Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology, Pune-37

(An Autonomous Institute of Savitribai Phule Pune University)

Department of AIDS

Division	A
Roll Number	78
PRN Number	12320056
Name	Mayank Kulkarni

Title: Write a program demonstrating use of different system calls.(In one program show the implementation of at least 4 system calls such as read(), write() ....)

Theory:

Basic Concepts of System Calls:

1. **System Calls:**
   - System calls are low-level functions used by a program to request services from the operating system (OS). They provide an interface between a running program and the OS, allowing the program to perform tasks like reading from or writing to a file, allocating memory, creating processes, and more.
   - In contrast to higher-level functions provided by libraries (like Python's built-in `open()` or `print()`), system calls directly interact with the kernel, which is the core part of the operating system.
2. **File Descriptors:**
   - A file descriptor is an integer that uniquely identifies an open file in a running process. When you open a file using a system call, the OS returns a file descriptor that can be used to perform subsequent operations (read, write, etc.) on that file.
3. **Key System Calls Used in the Program:**
   - **`open()`**: Opens a file and returns a file descriptor. It may create a file if it does not already exist (depending on the flags passed).
   - **`write()`**: Writes data to a file using the file descriptor. The data is written as a sequence of bytes.
   - **`lseek()`**: Moves the file descriptor's read/write pointer to a specific location within the file. This is useful for reading from or writing to a particular position in the file.
   - **`read()`**: Reads data from a file using the file descriptor. The data read is returned as a sequence of bytes.
   - **`close()`**: Closes an open file descriptor, freeing up system resources.

Program:

```python
import os

data = "Hello, this is a demonstration of system calls in Python."

fd = os.open("example.txt", os.O_RDWR | os.O_CREAT)

os.write(fd, data.encode())

os.lseek(fd, 0, os.SEEK_SET)

read_data = os.read(fd, len(data))
```

```
print("Data read from file:", read_data.decode())

os.close(fd)

print("System calls executed successfully!")
```

- **Data to be Written:**
  - The string `"Hello, this is a demonstration of system calls in Python."` is prepared to be written to a file. This data will later be read back from the file.
- **Opening the File:**
  - The `os.open()` system call is used to open a file named `example.txt`. The flags `os.O_RDWR` (open for reading and writing) and `os.O_CREAT` (create the file if it doesn't exist) are passed. The system returns a file descriptor (`fd`), which is used in subsequent operations.
  - **Why `os.open()`?**: This system call is a lower-level alternative to Python's built-in `open()` function, providing direct interaction with the OS.
- **Writing Data to the File:**
  - The `os.write()` system call writes the data to the file. The data must be encoded as bytes, which is done using `data.encode()`.
  - **Why `os.write()`?**: This call allows for writing raw bytes directly to a file, giving finer control over file I/O.
- **Repositioning the File Pointer:**
  - The `os.lseek()` system call moves the file pointer back to the beginning of the file (position 0). This is necessary because after writing to the file, the pointer is at the end, and we need it at the beginning to read the data back.
  - **Why `os.lseek()`?**: It provides precise control over where in the file subsequent read or write operations will occur.
- **Reading Data from the File:**
  - The `os.read()` system call reads the same number of bytes as the length of the original data from the file. The data read is decoded from bytes to a string using `decode()` and printed to verify that the write operation was successful.
  - **Why `os.read()`?**: Like `os.write()`, this call allows for low-level reading of raw bytes from the file.
- **Closing the File:**
  - The `os.close()` system call closes the file descriptor, releasing any system resources associated with it. This is a crucial step to prevent resource leaks in the system.

- **Why `os.close()`?**: It's important to close file descriptors when they are no longer needed to free up system resources.

Output:

```
[Running] python -u "d:\BTECH\VIT 3RD YEAR\VIT 3RD YEAR 1ST SEMESTER\STATISTICAL
Data read from file: Hello, this is a demonstration of system calls in Python.
System calls executed successfully!
```

Conclusion:

Thus, in this practical we have successfully performed and executed the different types of system calls.