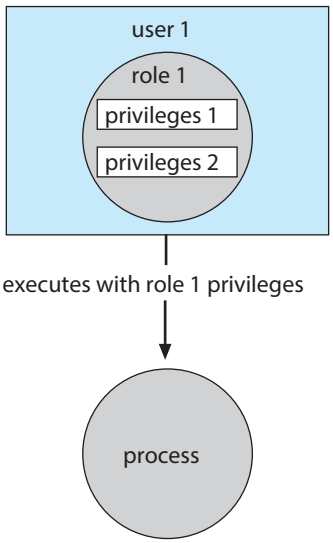

Figure 17.10 Role-based access control in Solaris 10.

role that enables a privilege, allowing the user to run a program to accomplish a specific task, as depicted in Figure 17.10. This implementation of privileges decreases the security risk associated with superusers and setuid programs.

Notice that this facility is similar to the access matrix described in Section 17.5. This relationship is further explored in the exercises at the end of the chapter.

17.9 Mandatory Access Control (MAC)

Operating systems have traditionally used **discretionary access control** (DAC) as a means of restricting access to files and other system objects. With DAC, access is controlled based on the identities of individual users or groups. In UNIX-based system, DAC takes the form of file permissions (settable by `chmod`, `chown`, and `chgrp`), whereas Windows (and some UNIX variants) allow finer granularity by means of access-control lists (ACLs).

DACs, however, have proved insufficient over the years. A key weakness lies in their discretionary nature, which allows the owner of a resource to set or modify its permissions. Another weakness is the unlimited access allowed for the administrator or root user. As we have seen, this design can leave the system vulnerable to both accidental and malicious attacks and provides no defense when hackers obtain root privileges.

The need arose, therefore, for a stronger form of protection, which was introduced in the form of **mandatory access control** (MAC). MAC is enforced as a system policy that even the root user cannot modify (unless the policy explicitly allows modifications or the system is rebooted, usually into an alternate configuration). The restrictions imposed by MAC policy rules are more powerful than the capabilities of the root user and can be used to make resources inaccessible to anyone but their intended owners.

Modern operating systems all provide MAC along with DAC, although implementations differ. Solaris was among the first to introduce MAC, which was part of Trusted Solaris (2.5). FreeBSD made DAC part of its TrustedBSD implementation (FreeBSD 5.0). The FreeBSD implementation was adopted by Apple in macOS 10.5 and has served as the substrate over which most of the security features of MAC and iOS are implemented. Linux's MAC implementation is part of the SELinux project, which was devised by the NSA, and has been integrated into most distributions. Microsoft Windows joined the trend with Windows Vista's Mandatory Integrity Control.

At the heart of MAC is the concept of **labels**. A label is an identifier (usually a string) assigned to an object (files, devices, and the like). Labels may also be applied to subjects (actors, such as processes). When a subject request to perform operations on the objects. When such requests are to be served by the operating system, it first performs checks defined in a policy, which dictates whether or not a given label holding subject is allowed to perform the operation on the labeled object.

As a brief example, consider a simple set of labels, ordered according to level of privilege: "unclassified," "secret," and "top secret." A user with "secret" clearance will be able to create similarly labeled processes, which will then have access to "unclassified" and "secret" files, but not to "top secret" files. Neither the user nor its processes would even be aware of the existence of "top secret" files, since the operating system would filter them out of all file operations (for example, they would not be displayed when listing directory contents). User processes would similarly be protected themselves in this way, so that an "unclassified" process would not be able to see or perform IPC requests to a "secret" (or "top secret") process. In this way, MAC labels are an implementation of the access matrix described earlier.

17.10 Capability-Based Systems

The concept of **capability-based protection** was introduced in the early 1970s. Two early research systems were Hydra and CAP. Neither system was widely used, but both provided interesting proving grounds for protection theories. For more details on these systems, see Section A.14.1 and Section A.14.2. Here, we consider two more contemporary approaches to capabilities.

17.10.1 Linux Capabilities

Linux uses capabilities to address the limitations of the UNIX model, which we described earlier. The POSIX standards group introduced capabilities in POSIX 1003.1e. Although POSIX.1e was eventually withdrawn, Linux was quick to adopt capabilities in Version 2.2 and has continued to add new developments.

In essence, Linux's capabilities "slice up" the powers of root into distinct areas, each represented by a bit in a bitmask, as shown in Figure 17.11. Fine-grained control over privileged operations can be achieved by toggling bits in the bitmask.

In practice, three bitmasks are used—denoting the capabilities ***permitted***, ***effective***, and ***inheritable***. Bitmasks can apply on a per-process or a per-thread basis. Furthermore, once revoked, capabilities cannot be reacquired. The usual