CS420: Operating Systems

OS Services & System Calls

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania

Operating System Services

- **Operating systems provide an environment for execution of programs and services to programs and users**
- **One set of operating-system services provides functions that are helpful to the user:**
  - **User interface** - Almost all operating systems have a user interface (UI).
    - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
  - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

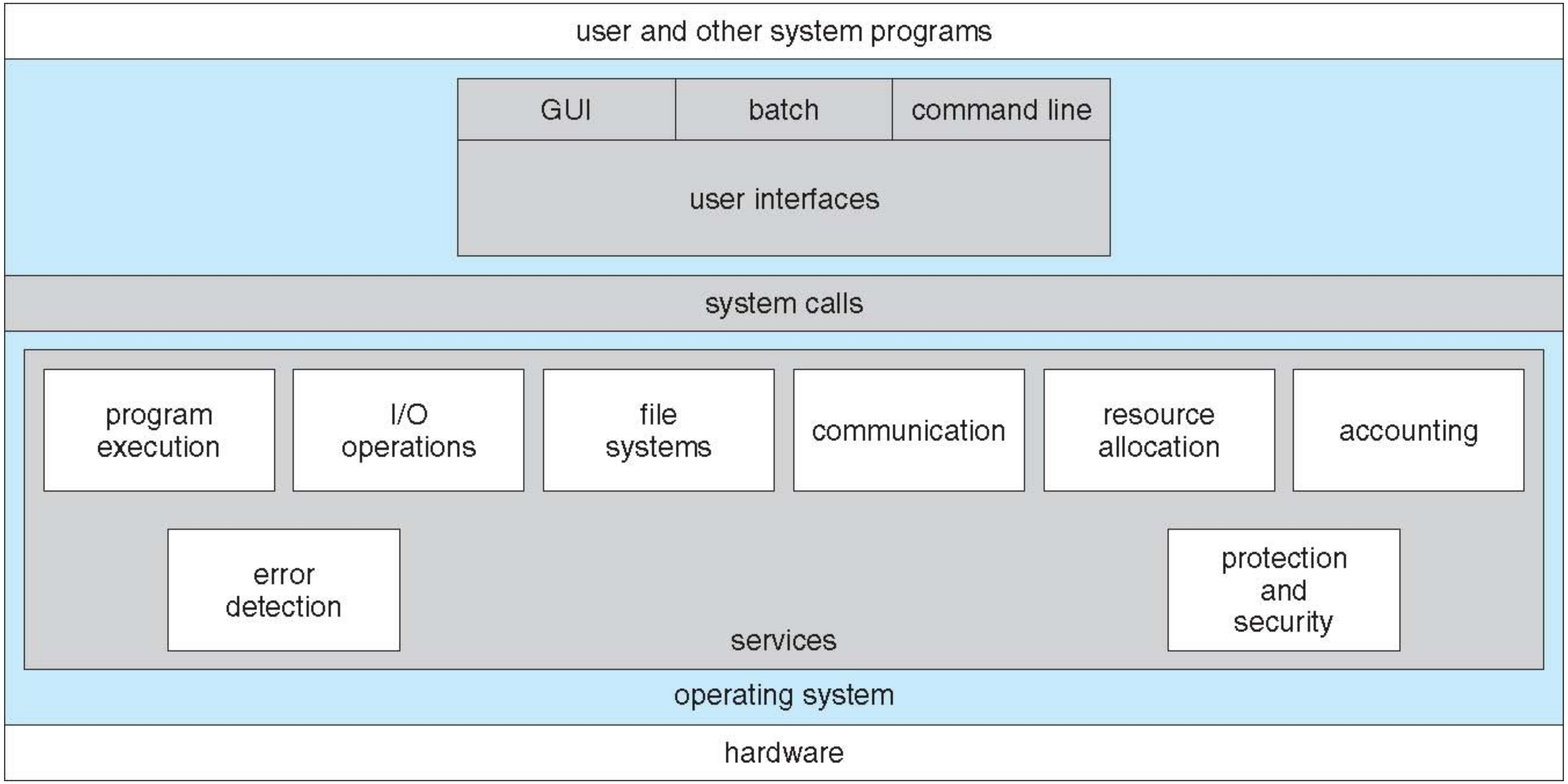
Operating System Services (Cont.)

- **Communications** – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)
- **Error detection** – OS needs to be constantly aware of possible errors
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
  - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont.)

- **Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing**
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code
  - **Accounting** - To keep track of the types and numbers of resources used by users
  - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
    - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

A View of Operating System Services



User Operating System Interface - CLI

- **Command Line Interface (CLI) or command interpreter allows direct command entry**
    - Sometimes implemented in kernel, sometimes by systems program
    - Sometimes multiple flavors implemented – shells
    - Primarily fetches a command from user and executes it
        - Sometimes commands are built-in, sometimes just names of programs
            - If the latter, adding new features doesn’t require shell modification

User Operating System Interface - GUI

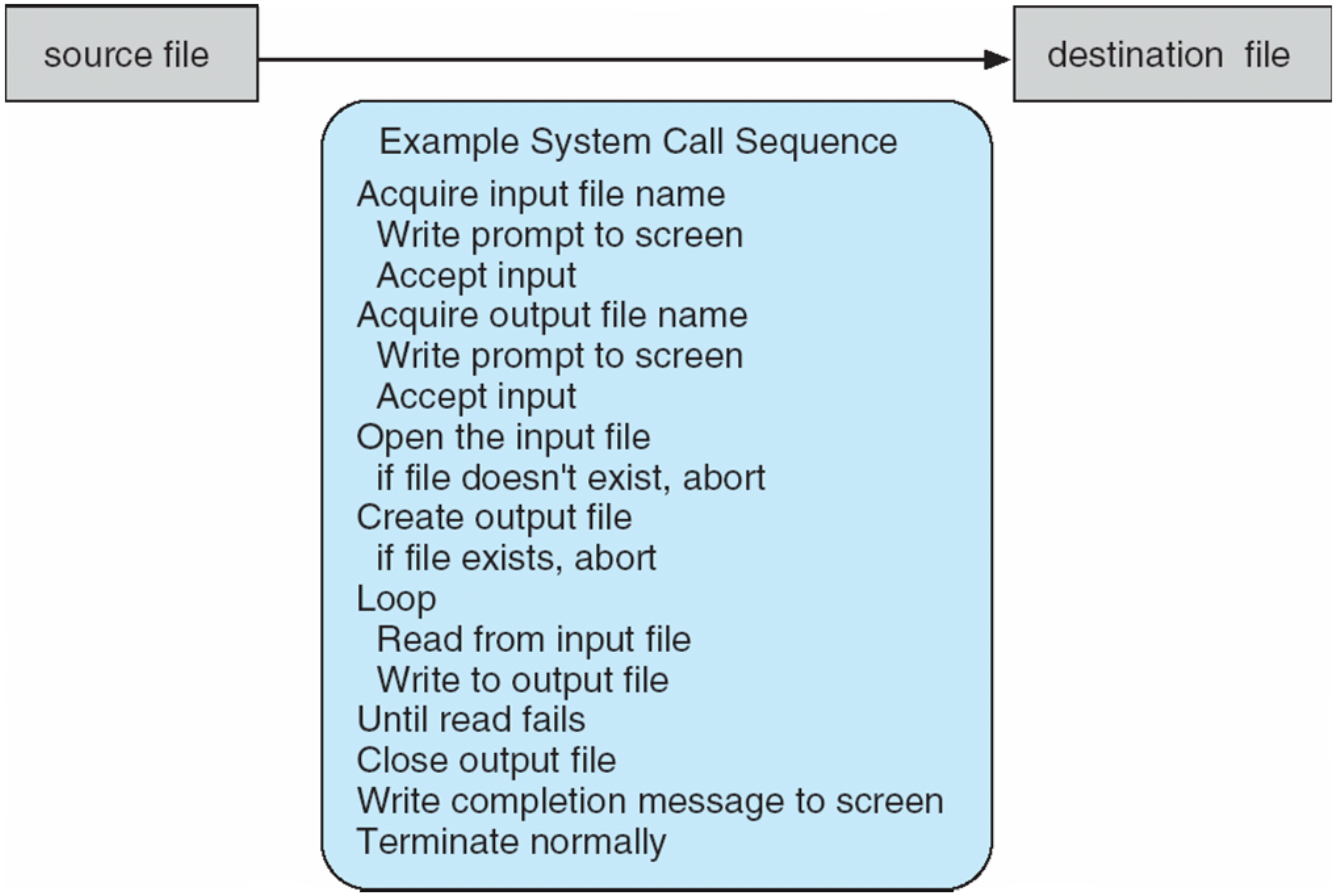
- **User-friendly desktop metaphor interface**
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc.
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory/folder
  - Invented at Xerox PARC in early 1970s
- **Many systems now include both CLI and GUI interfaces**
  - Microsoft Windows is GUI with CLI “command” shell
  - Apple Mac OS X (now macOS) has “Aqua” GUI interface with UNIX kernel underneath and shells available
  - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

System Calls

- **Programming interface to the services provided by the OS**
- **Typically written in a high-level language (C or C++)**
  - Some parts may be written in assembly language
- **Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use (e.g. `printf`)**
- **Three most common APIs are:**
  - Windows API for Windows systems
  - POSIX API for POSIX-based systems (virtually all versions of UNIX, Linux, and macOS),
  - Java API for the Java virtual machine (JVM)
- **Why use APIs rather than system calls?**

Example of System Calls

- **System call sequence to copy the contents of one file to another file**



```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

System Call Implementation

- **A system-call interface links function calls in an API to system calls provided by the OS**

- **The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values**

- **The caller need know nothing about how the system call is implemented**
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

System Call Parameter Passing

- **Three general methods used to pass parameters to the OS**

  1) Simplest: pass the parameters in registers
     - In some cases, may be more parameters than registers
  2) Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
     - This approach taken by Linux and Solaris
  3) Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system

Block and stack methods (2 & 3) do not limit the number or length of parameters being passed

Types of System Calls

- **Process control**
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- **File management**
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

Types of System Calls (Cont.)

- **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- **Information maintenance**
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes

Types of System Calls (Cont.)

- **Communications**
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach and detach remote devices
- **Protection**
  - get permissions, set permissions

Examples of System Calls

	Windows	Unix

Process control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File management	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device management	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

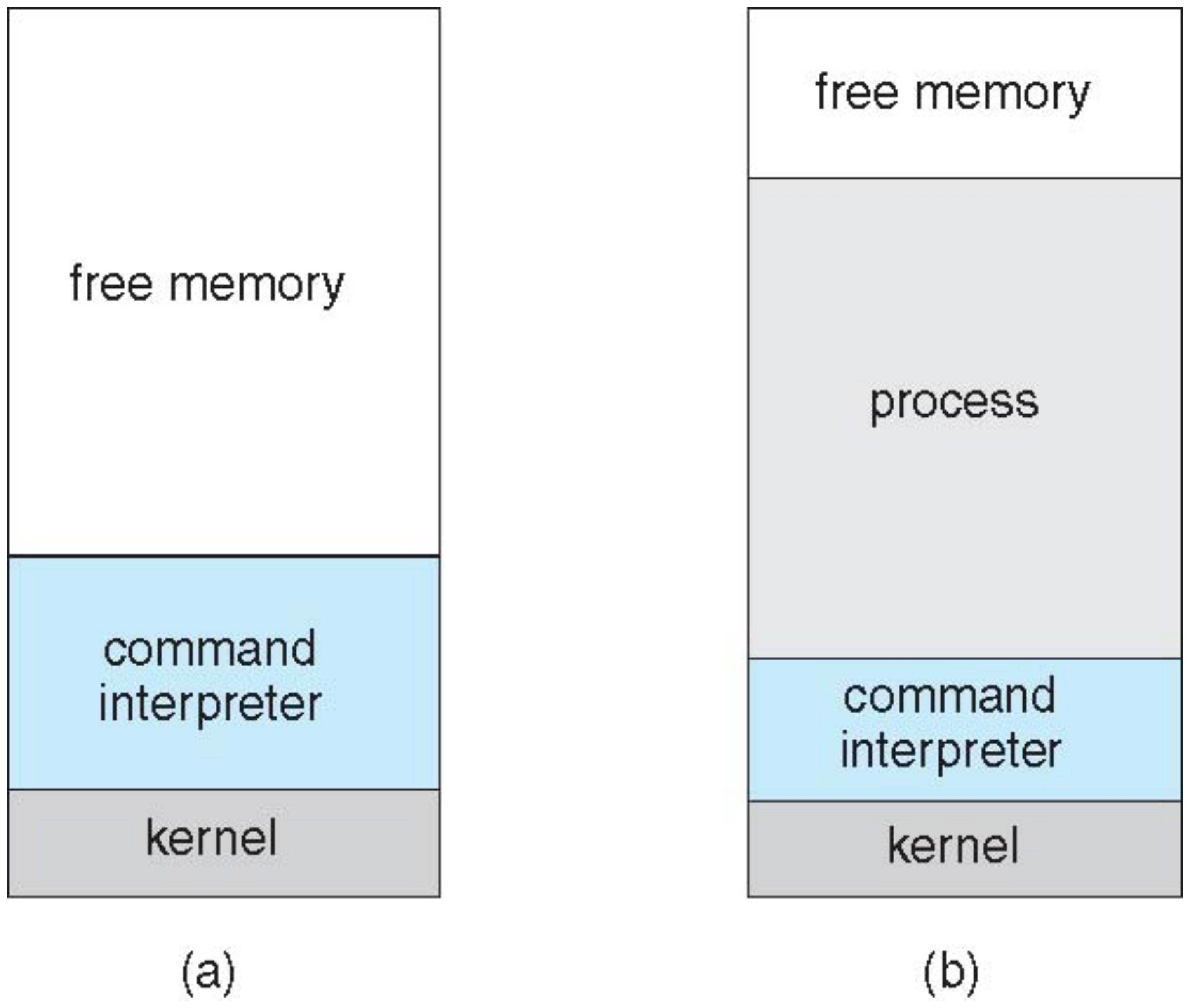
Communications	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shm_open()
		mmap()

Protection	SetFileSecurity()
	InitlializeSecurityDescriptor()
	SetSecurityDescriptorGroup()
		chmod()
		umask()
		chown()

Process Control In MS-DOS

- **Single-tasking**
- **Shell is invoked when system boots**
- **Simple method to run program**
  - No process created
- **Single memory space**
- **Loads program into memory, overwriting all but the kernel**
- **Program exit -> shell reloaded**
  - Program can pass information back to shell

Process Control In MS-DOS - CreateProcess

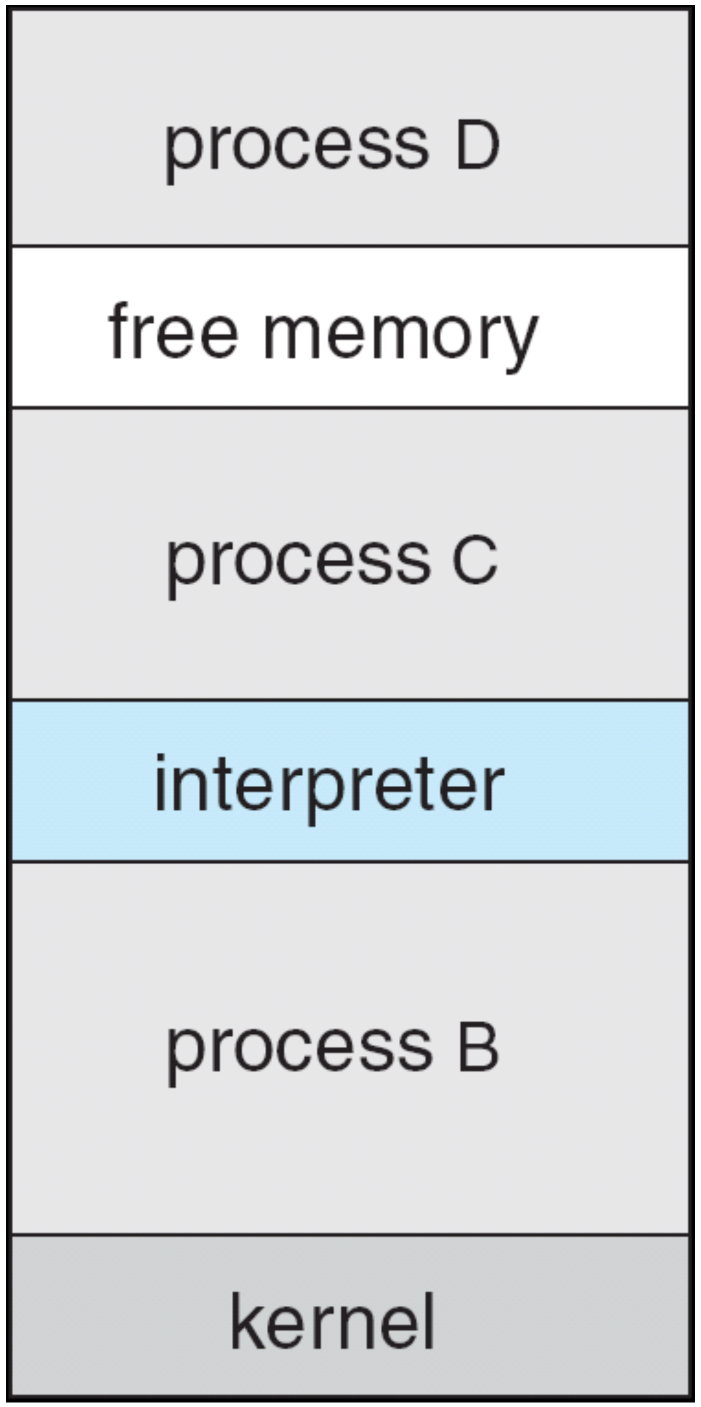


(a) At system startup (b) running a program

Process Control In FreeBSD

- **FreeBSD is a variant of Unix**
- **Supports multitasking**
- **User login -> invoke shell**
- **Shell executes `fork()` system call to create process**
  - Executes **`exec()`** to load program into memory
  - Shell waits for process to terminate or continues with user commands
- **Process exits with code of 0 if no error**
  - Exits with code > 0 if there is an error (error code identifies the type of error)

Process Control In FreeBSD


process D
free memory
process C
interpreter
process B
kernel