When a process is done with a shared resource that is controlled by a semaphore, the semaphore value is incremented by 1. If any other processes are asleep, waiting for the semaphore, they are awakened.

To implement semaphores correctly, the test of a semaphore's value and the decrementing of this value must be an atomic operation. For this reason, semaphores are normally implemented inside the kernel.

A common form of semaphore is called a *binary semaphore*. It controls a single resource and its value is initialized to 1. In general, however, a semaphore can be initialized to any positive value, with the value indicating how many of units of the shared resource are available for sharing.

System V semaphores are, unfortunately, more complicated than this. Three features contribute to this unnecessary complication.

1. A semaphore is not just a single nonnegative value. Instead we have to define a semaphore as a set of one or more semaphore values. When we create a semaphore we specify the number of values in the set.
2. The creation of a semaphore (`semget`) is independent of its initialization (`semctl`). This is a fatal flaw, since we cannot atomically create a new semaphore set and initialize all the values in the set.
3. Since all forms of System V IPC remain in existence even when no process is using them, we have to worry about a program that terminates without releasing the semaphores it has been allocated. The "undo" feature that we describe later is supposed to handle this.

The kernel maintains a `semid_ds` structure for each semaphore.

```
struct semid_ds {
  struct ipc_perm  sem_perm;  /* see Section 14.6.2 */
  struct sem  *sem_base;  /* ptr to first semaphore in set */
  ushort       sem_nsems; /* # of semaphores in set */
  time_t       sem_otime; /* last-semop() time */
  time_t       sem_ctime; /* last-change time */
};
```

The `sem_base` pointer is worthless to a user process, since it points to memory in the kernel. What it points to is an array of `sem` structures, containing `sem_nsems` elements, one element in the array for each semaphore value in the set.

```
struct sem {
  ushort  semval;   /* semaphore value, always >= 0 */
  pid_t   sempid;   /* pid for last operation */
  ushort  semncnt;  /* # processes awaiting semval > currval */
  ushort  semzcnt;  /* # processes awaiting semval = 0 */
};
```

Figure 14.18 lists the system limits (Section 14.6.3) that affect semaphore sets.