A common approach to managing system complexity is to identify a set of layers with well-defined

interfaces among them. The interfaces separate different levels of abstraction. Layering minimizes the

interactions among the subsystems and simplifies the description of the subsystems. Each subsystem

is abstracted through its interfaces with the other subsystems. Thus, we are able to design, implement,

and modify the individual subsystems independently.

The instruction set architecture (ISA) defines a processor’s set of instructions. For example, the

Intel architecture is represented by the x86-32 and x86-64 instruction sets for systems supporting 32-bit addressing and 64-bit addressing, respectively. The hardware supports two execution modes, a privileged,

or kernel,mode and a user mode. The instruction set consists of two sets of instructions, privileged

instructions that can only be executed in kernel mode and nonprivileged instructions that can be executed

in user mode. There are also sensitive instructions that can be executed in kernel and in user mode but

that behave differently (see Section 5.6).

Computer systems are fairly complex, and their operation is best understood when we consider a

model similar to the one in Figure 5.1, which shows the interfaces among the software components and

the hardware [325]. The hardware consists of one or more multicore processors, a system interconnect

(e.g., one or more buses), a memory translation unit, the main memory, and I/O devices, including one

or more networking interfaces. Applications written mostly in high-level languages (HLL) often call

library modules and are compiled into object code. Privileged operations, such as I/O requests, cannot

be executed in user mode; instead, application and library modules issue system calls and the operating

system determines whether the privileged operations required by the application do not violate system

security or integrity and, if they don’t, executes them on behalf of the user. The binaries resulting from

the translation of HLL programs are targeted to a specific hardware architecture.

The first interface we discuss is the instruction set architecture (ISA) at the boundary of the hardware

and the software. The next interface is the application binary interface (ABI),which allows the ensemble

consisting of the application and the library modules to access the hardware. The ABI does not include

privileged system instructions; instead it invokes system calls. Finally, the application program interface (API) defines the set of instructions the hardware was designed to execute and gives the application

access to the ISA. It includes HLL library calls, which often invoke system calls. A process is the

abstraction for the code of an application at execution time; a thread is a lightweight process. The ABI

is the projection of the computer system seen by the process, and the API is the projection of the system

from the perspective of the HLL program.

Clearly, the binaries created by a compiler for a specific ISA and a specific operating system are

not portable. Such code cannot run on a computer with a different ISA or on computers with the

same ISA but different operating systems. However, it is possible to compile an HLL program for a

VM environment, as shown in Figure 5.2, where portable code is produced and distributed and then

converted by binary translators to the ISA of the host system. A dynamic binary translation converts

blocks of guest instructions from the portable code to the host instruction and leads to a significant

performance improvement as such blocks are cached and reused.