Operating System Security

Mohammad Shahadat Hossain

Operating System Concepts

- An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

- An operating system is software that enables applications to interact with a computer's hardware. The software that contains the core components of the operating system is called the kernel.

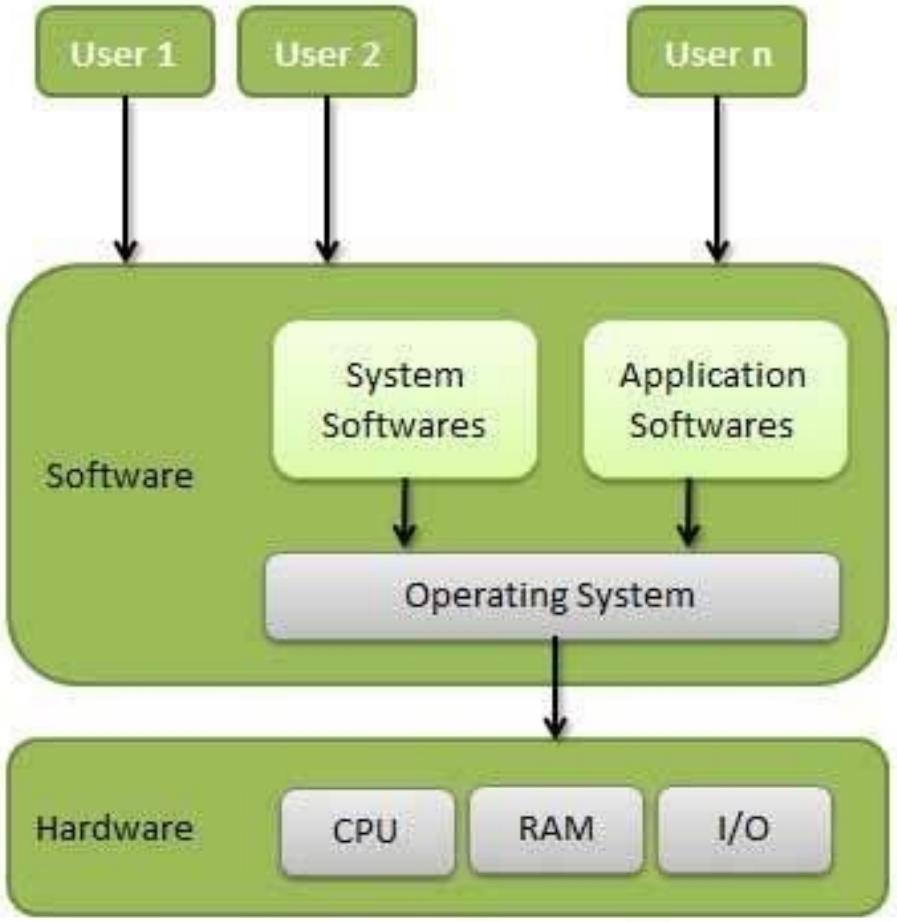
- The primary purposes of an Operating System are to enable applications (spftwares) to interact with a computer's hardware and to manage a system's hardware and software resources.

Operating System Definition and Basic Architecture

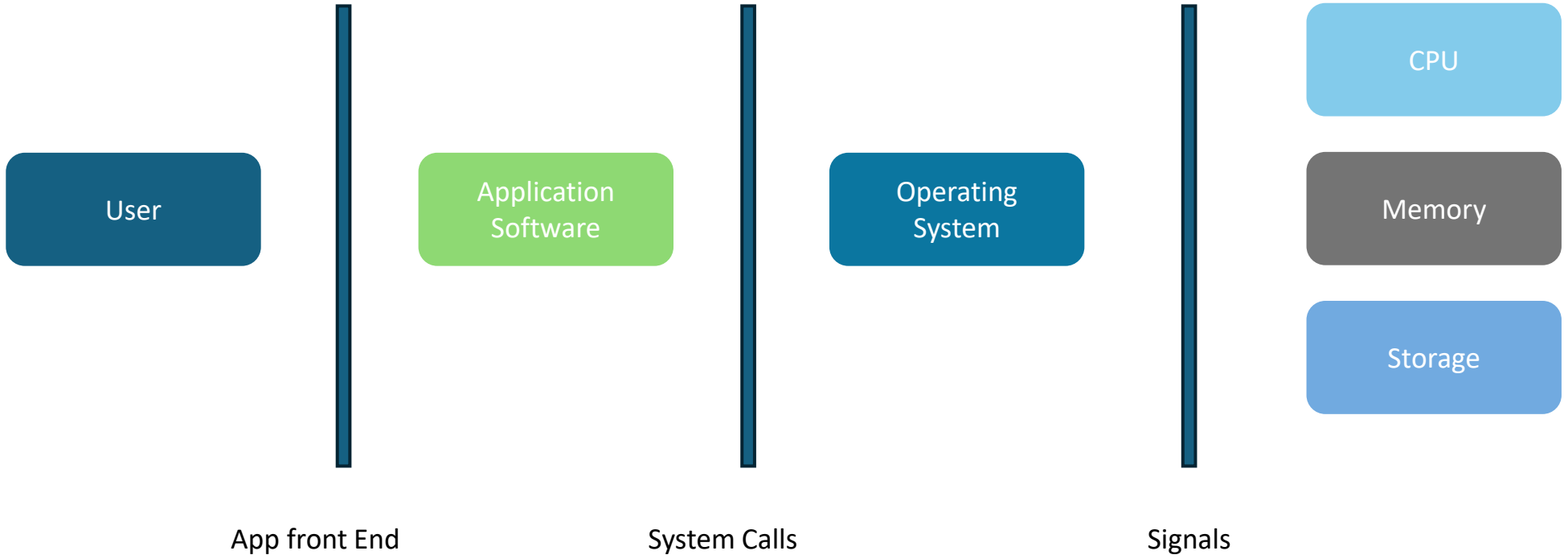
- An Operating System is the low-level software that supports a computer's basic functions, such as scheduling tasks and controlling peripherals.

- An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

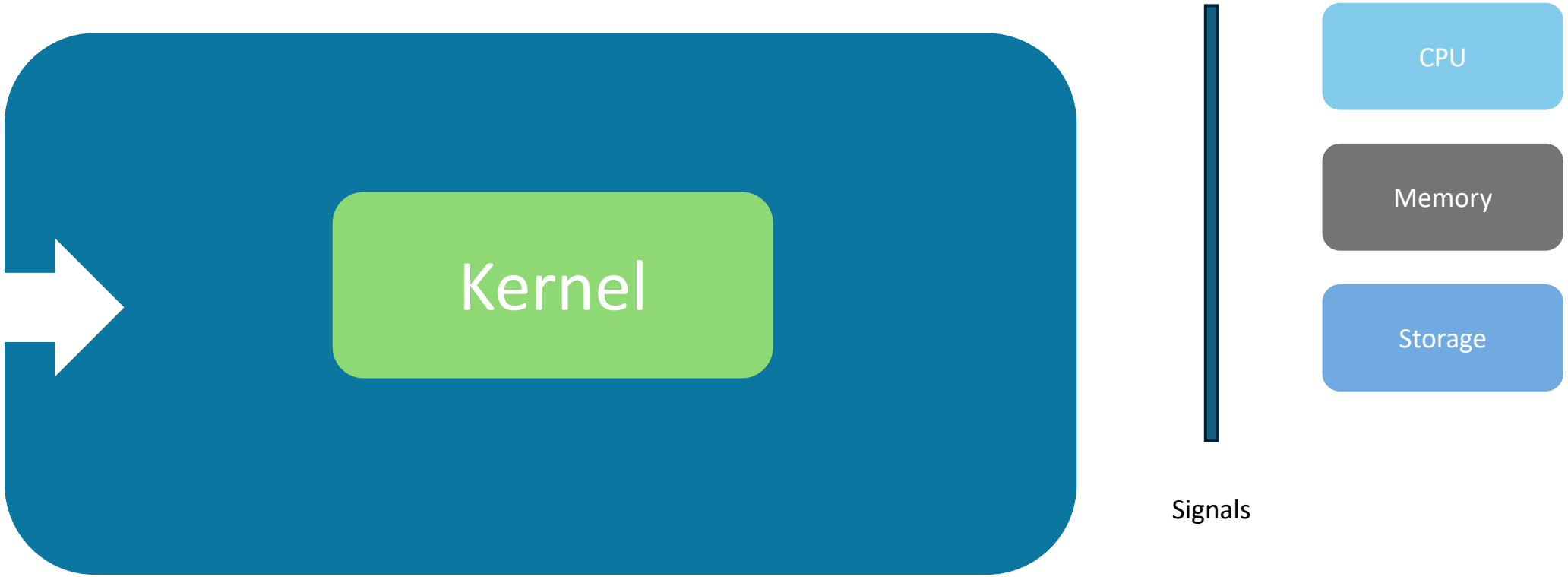
- An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.



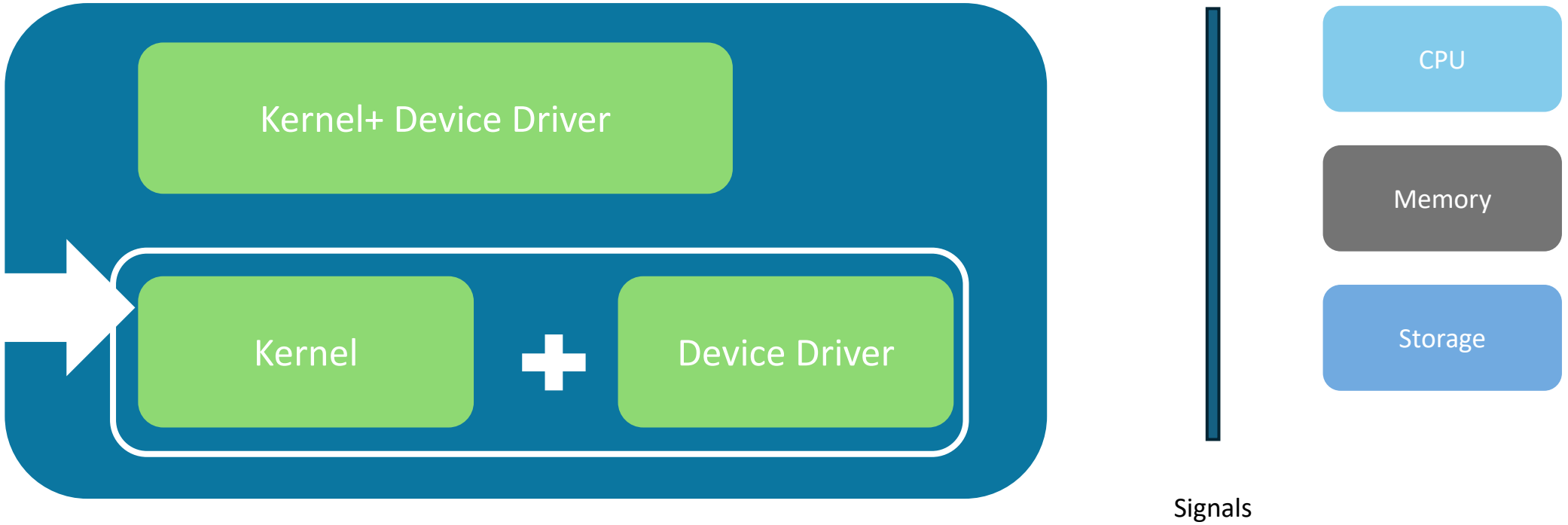
Operating System Definition and Basic Architecture



Operating System Definition and Basic Architecture



Operating System Definition and Basic Architecture

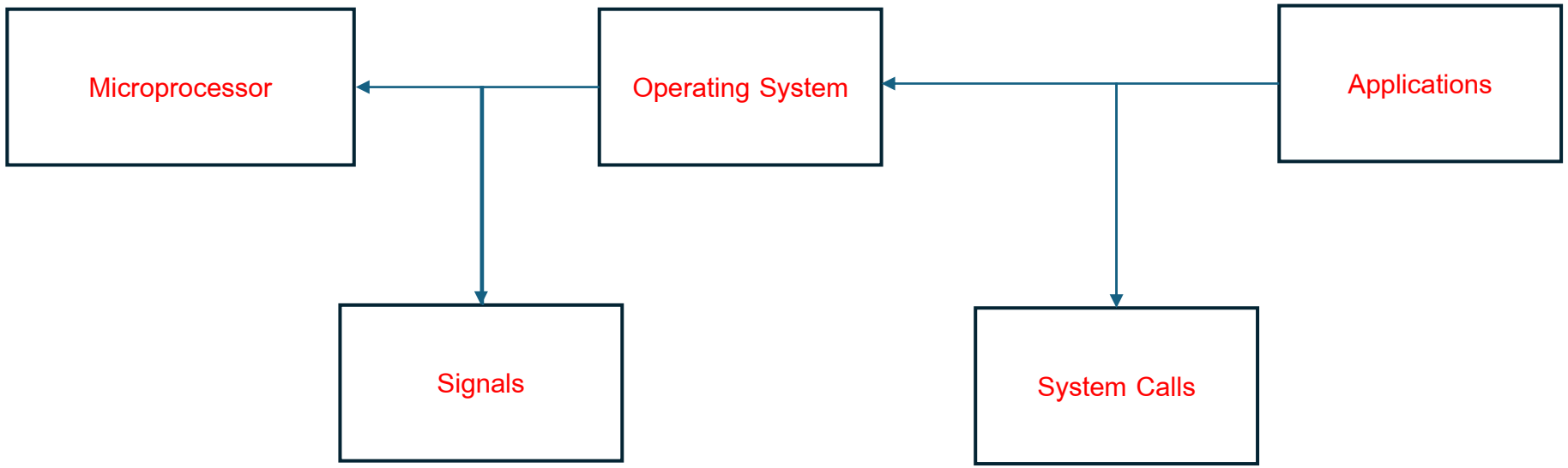


Operating System Components and Services

- Process Management
- Main Memory Management
- File System Management
- I/O System Management
- Storage Management
- Networking
- Security System

- Kernel
- Command Interpreter
- System Calls
- Signals

Operating System Components and Services



Process Management (Scheduling)

A program in running state is called a process. A process is program or a fraction of a program that is loaded in main memory. A process needs certain resources including CPU time, Memory, Files, and I/O devices to accomplish its task. The process management component manages the multiple processes running simultaneously on the Operating System.

Process Management is responsible for the following activities:-

- Create, load, execute, suspend, resume, and terminate processes.
- Switch system among multiple processes in main memory.
- Provides communication mechanisms so that processes can communicate with each others
- Provides synchronization mechanisms to control concurrent access to shared data to keep shared data consistent.
- Allocate/de-allocate resources properly to prevent or avoid deadlock situation.

A process is a program in execution. It consists of the followings:

- Executable program
- Program's data
- Stack and stack pointer
- Program counter and other CPU registers
- Details of opened files

Memory Management

Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.

Main memory is a volatile storage device which means it loses its contents in the case of system failure or as soon as system power goes down.

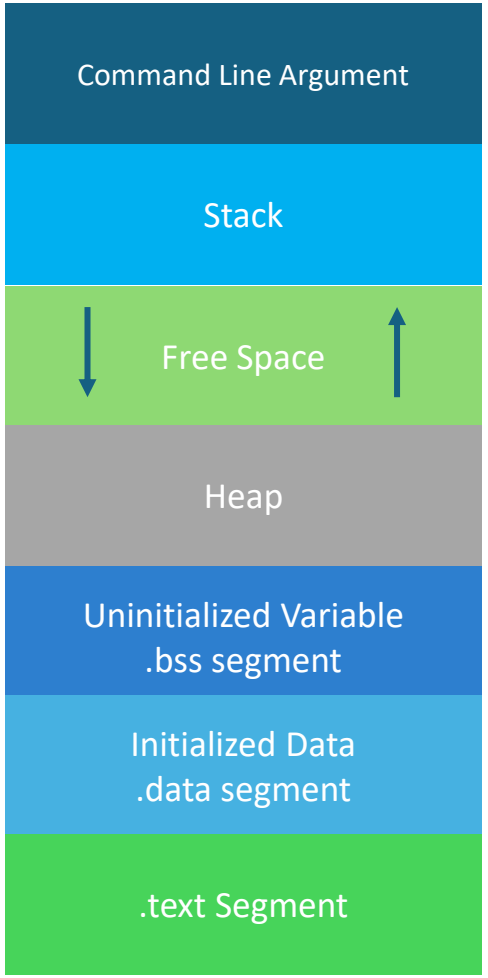
The operating system is responsible for the following activities in connections with memory management:

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes to load when memory space becomes available.
- Allocate and deallocate memory space as needed.

Memory layout for a process

A typical memory representation of C program consists of the following sections.

1. Text segment
2. Initialized data segment
3. Uninitialized data segment
4. Heap
5. Free Space
6. Stack
7. Command Line Argument



High Memory address

Low Memory address

Memory layout for a process

Text Segment

A text segment , also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.

As a memory region, a text segment may be placed below the heap or stack to prevent heaps and stack overflows from overwriting it.

Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on. Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

Memory layout for a process

Initialized Data Segment

Initialized data segment, usually called simply the Data Segment. A data segment is a portion of virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

Note that, data segment is not read-only, since the values of the variables can be altered at run time.

This segment can be further classified into initialized read-only area and initialized read-write area.

Memory layout for a process

Uninitialized Data Segment

Uninitialized data segment, often called the “bss” (Block Started by Symbol) segment, named after an ancient assembler operator that stood for “block started by symbol.” Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing

uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

Memory layout for a process

Heap

Heap is the segment where dynamic memory allocation usually takes place.

The heap area begins at the end of the BSS segment and grows to larger addresses from there. The Heap area is managed by malloc, realloc, and free system calls.

The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

Memory layout for a process

Stack

The stack area traditionally adjoined the heap area and grew the opposite direction; when the stack pointer met the heap pointer, free memory was exhausted. (With modern large address spaces and virtual memory techniques they may be placed almost anywhere, but they still typically grow opposite directions.)

Memory layout for a process

View the randomization setting

$cat /proc/sys/kernel/randomize_va_space

2

View Address Space

$ldd /bin/sleep

```
┌──(kali㉿kali)-[~/Desktop]
└─$ ldd /bin/sleep
        linux-vdso.so.1 (0×00007fffecbf6000)
        libc.so.6 ⇒ /lib/x86_64-linux-gnu/libc.so.6 (0×00007f94736dd000)
        /lib64/ld-linux-x86-64.so.2 (0×00007f94738e1000)

┌──(kali㉿kali)-[~/Desktop]
└─$ ldd /bin/sleep
        linux-vdso.so.1 (0×00007ffffcbbd000)
        libc.so.6 ⇒ /lib/x86_64-linux-gnu/libc.so.6 (0×00007fa720180000)
        /lib64/ld-linux-x86-64.so.2 (0×00007fa720384000)
```

Memory layout for a process

Change the randomization setting

echo 0 > /proc/sys/kernel/randomize_va_space

OR

sysctl -w kernel.randomize_va_space=0

```
┌──(root㉿kali)-[~]
└─# echo 0 > /proc/sys/kernel/randomize_va_space

┌──(root㉿kali)-[~]
└─# cat /proc/sys/kernel/randomize_va_space
0
```

View Address Space

$ldd /bin/sleep

```
┌──(root㉿kali)-[~]
└─# ldd /bin/sleep
        linux-vdso.so.1 (0x00007ffff7fc9000)
        libc.so.6 ⇒ /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7dc1000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ffff7fcb000)

┌──(root㉿kali)-[~]
└─# ldd /bin/sleep
        linux-vdso.so.1 (0x00007ffff7fc9000)
        libc.so.6 ⇒ /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7dc1000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ffff7fcb000)
```

Computer Memory Layout

Example

The size command reports the sizes (in bytes) of the text, data, and bss segments. Check the following simple C program

```
#include <stdio.h>
 int main(void)
{
   return 0;
}
```

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# size program1
   text    data     bss     dec     hex filename
   1216     528       8    1752     6d8 program1

┌──(root㉿kali)-[/home/kali/Desktop]
└─#
```

The size command reports the sizes (in bytes) of the text, data, and bss segments. Check the following simple C program

```
#include <stdio.h>
int global1; /* Uninitialized variable stored in bss*/
int global2;
int global3;
 int main(void)
{
   return 0;
}
```

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# size program2
   text    data     bss     dec     hex filename
   1216     528      24    1768     6e8 program2
```

Computer Memory Layout

Example

Let us add one static variable which is also stored in bss.

```
#include <stdio.h>
int global; /* Uninitialized variable stored in bss*/
int main(void)
{
   static int i; /* Uninitialized static variable stored in bss */
   return 0;
}
```

```
[shahadat@CentOS]$ gcc memory-layout.c -o memory-layout

[shahadat@CentOS]$ size memory-layout
 text      data       bss       dec       hex   filename
 960        248        16      1224       4c8   memory-layout
```

Computer Memory Layout

Example

Let us initialize the static variable which will then be stored in Data Segment (DS)

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/

int main(void)
{
   static int i = 100; /* Initialized static variable stored in DS*/
   return 0;
}
```

```
[shahadat@CentOS]$ gcc memory-layout.c -o memory-layout

[shahadat@CentOS]$ size memory-layout
text      data       bss       dec       hex    filename
960        252        12      1224       4c8    memory-layout
```

End of today’s class

File and File System Management

The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be in main memory during execution. Since the main memory is too small to permanently accommodate all data and program, the computer system must provide secondary storage to backup main memory.

Most modern computer systems use disks as the principle on-line storage medium, for both programs and data. Most programs, like compilers, assemblers, sort routines, editors, formatters, and so on, are stored on the disk until loaded into memory, and then use the disk as both the source and destination of their processing.

The operating system is responsible for the following activities in connection with disk management:

- Free Space Management
- Storage allocation
- Disk scheduling

- File Creation
- File Deletion
- Read and Write operations

The System Calls

System Calls :

System calls provide an interface to the services made by an operating system. The user interacts with the operating system programs through System calls. These calls are normally made available as library functions in high-level languages such as C, Java, Python etc. It provides a level of abstraction as the user is not aware of the implementation or execution of the call made. Details of the operating system is hidden from the user. Different hardware and software services can be availed through system calls.

System calls are available for the following operations:

- Process Management
- Memory Management
- File Operations
- Input / Output Operations

```
; Written by Shahadat
; Date: 16-Jan-2023

global _start:

section .text:
_start:
    mov eax, 0x04
    mov ebx, 0x01
    mov ecx, message
    mov edx, message_length
    int 0x80
; Gracefully Exit
    mov eax, 0x01
    mov ebx, 1
    int 0x80

section .data:
    message: db "Hello World!", 0xA
    message_length equ $-message
```

System Call

The System Calls

In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system's kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Services Provided by System Calls :

1.Process creation and management
2.Main memory management
3.File Access, Directory and File system management
4.Device handling(I/O)
5.Protection
6.Networking, etc.

The System Calls

Types of System Calls : There are 5 different categories of system calls –

1. **Process control:** end, abort, create, terminate, allocate and free memory.
2. **File management:** create, open, close, delete, read file etc.
3. Device management
4. Information maintenance
5. Communication

The Signals or Interrupts

An interrupt is a signal emitted by a device attached to a computer or from a program within the computer. It requires the operating system (OS) to stop and figure out what to do next. An interrupt temporarily stops or terminates a service or a current process.

Types of interrupts
Interrupts are classified into two types:

Hardware interrupt
A hardware interrupt is an electronic signal from an external hardware device that indicates it needs attention from the OS. One example of this is moving a mouse or pressing a keyboard key. In these examples of interrupts, the processor must stop to read the mouse position or keystroke at that instant.

In this type of interrupt, all devices are connected to the Interrupt Request Line (IRL). Typically, a hardware IRQ has a value that associates it with a particular device. This makes it possible for the processor to determine which device is requesting service by raising the IRQ, and then provide service accordingly.

The Signals or Interrupts

Software interrupts

A software interrupt occurs when an application program terminates or requests certain services from the OS. Usually, the processor requests a software interrupt when certain conditions are met by executing a special instruction. This instruction invokes the interrupt and functions like a subroutine call. Software interrupts are commonly used when the system interacts with device drivers or when a program requests OS services.

In some cases, software interrupts may be triggered unexpectedly by program execution errors rather than by design. These interrupts are known as exceptions or traps.

```
; Written by Shahadat
; Date: 16-Jan-2023

global _start:

section .text:
_start:
        mov eax, 0x04
        mov ebx, 0x01
        mov ecx, message
        mov edx, message_length
        int 0x80
; Gracefully Exit
        mov eax, 0x01
        mov ebx, 1
        int 0x80

section .data:
        message: db "Hello World!", 0xA
        message_length equ $-message
```

Software Interrupt

The File Systems and Permissions

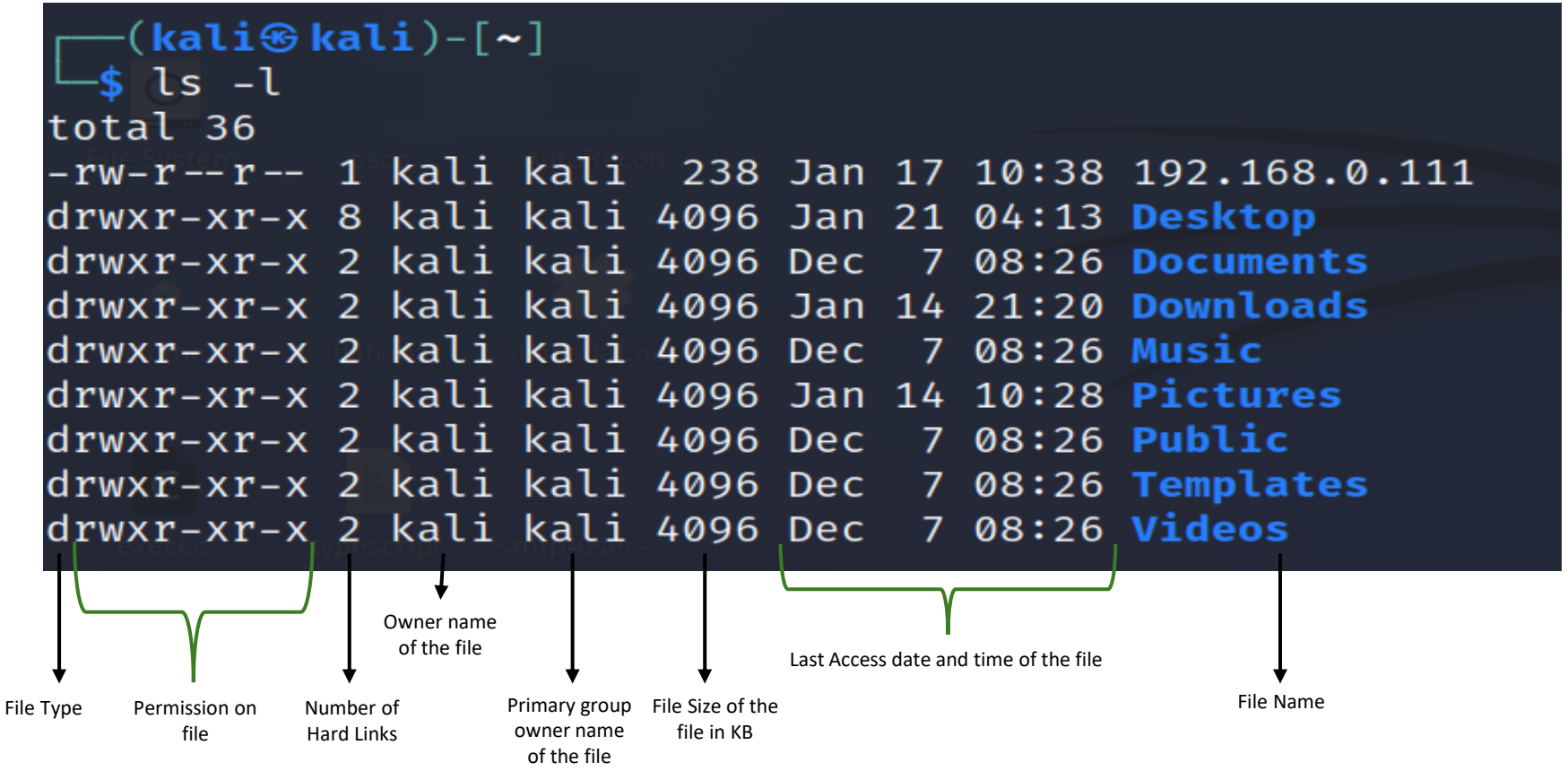
```
┌──(kali㉿kali)-[/]
└─$ ls
0    boot  etc   initrd.img      lib    lib64   lost+found  mnt  proc  run   srv       sys  usr  vmlinuz
bin  dev   home  initrd.img.old  lib32  libx32  media       opt  root  sbin  swapfile  tmp  var  vmlinuz.old

┌──(kali㉿kali)-[/]
└─$
```

Volume	Layout	Type	File System	Status	Capacity	Free Sp...	% Free
(E:)	Simple	Basic	NTFS (BitLocker Encrypted)	Healthy (Basic Data Partition)	187.50 GB	60.42 GB	32 %
(Disk 0 partition 1)	Simple	Basic		Healthy (EFI System Partition)	260 MB	260 MB	100 %
(Disk 0 partition 6)	Simple	Basic		Healthy (Recovery Partition)	1.17 GB	1.17 GB	100 %
New Volume (D:)	Simple	Basic	NTFS (BitLocker Encrypted)	Healthy (Basic Data Partition)	187.50 GB	63.78 GB	34 %
OS (C:)	Simple	Basic	NTFS (BitLocker Encrypted)	Healthy (Boot, Page File, Crash ...	100.50 GB	12.15 GB	12 %

Disk 0					
Basic 476.92 GB Online	260 MB Healthy (EFI S	**OS (C:)** 100.50 GB NTFS (BitLocker Encr Healthy (Boot, Page File, Crash	**New Volume (D:)** 187.50 GB NTFS (BitLocker Encryp Healthy (Basic Data Partition)	**(E:)** 187.50 GB NTFS (BitLocker Encryp Healthy (Basic Data Partition)	1.17 GB Healthy (Recovery

The File Systems and Permissions



Focus Points

File Type

Permissions

Links

The File Types

File Type	Command to create the File	Located in	Indicator	FILE command output
Regular FIle	touch	Any directory	–	Image data, ASCII Text, RAR archive data, etc
Directory File	mkdir	It is a directory	d	Directory
Block Files	fdisk	/dev	b	Block special
Character Files	mknod	/dev	c	Character special
Pipe Files	mkfifo	/dev	p	FIFO
Symbol Link Files	ln	/dev	l	Symbol link to <linkname>
Socket Files	socket() sys call OR exec	/dev	s	Socket

The File Types (Lab)

```
mkfifo /tmp/f;cat /tmp/f|/bin/bash -i 2>&1|nc -nvlp 4444 >/tmp/f
```

The File Permission

What are Linux File Permissions?

In Linux, file permissions, attributes, and ownership control the access level that the system processes and users have to files. This ensures that only authorized users and processes can access specific files and directories. This is an important part of maintaining and securing your systems.

Linux is a “top-down” environment. This means that if you deny “**group**” or “**other**” permissions to a directory, all subdirectories and files within that directory will be denied the permissions established at the directory level. The minimum permission for access to a directory is executed (x).

When Linux file permissions are represented by numbers, it's called numeric mode. In numeric mode, a three-digit value represents specific file permissions (for example, 744.) These are called octal values. The first digit is for owner permissions, the second digit is for group permissions, and the third is for other users. Each permission has a numeric value assigned to it:

- r (read): 4
- w (write): 2
- x (execute): 1

The File Permission

Linux file permissions

Read (r)
Read permission is used to access the file's contents. You can use a tool like cat or less on the file to display the file contents. You could also use a text editor like Vi or view on the file to display the contents of the file. Read permission is required to make copies of a file, because you need to access the file's contents to make a duplicate of it.

Write (w)
Write permission allows you to modify or change the contents of a file. Write permission also allows you to use the redirect or append operators in the shell (> or >>) to change the contents of a file. Without write permission, changes to the file's contents are not permitted.

Execute (x)
Execute permission allows you to execute the contents of a file. Typically, executables would be things like commands or compiled binary applications. However, execute permission also allows someone to run Bash shell scripts, Python programs, and a variety of interpreted languages.

The Directory Type File Permission

Linux directory type file permissions

Read (r)
Like regular files, this permission allows you to read the contents of the directory. However, that means that you can view the contents (or files) stored within the directory. This permission is required to have things like the ls command work.

Write (w)
As with regular files, this allows someone to modify the contents of the directory. When you are changing the contents of the directory, you are either adding files to the directory or removing files from the directory. As such, you must have write permission on a directory to move (mv) or remove (rm) files from it. You also need write permission to create new files (using touch or a file-redirect operator) or copy (cp) files into the directory.

Execute (x)
This permission is very different on directories compared to files. Essentially, you can think of it as providing access to the directory. Having execute permission on a directory authorizes you to look at extended information on files in the directory (using ls -l, for instance) but also allows you to change your working directory (using cd) or pass through this directory on your way to a subdirectory underneath.

Lacking execute permission on a directory can limit the other permissions in interesting ways. For example, how can you add a new file to a directory (by leveraging the write permission) if you can't access the directory's metadata to store the information for a new, additional file? You cannot. It is for this reason that directory-type files generally offer execute permission to one or more of the user owner, group owner, or others.

Class Test-1 Date: Feb 3, 2023

Answer any 4 questions| Each questions carry 5 marks | Total Marks 20

1. Explain the R W X permissions in a file and a Directory
2. Explain the Memory Layout when a process loaded into a memory region
3. How many types of files are there in a Linux System and what are there purpose ?
4. Explain the purpose of a syscall and a system Interrupt
5. Explain each column of the following output

```
┌──(kali㉿kali)-[/run]
└─$ ls -l initctl
prw------- 1 root root 0 Feb  3 04:15 initctl
```

Thank You All