open + ... + mount + ...

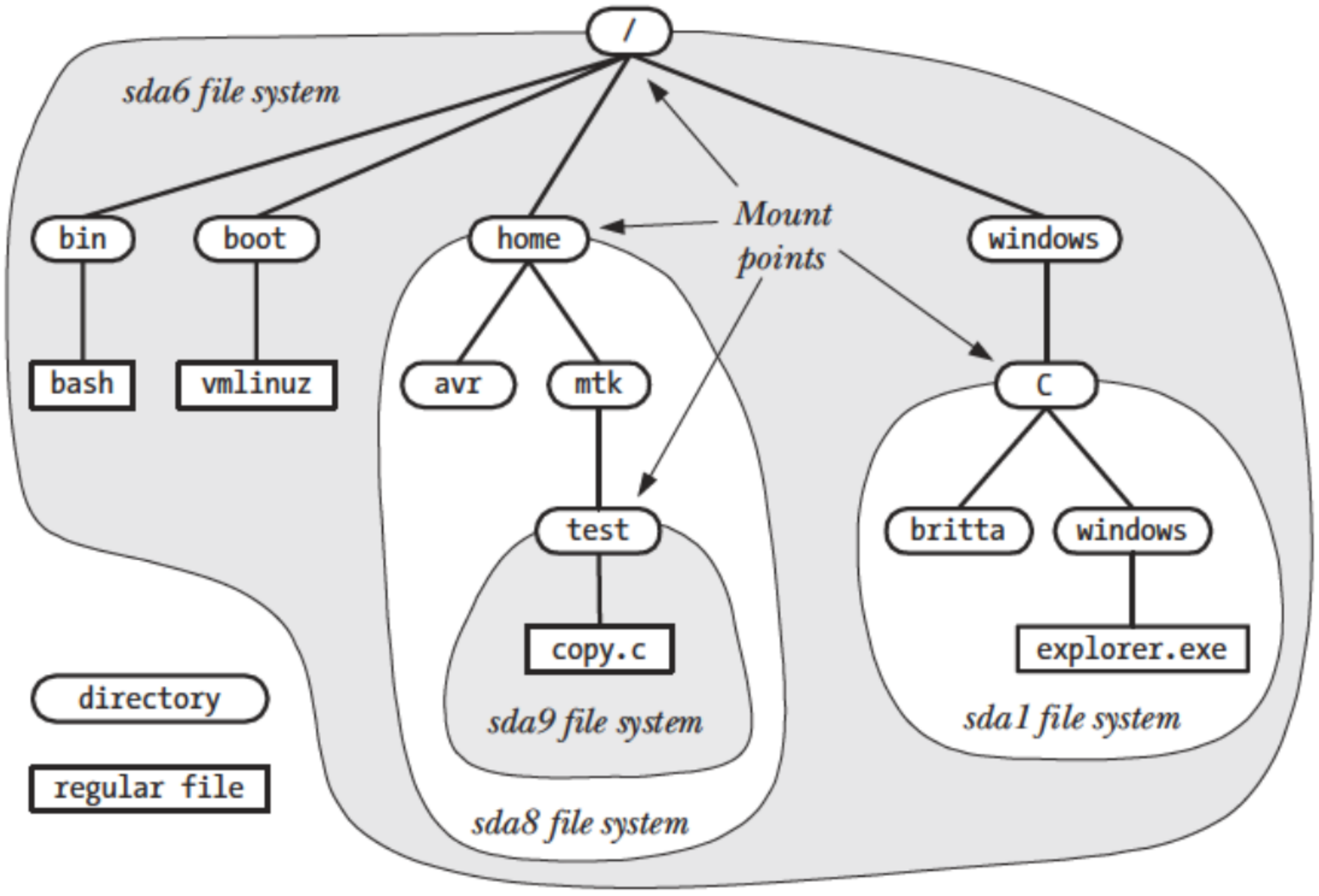
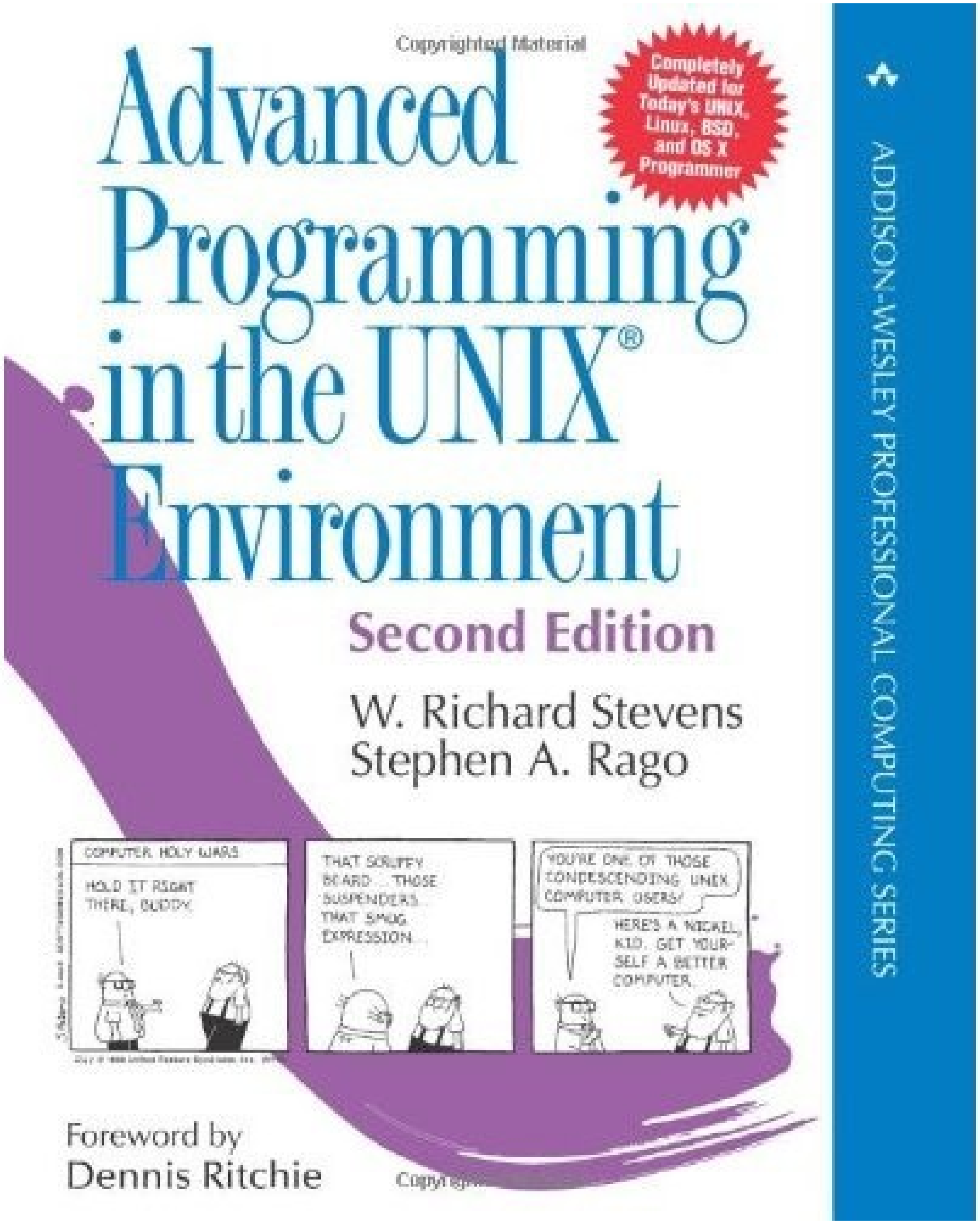


Figure 14-4: Example directory hierarchy showing file-system mount points



Copyrighted Material

Completely Updated for Today's UNIX, Linux, BSD, and OS X Programmer

Advanced Programming in the UNIX® Environment

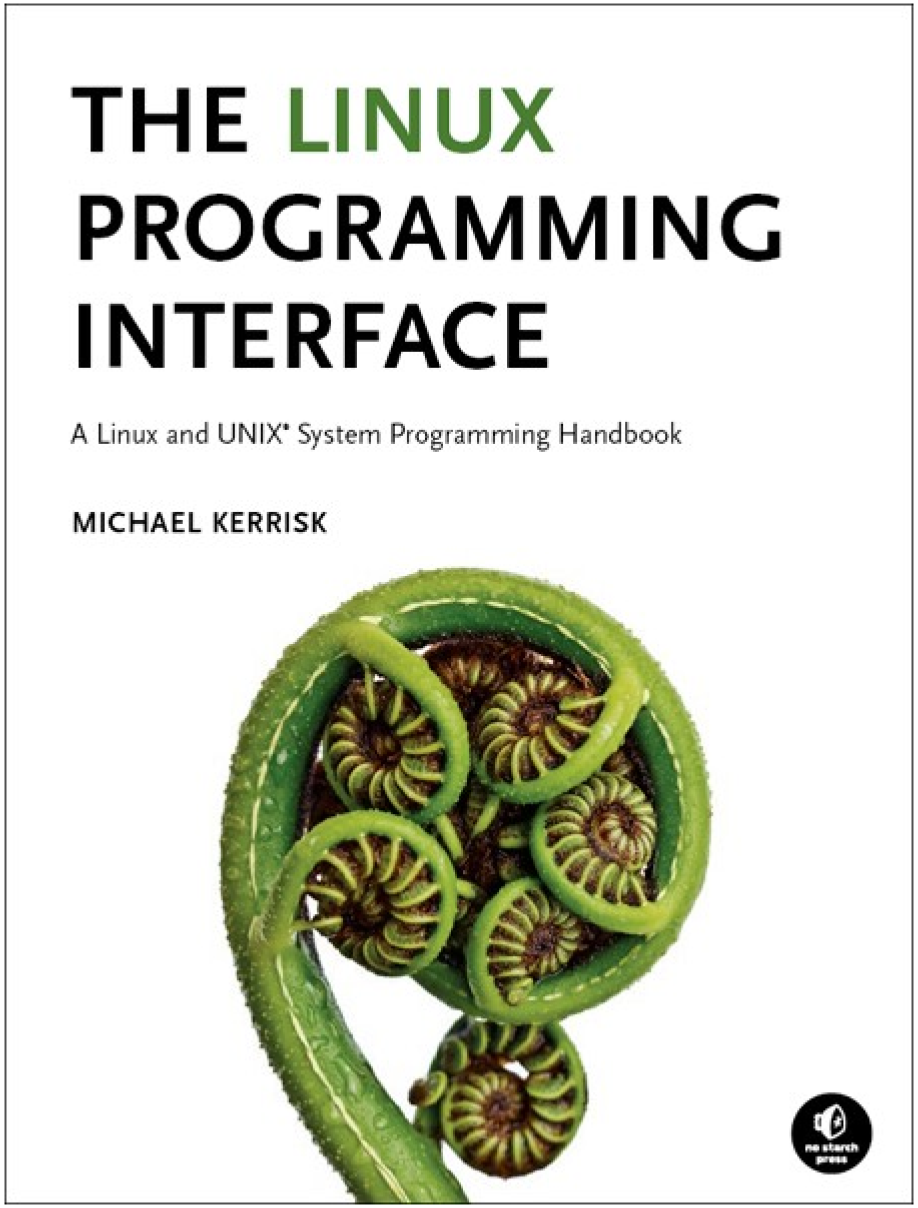
Second Edition

W. Richard Stevens
Stephen A. Rago

Foreword by
Dennis Ritchie

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

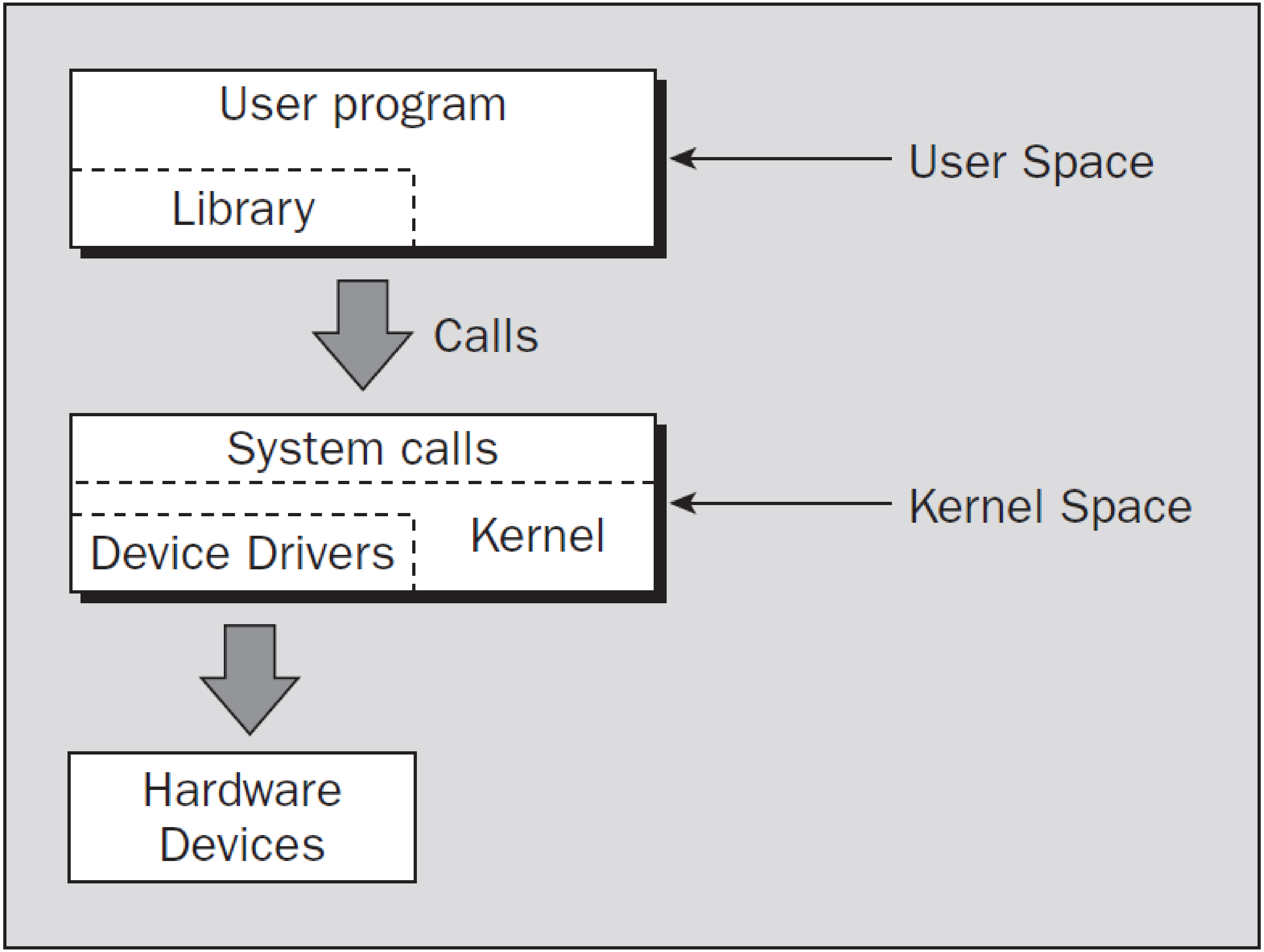
COMPUTER HOLY WARS
HOLD IT RIGHT THERE, BUDDY.
THAT SCRUFFY BEARD... THOSE SUSPENDERS... THAT SMUG EXPRESSION...
YOU'RE ONE OF THOSE CONDESCENDING UNIX COMPUTER USERS!
HERE'S A NICKEL, KID. GET YOURSELF A BETTER COMPUTER.


THE LINUX PROGRAMMING INTERFACE

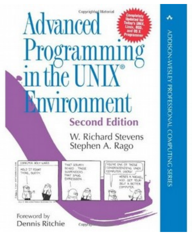
A Linux and UNIX® System Programming Handbook

MICHAEL KERRISK

no starch press


User program
Library
User Space
Calls
System calls
Kernel
Device Drivers
Kernel Space
Hardware Devices

A file is opened or created by calling the `open` function:



```
#include <fcntl.h>

int open(const char *pathname, int oflag, ... /* mode_t mode */ );
```

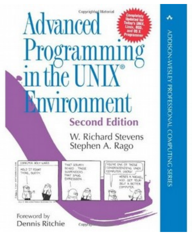
Returns: file descriptor if OK, -1 on error

One and only one of these three constants: `O_RDONLY`, `O_WRONLY`, `O_RDWR`.
Optional: `O_CREAT`, `O_EXCL`, `O_TRUNC`, `O_NOCTTY`, `O_NONBLOCK`, …

An open file is closed by calling the `close` function:

```
#include <unistd.h>

int close(int filedes);
```

Returns: 0 if OK, -1 on error

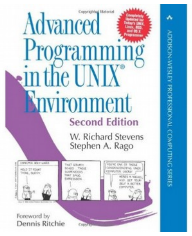


An open file's offset can be set explicitly by calling `lseek`:

```
#include <unistd.h>

off_t lseek(int filedes, off_t offset, int whence);
```

Returns: new file offset if OK, -1 on error

The value of the *whence* argument: `SEEK_SET`, `SEEK_CUR`, `SEEK_END`.



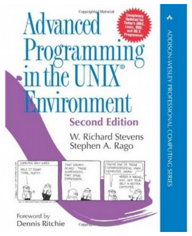
Data is read from an open file with the `read` function:

```
#include <unistd.h>

ssize_t read(int filedes, void *buf, size_t nbytes);
```

Returns: number of bytes read, 0 if end of file, -1 on error

`read` can block the caller forever if data isn't present with certain file types (pipes, terminal devices, and network devices).

If we `read` from a pipe whose write end has been closed, `read` returns 0 to indicate an end of file after all the data has been read. We should say that this end of file is not generated until there are no more writers for the pipe. It's possible to duplicate a pipe descriptor so that multiple processes have the pipe open for writing.



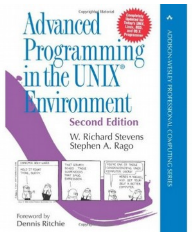
Data is written to an open file with the `write` function:

```
#include <unistd.h>

ssize_t write(int filedes, const void *buf, size_t nbytes);
```

Returns: number of bytes written if OK, -1 on error

`write` can block the caller forever if the data can't be accepted immediately by certain file types (no room in the pipe, network flow control, etc.).

If we `write` to a pipe whose read end has been closed, the signal `SIGPIPE` is generated. If we either ignore the signal or catch it and return from the signal handler, `write` returns -1 with `errno` set to `EPIPE`.

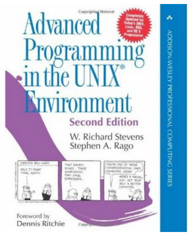


File's status information can be obtained with the `stat` function:

```
#include <sys/stat.h>

int stat(const char *restrict pathname, struct stat *restrict buf);
int fstat(int filedes, struct stat buf);
int lstat(const char *restrict pathname, struct stat *restrict buf);
```

All three return: 0 if OK, -1 on error

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```



Directories are created with the `mkdir` function:

```
#include <sys/stat.h>

int mkdir(const char *pathname, mode_t mode);
```

Returns: 0 if OK, -1 on error

An empty directory is deleted with the `rmdir` function:

```
#include <unistd.h>

int rmdir(const char *pathname);
```

Returns: 0 if OK, -1 on error

Hard links:

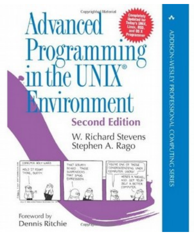
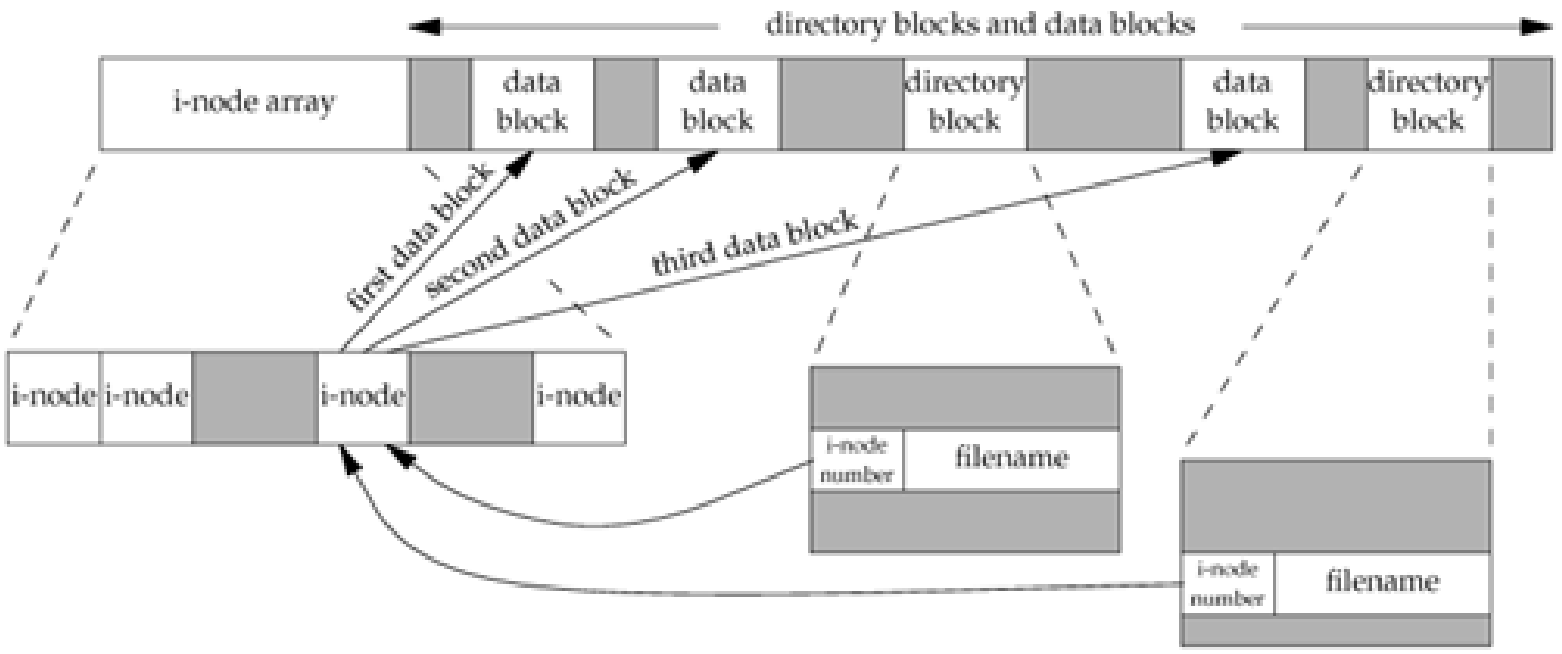


Figure 4.14. Cylinder group's i-nodes and data blocks in more detail



Any file can have multiple directory entries pointing to its i-node. The way we create a link to an existing file is with the `link` function:

```
#include <unistd.h>

int link(const char *existingpath, const char *newpath);
```

Returns: 0 if OK, -1 on error

To remove an existing directory entry, we call the `unlink` function:

```
#include <unistd.h>

int unlink(const char *pathname);
```

Returns: 0 if OK, -1 on error

A symbolic link is an indirect pointer to a file.
A symbolic link is created with the `symlink` function:

```
#include <unistd.h>

int symlink(const char *actualpath, const char *sympath);
```

Returns: 0 if OK, -1 on error

A new directory entry, *sympath*, is created that points to *actualpath*. It is not required that *actualpath* exist when the symbolic link is created.

Because the `open` function follows a symbolic link, we need a way to open the link itself and read the name in the link. The `readlink` function does this.

```
#include <unistd.h>

ssize_t readlink(const char* restrict pathname, char *restrict buf,
                 size_t bufsize);
```

Returns: number of bytes read if OK, -1 on error

Examen 2 (2011-1), parte de la pregunta 3:

```
$ mkdir d0 d1 d2
$ echo File a > d0/a
$ ln -s a d0/b
$ ln -s ../d1/c d0/c
$ ln -s ../d0/d d1/c
$ ln d0/a d0/d
$ ln -s ../d2/f d0/e
$ echo File e > d0/e
```

Indique los contenidos de los 3 directorios creados y los contenidos de sus archivos.

```
$ mkdir d0 d1 d2

$ ls -l
total 12
drwxrwxr-x 2 vk vk 4096 set  9 16:18 d0
drwxrwxr-x 2 vk vk 4096 set  9 16:18 d1
drwxrwxr-x 2 vk vk 4096 set  9 16:18 d2


$ echo File a > d0/a

$ ls -l d0/
total 4
-rw-rw-r-- 1 vk vk 7 set  9 16:18 a

$ cat d0/a
File a
```

```
$ ln -s a d0/b

$ ls -l d0/
total 4
-rw-rw-r-- 1 vk vk 7 set  9 16:18 a
lrwxrwxrwx 1 vk vk 1 set  9 16:20 b -> a

$ ls -l d0/b
lrwxrwxrwx 1 vk vk 1 set  9 16:20 d0/b -> a

$ cat d0/b
File a
```

```
$ ln -s ../d1/c d0/c

$ ls -l d0/
total 4
-rw-rw-r-- 1 vk vk 7 set  9 16:18 a
lrwxrwxrwx 1 vk vk 1 set  9 16:20 b -> a
lrwxrwxrwx 1 vk vk 7 set  9 16:21 c -> ../d1/c

$ ls -l d0/c
lrwxrwxrwx 1 vk vk 7 set  9 16:21 d0/c -> ../d1/c

$ cat d0/c
cat: d0/c: No existe el archivo o el directorio
```

dead link

```
$ ln -s ../d0/d d1/c

$ ls -l d1/
total 0
lrwxrwxrwx 1 vk vk 7 set  9 16:31 c -> ../d0/d
```

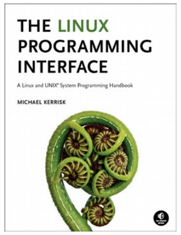
```
$ ln d0/a d0/d

$ ls -li d0/
total 8
6947875 -rw-rw-r-- 2 vk vk 7 set  9 16:18 a
6947876 lrwxrwxrwx 1 vk vk 1 set  9 16:20 b -> a
6947877 lrwxrwxrwx 1 vk vk 7 set  9 16:21 c -> ../d1/c
6947875 -rw-rw-r-- 2 vk vk 7 set  9 16:18 d
```

!

```
$ ln -s ../d2/f d0/e

$ ls -l d0/
total 8
-rw-rw-r-- 2 vk vk 7 set  9 16:18 a
lrwxrwxrwx 1 vk vk 1 set  9 16:20 b -> a
lrwxrwxrwx 1 vk vk 7 set  9 16:21 c -> ../d1/c
-rw-rw-r-- 2 vk vk 7 set  9 16:18 d
lrwxrwxrwx 1 vk vk 7 set  9 16:33 e -> ../d2/f


$ echo File e > d0/e

$ ls -l d0/
total 8
-rw-rw-r-- 2 vk vk 7 set  9 16:18 a
lrwxrwxrwx 1 vk vk 1 set  9 16:20 b -> a
lrwxrwxrwx 1 vk vk 7 set  9 16:21 c -> ../d1/c
-rw-rw-r-- 2 vk vk 7 set  9 16:18 d
lrwxrwxrwx 1 vk vk 7 set  9 16:33 e -> ../d2/f
```

```
$ ls -l d1/
total 0
lrwxrwxrwx 1 vk vk 7 set  9 16:31 c -> ../d0/d

$ ls -l d2/
total 4
-rw-rw-r-- 1 vk vk 7 set  9 16:34 f

$ cat d0/{a,b,c,d,e}
File a
File a
File a
File a
File e

$ cat d1/c
File a

$ cat d2/f
File e
```

```
$ mount
/dev/sda6 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
/dev/sda8 on /home type ext3 (rw,acl,user_xattr)
/dev/sda1 on /windows/C type vfat (rw,noexec,nosuid,nodev)
/dev/sda9 on /home/mtk/test type reiserfs (rw)
```



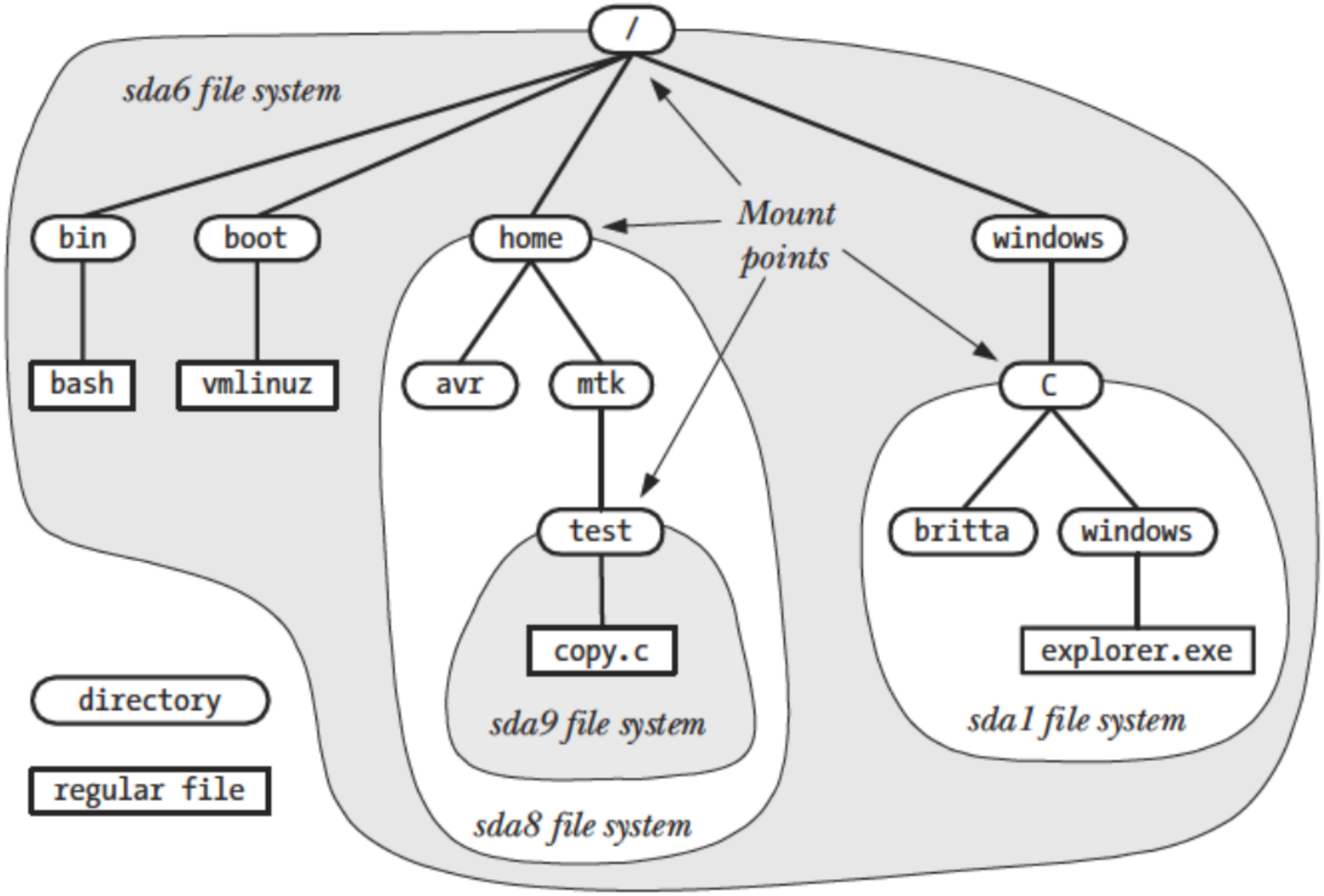


Figure 14-4: Example directory hierarchy showing file-system mount points

The `mount()` system call mounts the file system contained on the device specified by `source` under the directory (the *mount point*) specified by `target`:

```
#include <sys/mount.h>

int mount(const char *source, const char *target, const char *fstype,
unsigned long mountflags, const void *data);
```

Returns: 0 if OK, -1 on error

The `umount()` system call unmounts a mounted file system:

```
#include <sys/mount.h>

int umount(const char *target);
```

Returns: 0 if OK, -1 on error

Every process has a current working directory. We can change the current working directory of the calling process by calling the `chdir` or `fchdir` functions:

```
#include <unistd.h>

int chdir(const char *pathname);
int fchdir(int filedes);
```

Both return: 0 if OK, -1 on error

These two functions allow us to change the file access permissions for an existing file:

```
#include <sys/stat.h>

int chmod(const char *pathname, mode_t mode);
int fchmod(int filedes, mode_t mode);
```

Both return: 0 if OK, -1 on error

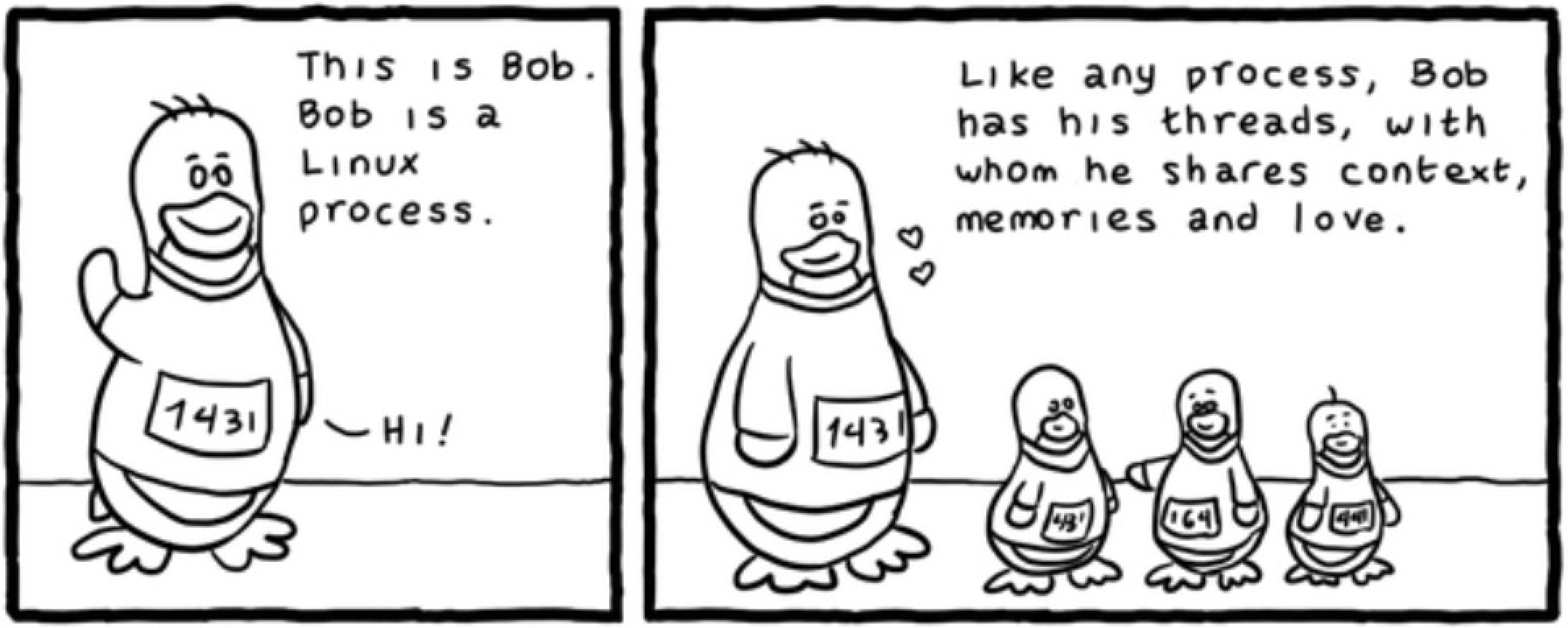
The `kill` function sends a signal to a process or a group of processes:

```
#include <signal.h>

int kill(pid_t pid, int signo);
```

Returns: 0 if OK, -1 on error

```
$ man 7 signal
...
       Signal     Value     Action   Comment
       ──────────────────────────────────────────────────────────────
       SIGHUP        1       Term    Hangup detected on controlling terminal
                                     or death of controlling process
       SIGINT        2       Term    Interrupt from keyboard
       SIGQUIT       3       Core    Quit from keyboard
       SIGILL        4       Core    Illegal Instruction
       SIGABRT       6       Core    Abort signal from abort(3)
       SIGFPE        8       Core    Floating point exception
       SIGKILL       9       Term    Kill signal
       SIGSEGV      11       Core    Invalid memory reference
       SIGPIPE      13       Term    Broken pipe: write to pipe with no
                                     readers
       SIGALRM      14       Term    Timer signal from alarm(2)
       SIGTERM      15       Term    Termination signal
       SIGUSR1   30,10,16    Term    User-defined signal 1
       SIGUSR2   31,12,17    Term    User-defined signal 2
       SIGCHLD   20,17,18    Ign     Child stopped or terminated
       SIGCONT   19,18,25    Cont    Continue if stopped
       SIGSTOP   17,19,23    Stop    Stop process
       SIGTSTP   18,20,24    Stop    Stop typed at tty
       SIGTTIN   21,21,26    Stop    tty input for background process
       SIGTTOU   22,22,27    Stop    tty output for background process

       The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.
```

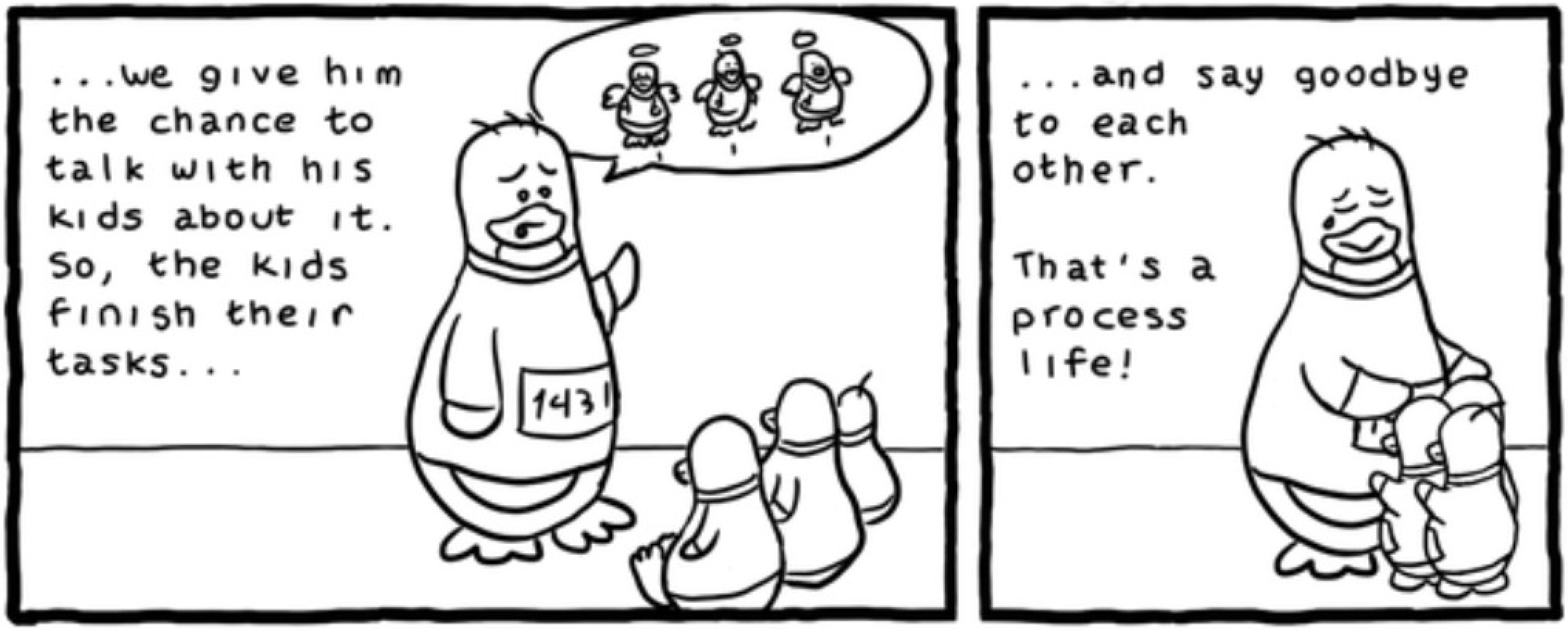
```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```


This is Bob. Bob is a Linux process.
1431
Hi!
Like any process, Bob has his threads, with whom he shares context, memories and love.
1431
1431
1641
1441

Daniel Stori (http://turnoff.us/geek/)

SIGTERM
1431
Sysadmin
And like all processes, inevitably sometime he will be killed.
When we gracefully kill a process with a soft SIGTERM...

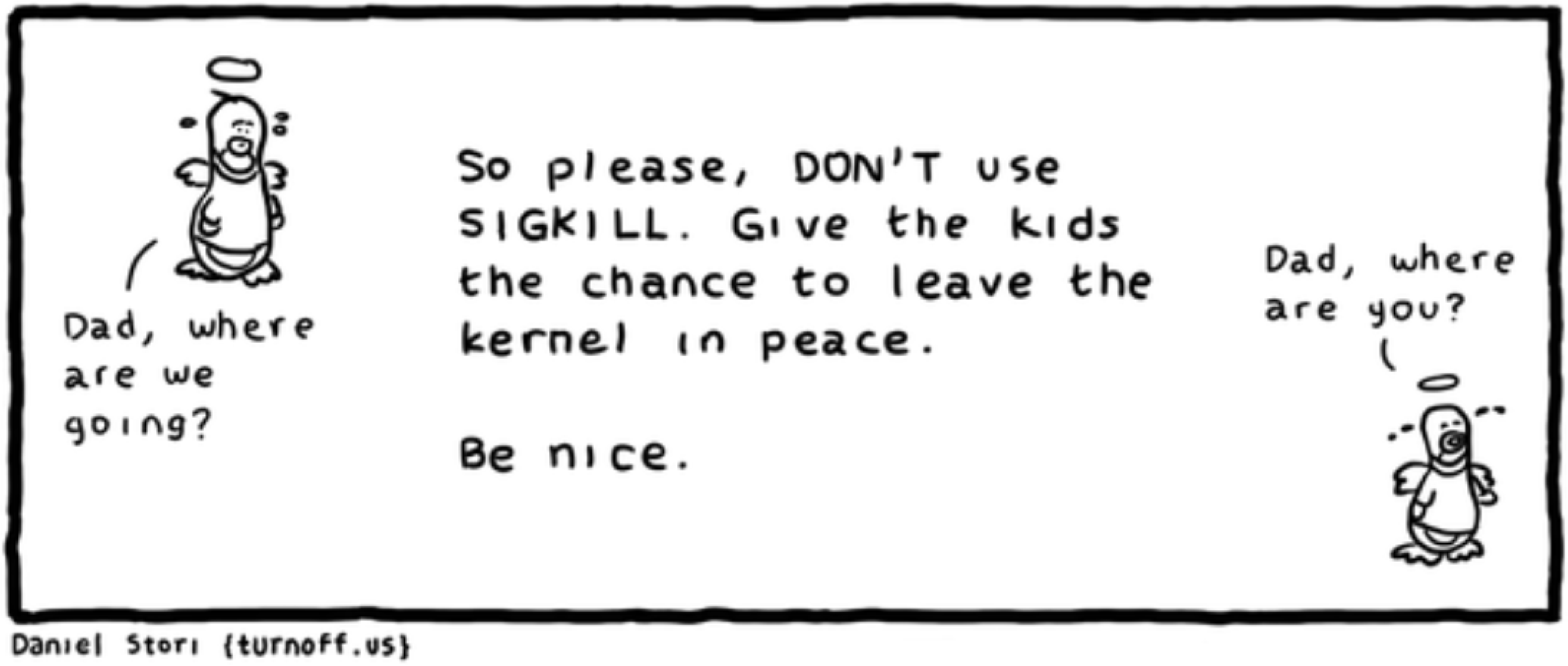
Daniel Stori (http://turnoff.us/geek/)


...we give him the chance to talk with his kids about it. So, the kids finish their tasks...
1431
...and say goodbye to each other.
That's a process life!

Daniel Stori (http://turnoff.us/geek/)

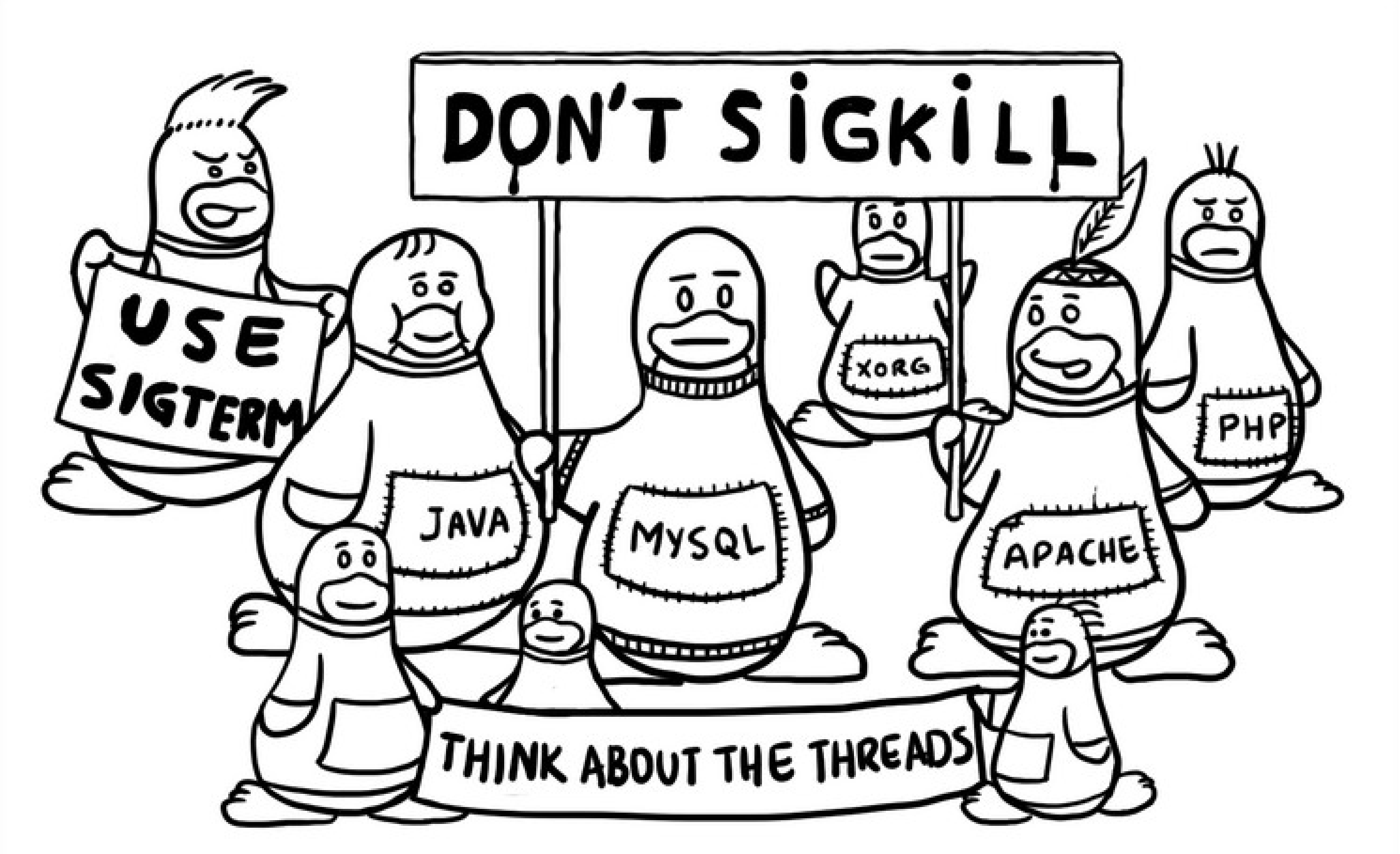
On the other hand, when we brutally kill a process with a SIGKILL, we prevent them from finishing their job and say goodbye...
SIGKILL
SysAdmin
1671
...and this is so SAD!
DAD!!
SIGKILL

Daniel Stori (http://turnoff.us/geek/)


Dad, where are we going?
So please, DON'T use SIGKILL. Give the kids the chance to leave the kernel in peace.
Be nice.
Dad, where are you?
Daniel Stori {turnoff.us}

Daniel Stori (http://turnoff.us/geek/)

`$ Adopt a good cause, DON'T SIGKILL`



Daniel Stori (http://turnoff.us/geek/)