Table 6.2 UNIX Signals

Value	Name	Description
01	SIGHUP	Hang up; sent to process when kernel assumes that the user of that process is doing no useful work
02	SIGINT	Interrupt
03	SIGQUIT	Quit; sent by user to induce halting of process and production of core dump
04	SIGILL	Illegal instruction
05	SIGTRAP	Trace trap; triggers the execution of code for process tracing
06	SIGIOT	IOT instruction
07	SIGEMT	EMT instruction
08	SIGFPE	Floating-point exception
09	SIGKILL	Kill; terminate process
10	SIGBUS	Bus error
11	SIGSEGV	Segmentation violation; process attempts to access location outside its virtual address space
12	SIGSYS	Bad argument to system call
13	SIGPIPE	Write on a pipe that has no readers attached to it
14	SIGALRM	Alarm clock; issued when a process wishes to receive a signal after a period of time
15	SIGTERM	Software termination
16	SIGUSR1	User-defined signal 1
17	SIGUSR2	User-defined signal 2
18	SIGCHLD	Death of a child
19	SIGPWR	Power failure

Table 6.3 Linux Atomic Operations

Atomic Integer Operations	
`ATOMIC_INIT (int i)`	At declaration: initialize an atomic_t to i
`int atomic_read(atomic_t *v)`	Read integer value of v
`void atomic_set(atomic_t *v, int i)`	Set the value of v to integer i
`void atomic_add(int i, atomic_t *v)`	Add i to v
`void atomic_sub(int i, atomic_t *v)`	Subtract i from v
`void atomic_inc(atomic_t *v)`	Add 1 to v
`void atomic_dec(atomic_t *v)`	Subtract 1 from v
`int atomic_sub_and_test(int i, atomic_t *v)`	Subtract i from v; return 1 if the result is zero; return 0 otherwise
`int atomic_add_negative(int i, atomic_t *v)`	Add i to v; return 1 if the result is negative; return 0 otherwise (used for implementing semaphores)
`int atomic_dec_and_test(atomic_t *v)`	Subtract 1 from v; return 1 if the result is zero; return 0 otherwise
`int atomic_inc_and_test(atomic_t *v)`	Add 1 to v; return 1 if the result is zero; return 0 otherwise
Atomic Bitmap Operations	
`void set_bit(int nr, void *addr)`	Set bit nr in the bitmap pointed to by addr
`void clear_bit(int nr, void *addr)`	Clear bit nr in the bitmap pointed to by addr
`void change_bit(int nr, void *addr)`	Invert bit nr in the bitmap pointed to by addr
`int test_and_set_bit(int nr, void *addr)`	Set bit nr in the bitmap pointed to by addr; return the old bit value
`int test_and_clear_bit(int nr, void *addr)`	Clear bit nr in the bitmap pointed to by addr; return the old bit value
`int test_and_change_bit(int nr, void *addr)`	Invert bit nr in the bitmap pointed to by addr; return the old bit value
`int test_bit(int nr, void *addr)`	Return the value of bit nr in the bitmap pointed to by addr

Table 6.6 Linux Memory Barrier Operations

`rmb()`	Prevents loads from being reordered across the barrier
`wmb()`	Prevents stores from being reordered across the barrier
`mb()`	Prevents loads and stores from being reordered across the barrier
`barrier()`	Prevents the compiler from reordering loads or stores across the barrier
`smp_rmb()`	On SMP, provides a `rmb()` and on UP provides a `barrier()`
`smp_wmb()`	On SMP, provides a `wmb()` and on UP provides a `barrier()`
`smp_mb()`	On SMP, provides a `mb()` and on UP provides a `barrier()`

SMP = symmetric multiprocessor
UP = uniprocessor