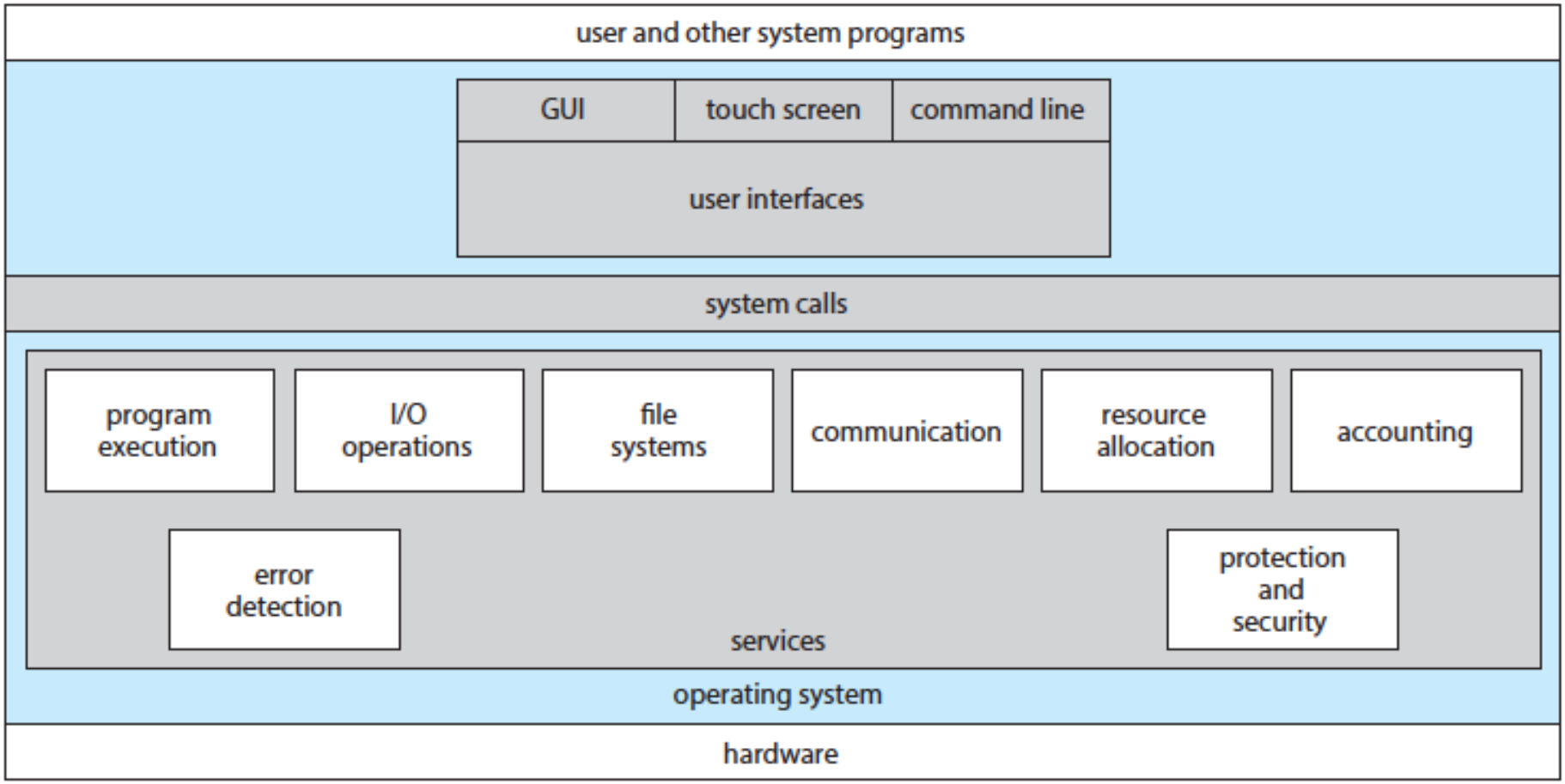
OS Structures

Systems II

Objectives

- Identify services provided by an operating system.
- Illustrate how system calls are used to provide operating system services.
- Compare monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems.
- Illustrate the process for booting an operating system.
- Apply tools for monitoring operating system performance.
- Design and implement kernel modules for interacting with a Linux kernel.

OS Services


user and other system programs
GUI
touch screen
command line
user interfaces
system calls
program execution
I/O operations
file systems
communication
resource allocation
accounting
error detection
protection and security
services
operating system
hardware

OS Services

- user interface (UI):
  graphical user interface (GUI),
  touch-screen interface,
  command-line interface (CLI)
- program execution: load a program into memory + run that program; program can end its execution (normally or by indicating error)
- I/O operations: users usually cannot control I/O devices directly (efficiency & protection)
- file-system manipulation: programs need to read and write files & directories; need to create and delete files; search for a given file; list file information; permissions management (allow or deny access to files or directories to other users); many OS provide a variety of file systems

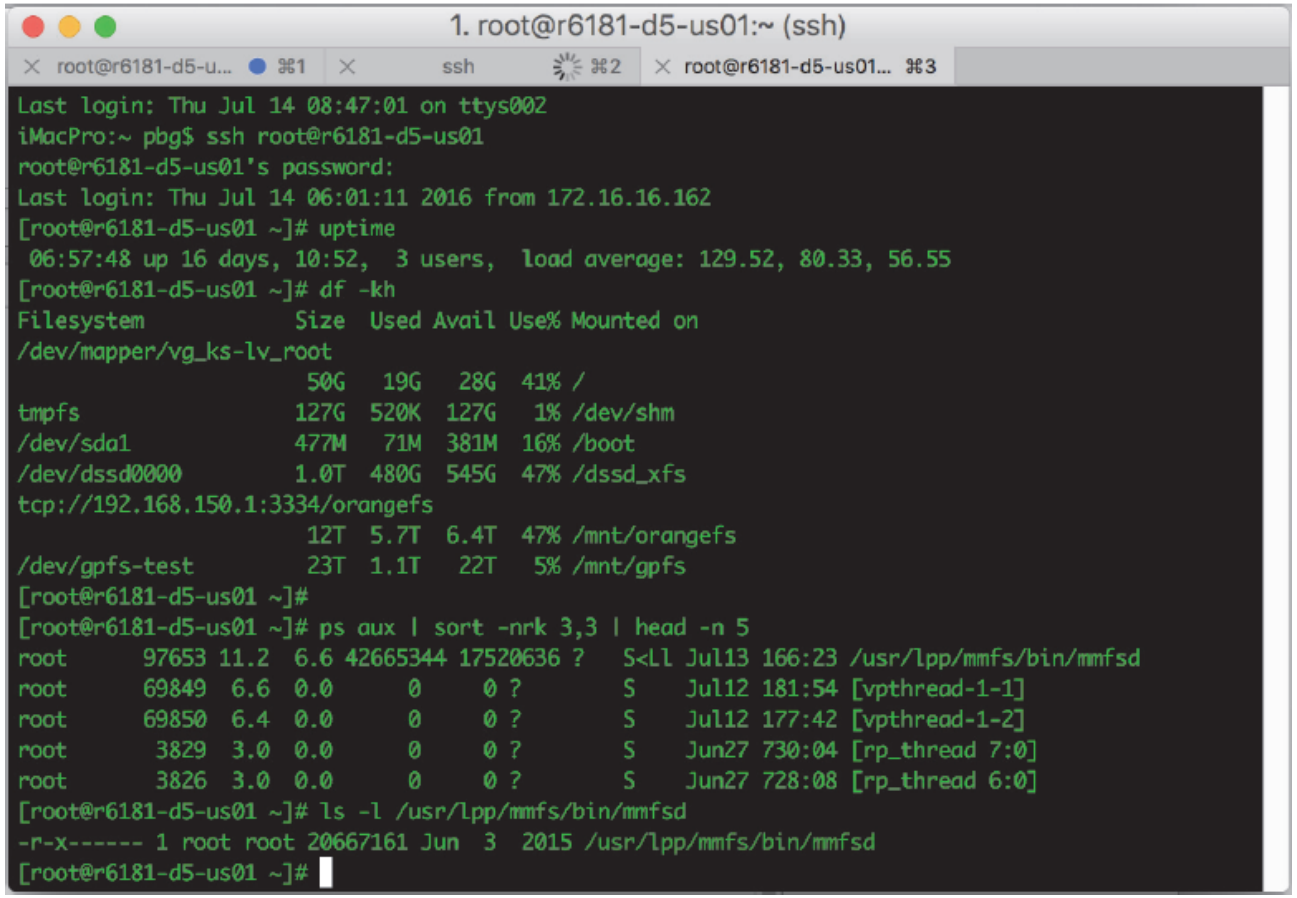
OS Services

- communications: one process can exchange information with another process (on the same computer or on different computer systems); may be implemented via shared memory or message passing
- error detection: OS needs to be detecting and correcting errors constantly: errors may occur in the CPU and memory (e.g. memory error or power failure), in I/O devices (e.g. parity error on disk, a connection failure on a network, lack of paper in the printer), in the user program (e.g. arithmetic overflow or accessing an illegal memory location).
  OS should take the appropriate action to ensure correct and consistent computing. Sometimes, it has no choice but to halt the system. It might terminate an error-causing process.

OS Services

- resource allocation: OS manages many different types of resources (e.g. CPU cycles, main memory, file storage, I/O devices) logging: keep track of which programs use how much and what kinds of computer resources
- protection & security: Protection = all access to system resources most be controlled. Security of the system from outsiders: authenticate to the system by password, record invalid access attempts.

Command interpreters (shells)

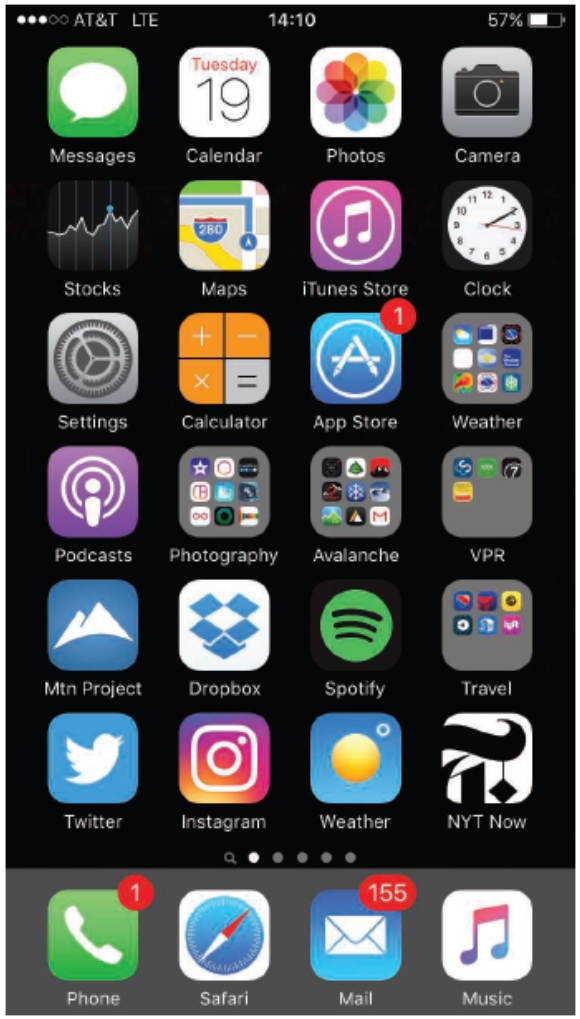


- Linux systems offer: C shell, Bourne-Again shell, Korn shell, ...

Graphical User Interface

- desktop, icons, menus, ...
- Xerox PARC, 1970s
- open-source interfaces: K Desktop Environment (KDE), GNOME, ...

Touch-Screen Interface



System Calls

System calls are an interface to the services of the OS. (Written in C and assembly language.)

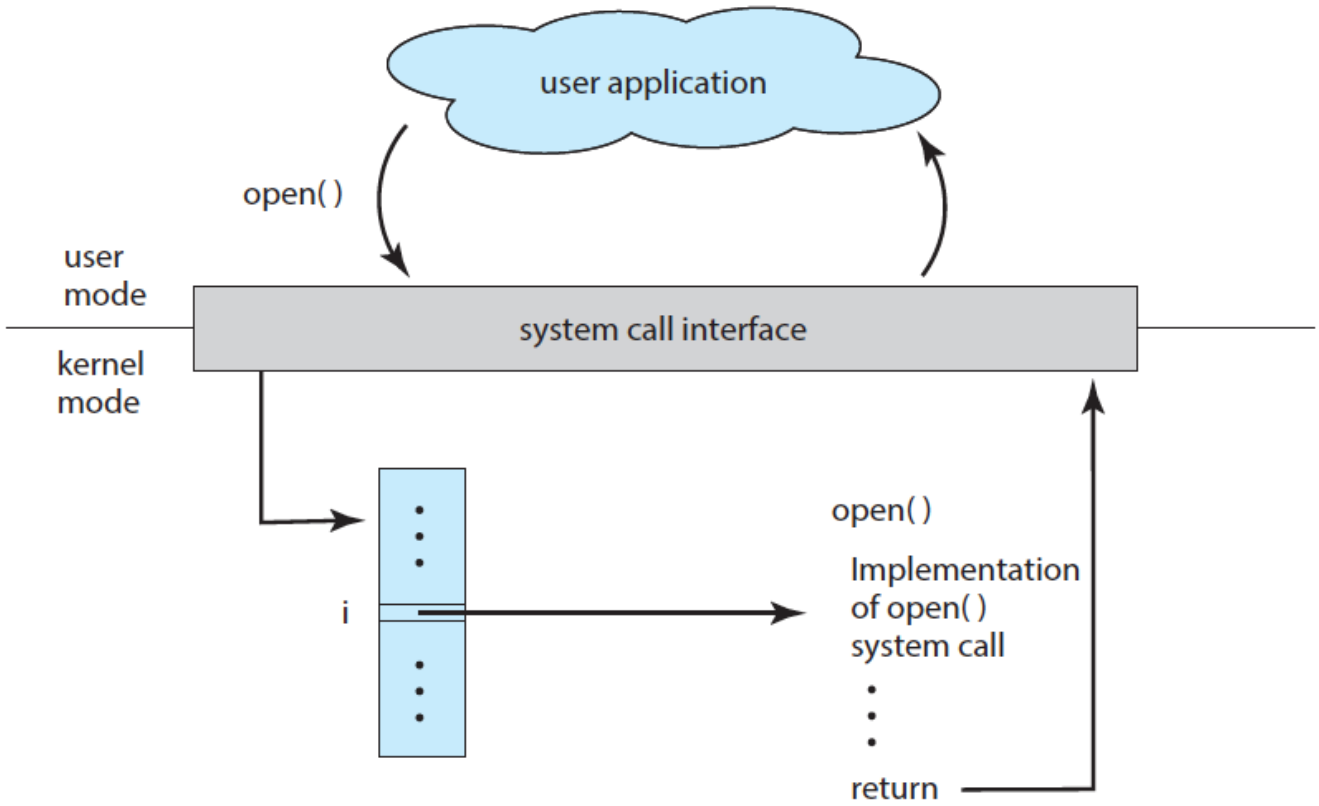
Application programming interface (API) is a set of functions available to an application programmer. Examples: Windows API, Java API, libc, ...

Why do we prefer API to system calls?

1. portability
2. system calls may be difficult to work with

System Calls

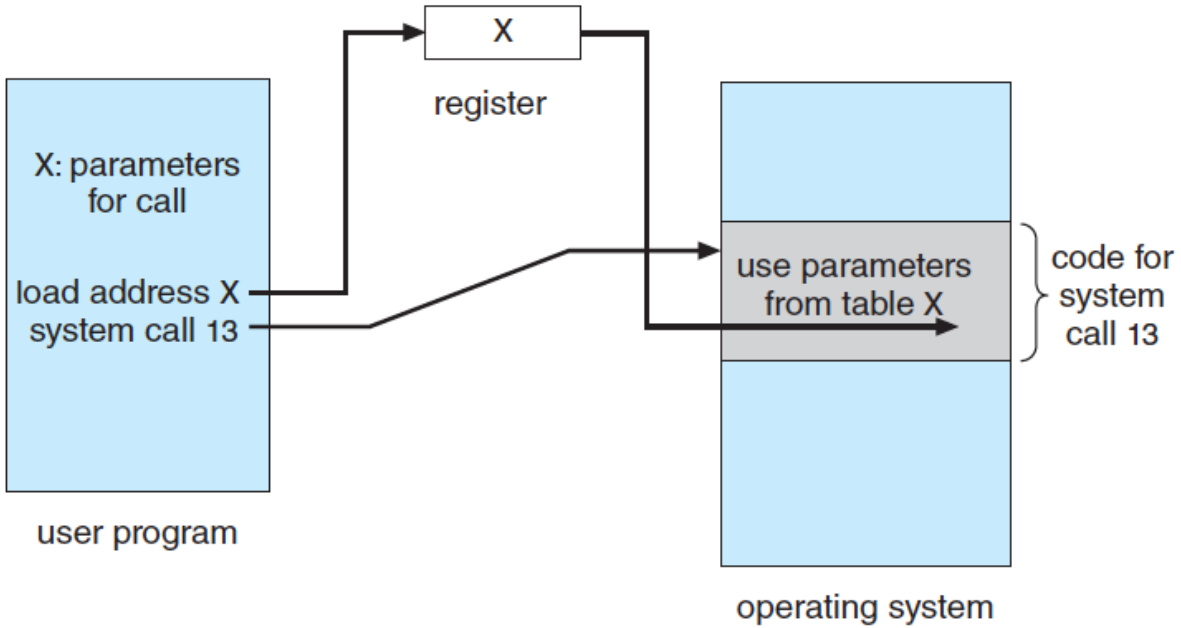
Run-time environment (RTE) software needed to execute applications written in a given language (compilers/interpreters, loaders, libraries, . . .)
RTE provides system-call interface.



System Calls

Passing parameters to the OS:

1. via registers (Linux: up to 5 parameters)
2. block method: parameters stored in a block in the memory, pointer is passed in a register (Linux: more than 5 parameters)
3. stack method: parameters are pushed onto and popped off a stack



Types of System Calls

1. Process control
    1. create process, terminate process
    2. load, execute
    3. get process attributes, set process attributes
    4. wait event, signal event
    5. allocate and free memory
2. File management
    1. create file, delete file
    2. open, close
    3. read, write, reposition
    4. get file attributes, set file attributes
3. Device management
    1. request device, release device
    2. read, write, reposition
    3. get device attributes, set device attributes
    4. logically attach or detach devices

Types of System Calls

1. Information maintenance
    1. get time or date, set time or date
    2. get system data, set system data
    3. get process, file, or device attributes
    4. set process, file, or device attributes
2. Communications
    1. create, delete communication connection
    2. send, receive messages
    3. transfer status information
    4. attach or detach remote devices
3. Protection
    1. get file permissions
    2. set file permissions

System Service (System Utilities)

1. File management
   Create, delete, copy, rename, print, list, and generally access and manipulate files and directories.
2. Status information
   Ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. May be more complex: provide detailed performance, logging, and debugging information. (Some systems also support a registry, which is used to store and retrieve configuration information.)
3. File modification
   Text editors to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.

System Service (System Utilities)

1. Programming-language support
   Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and Python) are often provided.
2. Program loading and execution
   The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.
3. Communications
   Mechanism for creating virtual connections among processes, users, and computer systems. Send messages to one another's screens (`wall`), browse web pages, send e-mail messages, log in remotely, transfer files from one machine to another.

System Service (System Utilities)

1. Background services

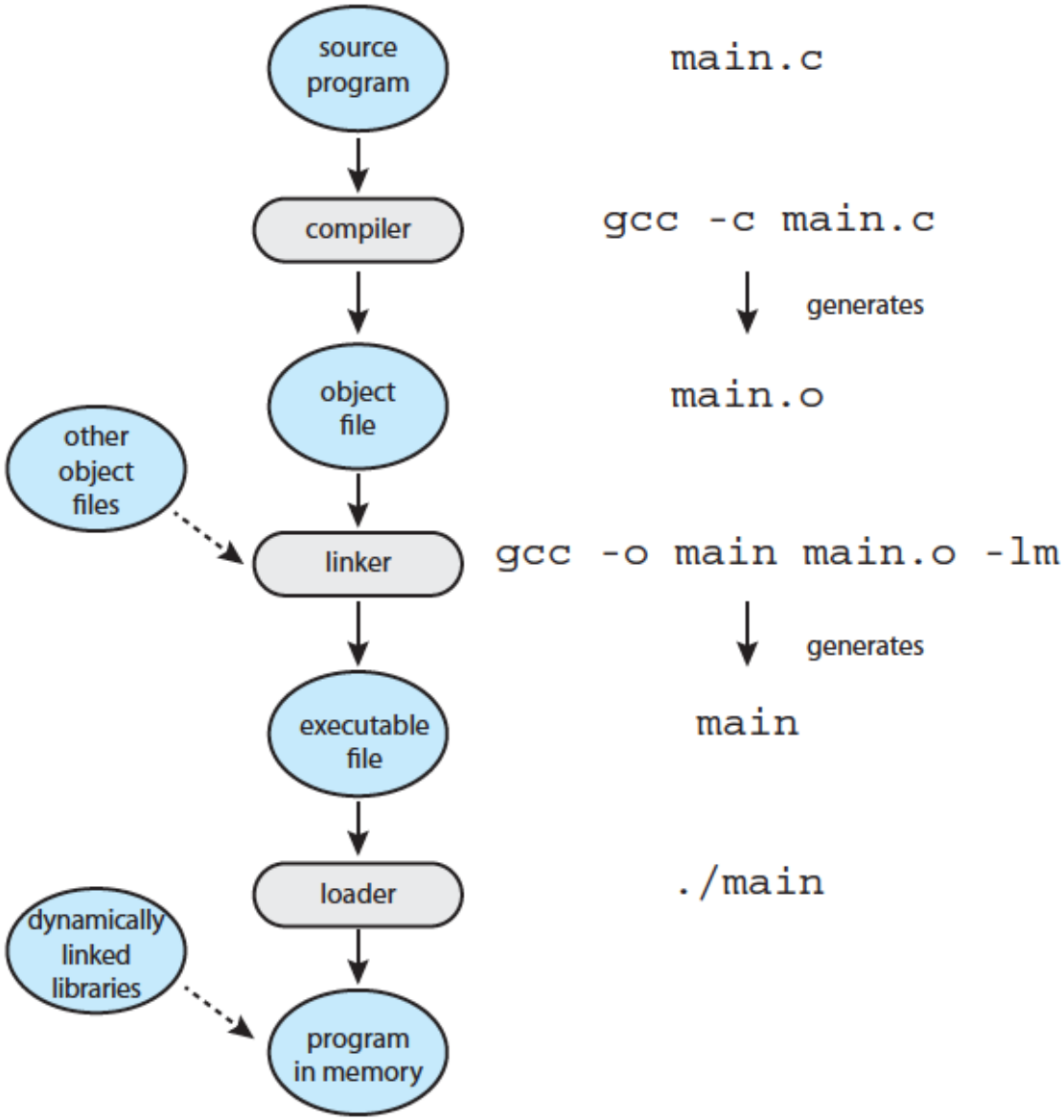
   General-purpose systems have methods for launching system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted. Constantly running system-program processes are known as services, subsystems, or daemons. Examples: process schedulers that start processes according to a specified schedule (`cron`), system error monitoring services, print servers, ...

Linkers & Loaders

1. source code is complied into: relocatable object files
   `$ gcc -c program.c`
2. linker combines object files into executable file
   (other libraries may be included; `-lm` for math library)
   `$ gcc -o program program.o -lm`
3. loader loads the binary file into memory (relocation)
4. dynamically linked libraries (DLLs)
5. standard formats for object and executable files
   Linux: ELF (Executable and Linkable Format)
   Windows: PS (Portable Executable)
   macOS: Mach-O

`file` command on Linux

Linkers & Loaders



OS Specific Applications?

How can be an application made to run on multiple OS?

1. interpreted language (Python, Ruby)
2. languages with RTE (Java virtual machine);
   RTE ported/developed for many OS
3. use a standard API (`POSIX API`)