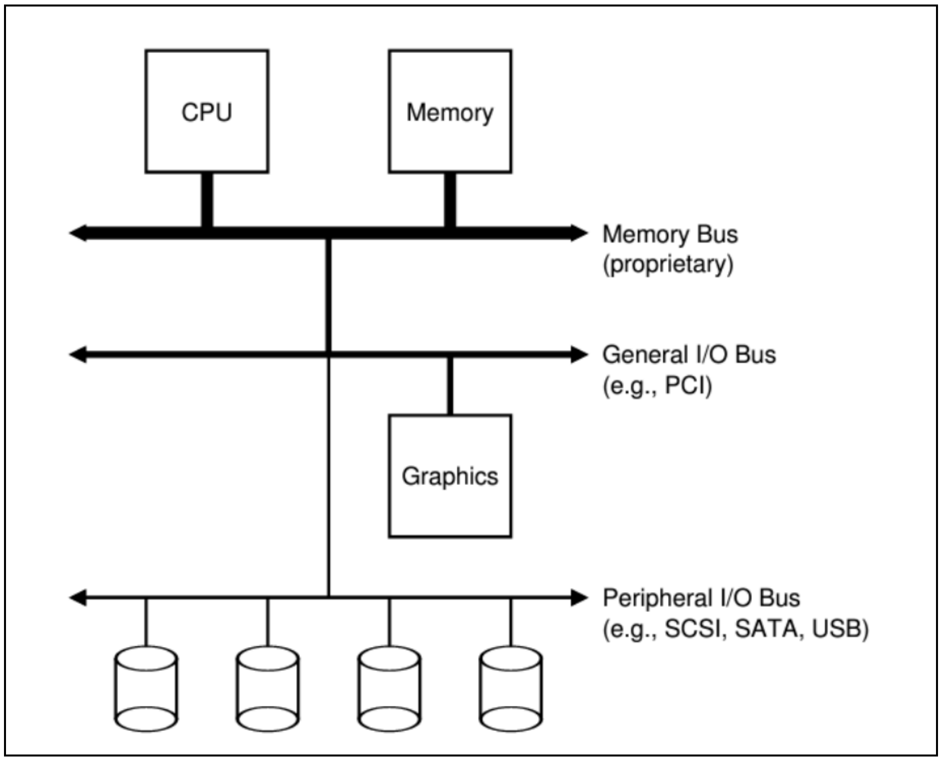
FINAL ASSIGNMENT [8] - Amina Srna

Task1: Explain the following diagram:



The diagram provided is a block diagram representing the architecture of a computer system, showing the interconnection between the CPU, Memory, Graphics, and various I/O (Input/Output) devices through different buses.

The CPU (Central Processing Unit) is the brain of the computer where most calculations take place. It communicates with other components via different buses.

Memory typically refers to the system's RAM (Random Access Memory), which temporarily stores data and instructions that the CPU needs while performing tasks.

The memory bus is a high-speed bus that connects the CPU directly to the memory. It's proprietary, meaning it is designed specifically for high-speed data transfer between these two components.

The General I/O Bus, such as the PCI (Peripheral Component Interconnect), connects the CPU to various I/O devices. It is used for connecting internal components like network cards, sound cards, and other peripherals.

The Graphics component, usually referring to the GPU (Graphics Processing Unit), is responsible for rendering images, videos, and animations. It is connected via the General I/O Bus to the CPU and memory.

The Peripheral I/O Bus is used to connect external devices and storage. Examples include SCSI (Small Computer System Interface), SATA (Serial ATA), and USB (Universal Serial Bus). This bus handles the communication between the CPU and external peripherals like hard drives, SSDs, printers, and other external devices.

The diagram shows several cylindrical shapes at the bottom, representing storage devices (like hard drives or SSDs) connected through the Peripheral I/O Bus.

Task 2: What are SATA ports?

SATA (Serial ATA) ports are connectors on a computer's motherboard used to connect storage devices such as hard drives (HDDs), solid-state drives (SSDs), and optical drives (like DVD or Blu-ray drives). SATA ports use serial communication, which transmits data one bit at a time, as opposed to the older parallel ATA (PATA) standard that transmits multiple bits simultaneously. This allows for higher data transfer rates and more efficient data handling.

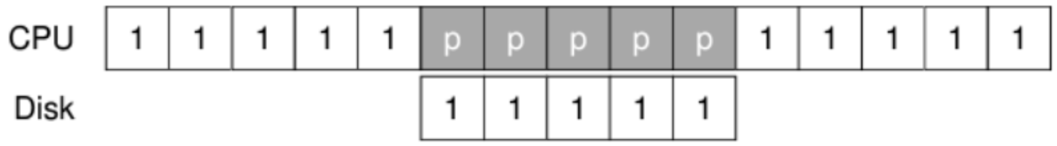
SATA supports hot swapping, which means you can connect or disconnect drives without shutting down the computer, provided the operating system and hardware support this feature. In addition to the data cables, SATA devices also require power, provided through a separate power connector from the power supply unit (PSU).

Task 3: What are PCI ports?

PCI (Peripheral Component Interconnect) ports are a type of hardware interface used to connect various internal components to a computer's motherboard. PCI is a standard for connecting peripheral devices to a computer's motherboard, introduced in the early 1990s to replace older standards like ISA and VESA.

PCI slots come in different sizes, with the most common being 32-bit and 64-bit slots. The 32-bit slots are shorter and have fewer pins, while the 64-bit slots are longer and support faster data transfer rates. PCI ports typically have a data transfer rate of up to 133 MB/s for the 32-bit version and up to 266 MB/s for the 64-bit version. These rates were sufficient for earlier peripheral devices but have been surpassed by newer standards like PCIe.

Task 4: The following communication between the CPU and the disk is performed without interrupts. Explain how this communication works.



The diagram illustrates a method of communication between the CPU and the disk that is performed without interrupts. Here's an explanation of how this communication works:

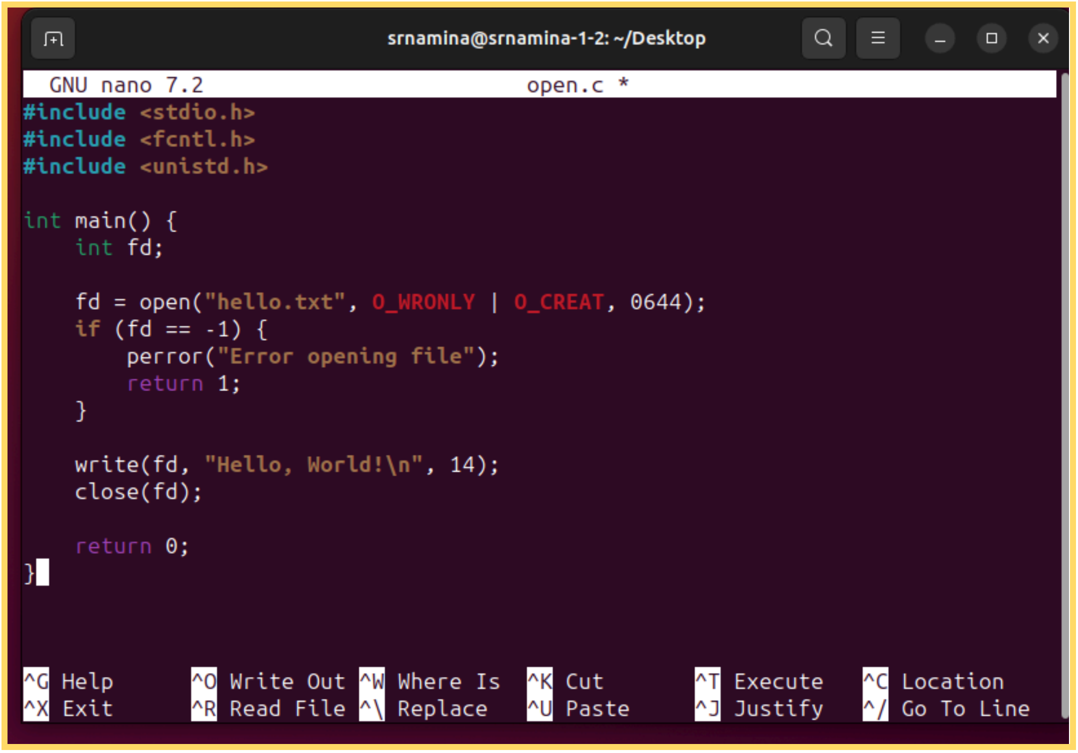
In the given diagram, we see a sequence of data units (represented as '1's) and placeholders (represented as 'p's) for both the CPU and the disk. The communication method shown is known as polling or busy-waiting. Polling is a method where the CPU continuously checks the status of an I/O device to see if it is ready for data transfer. In this scenario, the CPU continuously monitors the disk to determine when it is ready to receive or send data. The CPU does this by executing a loop that repeatedly checks the status of the disk.

In a communication method without interrupts, the CPU does not get interrupted by the disk when it is ready to transfer data. Instead, the CPU must regularly check the status of the disk. This can be less efficient compared to interrupt-driven I/O, where the CPU can perform other tasks and only be interrupted when the disk is ready. The '1's in the diagram likely represent data units that are ready to be transferred. The 'p's represent periods where the CPU is polling the disk, checking its status to see if it is ready for the next data unit. Once the disk signals that it is ready, the CPU sends or receives the next '1'.

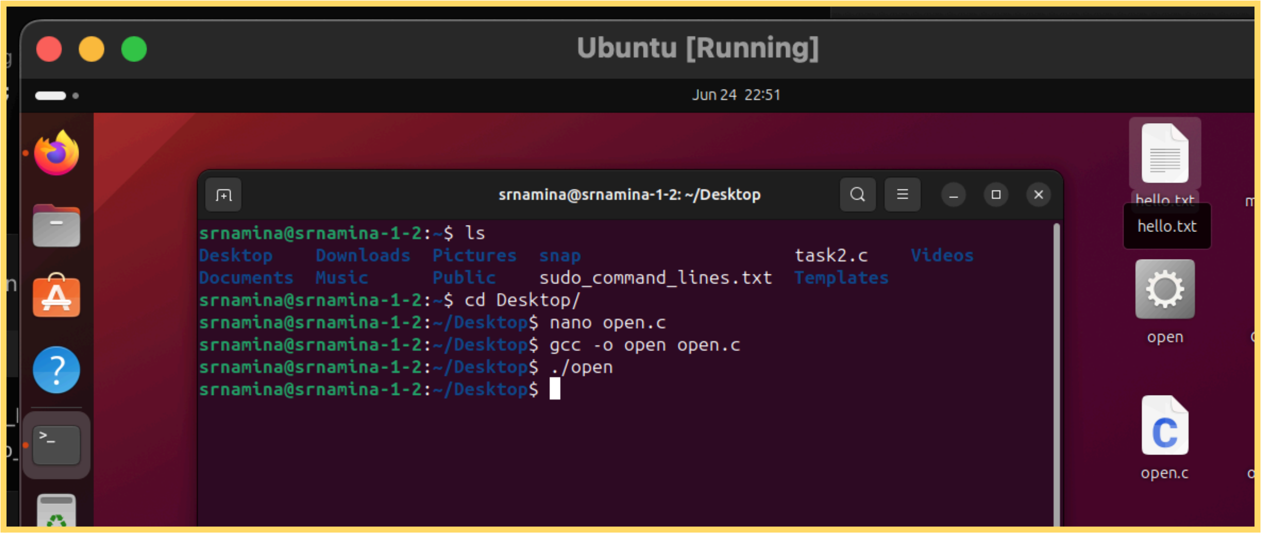
In this method, the CPU sends a '1' to the disk when it detects the disk is ready (during the polling phase 'p'). The disk processes the received data or prepares the next data unit for the CPU to read. The CPU continuously polls the disk (indicated by 'p') and transfers data when appropriate.

Task 5: Create a file with the name "hello.txt" by using open() and creat() system calls. Submit the code and explain the difference between the two system calls.

➔ Using open():

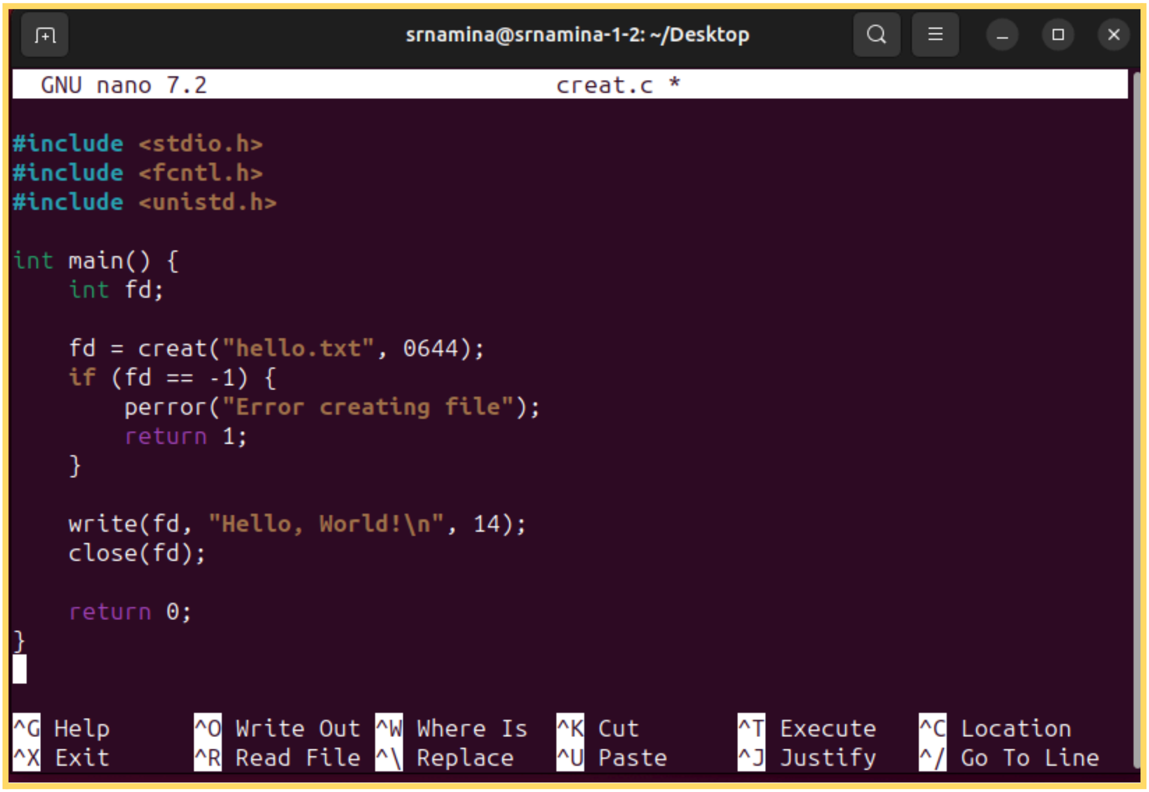


```
srnamina@srnamina-1-2: ~/Desktop
  GNU nano 7.2                             open.c *
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd;

    fd = open("hello.txt", O_WRONLY | O_CREAT, 0644);
    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }

    write(fd, "Hello, World!\n", 14);
    close(fd);

    return 0;
}

^G Help          ^O Write Out  ^W Where Is   ^K Cut         ^T Execute    ^C Location
^X Exit          ^R Read File  ^\ Replace    ^U Paste       ^J Justify    ^/ Go To Line
```

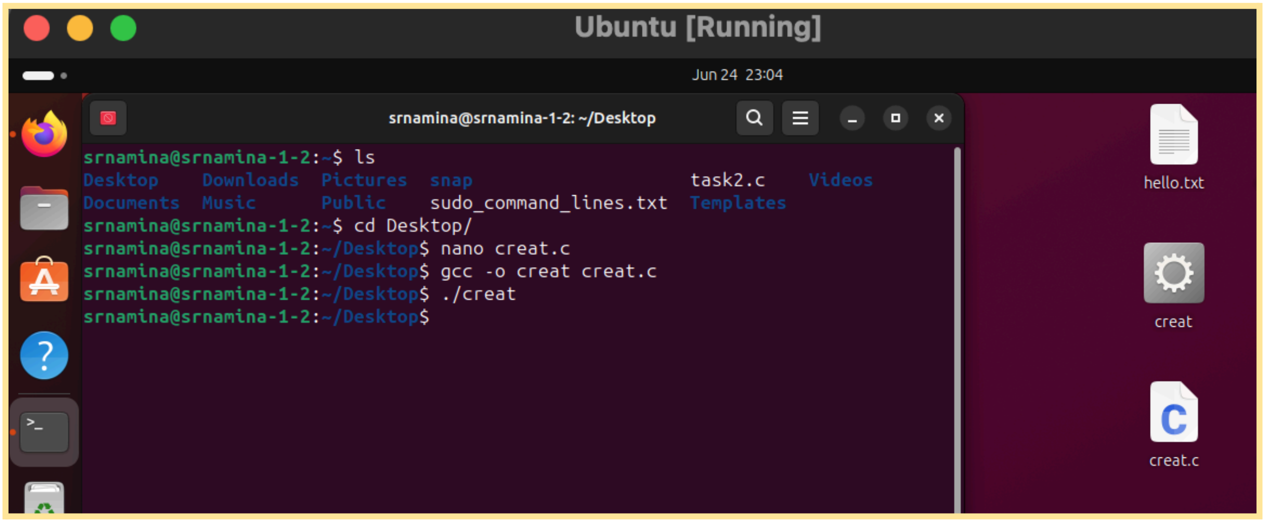


```
srnamina@srnamina-1-2: ~/Desktop
srnamina@srnamina-1-2:~$ ls
Desktop    Downloads  Pictures  snap                     task2.c    Videos
Documents  Music      Public    sudo_command_lines.txt  Templates
srnamina@srnamina-1-2:~$ cd Desktop/
srnamina@srnamina-1-2:~/Desktop$ nano open.c
srnamina@srnamina-1-2:~/Desktop$ gcc -o open open.c
srnamina@srnamina-1-2:~/Desktop$ ./open
srnamina@srnamina-1-2:~/Desktop$
```

➔ Using creat():



```
  GNU nano 7.2                       creat.c *

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd;

    fd = creat("hello.txt", 0644);
    if (fd == -1) {
        perror("Error creating file");
        return 1;
    }

    write(fd, "Hello, World!\n", 14);
    close(fd);

    return 0;
}

^G Help          ^O Write Out  ^W Where Is  ^K Cut          ^T Execute    ^C Location
^X Exit          ^R Read File  ^\ Replace   ^U Paste        ^J Justify    ^/ Go To Line
```



```
srnamina@srnamina-1-2:~$ ls
Desktop    Downloads  Pictures  snap                    task2.c    Videos
Documents  Music      Public    sudo_command_lines.txt  Templates
srnamina@srnamina-1-2:~$ cd Desktop/
srnamina@srnamina-1-2:~/Desktop$ nano creat.c
srnamina@srnamina-1-2:~/Desktop$ gcc -o creat creat.c
srnamina@srnamina-1-2:~/Desktop$ ./creat
srnamina@srnamina-1-2:~/Desktop$
```

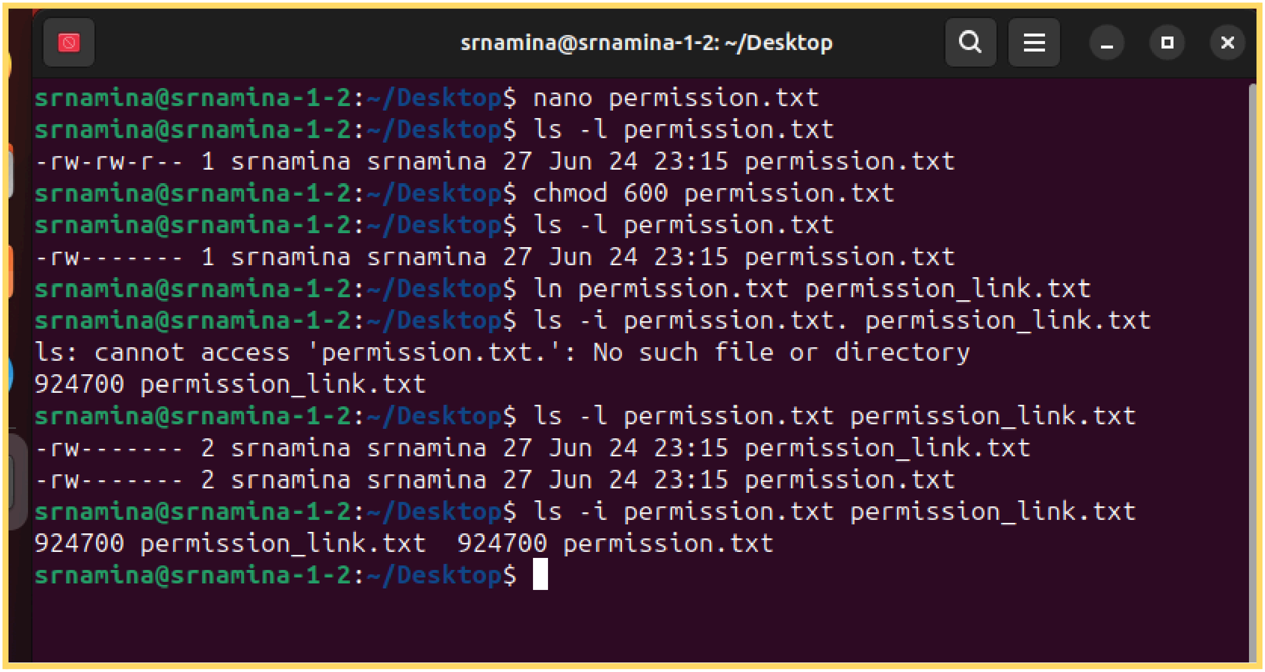
The open() system call is more versatile and can open files in various modes such as read, write, and append. It can also use various flags, like O_CREAT to create a file if it doesn't exist and O_EXCL to ensure it doesn't overwrite an existing file. open() can be used for opening, creating, and configuring file descriptors in many ways.

The creat() system call is simpler and specifically designed for creating a file (or truncating it if it exists) with write-only access. It is equivalent to using open() with the O_CREAT and O_WRONLY flags.

In practice, open() is more commonly used because it provides more flexibility. creat() is a more specialized function, effectively a shorthand for a common use case of open().

Task 6: Create a file with the name "permission.txt".

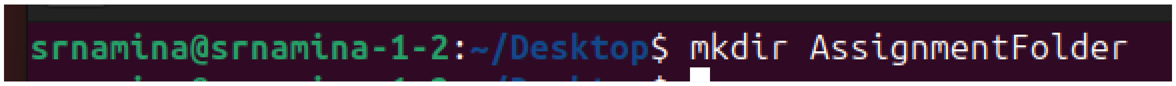
- Display the information about the file
- Change the permissions for the file so the administrator can only read and write the file.
- Create an new link to that file
- Display the inode numbers for the two links (check if they are the same or not, explain).



```
srnamina@srnamina-1-2:~/Desktop$ nano permission.txt
srnamina@srnamina-1-2:~/Desktop$ ls -l permission.txt
-rw-rw-r-- 1 srnamina srnamina 27 Jun 24 23:15 permission.txt
srnamina@srnamina-1-2:~/Desktop$ chmod 600 permission.txt
srnamina@srnamina-1-2:~/Desktop$ ls -l permission.txt
-rw------- 1 srnamina srnamina 27 Jun 24 23:15 permission.txt
srnamina@srnamina-1-2:~/Desktop$ ln permission.txt permission_link.txt
srnamina@srnamina-1-2:~/Desktop$ ls -i permission.txt. permission_link.txt
ls: cannot access 'permission.txt.': No such file or directory
924700 permission_link.txt
srnamina@srnamina-1-2:~/Desktop$ ls -l permission.txt permission_link.txt
-rw------- 2 srnamina srnamina 27 Jun 24 23:15 permission_link.txt
-rw------- 2 srnamina srnamina 27 Jun 24 23:15 permission.txt
srnamina@srnamina-1-2:~/Desktop$ ls -i permission.txt permission_link.txt
924700 permission_link.txt  924700 permission.txt
srnamina@srnamina-1-2:~/Desktop$ 
```

Task 7: How can we find the steps an operating system takes in order to execute a specific command? Find which steps are taken when we create a folder with the name of "AssignmentFolder".

1. **Write a command:**



```
srnamina@srnamina-1-2:~/Desktop$ mkdir AssignmentFolder
```

2. **Shell Process:**

   The shell interprets the command.
   The shell creates a child process using fork().
   The child process executes mkdir using execve().

3. **"Mkdir" Program:**

   The mkdir program calls the mkdir() system call.

4. **System Call "mkdir":**

   The kernel handles the `mkdir()` system call.
   It resolves the path, checks permissions, and interacts with the file system to create the directory.

5. **File System Interaction:**

   The file system driver allocates an inode for the new directory.
   Updates the parent directory to include the new directory entry.
   Updates relevant metadata (e.g., modification time).

6. **Completion:**

   The `mkdir` program exits.
   The shell displays a new prompt.

   By following these steps, the operating system successfully creates the directory AssignmentFolder. This process involves both user-level operations (handled by the shell and the mkdir program) and kernel-level operations (handled by system calls and the file system).

Task 8: Explain the difference between an absolute and a relative directory path.

An absolute and a relative directory path are two ways to specify the location of a file or directory in a file system. Here's a detailed explanation of the differences between them:

Absolute Path

An absolute path provides the complete path to a file or directory from the root of the file system. It is unambiguous and specifies the location of the file or directory regardless of the current working directory.

Key Characteristics:

- Always starts from the root directory, which is denoted by a / on Unix-like systems (e.g., Linux, macOS) or by a drive letter followed by a colon on Windows (e.g., C:\).
- It specifies the full path, including all directories from the root to the target file or directory.
- It is not dependent on the current working directory.

Relative Path

A relative path provides the location of a file or directory in relation to the current working directory. It does not start from the root directory but rather from the current directory you are working in.

Key Characteristics:

- Does not start with a root directory indicator (e.g., / or C:\).
- It specifies the path relative to the current working directory.
- It can include special symbols such as . (current directory) and .. (parent directory) to navigate the directory structure.

Absolute Path: Always begins from the root directory and provides the complete path to a file or directory. It is independent of the current working directory.

Relative Path: Begins from the current working directory and provides the path relative to it. It is dependent on where you are currently located within the directory structure.