Processes

Early computers allowed only one program to be executed at a time. This program had complete control of the system and had access to all the system's

resources. In contrast, contemporary computer systems allow multiple programs to be loaded into memory and executed concurrently. This evolution

required firmer control and more compartmentalization of the various programs; and these needs resulted in the notion of a process, which is a program

in execution. A process is the unit of work in a modern computing system.

The more complex the operating system is, the more it is expected to do on

behalf of its users. Although its main concern is the execution of user programs,

it also needs to take care of various system tasks that are best done in user space,

rather than within the kernel. A system therefore consists of a collection of

processes, some executing user code, others executing operating system code.

Potentially, all these processes can execute concurrently, with the CPU (or CPUs)

multiplexed among them. In this chapter, you will read about what processes

are, how they are represented in an operating system, and how they work.

CHAPTER OBJECTIVES

• Identify the separate components of a process and illustrate how they are

represented and scheduled in an operating system.

• Describe how processes are created and terminated in an operating system, including developing programs using the appropriate system calls

that perform these operations.

• Describe and contrast interprocess communication using shared memory

and message passing.

• Design programs that use pipes and POSIX shared memory to perform

interprocess communication.

• Describe client – server communication using sockets and remote procedure calls.

• Design kernel modules that interact with the Linux operating system.

One-Line Exam Definition

A process is a program in execution, representing all active CPU activities in an operating system.

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

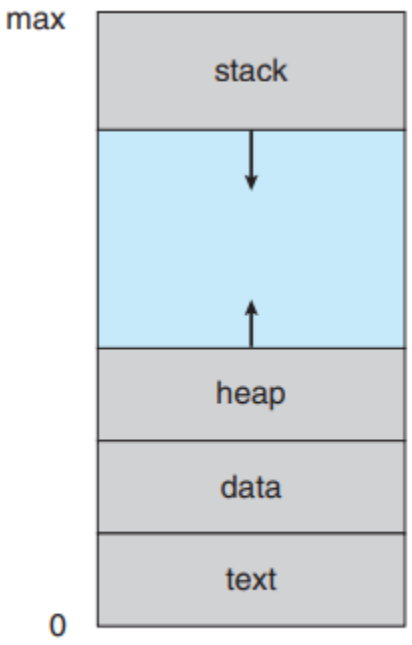


Figure 3.1 Layout of a process in memory.

• Heap section—memory that is dynamically allocated during program run time

• Stack section— temporary data storage when invoking functions (such as function parameters, return addresses, and local variables)

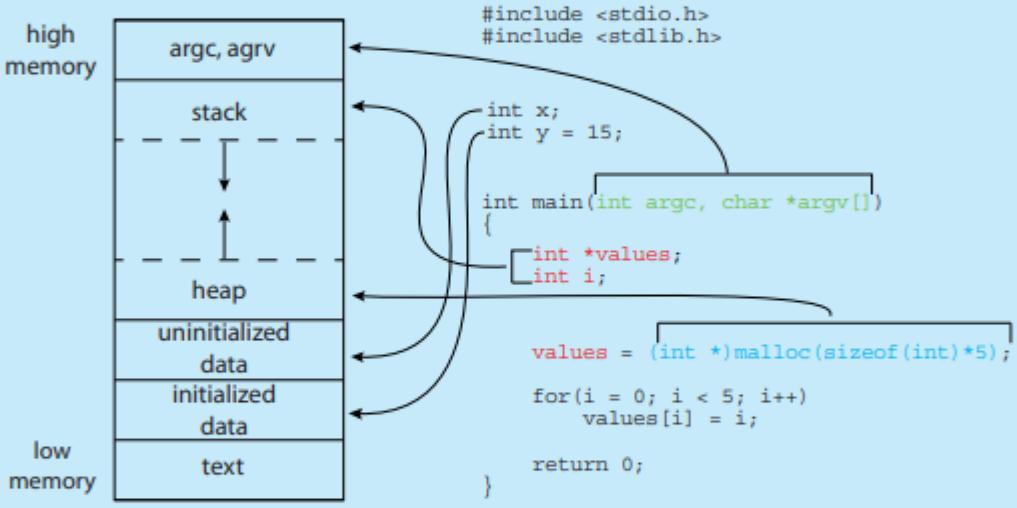
3.1.2 Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

MEMORY LAYOUT OF A C PROGRAM

The figure shown below illustrates the layout of a C program in memory, highlighting how the different sections of a process relate to an actual C program. This figure is similar to the general concept of a process in memory as shown in Figure 3.1, with a few differences:

- The global data section is divided into different sections for (a) initialized data and (b) uninitialized data.
- A separate section is provided for the `argc` and `argv` parameters passed to the `main()` function.



The GNU `size` command can be used to determine the size (in bytes) of some of these sections. Assuming the name of the executable file of the above C program is `memory`, the following is the output generated by entering the command `size memory`:

```
text    data    bss     dec     hex     filename
1158    284     8       1450    5aa     memory
```

The `data` field refers to uninitialized data, and `bss` refers to initialized data. (`bss` is a historical term referring to ***block started by symbol***.) The `dec` and `hex` values are the sum of the three sections represented in decimal and hexadecimal, respectively.

• New. The process is being created.

• Running. Instructions are being executed.

• Waiting. The process is waiting for some event to occur (such as an I/O

completion or reception of a signal).

• Ready. The process is waiting to be assigned to a processor

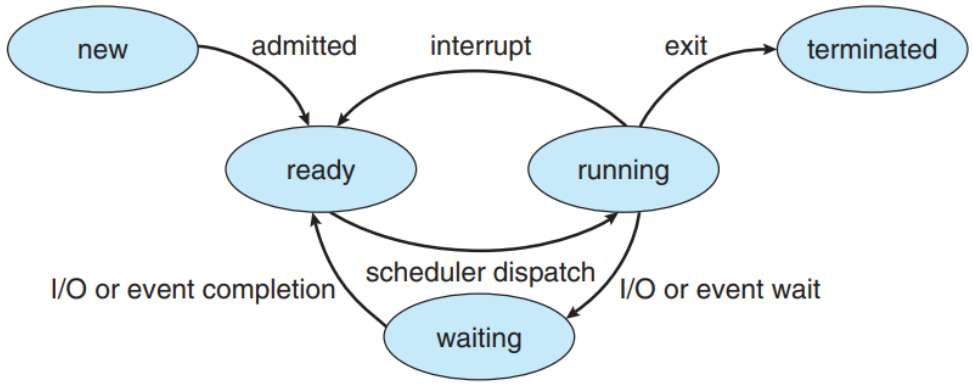


Figure 3.2 Diagram of process state.

• Terminated. The process has finished execution.

3.1.3 Process Control Block

Each process is represented in the operating system by a process control 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.

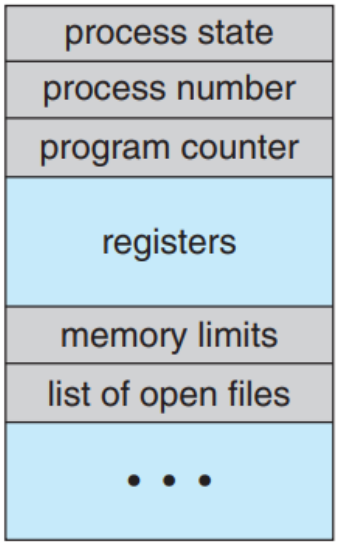


Figure 3.3 Process control block (PCB).

• CPU registers. The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward when it is rescheduled to run.

• CPU-scheduling information. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

(Chapter 5 describes process scheduling.)

• Memory-management information. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system (Chapter 9).

• Accounting information. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

• I/O status information. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

In brief, the PCB simply serves as the repository for all the data needed to start, or restart, a process, along with some accounting data.