

UNIT 7: INTRODUCTION TO OPERATING SYSTEMS

NOTES

Saltzer, J. & Schroeder, M. (1975) The Protection of Information in Computer Systems. Proceedings of the IEEE 63(9): 1278-1308. Available from:
<https://www.cs.virginia.edu/~evans/cs551/saltzer/> [Accessed 30 December 2021]

- controlled sharing of information among multiple users needed. Therefore, must ensure that the computer system helps implement the correct authority structure
- *The term "privacy" denotes a socially defined ability of an individual (or organization) to determine whether, when, and to whom personal (or organizational) information is to be released.*
- *The term "security" describes techniques that control who may use or modify the computer or the information contained in it.*
- a wide range of considerations are pertinent to the engineering of security of information. Historically, the literature of computer systems has more narrowly defined the term *protection* to be just those security techniques that control the access of executing programs to stored information. (e.g authentication)

Functional Levels of Information Protection

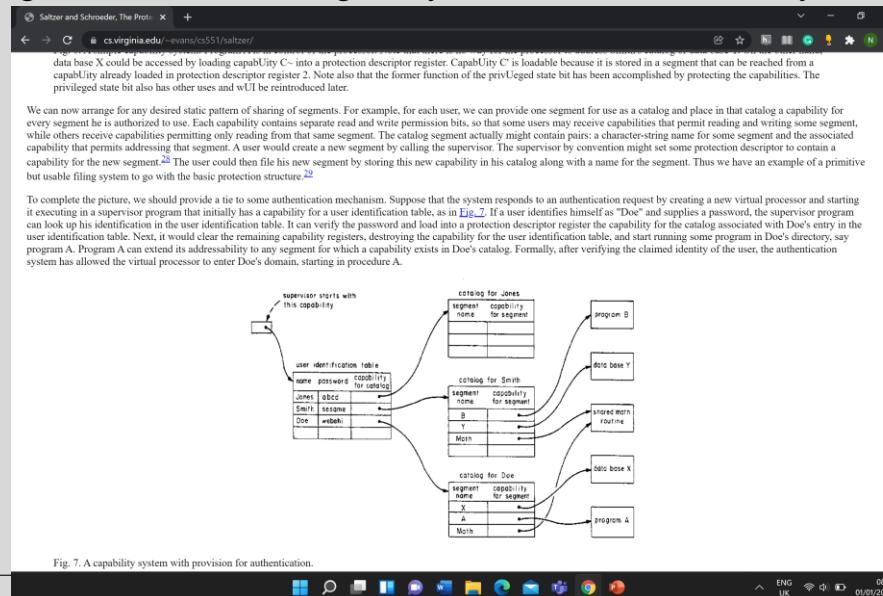
- Unprotected systems
- All-or-nothing systems: These are systems that provide isolation of users, sometimes moderated by total sharing of some pieces of information
- Controlled sharing: Significantly more complex machinery is required to control explicitly who may access each data item stored in the system.
- User-programmed sharing controls: A user may want to restrict access to a file in a way not provided in the standard facilities for controlling sharing.
- Putting strings on information: The printed labels on classified military information declaring a document to be "Top Secret" are another example of a constraint on information after its release to a person authorized to receive it.
- There is a consideration that cuts across all levels of functional capability: the *dynamics of use*. This term refers to how one establishes and changes the specification of who may access what. At any of the levels it is relatively easy to envision (and design) systems that statically express a particular protection intent. But the need to change access authorization dynamically and the need for such changes to be requested by executing programs introduces much complexity into protection systems.

Design Principles that apply particularly to protection mechanisms

- Economy of mechanism: Keep the design as simple and small as possible.
- Fail-safe defaults: Base access decisions on permission rather than exclusion. This principle, suggested by E. Glaser in 1965,⁸ means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. A conservative design must be based on arguments why objects should be accessible, rather than why they should not.
- Complete mediation: Every access to every object must be checked for authority. This principle, when systematically applied, is the primary underpinning of the protection system. It forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance. It implies that a foolproof method of identifying the source of every request must be devised.
- Open design: The design should not be secret [27]. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords.
- Separation of privilege: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key. The relevance of this observation to computer systems was pointed out by R. Needham in 1973. The reason is that, once the mechanism is locked, the two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them.
- Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
- Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users [28]. Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security.
- Psychological acceptability: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

Passwords and The use of descriptors for sharing information

- Passwords may have an expiration date, or usage count, to limit the length of usefulness of a compromised one.
- One solution to this problem is to provide a generator of "pronounceable" random passwords based on digraph or higher order frequency statistics [26] to make memorization easier.
- A more powerful authentication technique is sometimes used to protect against masquerading: *encryption* or *transformation* key.
- descriptors have been introduced here for the purpose of protecting information, although they are also used in some systems to organize addressing and storage allocation.
- To read on
 - **descriptors and segments.**
 - **Capability system or Tagged architecture**



Silberschatz, A., Galvin, P. & Gagne, G. (2018) *Operating System Concepts*. 10th ed. Hoboken, N.J: Wiley

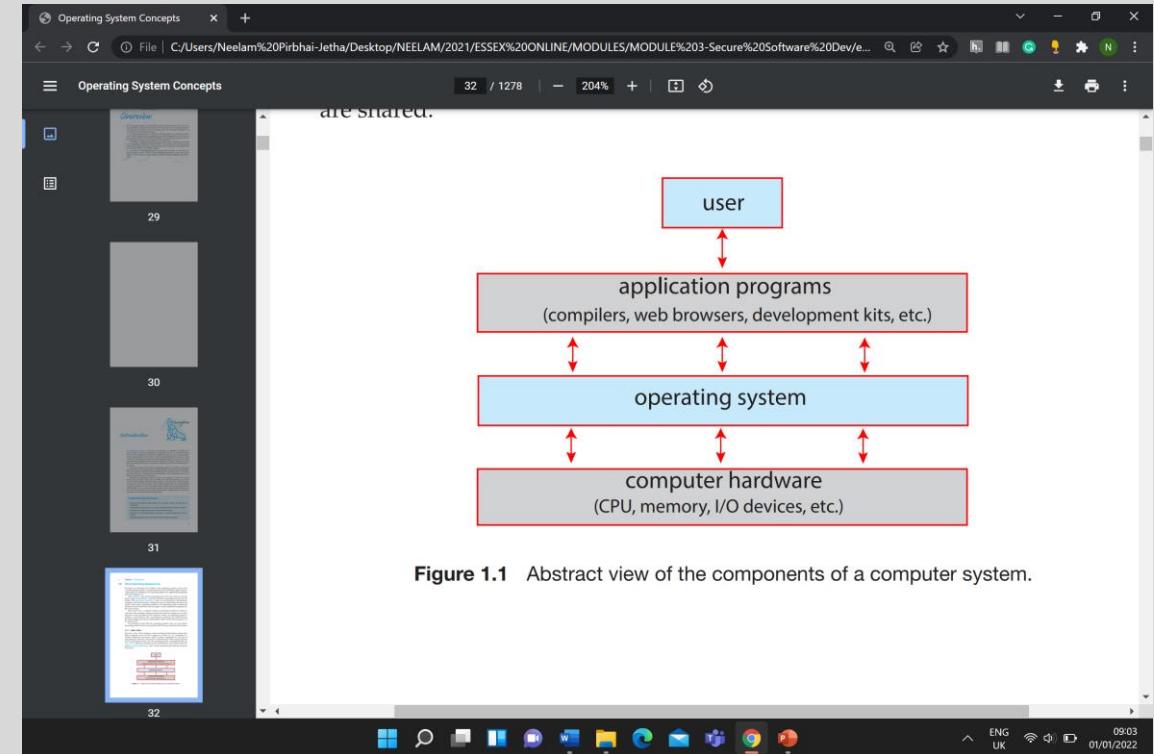
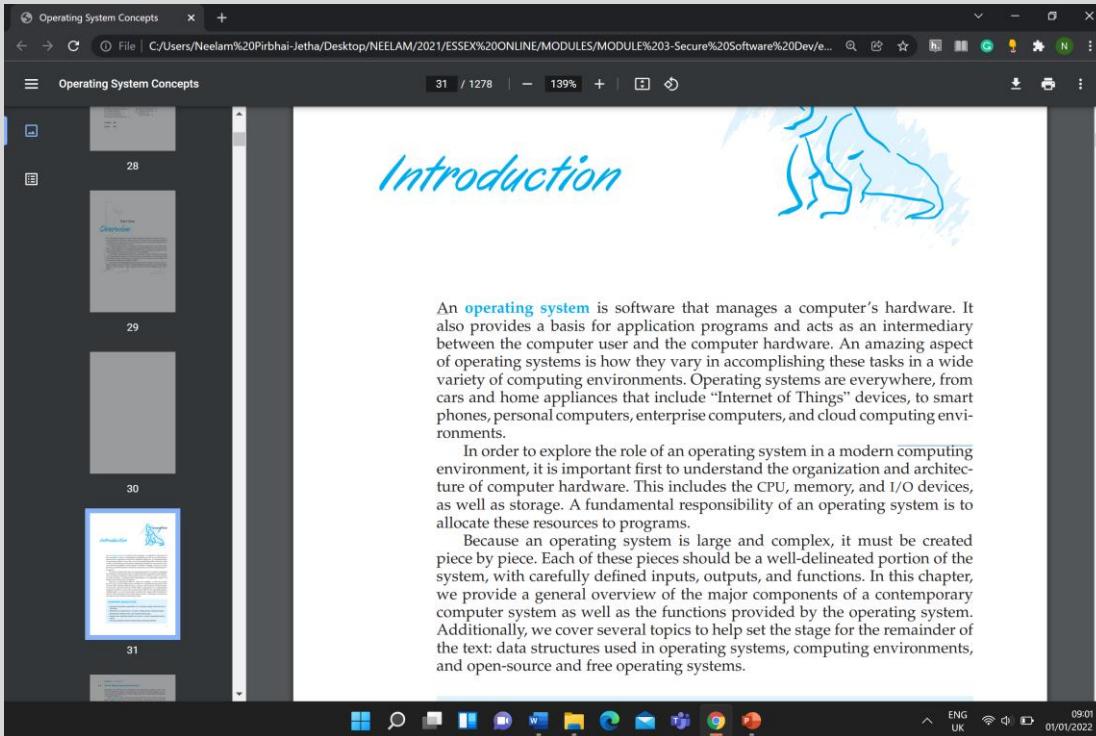


Figure 1.1 Abstract view of the components of a computer system.

Silberschatz, A., Galvin, P. & Gagne, G. (2018) *Operating System Concepts*. 10th ed. Hoboken, N.J: Wiley,

The screenshot shows a Windows desktop environment with two overlapping PDF documents from the book 'Operating System Concepts'. The left document is visible, showing page 62 of 1278, with the title '1.7 Virtualization'. The right document is partially visible, showing page 34 of 1278, with the title 'Entire mobile operating systems—Apple's iOS and Google's Android—features'. A blue callout box with a white border and a black arrow points from the bottom right towards the right-hand document. The callout box contains the text 'WHY STUDY OPERATING SYSTEMS?'. Below the callout box, there is a large amount of text about the importance of studying operating systems. The taskbar at the bottom shows various pinned icons and the date/time as 01/01/2022, 09:09. The system tray indicates the language is set to 'ENG UK'.

1.7 Virtualization

Virtualization is a technology that allows us to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate environment is running on its own private computer. These environments can be viewed as different individual operating systems (for example, Windows and UNIX) that may be running at the same time and may interact with each other. A user of a **virtual machine** can switch among the various operating systems in the same way a user can switch among the various processes running concurrently in a single operating system.

Virtualization allows operating systems to run as applications within other operating systems. At first blush, there seems to be little reason for such functionality. But the virtualization industry is vast and growing, which is a testament to its utility and importance.

Broadly speaking, virtualization software is one member of a class that also includes emulation. **Emulation**, which involves simulating computer hardware in software, is typically used when the source CPU type is different from the target CPU type. For example, when Apple switched from the IBM Power CPU to the Intel x86 CPU for its desktop and laptop computers, it included an emulation facility called "Rosetta," which allowed applications compiled for the IBM CPU to run on the Intel CPU. That same concept can be extended to allow an entire operating system written for one platform to run on another. Emulation comes at a heavy price, however. Every machine-level instruction that runs natively on the source system must be translated to the equivalent function on the target system, frequently resulting in several target instructions. If the source and target CPUs have similar performance levels, the emulated code may run much more slowly than the native code.

With virtualization, in contrast, an operating system that is natively compiled for a particular CPU architecture runs within another operating system also native to that CPU. Virtualization first came about on IBM mainframes as a method for multiple users to run tasks concurrently. Running multiple virtual machines allowed (and still allows) many users to run tasks on a system designed for a single user. Later, in response to problems with running multiple Microsoft Windows applications on the Intel x86 CPU, VMware created a new virtualization technology in the form of an application that ran on Windows. That application ran one or more **guest** copies of Windows or other native x86 operating systems, each running its own applications. (See Figure 1.16.)

Entire mobile operating systems—Apple's iOS and Google's Android—features

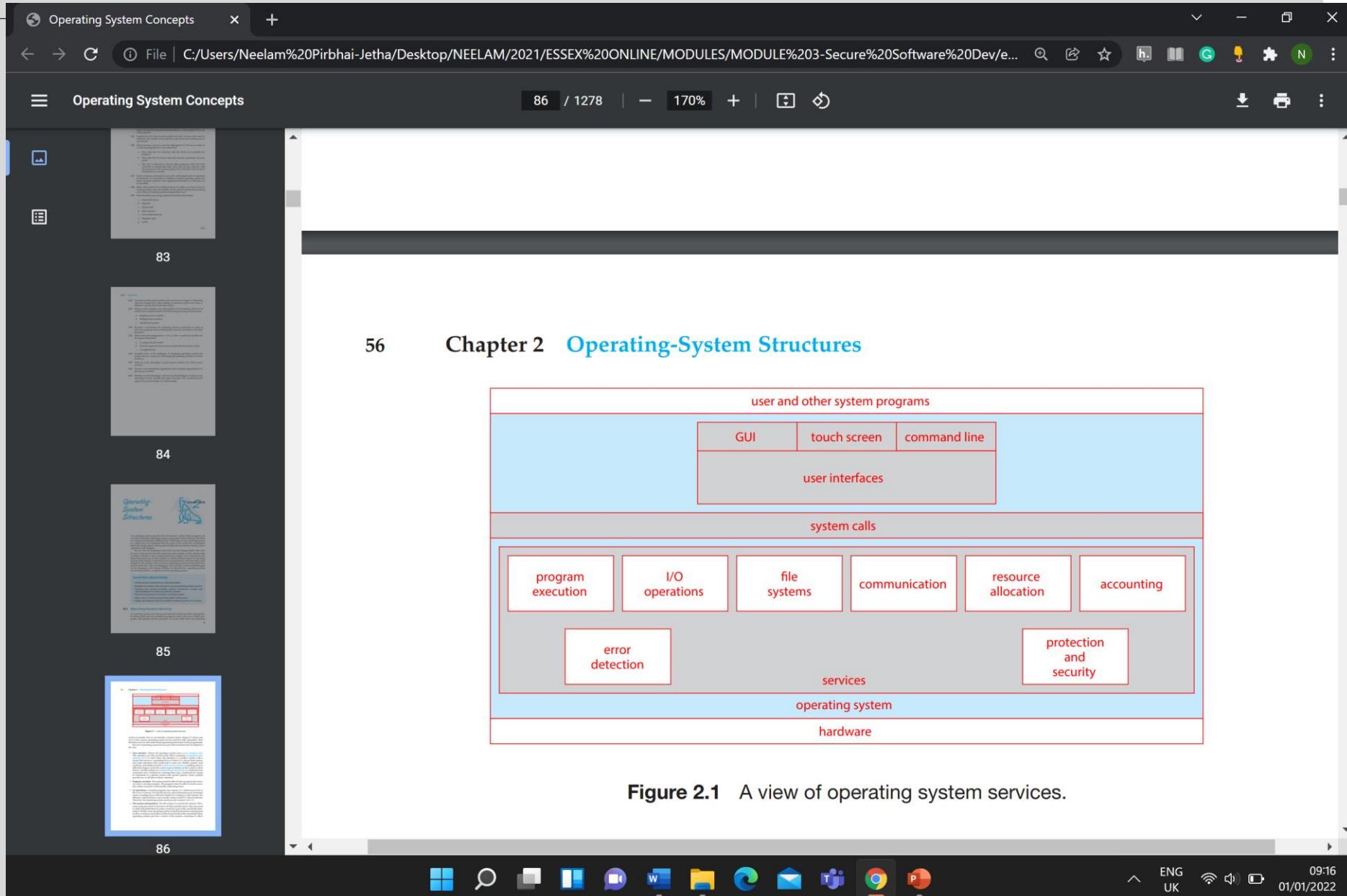
WHY STUDY OPERATING SYSTEMS?

Although there are many practitioners of computer science, only a small percentage of them will be involved in the creation or modification of an operating system. Why, then, study operating systems and how they work? Simply because, as almost all code runs on top of an operating system, knowledge of how operating systems work is crucial to proper, efficient, effective, and secure programming. Understanding the fundamentals of operating systems, how they drive computer hardware, and what they provide to applications is not only essential to those who program them but also highly useful to those who write programs on them and use them.

Summary

- An operating system is software that manages the computer hardware, as well as providing an environment for application programs to run.
 - Interrupts are a key way in which hardware interacts with the operating system. A hardware device triggers an interrupt by sending a signal to the CPU to alert the CPU that some event requires attention. The interrupt is managed by the interrupt handler.
 - For a computer to do its job of executing programs, the programs must be in main memory, which is the only large storage area that the processor can access directly.
 - The main memory is usually a volatile storage device that loses its contents when power is turned off or lost.
 - Nonvolatile storage is an extension of main memory and is capable of holding large quantities of data permanently.
 - The most common nonvolatile storage device is a hard disk, which can provide storage of both programs and data.
 - The wide variety of storage systems in a computer system can be organized in a hierarchy according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time generally increases.
 - Modern computer architectures are multiprocessor systems in which each CPU contains several computing cores.
 - To best utilize the CPU, modern operating systems employ multiprogramming, which allows several jobs to be in memory at the same time, thus ensuring that the CPU always has a job to execute.
 - Multitasking is an extension of multiprogramming wherein CPU scheduling algorithms rapidly switch between processes, providing users with a fast response time.
 - To prevent user programs from interfering with the proper operation of the system, the system hardware has two modes: user mode and kernel mode.
 - Various instructions are privileged and can be executed only in kernel mode. Examples include the instruction to switch to kernel mode, I/O control, timer management, and interrupt management.
 - A process is the fundamental unit of work in an operating system. Process management includes creating and deleting processes and providing mechanisms for processes to communicate and synchronize with each other.
 - An operating system manages memory by keeping track of what parts of memory are being used and by whom. It is also responsible for dynamically allocating and freeing memory space.
 - Storage space is managed by the operating system; this includes providing file systems for representing files and directories and managing space on mass-storage devices.
 - Operating systems provide mechanisms for protecting and securing the operating system and users. Protection measures control the access of processes or users to the resources made available by the computer system.
 - Virtualization involves abstracting a computer's hardware into several different execution environments.
 - Data structures that are used in an operating system include lists, stacks, queues, trees, and maps.
 - Computing takes place in a variety of environments, including traditional computing, mobile computing, client-server systems, peer-to-peer systems, cloud computing, and real-time embedded systems.
- Practice Exercises 53
- Free and open-source operating systems are available in source-code format. Free software is licensed to allow no-cost use, redistribution, and modification. GNU/Linux, FreeBSD, and Solaris are examples of popular open-source systems.

Chapter2



2.2 User and Operating-System Interface

We mentioned earlier that there are several ways for users to interface with the operating system. Here, we discuss three fundamental approaches. One provides a command-line interface, or **command interpreter**, that allows users to directly enter commands to be performed by the operating system. The other two allow users to interface with the operating system via a graphical user interface, or GUI.

2.2.1 Command Interpreters

Most operating systems, including Linux, UNIX, and Windows, treat the command interpreter as a special program that is running when a process is initiated or when a user first logs on (on interactive systems). On systems with multiple command interpreters to choose from, the interpreters are known as **shells**. For example, on UNIX and Linux systems, a user may choose among several different shells, including the *C shell*, *Bourne-Again shell*, *Korn shell*, and others. Third-party shells and free user-written shells are also available. Most shells provide similar functionality, and a user's choice of which shell to use is generally based on personal preference. Figure 2.2 shows the Bourne-Again (or bash) shell command interpreter being used on macOS.

The main function of the command interpreter is to get and execute the next user-specified command. Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute, and so on. The various shells available on UNIX systems operate in this way. These commands can be implemented in two general ways.

In one approach, the command interpreter itself contains the code to execute the command. For example, a command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call. In this case, the number of commands that can be given determines the size of the command interpreter, since each command requires its own implementing code.

An alternative approach—used by UNIX, among other operating systems—implements most commands through system programs. In this case, the command interpreter does not understand the command in any way; it merely uses the command to identify a file to be loaded into memory and executed. Thus, the UNIX command to delete a file

```
rm file.txt
```

would search for a file called `rm`, load the file into memory, and execute it with the parameter `file.txt`. The logic associated with the `rm` command would be

Chapter 2: User and Operating-System Interface

2.2.2 Graphical User Interface

A second strategy for interfacing with the operating system is through a user-friendly graphical user interface, or GUI. Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window-and-menu system characterized by a **desktop** metaphor. The user moves the mouse to position its pointer on images, or **icons**, on the screen (the desktop) that represent programs, files, directories, and system functions. Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory—known as a **folder**—or pull down a menu that contains commands.

Graphical user interfaces first appeared due in part to research taking place in the early 1970s at Xerox PARC research facility. The first GUI appeared on the Xerox Alto computer in 1973. However, graphical interfaces became more widespread with the advent of Apple Macintosh computers in the 1980s. The user interface for the Macintosh operating system has undergone various changes over the years, the most significant being the adoption of the *Aqua* interface that appeared with macOS. Microsoft's first version of Windows—Version 1.0—was based on the addition of a GUI interface to the MS-DOS operating system. Later versions of Windows have made significant changes in the appearance of the GUI along with several enhancements in its functionality.

Traditionally, UNIX systems have been dominated by command-line interfaces. Various GUI interfaces are available, however, with significant development in GUI designs from various open-source projects, such as *K Desktop Environment* (or *KDE*) and the *GNOME* desktop by the GNU project. Both the KDE and GNOME desktops run on Linux and various UNIX systems and are available under open-source licenses, which means their source code is readily available for reading and for modification under specific license terms.

2.2.3 Touch-Screen Interface

Because either a command-line interface or a mouse-and-keyboard system is impractical for most mobile systems, smartphones and handheld tablet computers typically use touch-screen interface. Here, users interact by making **gestures** on the touch screen—for example, pressing and swiping fingers across the screen. Although earlier smartphones included a physical keyboard, most smartphones and tablets now simulate a keyboard on the touch screen. Figure 2.3 illustrates the touch screen of the Apple iPhone. Both the iPad and the iPhone use the **Springboard** touch-screen interface.

2.3 System Calls

System calls provide an interface to the services made available by an operating system. These calls are generally available as functions written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly) may have to be written using assembly-language instructions.

2.3.1 Example

Before we discuss how an operating system makes system calls available, let's first use an example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file. The first input that the program will need is the names of the two files: the input file and the output file. These names can be specified in many ways, depending on the operating-system design. One approach is to pass the names of the two files as part of the command—for example, the UNIX cp command:

```
cp in.txt out.txt
```

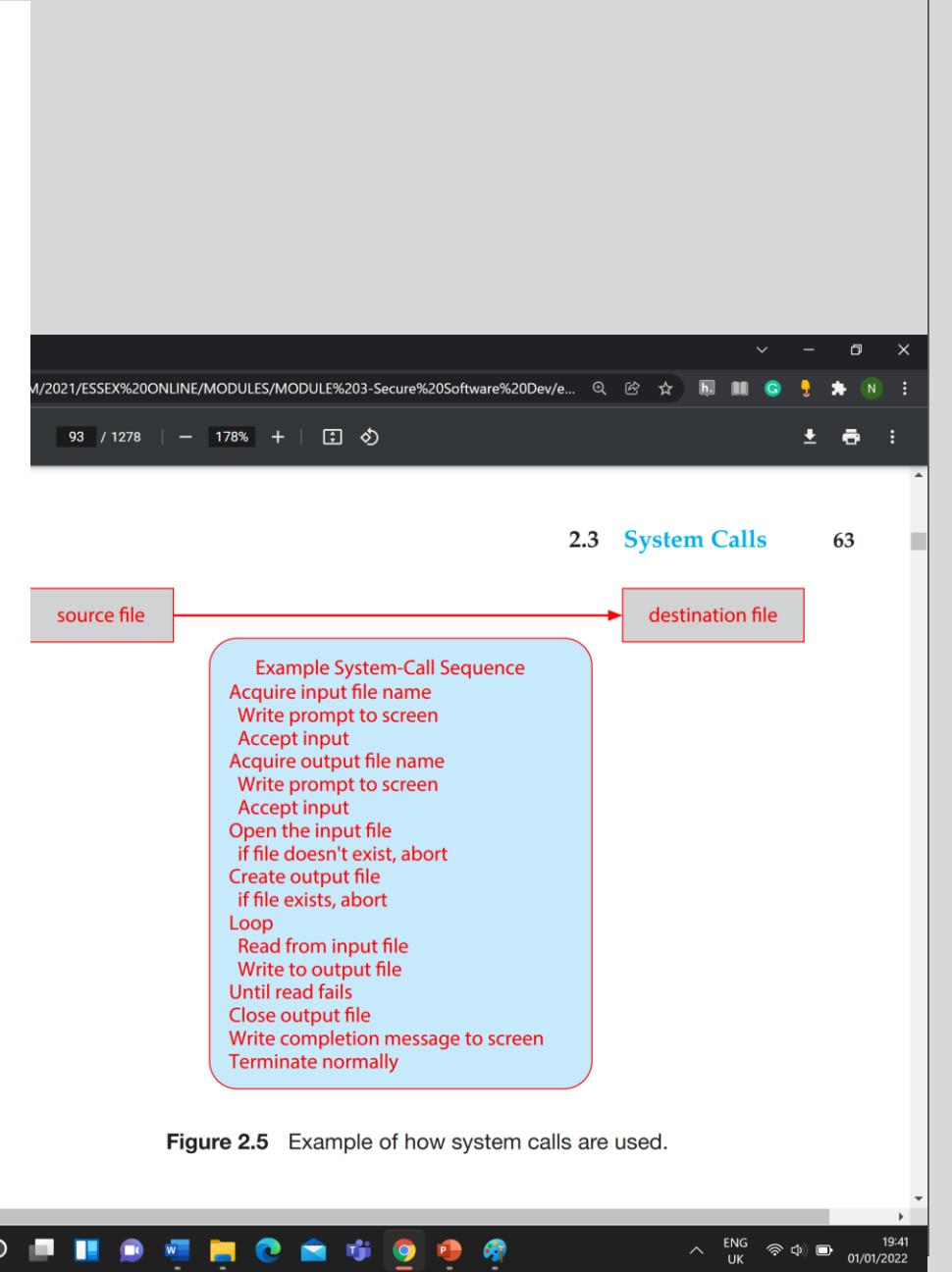
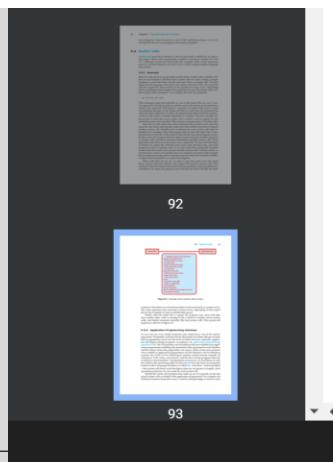
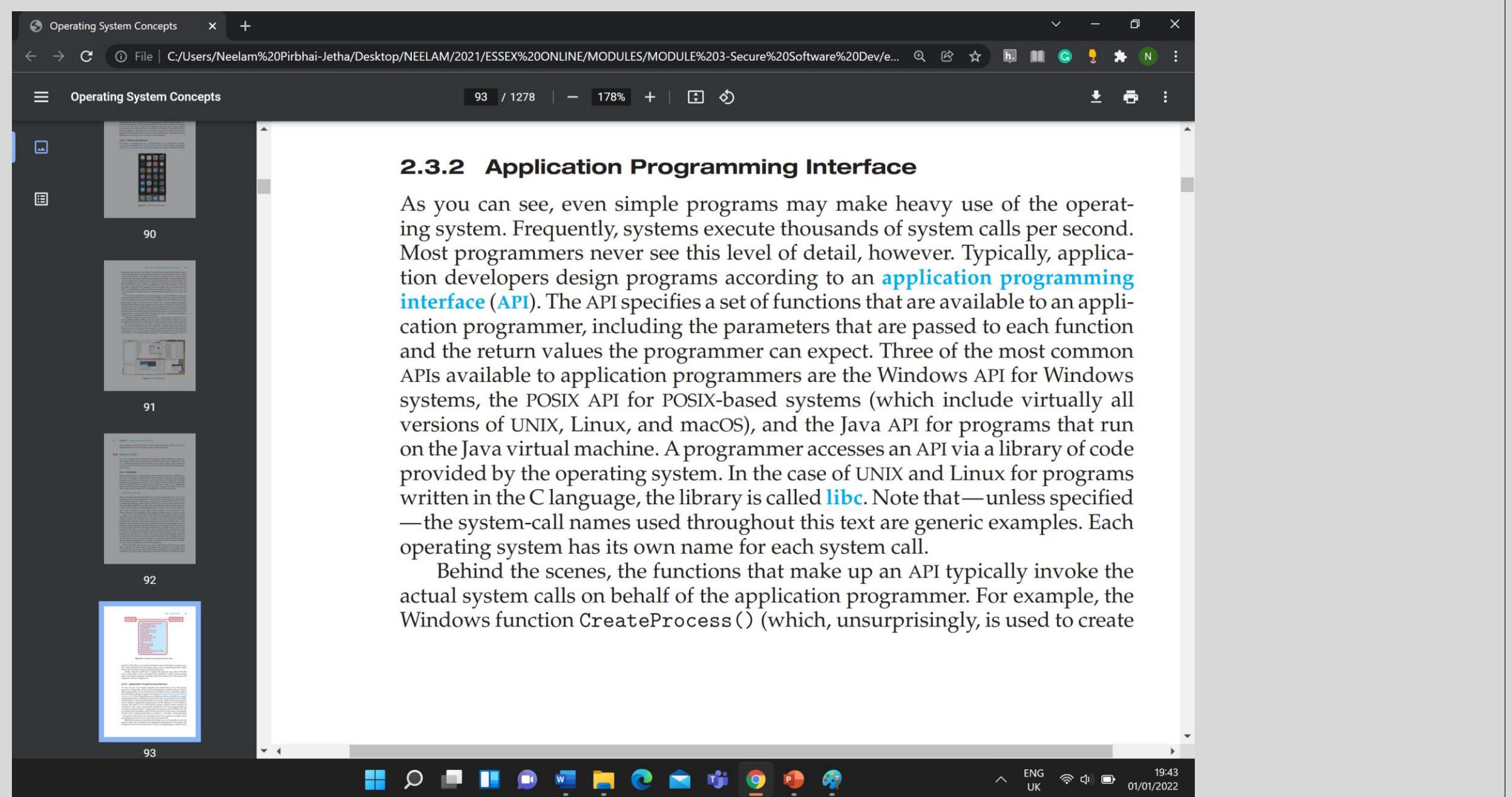


Figure 2.5 Example of how system calls are used.



2.3.3.6 Protection

Protection provides a mechanism for controlling access to the resources provided by a computer system. Historically, protection was a concern only on multiprogrammed computer systems with several users. However, with the advent of networking and the Internet, all computer systems, from servers to mobile handheld devices, must be concerned with protection.

Typically, system calls providing protection include `set_permission()` and `get_permission()`, which manipulate the permission settings of

resources such as files and disks. The `allow_user()` and `deny_user()` system calls specify whether particular users can—or cannot—be allowed access to certain resources. We cover protection in Chapter 17 and the much larger issue of security—which involves using protection against external threats—in Chapter 16.

2.11 Summary

- An operating system provides an environment for the execution of programs by providing services to users and programs.
- The three primary approaches for interacting with an operating system are (1) command interpreters, (2) graphical user interfaces, and (3) touch-screen interfaces.
- System calls provide an interface to the services made available by an operating system. Programmers use a system call's application programming interface (API) for accessing system-call services.
- System calls can be divided into six major categories: (1) process control, (2) file management, (3) device management, (4) information maintenance, (5) communications, and (6) protection.
- The standard C library provides the system-call interface for UNIX and Linux systems.
- Operating systems also include a collection of system programs that provide utilities to users.
- A linker combines several relocatable object modules into a single binary executable file. A loader loads the executable file into memory, where it becomes eligible to run on an available CPU.
- There are several reasons why applications are operating-system specific. These include different binary formats for program executables, different instruction sets for different CPUs, and system calls that vary from one operating system to another.
- An operating system is designed with specific goals in mind. These goals ultimately determine the operating system's policies. An operating system implements these policies through specific mechanisms.
- A monolithic operating system has no structure; all functionality is provided in a single, static binary file that runs in a single address space. Although such systems are difficult to modify, their primary benefit is efficiency.
- A layered operating system is divided into a number of discrete layers, where the bottom layer is the hardware interface and the highest layer is the user interface. Although layered software systems have had some suc-

cess, this approach is generally not ideal for designing operating systems due to performance problems.

- The microkernel approach for designing operating systems uses a minimal kernel; most services run as user-level applications. Communication takes place via message passing.
- A modular approach for designing operating systems provides operating-system services through modules that can be loaded and removed during run time. Many contemporary operating systems are constructed as hybrid systems using a combination of a monolithic kernel and modules.
- A boot loader loads an operating system into memory, performs initialization, and begins system execution.
- The performance of an operating system can be monitored using either counters or tracing. Counters are a collection of system-wide or per-process statistics, while tracing follows the execution of a program through the operating system.

3.9 Summary

- A process is a program in execution, and the status of the current activity of a process is represented by the program counter, as well as other registers.
- The layout of a process in memory is represented by four different sections: (1) text, (2) data, (3) heap, and (4) stack.
- As a process executes, it changes state. There are four general states of a process: (1) ready, (2) running, (3) waiting, and (4) terminated.
- A process control block (PCB) is the kernel data structure that represents a process in an operating system.
- The role of the process scheduler is to select an available process to run on a CPU.
- An operating system performs a context switch when it switches from running one process to running another.

Chapter 3

154 Chapter 3 Processes

- The `fork()` and `CreateProcess()` system calls are used to create processes on UNIX and Windows systems, respectively.
- When shared memory is used for communication between processes, two (or more) processes share the same region of memory. POSIX provides an API for shared memory.
- Two processes may communicate by exchanging messages with one another using message passing. The Mach operating system uses message passing as its primary form of interprocess communication. Windows provides a form of message passing as well.
- A pipe provides a conduit for two processes to communicate. There are two forms of pipes, ordinary and named. Ordinary pipes are designed for communication between processes that have a parent-child relationship. Named pipes are more general and allow several processes to communicate.
- UNIX systems provide ordinary pipes through the `pipe()` system call. Ordinary pipes have a read end and a write end. A parent process can, for example, send data to the pipe using its write end, and the child process can read it from its read end. Named pipes in UNIX are termed FIFOs.
- Windows systems also provide two forms of pipes—anonymous and named pipes. Anonymous pipes are similar to UNIX ordinary pipes. They are unidirectional and employ parent-child relationships between the communicating processes. Named pipes offer a richer form of interprocess communication than the UNIX counterpart, FIFOs.
- Two common forms of client-server communication are sockets and remote procedure calls (RPCs). Sockets allow two processes on different machines to communicate over a network. RPCs abstract the concept of function (procedure) calls in such a way that a function can be invoked on another process that may reside on a separate computer.
- The Android operating system uses RPCs as a form of interprocess communication using its binder framework.

4.8 Summary

- A thread represents a basic unit of CPU utilization, and threads belonging to the same process share many of the process resources, including code and data.
- There are four primary benefits to multithreaded applications: (1) responsiveness, (2) resource sharing, (3) economy, and (4) scalability.
- Concurrency exists when multiple threads are making progress, whereas parallelism exists when multiple threads are making progress simultaneously.

Chapter 4

Practice Exercises 197

neously. On a system with a single CPU, only concurrency is possible; parallelism requires a multicore system that provides multiple CPUs.

- There are several challenges in designing multithreaded applications. They include dividing and balancing the work, dividing the data between the different threads, and identifying any data dependencies. Finally, multithreaded programs are especially challenging to test and debug.
- Data parallelism distributes subsets of the same data across different computing cores and performs the same operation on each core. Task parallelism distributes not data but tasks across multiple cores. Each task is running a unique operation.
- User applications create user-level threads, which must ultimately be mapped to kernel threads to execute on a CPU. The many-to-one model maps many user-level threads to one kernel thread. Other approaches include the one-to-one and many-to-many models.
- A thread library provides an API for creating and managing threads. Three common thread libraries include Windows, Pthreads, and Java threading. Windows is for the Windows system only, while Pthreads is available for POSIX-compatible systems such as UNIX, Linux, and macOS. Java threads will run on any system that supports a Java virtual machine.
- Implicit threading involves identifying tasks—not threads—and allowing languages or API frameworks to create and manage threads. There are several approaches to implicit threading, including thread pools, fork-join frameworks, and Grand Central Dispatch. Implicit threading is becoming an increasingly common technique for programmers to use in developing concurrent and parallel applications.
- Threads may be terminated using either asynchronous or deferred cancellation. Asynchronous cancellation stops a thread immediately, even if it is in the middle of performing an update. Deferred cancellation informs a thread that it should terminate but allows the thread to terminate in an orderly fashion. In most circumstances, deferred cancellation is preferred to asynchronous termination.
- Unlike many other operating systems, Linux does not distinguish between processes and threads; instead, it refers to each as a task. The Linux `clone()` system call can be used to create tasks that behave either more like processes or more like threads.

5.9 Summary

- CPU scheduling is the task of selecting a waiting process from the ready queue and allocating the CPU to it. The CPU is allocated to the selected process by the dispatcher.
- Scheduling algorithms may be either preemptive (where the CPU can be taken away from a process) or nonpreemptive (where a process must voluntarily relinquish control of the CPU). Almost all modern operating systems are preemptive.
- Scheduling algorithms can be evaluated according to the following five criteria: (1) CPU utilization, (2) throughput, (3) turnaround time, (4) waiting time, and (5) response time.
- First-come, first-served (FCFS) scheduling is the simplest scheduling algorithm, but it can cause short processes to wait for very long processes.
- Shortest-job-first (SJF) scheduling is provably optimal, providing the shortest average waiting time. Implementing SJF scheduling is difficult, however, because predicting the length of the next CPU burst is difficult.
- Round-robin (RR) scheduling allocates the CPU to each process for a time quantum. If the process does not relinquish the CPU before its time quantum expires, the process is preempted, and another process is scheduled to run for a time quantum.
- Priority scheduling assigns each process a priority, and the CPU is allocated to the process with the highest priority. Processes with the same priority can be scheduled in FCFS order or using RR scheduling.
- Multilevel queue scheduling partitions processes into several separate queues arranged by priority, and the scheduler executes the processes in the highest-priority queue. Different scheduling algorithms may be used in each queue.

- Multilevel feedback queues are similar to multilevel queues, except that a process may migrate between different queues.
- Multicore processors place one or more CPUs on the same physical chip, and each CPU may have more than one hardware thread. From the perspective of the operating system, each hardware thread appears to be a logical CPU.
- Load balancing on multicore systems equalizes loads between CPU cores, although migrating threads between cores to balance loads may invalidate cache contents and therefore may increase memory access times.
- Soft real-time scheduling gives priority to real-time tasks over non-real-time tasks. Hard real-time scheduling provides timing guarantees for real-time tasks,
- Rate-monotonic real-time scheduling schedules periodic tasks using a static priority policy with preemption.

Practice Exercises 251

- Earliest-deadline-first (EDF) scheduling assigns priorities according to deadline. The earlier the deadline, the higher the priority; the later the deadline, the lower the priority.
- Proportional share scheduling allocates T shares among all applications. If an application is allocated N shares of time, it is ensured of having N/T of the total processor time.
- Linux uses the completely fair scheduler (CFS), which assigns a proportion of CPU processing time to each task. The proportion is based on the virtual runtime (`vruntime`) value associated with each task.
- Windows scheduling uses a preemptive, 32-level priority scheme to determine the order of thread scheduling.
- Solaris identifies six unique scheduling classes that are mapped to a global priority. CPU-intensive threads are generally assigned lower priorities (and longer time quantum), and I/O-bound threads are usually assigned higher priorities (with shorter time quantum).
- Modeling and simulations can be used to evaluate a CPU scheduling algorithm.

6.10 Summary

- A race condition occurs when processes have concurrent access to shared data and the final result depends on the particular order in which concurrent accesses occur. Race conditions can result in corrupted values of shared data.
- A critical section is a section of code where shared data may be manipulated and a possible race condition may occur. The critical-section problem

Practice Exercises 287

is to design a protocol whereby processes can synchronize their activity to cooperatively share data.

- A solution to the critical-section problem must satisfy the following three requirements: (1) mutual exclusion, (2) progress, and (3) bounded waiting. Mutual exclusion ensures that only one process at a time is active in its critical section. Progress ensures that programs will cooperatively determine what process will next enter its critical section. Bounded waiting limits how much time a program will wait before it can enter its critical section.
- Software solutions to the critical-section problem, such as Peterson's solution, do not work well on modern computer architectures.
- Hardware support for the critical-section problem includes memory barriers; hardware instructions, such as the compare-and-swap instruction; and atomic variables.

- A mutex lock provides mutual exclusion by requiring that a process acquire a lock before entering a critical section and release the lock on exiting the critical section.
- Semaphores, like mutex locks, can be used to provide mutual exclusion. However, whereas a mutex lock has a binary value that indicates if the lock is available or not, a semaphore has an integer value and can therefore be used to solve a variety of synchronization problems.
- A monitor is an abstract data type that provides a high-level form of process synchronization. A monitor uses condition variables that allow processes to wait for certain conditions to become true and to signal one another when conditions have been set to true.
- Solutions to the critical-section problem may suffer from liveness problems, including deadlock.
- The various tools that can be used to solve the critical-section problem as well as to synchronize the activity of processes can be evaluated under varying levels of contention. Some tools work better under certain contention loads than others.

7.6 Summary

- Classic problems of process synchronization include the bounded-buffer, readers-writers, and dining-philosophers problems. Solutions to these problems can be developed using the tools presented in Chapter 6, including mutex locks, semaphores, monitors, and condition variables.
- Windows uses dispatcher objects as well as events to implement process synchronization tools.
- Linux uses a variety of approaches to protect against race conditions, including atomic variables, spinlocks, and mutex locks.
- The POSIX API provides mutex locks, semaphores, and condition variables. POSIX provides two forms of semaphores: named and unnamed. Several unrelated processes can easily access the same named semaphore by simply referring to its name. Unnamed semaphores cannot be shared as easily, and require placing the semaphore in a region of shared memory.
- Java has a rich library and API for synchronization. Available tools include monitors (which are provided at the language level) as well as reentrant locks, semaphores, and condition variables (which are supported by the API).
- Alternative approaches to solving the critical-section problem include transactional memory, OpenMP, and functional languages. Functional languages are particularly intriguing, as they offer a different programming paradigm from procedural languages. Unlike procedural languages, functional languages do not maintain state and therefore are generally immune from race conditions and critical sections.

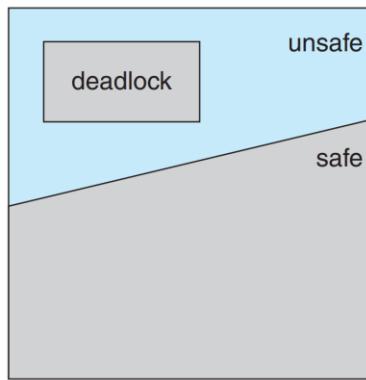


Figure 8.8 Safe, unsafe, and deadlocked state spaces.

A screenshot of a Microsoft Word document window. The title bar shows the file path: C:/Users/Neelam%20Pirbhai-Jetha/Desktop/NEELAM/2021/ESSEX%20ONLINE/MODULES/MODULE%203-Secure%20Software%20Dev/ebo... The main content area contains the following text:

8.8 Recovery from Deadlock 341

MANAGING DEADLOCK IN DATABASES

Database systems provide a useful illustration of how both open-source and commercial software manage deadlock. Updates to a database may be performed as *transactions*, and to ensure data integrity, locks are typically used. A transaction may involve several locks, so it comes as no surprise that deadlocks are possible in a database with multiple concurrent transactions. To manage deadlock, most transactional database systems include a deadlock detection and recovery mechanism. The database server will periodically search for cycles in the wait-for graph to detect deadlock among a set of transactions. When deadlock is detected, a victim is selected and the transaction is aborted and rolled back, releasing the locks held by the victim transaction and freeing the remaining transactions from deadlock. Once the remaining transactions have resumed, the aborted transaction is reissued. Choice of a victim transaction depends on the database system; for instance, MySQL attempts to select transactions that minimize the number of rows being inserted, updated, or deleted.

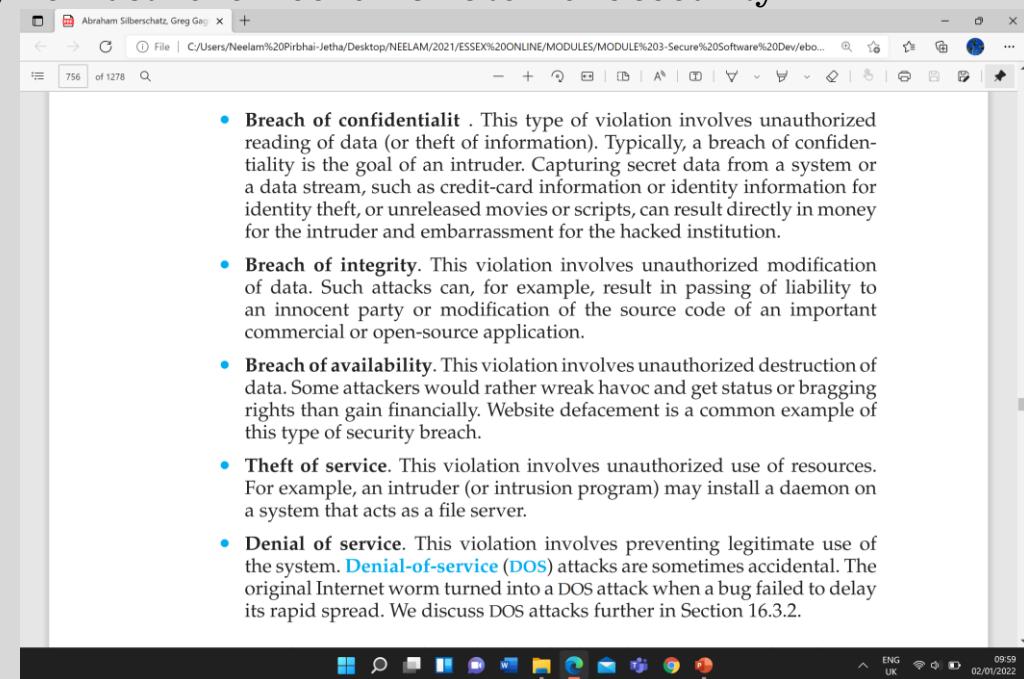
The Word ribbon tabs are visible at the top, and the Windows taskbar is visible at the bottom, showing various application icons and the date/time (09:30, 02/01/2022).

8.9 Summary

- Deadlock occurs in a set of processes when every process in the set is waiting for an event that can only be caused by another process in the set.
- There are four necessary conditions for deadlock: (1) mutual exclusion, (2) hold and wait, (3) no preemption, and (4) circular wait. Deadlock is only possible when all four conditions are present.
- Deadlocks can be modeled with resource-allocation graphs, where a cycle indicates deadlock.
- Deadlocks can be prevented by ensuring that one of the four necessary conditions for deadlock cannot occur. Of the four necessary conditions, eliminating the circular wait is the only practical approach.
- Deadlock can be avoided by using the banker's algorithm, which does not grant resources if doing so would lead the system into an unsafe state where deadlock would be possible.
- A deadlock-detection algorithm can evaluate processes and resources on a running system to determine if a set of processes is in a deadlocked state.
- If deadlock does occur, a system can attempt to recover from the deadlock by either aborting one of the processes in the circular wait or preempting resources that have been assigned to a deadlocked process.

Chapter 16

- Both protection and security are vital to computer systems. We distinguish between these two concepts in the following way: Security is a measure of confidence that the integrity of a system and its data will be preserved. Protection is the set of mechanisms that control the access of processes and users to the resources defined by a computer system.
- We say that a system is secure if its resources are used and accessed as intended under all circumstances. Unfortunately, total security cannot be achieved. Nonetheless, we must have mechanisms to make security breaches a rare occurrence, rather than the norm.



To protect a system, we must take security measures at four levels:

1. **Physical.** The site or sites containing the computer systems must be physically secured against entry by intruders. Both the machine rooms and the terminals or computers that have access to the target machines must be secured, for example by limiting access to the building they reside in, or locking them to the desk on which they sit.
2. **Network.** Most contemporary computer systems—from servers to mobile devices to Internet of Things (IoT) devices—are networked. Networking provides a means for the system to access external resources but also provides a potential vector for unauthorized access to the system itself.

Further, computer data in modern systems frequently travel over private leased lines, shared lines like the Internet, wireless connections, and dial-up lines. Intercepting these data can be just as harmful as breaking into a computer, and interruption of communications can constitute a remote denial-of-service attack, diminishing users' use of and trust in the system.

3. **Operating system.** The operating system and its built-in set of applications and services comprise a huge code base that may harbor many vulnerabilities. Insecure default settings, misconfigurations, and security

bugs are only a few potential problems. Operating systems must thus be kept up to date (via continuous patching) and “hardened”—configured and modified to decrease the attack surface and avoid penetration. The **attack surface** is the set of points at which an attacker can try to break into the system.

4. **Application.** Third-party applications may also pose risks, especially if they possess significant privileges. Some applications are inherently malicious, but even benign applications may contain security bugs. Due to the vast number of third-party applications and their disparate code bases, it is virtually impossible to ensure that all such applications are secure.

THE PRINCIPLE OF LEAST PRIVILEGE

“The principle of least privilege. Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job. The purpose of this principle is to reduce the number of potential interactions among privileged programs to the minimum necessary to operate correctly, so that one may develop confidence that unintentional, unwanted, or improper uses of privilege do not occur.”—Jerome H. Saltzer, describing a design principle of the Multics operating system in 1974: <https://pdfs.semanticscholar.org/1c8d/06510ad449ad24fbdd164f8008cc730cab47.pdf>.

STRONG AND EASY TO REMEMBER PASSWORDS

It is extremely important to use strong (hard to guess and hard to shoulder surf) passwords on critical systems like bank accounts. It is also important to not use the same password on lots of systems, as one less important, easily hacked system could reveal the password you use on more important systems. A good technique is to generate your password by using the first letter of each word of an easily remembered phrase using both upper and lower characters with a number or punctuation mark thrown in for good measure. For example, the phrase “My girlfriend’s name is Katherine” might yield the password “Mgn.isK!”. The password is hard to crack but easy for the user to remember. A more secure system would allow more characters in its passwords. Indeed, a system might also allow passwords to include the space character, so that a user could create a **passphrase** which is easy to remember but difficult to break.

16.6.8 Security Defenses Summarized

By applying appropriate layers of defense, we can keep systems safe from all but the most persistent attackers. In summary, these layers may include the following:

- Educate users about safe computing—don’t attach devices of unknown origin to the computer, don’t share passwords, use strong passwords, avoid falling for social engineering appeals, realize that an e-mail is not necessarily a private communication, and so on
- Educate users about how to prevent phishing attacks—don’t click on e-mail attachments or links from unknown (or even known) senders; authenticate (for example, via a phone call) that a request is legitimate.
- Use secure communication when possible.
- Physically protect computer hardware.
- Configure the operating system to minimize the attack surface; disable all unused services.
- Configure system daemons, privileges applications, and services to be as secure as possible.
- Use modern hardware and software, as they are likely to have up-to-date security features.
- Keep systems and applications up to date and patched.
- Only run applications from trusted sources (such as those that are code signed).

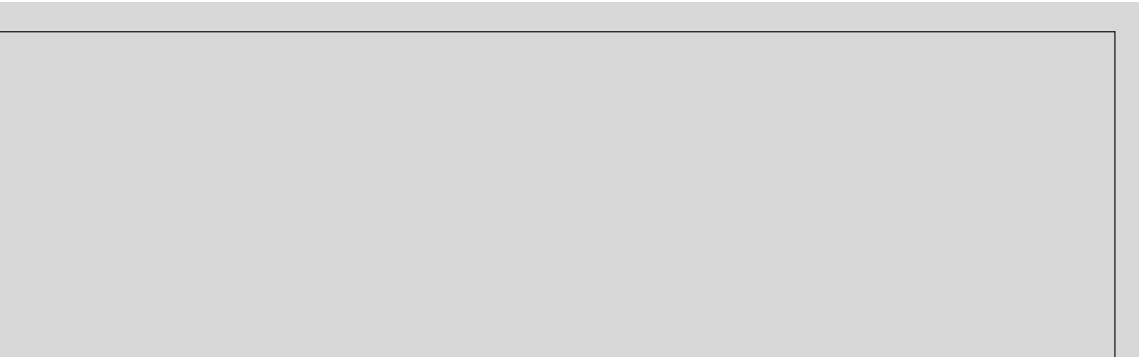
662

Chapter 16 Security

- Enable logging and auditing; review the logs periodically, or automate alerts.
- Install and use antivirus software on systems susceptible to viruses, and keep the software up to date.
- Use strong passwords and passphrases, and don’t record them where they could be found.
- Use intrusion detection, firewalling, and other network-based protection systems as appropriate.
- For important facilities, use periodic vulnerability assessments and other testing methods to test security and response to incidents.
- Encrypt mass-storage devices, and consider encrypting important individual files as well.
- Have a security policy for important systems and facilities, and keep it up to date

16.8 Summary

- Protection is an internal problem. Security, in contrast, must consider both the computer system and the environment—people, buildings, businesses, valuable objects, and threats—with which the system is used.
- The data stored in the computer system must be protected from unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency. It is easier to protect against accidental loss of data consistency than to protect against malicious access to the data. Absolute protection of the information stored in a computer system from malicious abuse is not possible; but the cost to the perpetrator can be made sufficiently high to deter most, if not all, attempts to access that information without proper authority.
- Several types of attacks can be launched against programs and against individual computers or the masses. Stack- and buffer-overflow tech-



Further Reading

665

niques allow successful attackers to change their level of system access. Viruses and malware require human interaction, while worms are self-perpetuating, sometimes infecting thousands of computers. Denial-of-service attacks prevent legitimate use of target systems.

- Encryption limits the domain of receivers of data, while authentication limits the domain of senders. Encryption is used to provide confidentiality of data being stored or transferred. Symmetric encryption requires a shared key, while asymmetric encryption provides a public key and a private key. Authentication, when combined with hashing, can prove that data have not been changed.
- User authentication methods are used to identify legitimate users of a system. In addition to standard user-name and password protection, several authentication methods are used. One-time passwords, for example, change from session to session to avoid replay attacks. Two-factor authentication requires two forms of authentication, such as a hardware calculator with an activation PIN, or one that presents a different response based on the time. Multifactor authentication uses three or more forms. These methods greatly decrease the chance of authentication forgery.
- Methods of preventing or detecting security incidents include an up-to-date security policy, intrusion-detection systems, antivirus software, auditing and logging of system events, system-call monitoring, code signing, sandboxing, and firewalls.

17.13 Summary

- System protection features are guided by the principle of need-to-know and implement mechanisms to enforce the principle of least privilege.
- Computer systems contain objects that must be protected from misuse. Objects may be hardware (such as memory, CPU time, and I/O devices) or software (such as files, programs, and semaphores).
- An access right is permission to perform an operation on an object. A domain is a set of access rights. Processes execute in domains and may use any of the access rights in the domain to access and manipulate objects. During its lifetime, a process may be either bound to a protection domain or allowed to switch from one domain to another.
- A common method of securing objects is to provide a series of protection rings, each with more privileges than the last. ARM, for example, provides four protection levels. The most privileged, TrustZone, is callable only from kernel mode.
- The access matrix is a general model of protection that provides a mechanism for protection without imposing a particular protection policy on the system or its users. The separation of policy and mechanism is an important design property.

Chapter 17: Protection

- The access matrix is sparse. It is normally implemented either as access lists associated with each object or as capability lists associated with each domain. We can include dynamic protection in the access-matrix model by considering domains and the access matrix itself as objects. Revocation of access rights in a dynamic protection model is typically easier to implement with an access-list scheme than with a capability list.
- Real systems are much more limited than the general model. Older UNIX distributions are representative, providing discretionary access controls of read, write, and execution protection separately for the owner, group, and general public for each file. More modern systems are closer to the general model, or at least provide a variety of protection features to protect the system and its users.
- Solaris 10 and beyond, among other systems, implement the principle of least privilege via role-based access control, a form of access matrix.

Another protection extension is mandatory access control, a form of system policy enforcement.

- Capability-based systems offer finer-grained protection than older models, providing specific abilities to processes by “slicing up” the powers of root into distinct areas. Other methods of improving protection include System Integrity Protection, system-call filtering, sandboxing, and code signing.
- Language-based protection provides finer-grained arbitration of requests and privileges than the operating system is able to provide. For example, a single Java JVM can run several threads, each in a different protection class. It enforces the resource requests through sophisticated stack inspection and via the type safety of the language.

18.9 Summary

- Virtualization is a method for providing a guest with a duplicate of a system's underlying hardware. Multiple guests can run on a given system, each believing that it is the native operating system and is in full control.
- Virtualization started as a method to allow IBM to segregate users and provide them with their own execution environments on IBM mainframes. Since then, thanks to improvements in system and CPU performance and innovative software techniques, virtualization has become a common feature in data centers and even on personal computers. Because of its popularity, CPU designers have added features to support virtualization. This snowball effect is likely to continue, with virtualization and its hardware support increasing over time.
- The virtual machine manager, or hypervisor, creates and runs the virtual machine. Type 0 hypervisors are implemented in the hardware and require modifications to the operating system to ensure proper operation. Some

730

Chapter 18 Virtual Machines

type 0 hypervisors offer an example of paravirtualization, in which the operating system is aware of virtualization and assists in its execution.

- Type 1 hypervisors provide the environment and features needed to create, run, and manage guest virtual machines. Each guest includes all of the software typically associated with a full native system, including the operating system, device drivers, applications, user accounts, and so on.

- Type 2 hypervisors are simply applications that run on other operating systems, which do not know that virtualization is taking place. These hypervisors do not have hardware or host support so must perform all virtualization activities in the context of a process.
- Programming-environment virtualization is part of the design of a programming language. The language specifies a containing application in which programs run, and this application provides services to the programs.
- Emulation is used when a host system has one architecture and the guest was compiled for a different architecture. Every instruction the guest wants to execute must be translated from its instruction set to that of the native hardware. Although this method involves some performance penalty, it is balanced by the usefulness of being able to run old programs on newer, incompatible hardware or run games designed for old consoles on modern hardware.
- Implementing virtualization is challenging, especially when hardware support is minimal. The more features provided by the system, the easier virtualization is to implement and the better the performance of the guests.
- VMMs take advantage of whatever hardware support is available when optimizing CPU scheduling, memory management, and I/O modules to provide guests with optimum resource use while protecting the VMM from the guests and the guests from one another.
- Current research is extending the uses of virtualization. Unikernels aim to increase efficiency and decrease security attack surface by compiling an application, its libraries, and the kernel resources the application needs into one binary with one address space that runs within a virtual machine. Partitioning hypervisors provide secure execution, real-time operation, and other features traditionally only available to applications running on dedicated hardware.

Chapter 18: Virtual Machines

19.10 Summary

- A distributed system is a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory, and the processors communicate with one another through various communication lines, such as high-speed buses and the Internet. The processors in a distributed system vary in size and function.
- A distributed system provides the user with access to all system resources. Access to a shared resource can be provided by data migration, computation migration, or process migration. The access can be specified by the user or implicitly supplied by the operating system and applications.

Practice Exercises 769

- Protocol stacks, as specified by network layering models, add information to a message to ensure that it reaches its destination.
- A naming system (such as DNS) must be used to translate from a host name to a network address, and another protocol (such as ARP) may be needed to translate the network number to a network device address (an Ethernet address, for instance).
- If systems are located on separate networks, routers are needed to pass packets from source network to destination network.
- The transport protocols UDP and TCP direct packets to waiting processes through the use of unique system-wide port numbers. In addition, the TCP protocol allows the flow of packets to become a reliable, connection-oriented byte stream.

Chapter 19: Networks and Distributed Systems

- There are many challenges to overcome for a distributed system to work correctly. Issues include naming of nodes and processes in the system, fault tolerance, error recovery, and scalability. Scalability issues include handling increased load, being fault tolerant, and using efficient storage schemes, including the possibility of compression and/or deduplication.
- A DFS is a file-service system whose clients, servers, and storage devices are dispersed among the sites of a distributed system. Accordingly, service activity has to be carried out across the network; instead of a single centralized data repository, there are multiple independent storage devices.
- There are two main types of DFS models: the client–server model and the cluster-based model. The client-server model allows transparent file sharing among one or more clients. The cluster-based model distributes the files among one or more data servers and is built for large-scale parallel data processing.
- Ideally, a DFS should look to its clients like a conventional, centralized file system (although it may not conform exactly to traditional file-system interfaces such as POSIX). The multiplicity and dispersion of its servers and storage devices should be transparent. A transparent DFS facilitates client mobility by bringing the client's environment to the site where the client logs in.
- There are several approaches to naming schemes in a DFS. In the simplest approach, files are named by some combination of their host name and local name, which guarantees a unique system-wide name. Another approach, popularized by NFS, provides a means to attach remote directories to local directories, thus giving the appearance of a coherent directory tree.
- Requests to access a remote file are usually handled by two complementary methods. With remote service, requests for accesses are delivered to the server. The server machine performs the accesses, and the results are forwarded back to the client. With caching, if the data needed to satisfy the access request are not already cached, then a copy of the data is brought from the server to the client. Accesses are performed on the cached copy. The problem of keeping the cached copies consistent with the master file is the cache-consistency problem.

← → C code-maven.com/interactive-shell-with-cmd-in-python

Code Maven Login Register Exercises Slides Books Courses Services Type keyword:

Python examples/python/cli/full.py

12. Python unittest fails, but return exit code 0 - how to fix

13. Testing with PyTest

14. Parsing test results from JUnit XML files with Python

15. Combine pytest reports

```
1. from cmd import Cmd
2.
3. class MyPrompt(Cmd):
4.     prompt = 'pb> '
5.     intro = "Welcome! Type ? to list commands"
6.
7.     def do_exit(self, inp):
8.         print("Bye")
9.         return True
10.
11.    def help_exit(self):
12.        print('exit the application. Shorthand: x q Ctrl-D.')
13.
14.    def do_add(self, inp):
15.        print("adding '{}'".format(inp))
16.
17.    def help_add(self):
18.        print("Add a new entry to the system.")
19.
20.    def default(self, inp):
21.        if inp == 'x' or inp == 'q':
22.            return self.do_exit(inp)
23.
24.        print("Default: {}".format(inp))
25.
26.    do_EOF = do_exit
27.    help_EOF = help_exit
28.
29. if __name__ == '__main__':
30.     MyPrompt().cmdloop()
```

- Check this video:
https://www.youtube.com/watch?v=B55roi7rE_Y

Szabo, G. (2018) Create your own interactive shell with cmd in Python. Available from: <https://code-maven.com/interactive-shell-with-cmd-in-python>