

Figure 1.13 Transition from user to kernel mode.

the request. This is shown in Figure 1.13. As we shall see, this architectural enhancement is useful for many other aspects of system operation as well.

1. How does the user process write to a device show the diagram for program that goes from user mode to kernel mode and back

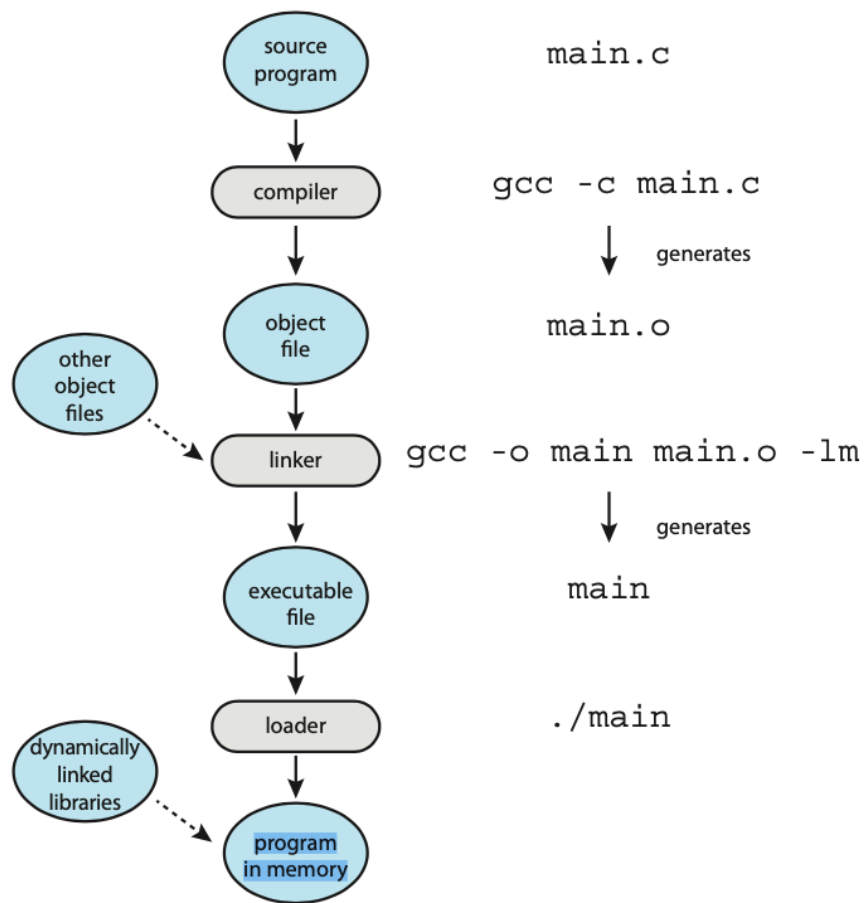


Figure 2.11 The role of the linker and loader.

2. Draw diagram of how the user writes a program and how does it get compiled, linked and loaded in memory and how does DLL get used.

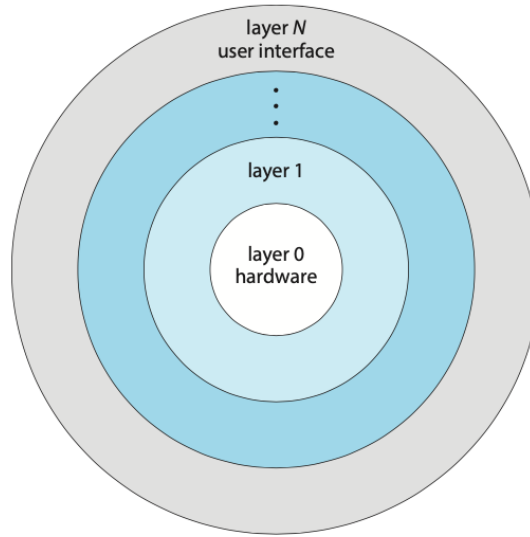


Figure 2.14 A layered operating system.

and services of only lower-level layers. This approach simplifies debugging

3. Draw diagram of layer approach OS and benefits of layered approach

2.8 Operating-System Structure

85

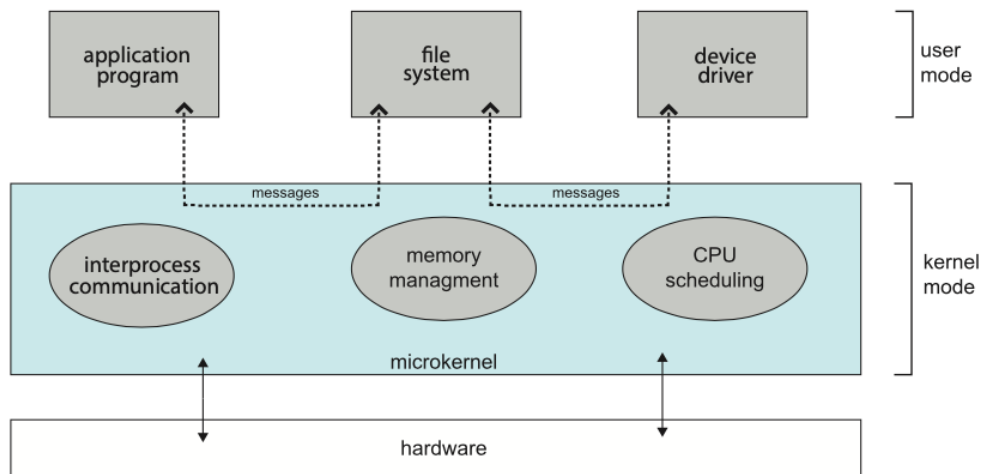


Figure 2.15 Architecture of a typical microkernel.

4. Draw diagram of a micro kernel OS

3.1.1 The Process

Informally, as mentioned earlier, a process is a program in execution. The status of the current activity of a process is represented by the value of the **program counter** and the contents of the processor's registers. The memory layout of a process is typically divided into multiple sections, and is shown in Figure 3.1. These sections include:

- **Text section**—the executable code
- **Data section**—global variables

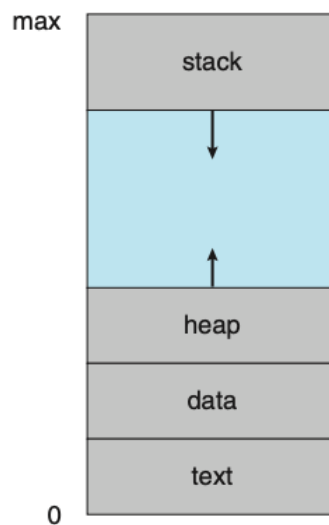


Figure 3.1 Layout of a process in memory.

5. What is a process and write different sections of a process along with diagram

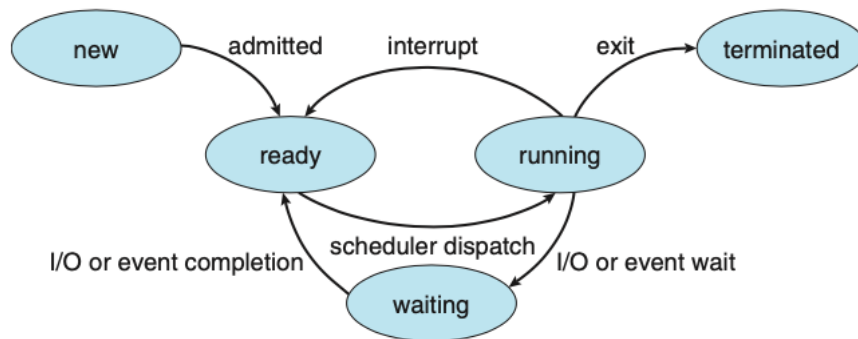


Figure 3.2 Diagram of process state.

- **Terminated.** The process has finished execution.

6. What are different process states draw a diagram of the different states
block (PCB)—also called a **task control block**. A PCB is shown in Figure 3.3. It contains many pieces of information associated with a specific process, including these:

- **Process state.** The state may be new, ready, running, waiting, halted, and so on.
- **Program counter.** The counter indicates the address of the next instruction to be executed for this process.

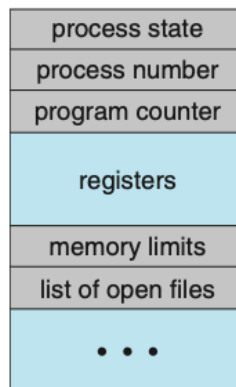


Figure 3.3 Process control block (PCB).

7. What is a process control block of a process draw diagram and describe 3 parts of PCB

thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

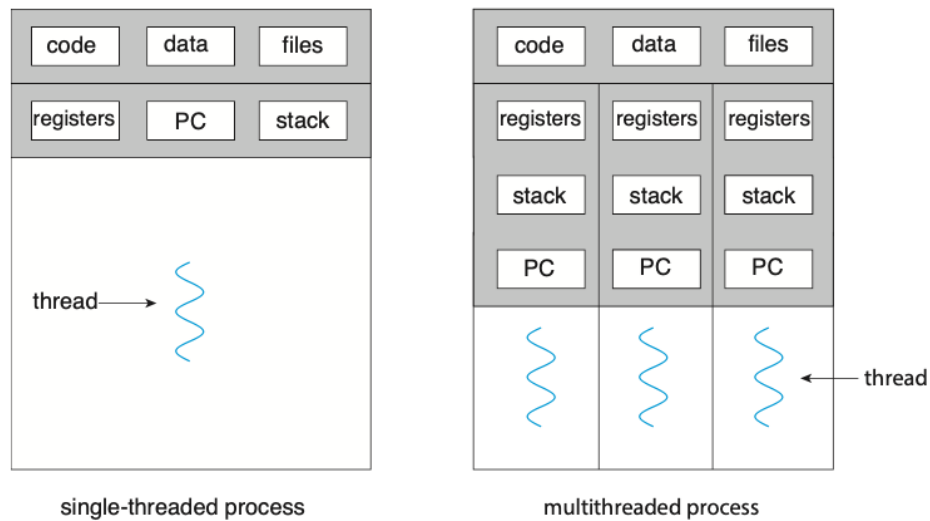


Figure 4.1 Single-threaded and multithreaded processes.

8. Difference between process and thread draw diagram of single threaded vs multi threaded process. What are the shared parts of a multi threaded program

The benefits of multithreaded programming can be broken down into four major categories:

1. **Responsiveness.** Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. This quality is especially useful in designing user interfaces. For instance, consider what happens when a user clicks a button that results in the performance of a time-consuming operation. A single-threaded application would be unresponsive to the user until the operation had been completed. In contrast, if the time-consuming operation is performed in a separate, asynchronous thread, the application remains responsive to the user.
2. **Resource sharing.** Processes can share resources only through techniques such as shared memory and message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
3. **Economy.** Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general thread creation consumes less time and memory than process creation. Additionally, context switching is typically faster between threads than between processes.
4. **Scalability.** The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores. A single-threaded process can run on only one processor, regardless how many are available. We explore this issue further in the following section.

9. Name 3 benefits of multithreaded program

5.2 Scheduling Criteria

Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.

Many criteria have been suggested for comparing CPU-scheduling algorithms. Which characteristics are used for comparison can make a substantial difference in which algorithm is judged to be best. The criteria include the following:


- **CPU utilization.** We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system). (CPU utilization can be obtained by using the `top` command on Linux, macOS, and UNIX systems.)
- **Throughput.** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed

5.3 Scheduling Algorithms 205

per time unit, called **throughput**. For long processes, this rate may be one process over several seconds; for short transactions, it may be tens of processes per second.

- **Turnaround time.** From the point of view of a particular process, the

10. What are different scheduling criteria for processes, throughput, turn around time etc.
11. One scheduling algo find turn around time etc.



Solution to Deadlock for Dining Philosophers Problem

- Allow at most four philosophers to be sitting simultaneously at the table.
- Allow a philosopher to pick up her chopsticks only if both chopsticks are available (to do this, she must pick them up in a critical section).
- Use an asymmetric solution—that is, odd-numbered philosopher picks up first her left chopstick and then her right chopstick, whereas an even numbered philosopher picks up her right chopstick and then her left chopstick.

12. What are different ways in which you can solve dining philosopher problem.

13. What is a binary semaphore. What is Wait and signal

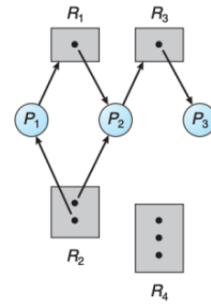
Necessary Conditions

- **Mutual exclusion:** At least one resource must be held in a non sharable mode. If another process requests that resource, the requesting process must be delayed until the resource has been released.
- **Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
- **No preemption:** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** A set $\{P_0, P_1, \dots, P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource held by P_1 , P_1 is waiting for a resource held by P_2 , ..., P_{n-1} is waiting for a resource held by P_n , and P_n is waiting for a resource held by P_0 .

14. Necessary conditions of deadlock

Resource Allocation graph...

- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$
- No Deadlock in this situation since there is no cycle.



15. Resource allocation graph

16. Bankers algorithm