SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya
TRUST

K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Date: 06-09-2024

Batch: D-2 **Roll No.:** 16010122151

Experiment No. 03

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

Signature of the Staff In-charge with date

TITLE: System calls

AIM: To understand the working Process based system calls.

Expected Outcome of Experiment:

CO 1. To introduce basic concepts and functions of operating systems.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **William Stallings "Operating Systems" Person, Seventh Edition Edition.**
3. **Sumitabha Das " UNIX Concepts & Applications", McGraw Hill Second Edition.**

Pre Lab/ Prior Concepts:

System Calls Provide the Interface between a process and the OS.

System calls are usually made when a process in user mode requires access to a resource.

Then it requests the kernel to provide the resource via a system call.

System calls are required in the following situations −

1) If a file system requires the creation or deletion of files.

2) Reading and writing from files also require a system call.

3) Creation and management of new processes.

4) Network connections also require system calls. This includes sending and receiving packets.

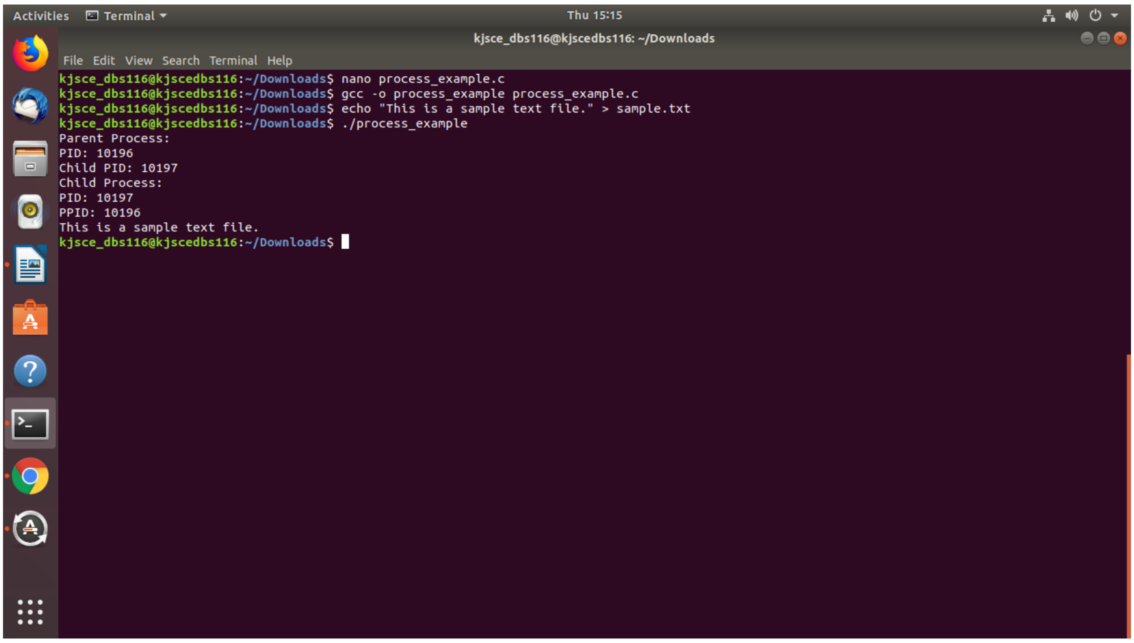
5) Access to a hardware devices such as a printer, scanner etc. requires a system call.

Description of the application to be implemented:

Program for System Call:

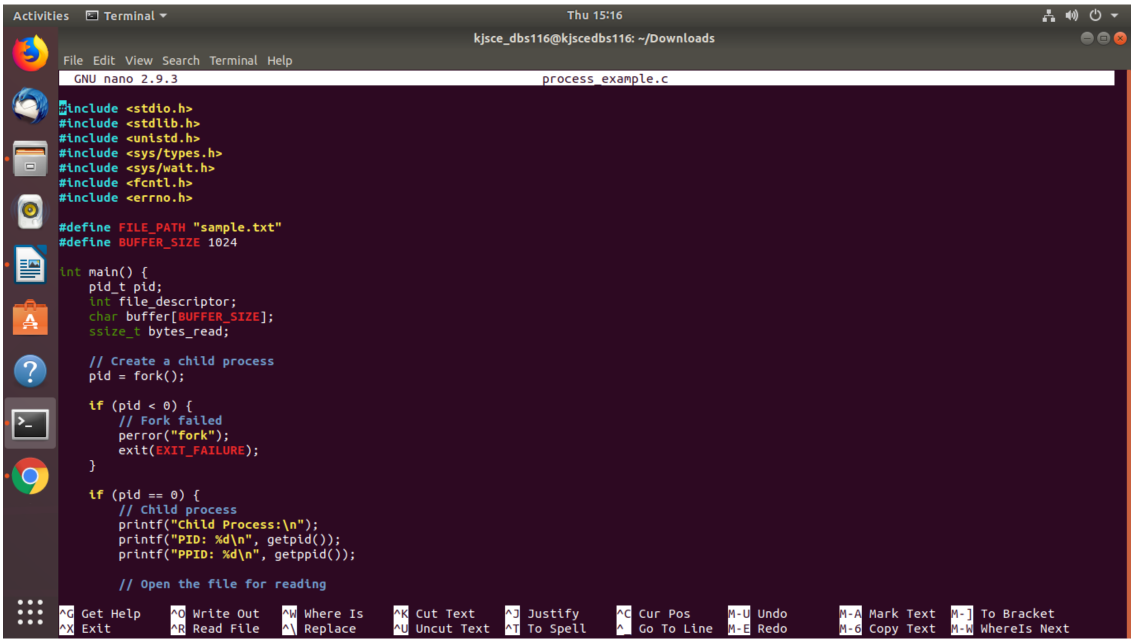
1. Write a Program for creating process using System call (E.g fork()) Create a child process. Display the details about that process using getpid and getppid functions. In a child process, Open the file using file system calls and read the contents and display.

Implementation details: (printout of code / screen shot)



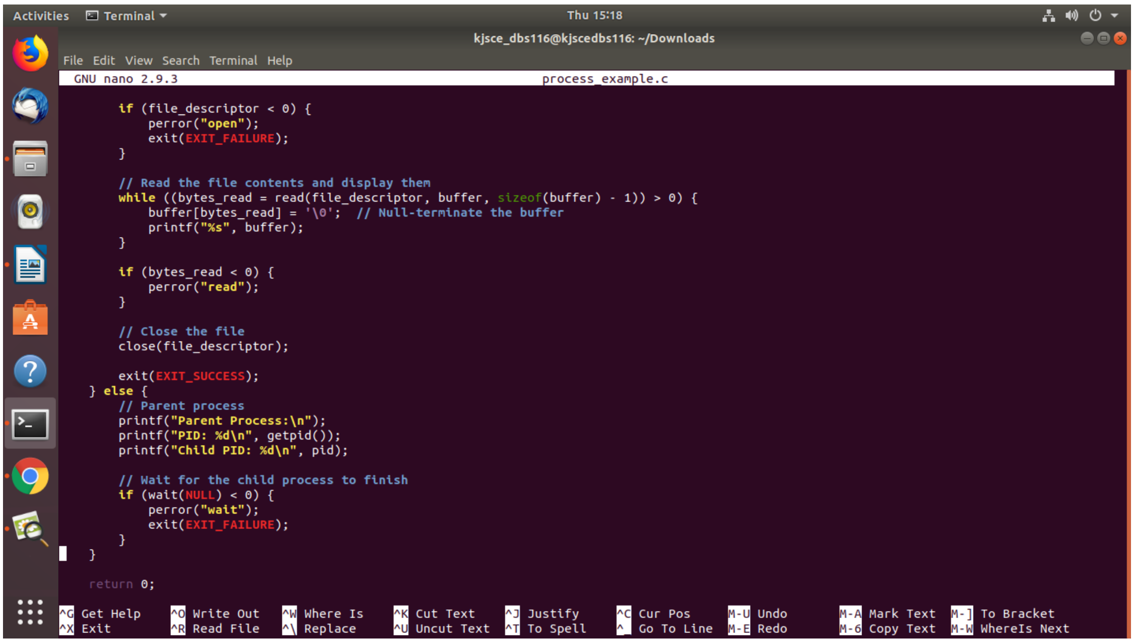
Activities
Terminal
Thu 15:15
kjsce_dbs116@kjscedbs116: ~/Downloads
File Edit View Search Terminal Help

```
kjsce_dbs116@kjscedbs116:~/Downloads$ nano process_example.c
kjsce_dbs116@kjscedbs116:~/Downloads$ gcc -o process_example process_example.c
kjsce_dbs116@kjscedbs116:~/Downloads$ echo "This is a sample text file." > sample.txt
kjsce_dbs116@kjscedbs116:~/Downloads$ ./process_example
Parent Process:
PID: 10196
Child PID: 10197
Child Process:
PID: 10197
PPID: 10196
This is a sample text file.
kjsce_dbs116@kjscedbs116:~/Downloads$ 
```



Activities
Terminal
Thu 15:16
kjsce_dbs116@kjscedbs116: ~/Downloads
File Edit View Search Terminal Help

```
  GNU nano 2.9.3                                              process_example.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>

#define FILE_PATH "sample.txt"
#define BUFFER_SIZE 1024

int main() {
    pid_t pid;
    int file_descriptor;
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // Child process
        printf("Child Process:\n");
        printf("PID: %d\n", getpid());
        printf("PPID: %d\n", getppid());

        // Open the file for reading

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text  M-] To Bracket
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text  M-W WhereIs Next
```



Activities
Terminal
Thu 15:18
kjsce_dbs116@kjscedbs116: ~/Downloads
File Edit View Search Terminal Help
GNU nano 2.9.3
process_example.c

```
        if (file_descriptor < 0) {
            perror("open");
            exit(EXIT_FAILURE);
        }

        // Read the file contents and display them
        while ((bytes_read = read(file_descriptor, buffer, sizeof(buffer) - 1)) > 0) {
            buffer[bytes_read] = '\0';  // Null-terminate the buffer
            printf("%s", buffer);
        }

        if (bytes_read < 0) {
            perror("read");
        }

        // Close the file
        close(file_descriptor);

        exit(EXIT_SUCCESS);
    } else {
        // Parent process
        printf("Parent Process:\n");
        printf("PID: %d\n", getpid());
        printf("Child PID: %d\n", pid);

        // Wait for the child process to finish
        if (wait(NULL) < 0) {
            perror("wait");
            exit(EXIT_FAILURE);
        }
    }

    return 0;
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo M-A Mark Text M-] To Bracket
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line M-E Redo M-6 Copy Text M-W WhereIs Next

Conclusion : Learnt how to create a child process using **fork() ,** manage process IDs with `getpid()` and `getppid()`, and performing file operations within the child process, while ensuring proper synchronization between parent and child processes.

Post Lab Descriptive Questions

1) Describe System Call Interface.

The **System Call Interface (SCI)** is a critical component of an operating system that allows user-space programs to request services from the kernel. It acts as a bridge between user applications and the underlying hardware or system resources, providing a controlled mechanism for executing privileged operations.

2)List the types of System Calls.

a)Process Control:

fork(): Create a new process.
exec(): Replace the current process image with a new one.
exit(): Terminate the current process.
wait(): Wait for a child process to change state.
getpid(): Get the process ID.
getppid(): Get the parent process ID.

b)File Management:

open(): Open a file or device.
read(): Read data from a file descriptor.
write(): Write data to a file descriptor.
close(): Close a file descriptor.
lseek(): Reposition the file offset.
unlink(): Delete a file or directory.
rename(): Rename a file or directory.
stat(): Get file status.

c)Device Management:

ioctl(): Control device-specific operations.
read(): Read from a device.
write(): Write to a device.
Information Maintenance:

gettimeofday(): Get the current time.
settimeofday(): Set the system time.

uname(): Get system information.

d)Communication:

pipe(): Create a pipe for inter-process communication.
shmget(): Allocate shared memory.
shmat(): Attach shared memory to the process.
msgget(): Create or access a message queue.
msgsnd(): Send a message to a message queue.
msgrcv(): Receive a message from a message queue.
semget(): Create or access a semaphore set.
semop(): Operate on a semaphore set.

e)Memory Management:

mmap(): Map files or devices into memory.
munmap(): Unmap memory regions.
brk(): Change the data segment size.
sbrk(): Adjust the program's data space.

Date: 06-09-2024 **Signature of faculty in-charge**