现系统功能的原因。

本书的第一部分以易于阅读和交叉引用的格式复制了一个经过筛选的Linux 内核源代码的子集。在这本书稍后的源代码分析中，无论这行代码在何处被引用，都会在这一行前面发现一个小箭头。这个箭头指出了对此行进行分析处的页号。

本书的第二部分，即源代码分析部分，对源代码进行了讨论。每一章讨论了一个不同的内核子系统，或者是其他的功能性逻辑组件，例如系统调用或内存管理。大量的行号引用为读者指明了所讨论代码行的确切行号。

在本书正文后的附录部分，简洁地覆盖了自本书主要部分完成以后内核的变化。在附录中还包含了被内核用做软件许可证的完整的GNU通用公共许可证。

本书的使用对象

本书假设读者能阅读C语言的代码,不怕偶尔读一些汇编语言代码。并且想知道一个快速的、坚固的、可靠的、健壮的、现代的、实用的操作系统是如何工作的。一些读者也许想为前进中的Linux内核提供他们自己的改进和添加内容。

试读结束，需要全本PDF请购买 www.ertongbook.com

如何使用本书

用最适合自己的方法放松地去看本书。因为写这本书的目的是为读者提供一个参考资料，读者不必从头看到尾。因为注释和代码是一一对应的，可以从另外一个方向接近内核。

欢迎对我的书提出意见。你可以通过e-mail和我联系。地址是：lckc@ScottMaxwell.org。勘误表、更新和其他一些有用信息可以通过访问 http://www.ScottMaxwell.org/lckc.html 得到。

原书书号：ISBN 1-57610-469-9
原出版社网址：http://www.coriolis.com

目　录

第二部分 Linux 内核源代码分析

第一部分　Linux 内核源代码

试读结束，需要全本PDF请购买 www.ertongbook.com

arch/i386/kernel/entry.S

```
1 /*
2  *  linux/arch/i386/entry.S
3  *
4  *  Copyright (C) 1991, 1992  Linus Torvalds
5  */
6
7 /*
8  * entry.S contains the system-call and fault low-level
9  * handling routines.  This also contains the
10  * timer-interrupt handler, as well as all interrupts and
11  * faults that can result in a task-switch.
12  *
13  * NOTE: This code handles signal-recognition, which
14  * happens every time after a timer-interrupt and after
15  * each system call.
16  *
17  * I changed all the .align's to 4 (16 byte alignment),
18  * as that's faster on a 486.
19  *
20  * Stack layout in 'ret_from_system_call':
21  *      ptrace needs to have all regs on the stack.
22  *      if the order here is changed, it needs to be
23  *      updated in fork.c:copy_process,
24  *       signal.c:do_signal, ptrace.c and ptrace.h
25  *
26  *       0(%esp) - %ebx
27  *       4(%esp) - %ecx
28  *       8(%esp) - %edx
29  *       C(%esp) - %esi
30  *      10(%esp) - %edi
31  *      14(%esp) - %ebp
32  *      18(%esp) - %eax
33  *      1C(%esp) - %ds
34  *      20(%esp) - %es
35  *      24(%esp) - orig_eax
36  *      28(%esp) - %eip
37  *      2C(%esp) - %cs
38  *      30(%esp) - %eflags
39  *      34(%esp) - %oldesp
40  *      38(%esp) - %oldss
41  *
42  * "current" is in register %ebx during any slow entries.
43  */
44
45 #include <linux/sys.h>
46 #include <linux/linkage.h>
47 #include <asm/segment.h>
48 #define ASSEMBLY
49 #include <asm/smp.h>
50
51 EBX             = 0x00
52 ECX             = 0x04
53 EDX             = 0x08
54 ESI             = 0x0C
55 EDI             = 0x10
56 EBP             = 0x14
57 EAX             = 0x18
58 DS              = 0x1C
59 ES              = 0x20
60 ORIG_EAX        = 0x24
61 EIP             = 0x28
62 CS              = 0x2C
63 EFLAGS          = 0x30
64 OLDESP          = 0x34
65 OLDSS           = 0x38
66
67 CF_MASK         = 0x00000001
68 IF_MASK         = 0x00000200
69 NT_MASK         = 0x00004000
70 VM_MASK         = 0x00020000
71
72 /*
73  * these are offsets into the task-struct.
74  */
75 state           =  0
76 flags           =  4
77 sigpending      =  8
78 addr_limit      = 12
79 exec_domain     = 16
80 need_resched    = 20
81
82 ENOSYS = 38
83
84
85 #define SAVE_ALL                                \
86   cld;                                          \
87   pushl %es;                                    \
88   pushl %ds;                                    \
89   pushl %eax;                                   \
90   pushl %ebp;                                   \
91   pushl %edi;                                   \
92   pushl %esi;                                   \
93   pushl %edx;                                   \
94   pushl %ecx;                                   \
95   pushl %ebx;                                   \
```

```
96   movl $(__KERNEL_DS),%edx;                          \
97   movl %dx,%ds;                                      \
98   movl %dx,%es;
99
100 #define RESTORE_ALL                                  \
101   popl %ebx;                                         \
102   popl %ecx;                                         \
103   popl %edx;                                         \
104   popl %esi;                                         \
105   popl %edi;                                         \
106   popl %ebp;                                         \
107   popl %eax;                                         \
108 1:      popl %ds;                                    \
109 2:      popl %es;                                    \
110   addl $4,%esp;                                      \
111 3:      iret;                                        \
112 .section .fixup,"ax";                                \
113 4:      movl $0,(%esp);                              \
114   jmp 1b;                                            \
115 5:      movl $0,(%esp);                              \
116   jmp 2b;                                            \
117 6:      pushl %ss;                                   \
118   popl %ds;                                          \
119   pushl %ss;                                         \
120   popl %es;                                          \
121   pushl $11;                                         \
122   call do_exit;                                      \
123 .previous;                                           \
124 .section __ex_table,"a";                             \
125   .align 4;                                          \
126   .long 1b,4b;                                       \
127   .long 2b,5b;                                       \
128   .long 3b,6b;                                       \
129 .previous
130
131 #define GET_CURRENT(reg)                             \
132   movl %esp, reg;                                    \
133   andl $-8192, reg;
134
135 ENTRY(lcall7)
136   pushfl     # We get a different stack layout with call
137   pushl %eax # gates, which has to be cleaned up later..
138   SAVE_ALL
139   movl EIP(%esp),%eax     # this is eflags, not eip..
140   movl CS(%esp),%edx      # this is eip..
141   movl EFLAGS(%esp),%ecx  # and this is cs..
142   movl %eax,EFLAGS(%esp)  #
143   movl %edx,EIP(%esp)     # move to their "normal" places
144   movl %ecx,CS(%esp)      #
145   movl %esp,%ebx
146   pushl %ebx
147   andl $-8192,%ebx        # GET_CURRENT
148   movl exec_domain(%ebx),%edx  # Get the execution domain
149   movl 4(%edx),%edx       # Get lcall7 handler for domain
150   call *%edx
151   popl %eax
152   jmp ret_from_sys_call
153
154
155   ALIGN
156   .globl  ret_from_fork
157 ret_from_fork:
158 #ifdef __SMP__
159   call SYMBOL_NAME(schedule_tail)
160 #endif /* __SMP__ */
161   GET_CURRENT(%ebx)
162   jmp     ret_from_sys_call
163
164 /*
165  * Return to user mode is not as complex as all this
166  * looks, but we want the default path for a system call
167  * return to go as quickly as possible which is why some
168  * of this is less clear than it otherwise should be.
169  */
170
171 ENTRY(system_call)
172   pushl %eax                      # save orig_eax
173   SAVE_ALL
174   GET_CURRENT(%ebx)
175   cmpl $(NR_syscalls),%eax
176   jae badsys
177   testb $0x20,flags(%ebx)         # PF_TRACESYS
178   jne tracesys
179   call *SYMBOL_NAME(sys_call_table)(,%eax,4)
180   movl %eax,EAX(%esp)             # save the return value
181   ALIGN
182   .globl ret_from_sys_call
183   .globl ret_from_intr
184 ret_from_sys_call:
185   movl SYMBOL_NAME(bh_mask),%eax
186   andl SYMBOL_NAME(bh_active),%eax
187   jne handle_bottom_half
188 ret_with_reschedule:
189   cmpl $0,need_resched(%ebx)
190   jne reschedule
191   cmpl $0,sigpending(%ebx)
```

```
192   jne signal_return
193 restore_all:
194   RESTORE_ALL
195
196   ALIGN
197 signal_return:
198   sti         # we can get here from an interrupt handler
199   testl $(VM_MASK),EFLAGS(%esp)
200   movl %esp,%eax
201   jne v86_signal_return
202   xorl %edx,%edx
203   call SYMBOL_NAME(do_signal)
204   jmp restore_all
205
206   ALIGN
207 v86_signal_return:
208   call SYMBOL_NAME(save_v86_state)
209   movl %eax,%esp
210   xorl %edx,%edx
211   call SYMBOL_NAME(do_signal)
212   jmp restore_all
213
214   ALIGN
215 tracesys:
216   movl $-ENOSYS,EAX(%esp)
217   call SYMBOL_NAME(syscall_trace)
218   movl ORIG_EAX(%esp),%eax
219   call *SYMBOL_NAME(sys_call_table)(,%eax,4)
220   movl %eax,EAX(%esp)             # save the return value
221   call SYMBOL_NAME(syscall_trace)
222   jmp ret_from_sys_call
223 badsys:
224   movl $-ENOSYS,EAX(%esp)
225   jmp ret_from_sys_call
226
227   ALIGN
228 ret_from_exception:
229   movl SYMBOL_NAME(bh_mask),%eax
230   andl SYMBOL_NAME(bh_active),%eax
231   jne handle_bottom_half
232   ALIGN
233 ret_from_intr:
234   GET_CURRENT(%ebx)
235   movl EFLAGS(%esp),%eax          # mix EFLAGS and CS
236   movb CS(%esp),%al
237   testl $(VM_MASK | 3),%eax # rtn to VM86 mode|non-super?
238   jne ret_with_reschedule
239   jmp restore_all
240
241   ALIGN
242 handle_bottom_half:
243   call SYMBOL_NAME(do_bottom_half)
244   jmp ret_from_intr
245
246   ALIGN
247 reschedule:
248   call SYMBOL_NAME(schedule)    # test
249   jmp ret_from_sys_call
250
251 ENTRY(divide_error)
252   pushl $0                # no error code
253   pushl $ SYMBOL_NAME(do_divide_error)
254   ALIGN
255 error_code:
256   pushl %ds
257   pushl %eax
258   xorl %eax,%eax
259   pushl %ebp
260   pushl %edi
261   pushl %esi
262   pushl %edx
263   decl %eax                       # eax = -1
264   pushl %ecx
265   pushl %ebx
266   cld
267   movl %es,%cx
268   xchgl %eax, ORIG_EAX(%esp) # orig_eax (get error code.)
269   movl %esp,%edx
270   xchgl %ecx, ES(%esp)       # get the addr and save es.
271   pushl %eax                 # push the error code
272   pushl %edx
273   movl $(__KERNEL_DS),%edx
274   movl %dx,%ds
275   movl %dx,%es
276   GET_CURRENT(%ebx)
277   call *%ecx
278   addl $8,%esp
279   jmp ret_from_exception
280
281 ENTRY(coprocessor_error)
282   pushl $0
283   pushl $ SYMBOL_NAME(do_coprocessor_error)
284   jmp error_code
285
286 ENTRY(device_not_available)
287   pushl $-1               # mark this as an int
```

```
288   SAVE_ALL
289   GET_CURRENT(%ebx)
290   pushl $ret_from_exception
291   movl %cr0,%eax
292   testl $0x4,%eax        # EM (math emulation bit)
293   je SYMBOL_NAME(math_state_restore)
294   pushl $0               # temp storage for ORIG_EIP
295   call  SYMBOL_NAME(math_emulate)
296   addl $4,%esp
297   ret
298
299 ENTRY(debug)
300   pushl $0
301   pushl $ SYMBOL_NAME(do_debug)
302   jmp error_code
303
304 ENTRY(nmi)
305   pushl $0
306   pushl $ SYMBOL_NAME(do_nmi)
307   jmp error_code
308
309 ENTRY(int3)
310   pushl $0
311   pushl $ SYMBOL_NAME(do_int3)
312   jmp error_code
313
314 ENTRY(overflow)
315   pushl $0
316   pushl $ SYMBOL_NAME(do_overflow)
317   jmp error_code
318
319 ENTRY(bounds)
320   pushl $0
321   pushl $ SYMBOL_NAME(do_bounds)
322   jmp error_code
323
324 ENTRY(invalid_op)
325   pushl $0
326   pushl $ SYMBOL_NAME(do_invalid_op)
327   jmp error_code
328
329 ENTRY(coprocessor_segment_overrun)
330   pushl $0
331   pushl $ SYMBOL_NAME(do_coprocessor_segment_overrun)
332   jmp error_code
333
334 ENTRY(reserved)
335   pushl $0
336   pushl $ SYMBOL_NAME(do_reserved)
337   jmp error_code
338
339 ENTRY(double_fault)
340   pushl $ SYMBOL_NAME(do_double_fault)
341   jmp error_code
342
343 ENTRY(invalid_TSS)
344   pushl $ SYMBOL_NAME(do_invalid_TSS)
345   jmp error_code
346
347 ENTRY(segment_not_present)
348   pushl $ SYMBOL_NAME(do_segment_not_present)
349   jmp error_code
350
351 ENTRY(stack_segment)
352   pushl $ SYMBOL_NAME(do_stack_segment)
353   jmp error_code
354
355 ENTRY(general_protection)
356   pushl $ SYMBOL_NAME(do_general_protection)
357   jmp error_code
358
359 ENTRY(alignment_check)
360   pushl $ SYMBOL_NAME(do_alignment_check)
361   jmp error_code
362
363 ENTRY(page_fault)
364   pushl $ SYMBOL_NAME(do_page_fault)
365   jmp error_code
366
367 ENTRY(spurious_interrupt_bug)
368   pushl $0
369   pushl $ SYMBOL_NAME(do_spurious_interrupt_bug)
370   jmp error_code
371
372 .data
373 ENTRY(sys_call_table)
374   .long SYMBOL_NAME(sys_ni_syscall)       /* 0 */
375   .long SYMBOL_NAME(sys_exit)
376   .long SYMBOL_NAME(sys_fork)
377   .long SYMBOL_NAME(sys_read)
378   .long SYMBOL_NAME(sys_write)
379   .long SYMBOL_NAME(sys_open)             /* 5 */
380   .long SYMBOL_NAME(sys_close)
381   .long SYMBOL_NAME(sys_waitpid)
382   .long SYMBOL_NAME(sys_creat)
383   .long SYMBOL_NAME(sys_link)
```

```
384   .long SYMBOL_NAME(sys_unlink)           /* 10 */
385   .long SYMBOL_NAME(sys_execve)
386   .long SYMBOL_NAME(sys_chdir)
387   .long SYMBOL_NAME(sys_time)
388   .long SYMBOL_NAME(sys_mknod)
389   .long SYMBOL_NAME(sys_chmod)            /* 15 */
390   .long SYMBOL_NAME(sys_lchown)
391   .long SYMBOL_NAME(sys_ni_syscall) /*old break holder*/
392   .long SYMBOL_NAME(sys_stat)
393   .long SYMBOL_NAME(sys_lseek)
394   .long SYMBOL_NAME(sys_getpid)           /* 20 */
395   .long SYMBOL_NAME(sys_mount)
396   .long SYMBOL_NAME(sys_oldumount)
397   .long SYMBOL_NAME(sys_setuid)
398   .long SYMBOL_NAME(sys_getuid)
399   .long SYMBOL_NAME(sys_stime)            /* 25 */
400   .long SYMBOL_NAME(sys_ptrace)
401   .long SYMBOL_NAME(sys_alarm)
402   .long SYMBOL_NAME(sys_fstat)
403   .long SYMBOL_NAME(sys_pause)
404   .long SYMBOL_NAME(sys_utime)            /* 30 */
405   .long SYMBOL_NAME(sys_ni_syscall) /* old stty holder */
406   .long SYMBOL_NAME(sys_ni_syscall) /* old gtty holder */
407   .long SYMBOL_NAME(sys_access)
408   .long SYMBOL_NAME(sys_nice) /*next: old ftime holder*/
409   .long SYMBOL_NAME(sys_ni_syscall)       /* 35 */
410   .long SYMBOL_NAME(sys_sync)
411   .long SYMBOL_NAME(sys_kill)
412   .long SYMBOL_NAME(sys_rename)
413   .long SYMBOL_NAME(sys_mkdir)
414   .long SYMBOL_NAME(sys_rmdir)            /* 40 */
415   .long SYMBOL_NAME(sys_dup)
416   .long SYMBOL_NAME(sys_pipe)
417   .long SYMBOL_NAME(sys_times)
418   .long SYMBOL_NAME(sys_ni_syscall) /* old prof holder */
419   .long SYMBOL_NAME(sys_brk)              /* 45 */
420   .long SYMBOL_NAME(sys_setgid)
421   .long SYMBOL_NAME(sys_getgid)
422   .long SYMBOL_NAME(sys_signal)
423   .long SYMBOL_NAME(sys_geteuid)
424   .long SYMBOL_NAME(sys_getegid)          /* 50 */
425   .long SYMBOL_NAME(sys_acct)
426   .long SYMBOL_NAME(sys_umount) /*recyc never used phys*/
427   .long SYMBOL_NAME(sys_ni_syscall) /* old lock holder */
428   .long SYMBOL_NAME(sys_ioctl)
429   .long SYMBOL_NAME(sys_fcntl)            /* 55 */
430   .long SYMBOL_NAME(sys_ni_syscall) /* old mpx holder */
431   .long SYMBOL_NAME(sys_setpgid)
432   .long SYMBOL_NAME(sys_ni_syscall) /*old ulimit holder*/
433   .long SYMBOL_NAME(sys_olduname)
434   .long SYMBOL_NAME(sys_umask)            /* 60 */
435   .long SYMBOL_NAME(sys_chroot)
436   .long SYMBOL_NAME(sys_ustat)
437   .long SYMBOL_NAME(sys_dup2)
438   .long SYMBOL_NAME(sys_getppid)
439   .long SYMBOL_NAME(sys_getpgrp)          /* 65 */
440   .long SYMBOL_NAME(sys_setsid)
441   .long SYMBOL_NAME(sys_sigaction)
442   .long SYMBOL_NAME(sys_sgetmask)
443   .long SYMBOL_NAME(sys_ssetmask)
444   .long SYMBOL_NAME(sys_setreuid)         /* 70 */
445   .long SYMBOL_NAME(sys_setregid)
446   .long SYMBOL_NAME(sys_sigsuspend)
447   .long SYMBOL_NAME(sys_sigpending)
448   .long SYMBOL_NAME(sys_sethostname)
449   .long SYMBOL_NAME(sys_setrlimit)        /* 75 */
450   .long SYMBOL_NAME(sys_getrlimit)
451   .long SYMBOL_NAME(sys_getrusage)
452   .long SYMBOL_NAME(sys_gettimeofday)
453   .long SYMBOL_NAME(sys_settimeofday)
454   .long SYMBOL_NAME(sys_getgroups)        /* 80 */
455   .long SYMBOL_NAME(sys_setgroups)
456   .long SYMBOL_NAME(old_select)
457   .long SYMBOL_NAME(sys_symlink)
458   .long SYMBOL_NAME(sys_lstat)
459   .long SYMBOL_NAME(sys_readlink)         /* 85 */
460   .long SYMBOL_NAME(sys_uselib)
461   .long SYMBOL_NAME(sys_swapon)
462   .long SYMBOL_NAME(sys_reboot)
463   .long SYMBOL_NAME(old_readdir)
464   .long SYMBOL_NAME(old_mmap)             /* 90 */
465   .long SYMBOL_NAME(sys_munmap)
466   .long SYMBOL_NAME(sys_truncate)
467   .long SYMBOL_NAME(sys_ftruncate)
468   .long SYMBOL_NAME(sys_fchmod)
469   .long SYMBOL_NAME(sys_fchown)           /* 95 */
470   .long SYMBOL_NAME(sys_getpriority)
471   .long SYMBOL_NAME(sys_setpriority)
472   .long SYMBOL_NAME(sys_ni_syscall) /*old profil holder*/
473   .long SYMBOL_NAME(sys_statfs)
474   .long SYMBOL_NAME(sys_fstatfs)          /* 100 */
475   .long SYMBOL_NAME(sys_ioperm)
476   .long SYMBOL_NAME(sys_socketcall)
477   .long SYMBOL_NAME(sys_syslog)
478   .long SYMBOL_NAME(sys_setitimer)
479   .long SYMBOL_NAME(sys_getitimer)        /* 105 */
```

```
480   .long SYMBOL_NAME(sys_newstat)
481   .long SYMBOL_NAME(sys_newlstat)
482   .long SYMBOL_NAME(sys_newfstat)
483   .long SYMBOL_NAME(sys_uname)
484   .long SYMBOL_NAME(sys_iopl)             /* 110 */
485   .long SYMBOL_NAME(sys_vhangup)
486   .long SYMBOL_NAME(sys_idle)
487   .long SYMBOL_NAME(sys_vm86old)
488   .long SYMBOL_NAME(sys_wait4)
489   .long SYMBOL_NAME(sys_swapoff)          /* 115 */
490   .long SYMBOL_NAME(sys_sysinfo)
491   .long SYMBOL_NAME(sys_ipc)
492   .long SYMBOL_NAME(sys_fsync)
493   .long SYMBOL_NAME(sys_sigreturn)
494   .long SYMBOL_NAME(sys_clone)            /* 120 */
495   .long SYMBOL_NAME(sys_setdomainname)
496   .long SYMBOL_NAME(sys_newuname)
497   .long SYMBOL_NAME(sys_modify_ldt)
498   .long SYMBOL_NAME(sys_adjtimex)
499   .long SYMBOL_NAME(sys_mprotect)         /* 125 */
500   .long SYMBOL_NAME(sys_sigprocmask)
501   .long SYMBOL_NAME(sys_create_module)
502   .long SYMBOL_NAME(sys_init_module)
503   .long SYMBOL_NAME(sys_delete_module)
504   .long SYMBOL_NAME(sys_get_kernel_syms)  /* 130 */
505   .long SYMBOL_NAME(sys_quotactl)
506   .long SYMBOL_NAME(sys_getpgid)
507   .long SYMBOL_NAME(sys_fchdir)
508   .long SYMBOL_NAME(sys_bdflush)
509   .long SYMBOL_NAME(sys_sysfs)            /* 135 */
510   .long SYMBOL_NAME(sys_personality)
511   .long SYMBOL_NAME(sys_ni_syscall) /* for afs_syscall */
512   .long SYMBOL_NAME(sys_setfsuid)
513   .long SYMBOL_NAME(sys_setfsgid)
514   .long SYMBOL_NAME(sys_llseek)           /* 140 */
515   .long SYMBOL_NAME(sys_getdents)
516   .long SYMBOL_NAME(sys_select)
517   .long SYMBOL_NAME(sys_flock)
518   .long SYMBOL_NAME(sys_msync)
519   .long SYMBOL_NAME(sys_readv)            /* 145 */
520   .long SYMBOL_NAME(sys_writev)
521   .long SYMBOL_NAME(sys_getsid)
522   .long SYMBOL_NAME(sys_fdatasync)
523   .long SYMBOL_NAME(sys_sysctl)
524   .long SYMBOL_NAME(sys_mlock)            /* 150 */
525   .long SYMBOL_NAME(sys_munlock)
526   .long SYMBOL_NAME(sys_mlockall)
527   .long SYMBOL_NAME(sys_munlockall)
528   .long SYMBOL_NAME(sys_sched_setparam)
529   .long SYMBOL_NAME(sys_sched_getparam)   /* 155 */
530   .long SYMBOL_NAME(sys_sched_setscheduler)
531   .long SYMBOL_NAME(sys_sched_getscheduler)
532   .long SYMBOL_NAME(sys_sched_yield)
533   .long SYMBOL_NAME(sys_sched_get_priority_max)
534   .long SYMBOL_NAME(sys_sched_get_priority_min) /* 160 */
535   .long SYMBOL_NAME(sys_sched_rr_get_interval)
536   .long SYMBOL_NAME(sys_nanosleep)
537   .long SYMBOL_NAME(sys_mremap)
538   .long SYMBOL_NAME(sys_setresuid)
539   .long SYMBOL_NAME(sys_getresuid)        /* 165 */
540   .long SYMBOL_NAME(sys_vm86)
541   .long SYMBOL_NAME(sys_query_module)
542   .long SYMBOL_NAME(sys_poll)
543   .long SYMBOL_NAME(sys_nfsservctl)
544   .long SYMBOL_NAME(sys_setresgid)        /* 170 */
545   .long SYMBOL_NAME(sys_getresgid)
546   .long SYMBOL_NAME(sys_prctl)
547   .long SYMBOL_NAME(sys_rt_sigreturn)
548   .long SYMBOL_NAME(sys_rt_sigaction)
549   .long SYMBOL_NAME(sys_rt_sigprocmask)   /* 175 */
550   .long SYMBOL_NAME(sys_rt_sigpending)
551   .long SYMBOL_NAME(sys_rt_sigtimedwait)
552   .long SYMBOL_NAME(sys_rt_sigqueueinfo)
553   .long SYMBOL_NAME(sys_rt_sigsuspend)
554   .long SYMBOL_NAME(sys_pread)            /* 180 */
555   .long SYMBOL_NAME(sys_pwrite)
556   .long SYMBOL_NAME(sys_chown)
557   .long SYMBOL_NAME(sys_getcwd)
558   .long SYMBOL_NAME(sys_capget)
559   .long SYMBOL_NAME(sys_capset)           /* 185 */
560   .long SYMBOL_NAME(sys_sigaltstack)
561   .long SYMBOL_NAME(sys_sendfile)
562   .long SYMBOL_NAME(sys_ni_syscall)  /* streams1 */
563   .long SYMBOL_NAME(sys_ni_syscall)  /* streams2 */
564   .long SYMBOL_NAME(sys_vfork)            /* 190 */
565
566   /*
567    * NOTE!! This doesn't have to be exact - we just have
568    * to make sure we have _enough_ of the sys_ni_syscall
569    * entries. Don't panic if you notice that this hasn't
570    * been shrunk every time we add a new system call.
571    */
572   .rept NR_syscalls-190
573     .long SYMBOL_NAME(sys_ni_syscall)
574   .endr
```


试读结束，需要全本PDF请购买 www.ertongbook.com