**Accessing the System Call from User-Space**

Generally, the C library provides support for system calls. User applications can pull in function prototypes from the standard headers and link with the C library to use your system call (or the library routine that, in turn, uses your syscall call). If you just wrote the system call, however, it is doubtful that glibc already supports it!

Thankfully, Linux provides a set of macros for wrapping access to system calls. It sets up the register contents and issues the trap instructions.These macros are named

\_syscall n () , where n is between 0 and 6.The number corresponds to the number of

parameters passed into the syscall because the macro needs to know how many parameters to expect and, consequently, push into registers. For example, consider the system call open() , defined as

long open(const char \*filename, int flags, int mode)

The syscall macro to use this system call without explicit library support would be

#define \_\_NR\_open 5

\_syscall3(long, open, const char \*, filename, int, flags, int, mode)

Then, the application can simply call open() .

For each macro, there are 2 + 2 × n parameters.The first parameter corresponds to the

return type of the syscall.The second is the name of the system call. Next follows the

type and name for each parameter in order of the system call.The \_\_NR\_open define is in <asm/unistd.h> ; it is the system call number.