Experiment No. 3

Title: File and Process handling using system calls

Batch: B-2 Roll No: 16010422234 Name: Chandana Galgali Date: 06/08/2024

Experiment No: 3

Aim: Implementation of basic commands in Linux and write a program to show file and process handling using system calls in Linux.

Resources needed: Ubuntu 15.04 GNU.

Theory:
In the realm of operating systems, Linux provides a rich set of system calls for managing files and processes. System calls serve as the fundamental interface between a process and the operating system, allowing user-space applications to request services such as file manipulation and process control from the kernel.

File Management System Calls:
open(): Open or create a file.
read(): Read data from a file.
write(): Write data to a file.
close(): Close an open file.
lseek(): Move the file pointer to a specified location.
stat(): Get file status.
fstat(): Get file status using a file descriptor.
chmod(): Change file permissions.
unlink(): Delete a file.
rename(): Rename a file.

Process Management System Calls:
fork(): Create a new process.
exec(): Replace the current process image with a new one.
wait(): Wait for a child process to change state.
exit(): Terminate a process.
getpid(): Get the process ID of the current process.
getppid(): Get the parent process ID.
kill(): Send a signal to a process.
sleep(): Suspend execution for an interval of time.
system(): Execute a shell command.
nice(): Change process priority.

Pre lab/Prior concepts:
Study the commands given.

Activity:

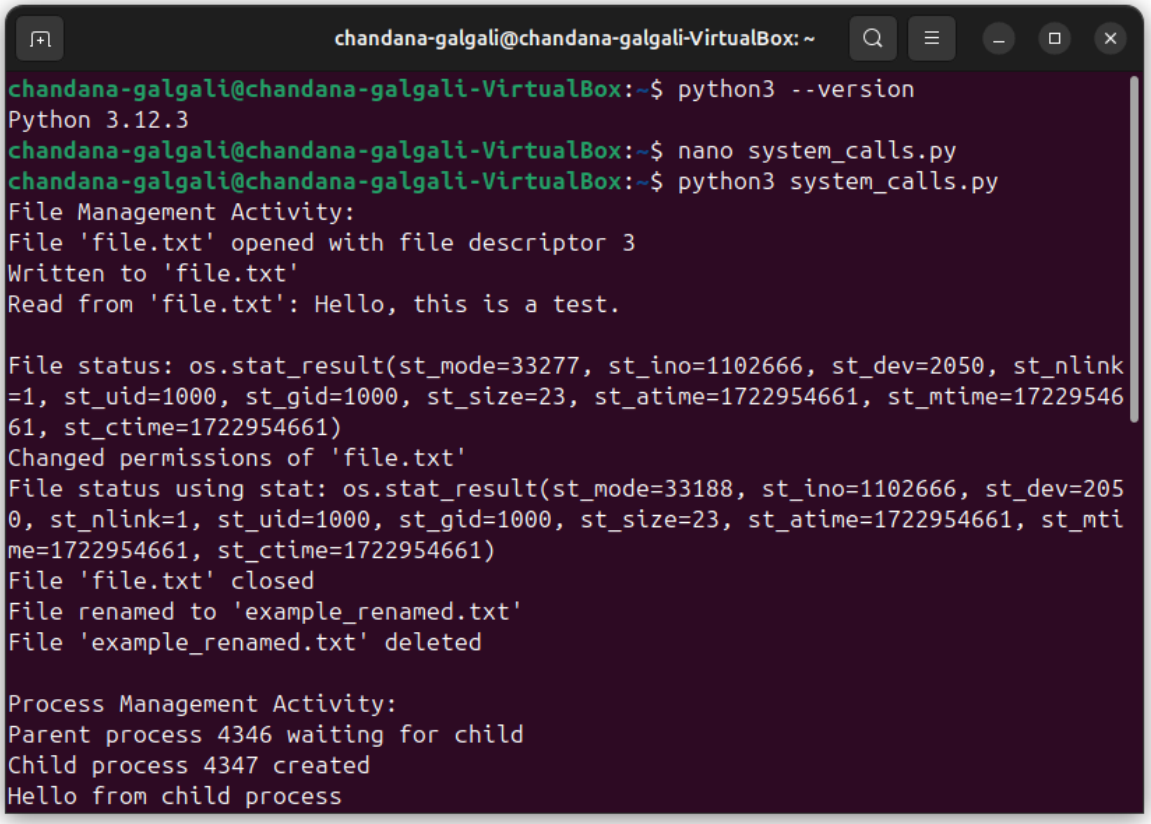
1. Write a program to show file management and process management using system calls.

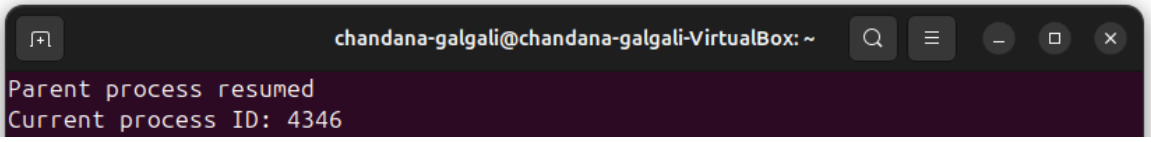
```
import os
import sys
import stat
import time

def file_management():
    filename = "file.txt"

    # 1. Open a file
    fd = os.open(filename, os.O_RDWR | os.O_CREAT)
    print(f"File '{filename}' opened with file descriptor {fd}")

    # 2. Write to a file
    os.write(fd, b"Hello, this is a test.\n")
    print(f"Written to '{filename}'")

    # 3. Read from a file
    os.lseek(fd, 0, os.SEEK_SET)
    data = os.read(fd, 100)
    print(f"Read from '{filename}': {data.decode()}")

    # 4. Get file status
    file_stat = os.fstat(fd)
    print(f"File status: {file_stat}")

    # 5. Change file permissions
    os.chmod(filename, stat.S_IRUSR | stat.S_IWUSR | stat.S_IRGRP | stat.S_IROTH)
    print(f"Changed permissions of '{filename}'")

    # 6. Move the file pointer
    os.lseek(fd, 0, os.SEEK_SET)

    # 7. Get file status using stat
    file_stat = os.stat(filename)
    print(f"File status using stat: {file_stat}")

    # 8. Close the file
    os.close(fd)
    print(f"File '{filename}' closed")

    # 9. Rename the file
    os.rename(filename, "example_renamed.txt")
    print(f"File renamed to 'example_renamed.txt'")

    # 10. Delete the file
    os.unlink("example_renamed.txt")
```

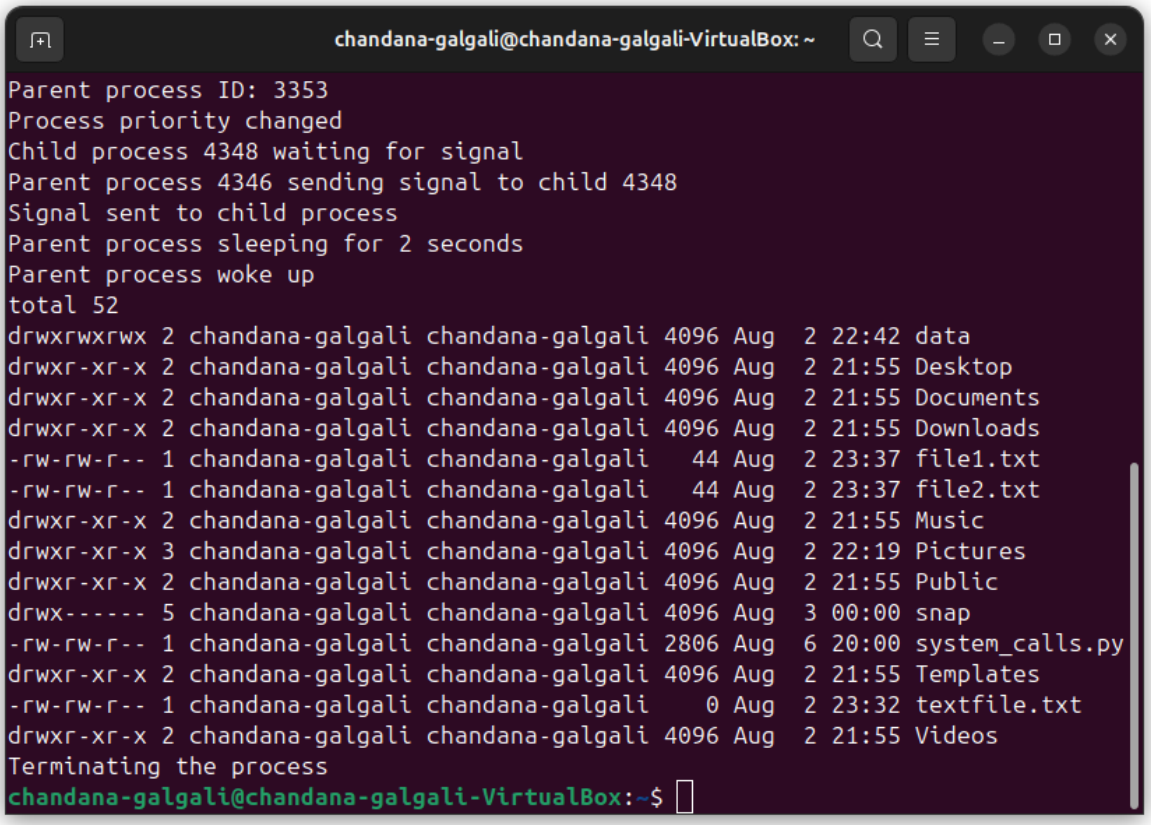
```
    print("File 'example_renamed.txt' deleted")

def process_management():
    # 1. Create a new process
    pid = os.fork()
    if pid == 0:
        # Child process
        print(f"Child process {os.getpid()} created")
        # 2. Replace the process image
        os.execlp("echo", "echo", "Hello from child process")
    else:
        # Parent process
        print(f"Parent process {os.getpid()} waiting for child")
        # 3. Wait for child process to change state
        os.wait()
        print("Parent process resumed")

    # 4. Get process ID
    pid = os.getpid()
    print(f"Current process ID: {pid}")

    # 5. Get parent process ID
    ppid = os.getppid()
    print(f"Parent process ID: {ppid}")

    # 6. Change process priority
    os.nice(10)
    print(f"Process priority changed")

    # 7. Send a signal to a process
    pid = os.fork()
    if pid == 0:
        # Child process
        print(f"Child process {os.getpid()} waiting for signal")
        time.sleep(10)
    else:
        # Parent process
        print(f"Parent process {os.getpid()} sending signal to child {pid}")
        os.kill(pid, 9)
        print("Signal sent to child process")

    # 8. Sleep for a while
    print("Parent process sleeping for 2 seconds")
    time.sleep(2)
    print("Parent process woke up")

    # 9. Execute a shell command
    os.system("ls -l")
```

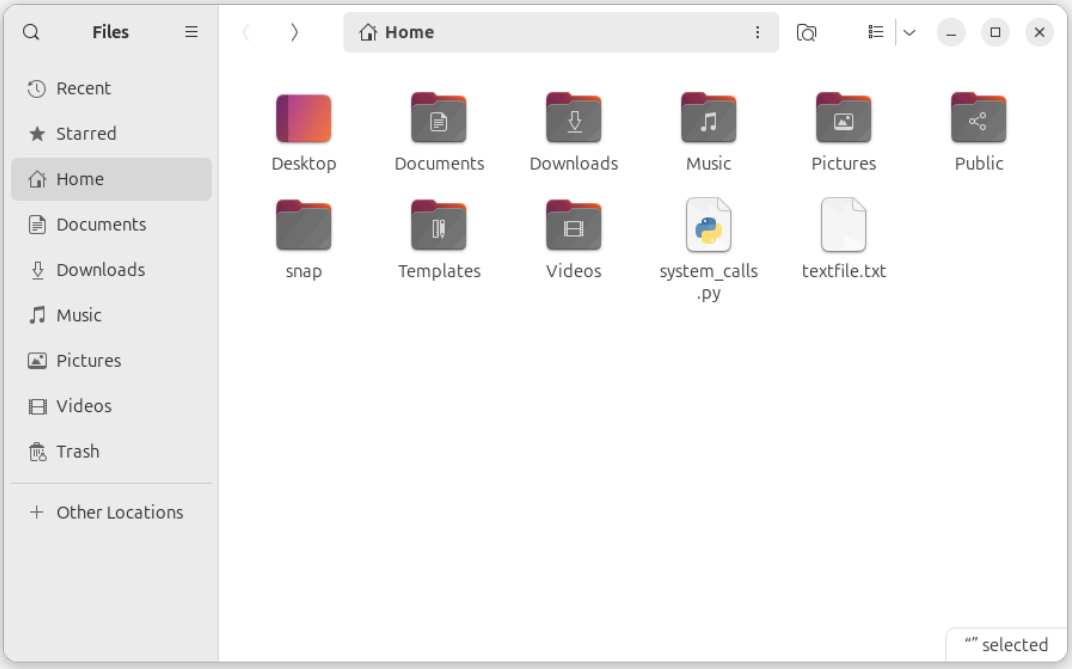
```
    # 10. Terminate the process
    print("Terminating the process")
    sys.exit(0)

if __name__ == "__main__":
    print("File Management Activity:")
    file_management()
    print("\nProcess Management Activity:")
    process_management()
```

Results: Perform the activity task and attach the snapshots here.









Outcomes: CO1 - Understand basic structure of modern operating system

---

Conclusion:

This experiment demonstrates the use of various Linux system calls for file and process management. Through the practical implementation of these system calls, we gain insight into how the operating system handles fundamental operations. Understanding these concepts is crucial for efficient system-level programming and contributes to a deeper understanding of operating systems.

Grade: AA/AB/BB/BC/CC/CD/DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Richard Blum and Christine Bresnahan, “Linux Command Line & Shell Scripting”, II Edition edition, Wiley, 2012.
