Amirkabir University of Technology
(Tehran Polytechnic)

Operating Systems

System Calls

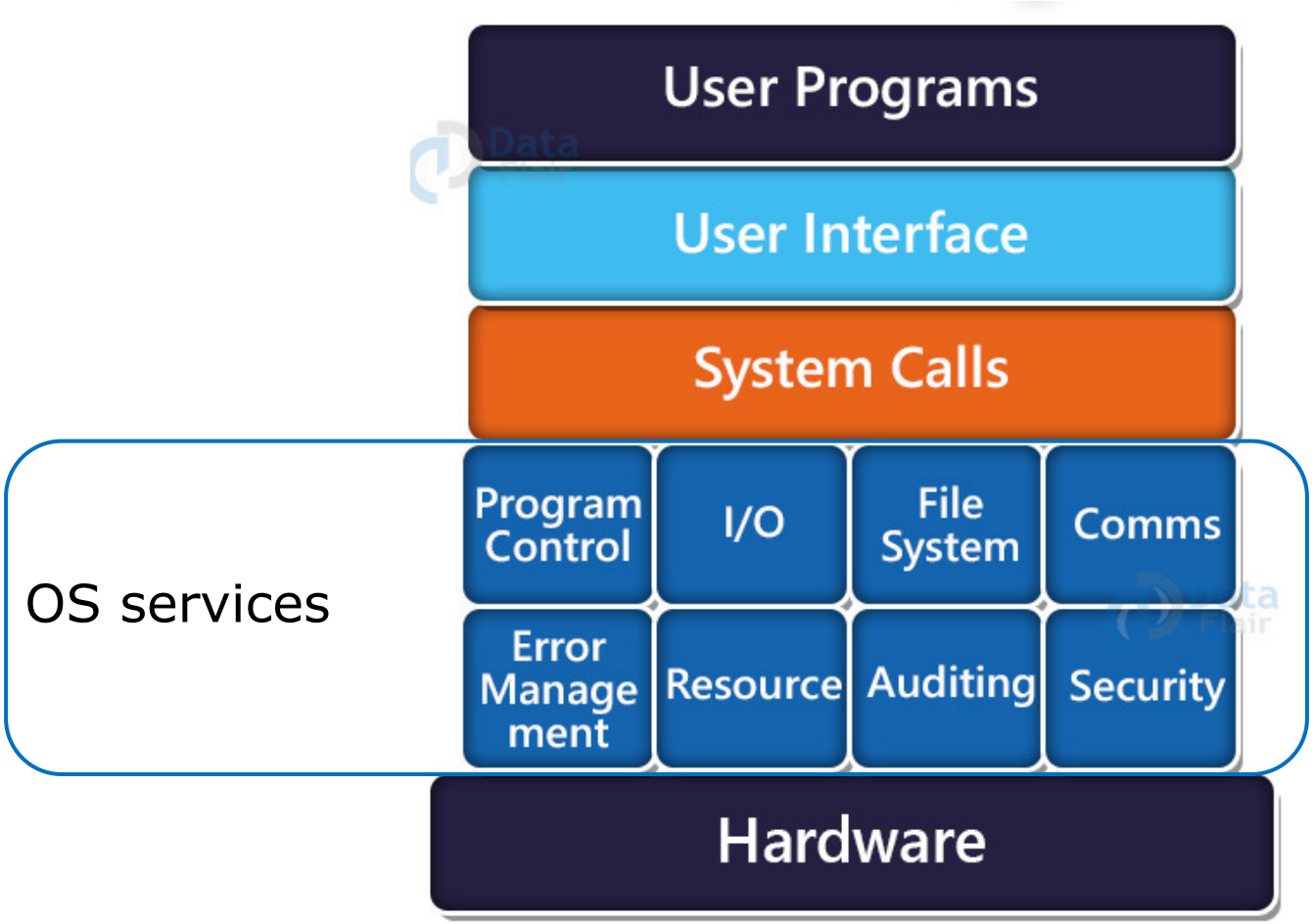
Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Fall 2024

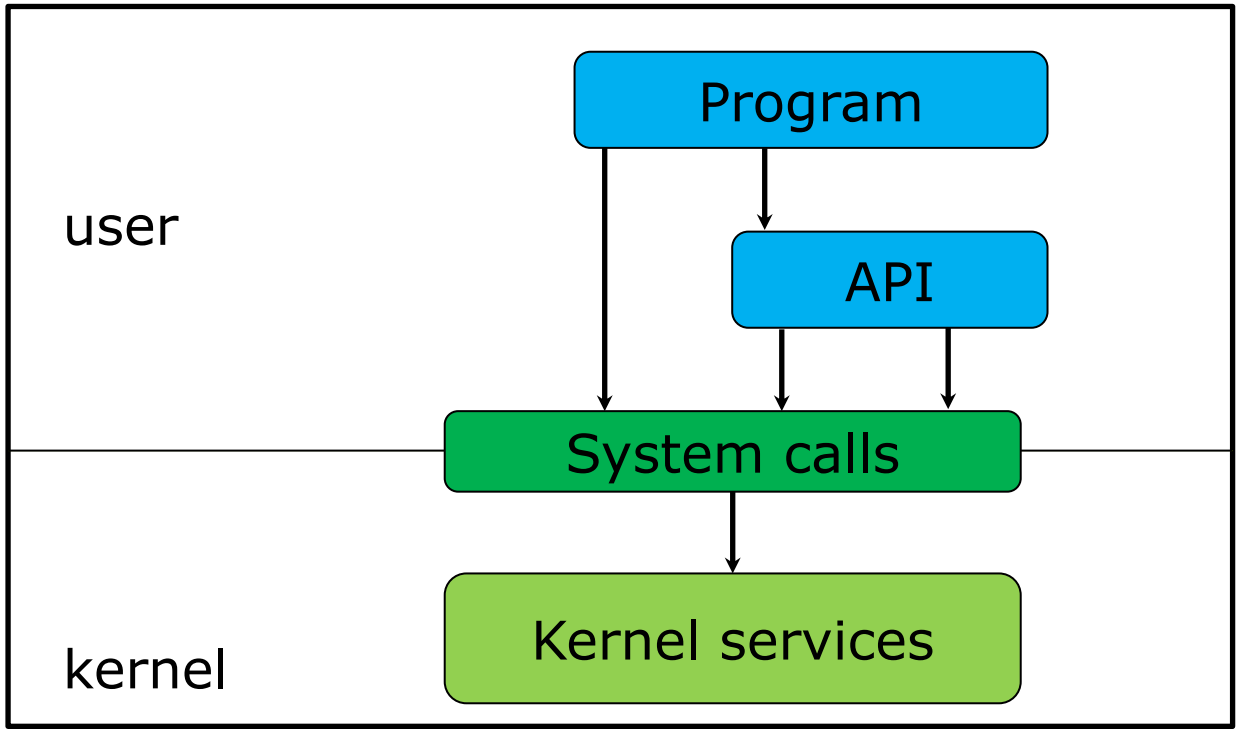
System Calls

- Programming interface to the services provided by the OS.



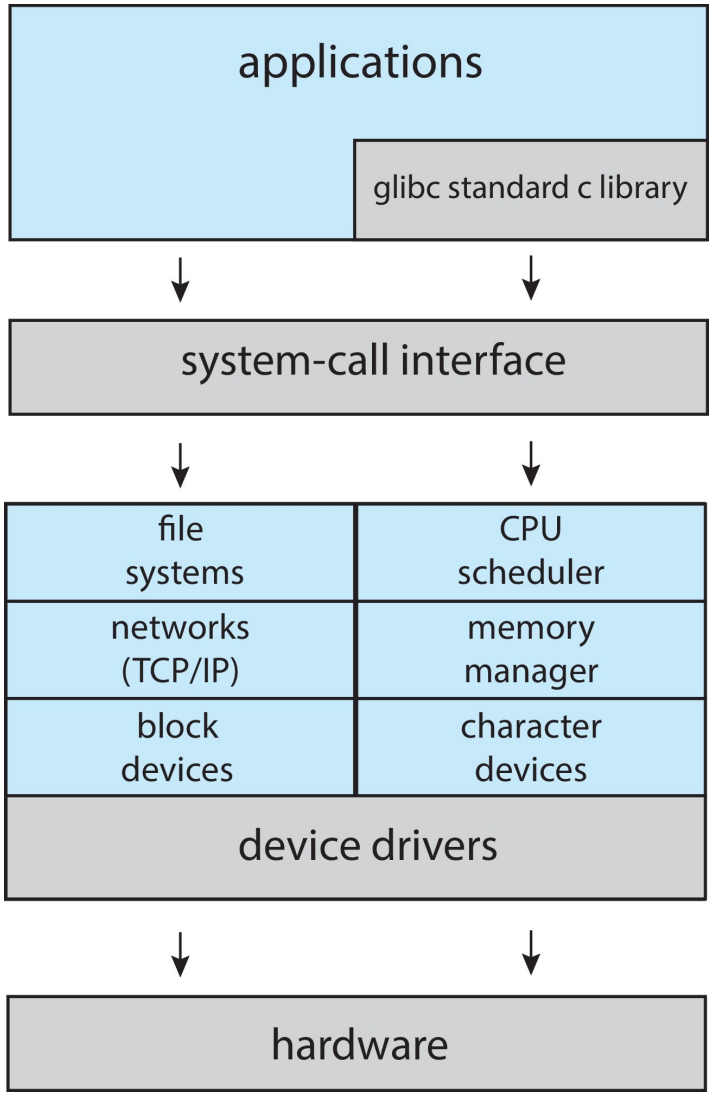
System Calls (cont.)

- Typically written in a high-level language (C or C++ or Assembly).
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use.



Three most common APIs

- Win32 API for Windows (Win API)
- POSIX API for POSIX-based systems
  - Including virtually all versions of UNIX, Linux (***unistd.h***), and Mac OS X
- Java API for the Java virtual machine.



Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

Example of Standard API (cont.)

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

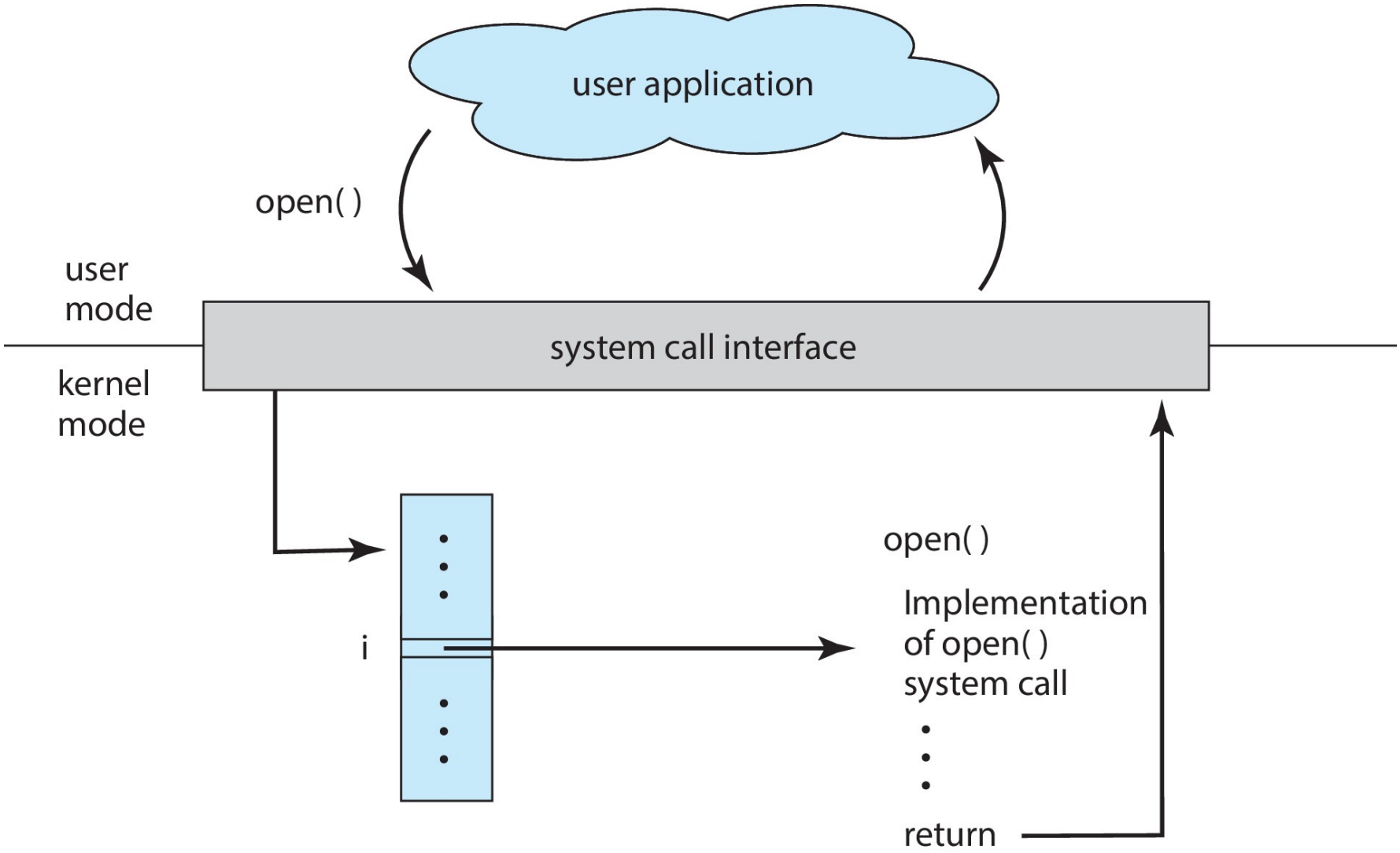
System Call Implementation

- Typically, a number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers.
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values

System Call Implementation (cont.)

- 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

API – System Call – OS Relationship



System calls in assembly programs (demo)

- Put the system call number in the ***EAX register***.
- Store the arguments to the system call in the registers EBX, ECX,...
- Call the relevant interrupt (***80h***).
- The result is ***usually*** returned in the ***EAX*** register.

https://www.tutorialspoint.com/assembly_programming/assembly_system_calls.htm

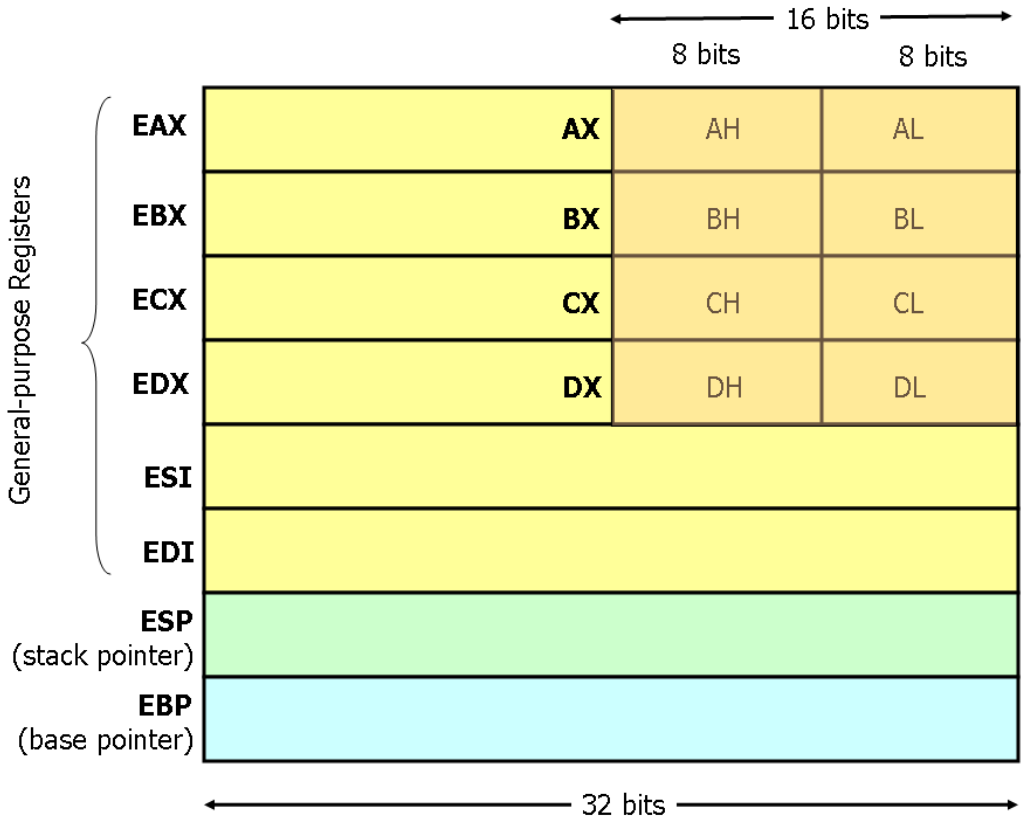
http://faculty.nps.edu/cseagle/assembly/sys_call.html

System Call Parameter Passing

- **Parameter Passing**
  - Register
  - Register pointer to mem. block
  - Stack (Push, Pop)
- Often, more information is required than simply identity of desired system call.
- Exact type and amount of information vary according to OS and call.

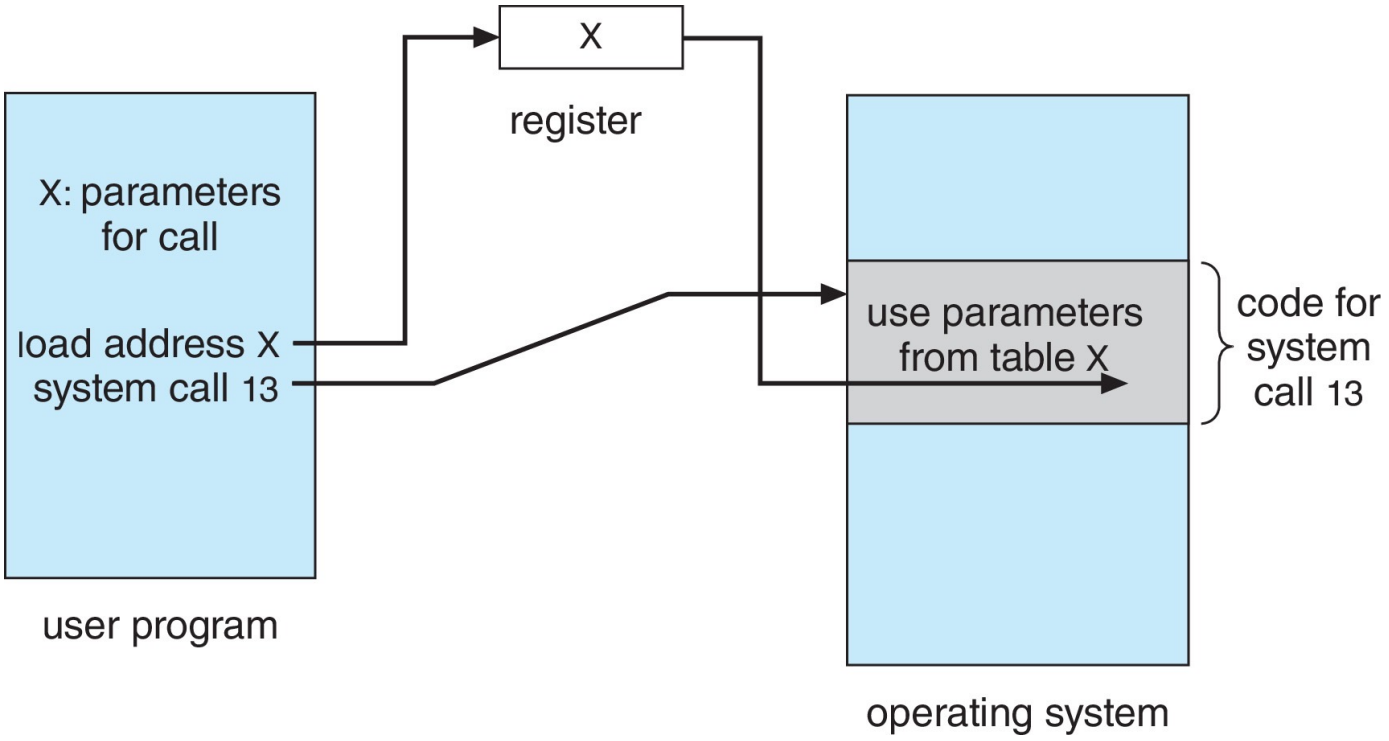
System Call Parameter Passing--Methods

- **Simplest**: pass the parameters in registers.
  - In some cases, may be more parameters than registers.



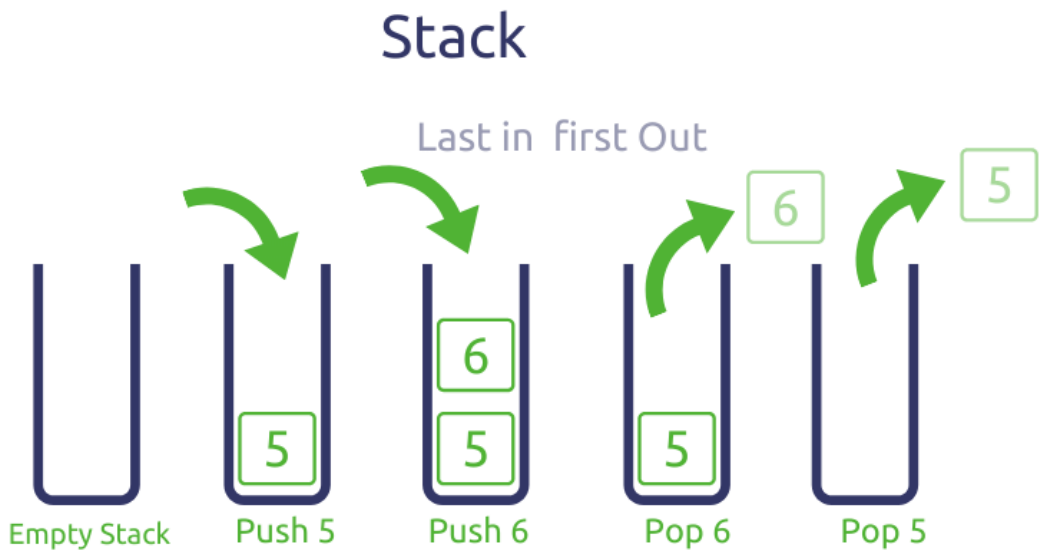
System Call Parameter Passing—Methods (cont.)

- 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.**



System Call Parameter Passing—Methods (cont.)

- Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system.



System Call Parameter Passing—Methods (cont.)

Methods\features	Is there a limitation on the number of parameters?	Is there a limitation on the length of the parameters?
Register		
Register pointer to mem. block		
Stack (Push, Pop)		

System Call Parameter Passing—Methods (cont.)

Methods\features	Is there a limitation on the number of parameters?	Is there a limitation on the length of the parameters?
Register	YES	YES
Register pointer to mem. block	NO	NO
Stack (Push, Pop)	NO	NO

Types of System Calls

- **Process control**
  - Create process, terminate process
  - …
- **File management**
  - create file, delete file
  - …
- **Device management**
  - request device, release device
  - …
- **Please study the reference book for more details**

Types of System Calls (Cont.)

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Why Applications are Operating System Specific

- Apps compiled on one system usually not executable on other OSs.
- Each OS provides its own unique system calls
  - Own file formats, etc.
- Apps can be multi-operating system
  - Written in interpreted language like Python, Ruby, and interpreter available on multiple OSs.
  - App written in language that includes a VM containing the running app (like Java).
  - Use standard language (like C), compile separately on each operating system to run on each.