|  |  |  |  |
| --- | --- | --- | --- |
| **Fr. Conceicao Rodrigues College of Engineering**  **Department of Computer Engineering** | | | |
| **Student’s Roll No** |  | **Students Name** |  |
| **Date of Performance** |  | **SE Computer – Div** | **A / B** |

**Aim:** To study Process and File Management System Calls

**Lab Outcome:**

**CSL403.1: Demonstrate basic Operating system Commands, Shell scripts, System Calls and API wrt Linux.**

**Problem Statements:**

(1.) Process related System Calls.

a) Create a child process in Linux using the fork system call. From the child process obtain the process ID of both child and parent by using getpid and getppid system call.

b) Explore wait and waitpid before termination of process.

c) Explain ps command and output in detail. What is Zombie and Orphan Process? Show the output.

d) Explain fork(), getpid(), getppid(),wait() and waitpid() with syntax.

(2) File related system calls

a) Program to copy contents of one file (source) to another file (destination). Finally displaying contents of destination file.

b) 2. Explain creat(), open(), close(), read() and write() with syntax.

**References:**

<https://www.geeksforgeeks.org/fork-system-call/>

<https://www.geeksforgeeks.org/getppid-getpid-linux/>

<https://www.geeksforgeeks.org/wait-system-call-c/>

<https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/>

<https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/>

|  |  |  |  |
| --- | --- | --- | --- |
| **On time Submission(2)** | **Knowledge of Topic(4)** | **Implementation and Demonstraion(4)** | **Total (10)** |
|  |  |  |  |
| **Signature of Faculty** |  | **Date of Submission** |  |

Q1.

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    fork();

    fork();

    fork();

    printf("hello\n");

    return 0;

}

/\*

hello

hello

hello

hello

hello

hello

universe@dell16:~/Desktop/9555$ hello

hello

\*/

**Q1.a)**

#include <stdio.h>

int main()

{

    int pid;

    pid = fork();

    if (pid == 0)

    {

        printf("\nParent Process id :");

        printf("%d",getppid());

        printf("\nChild Process with parent id :");

        printf("%d\n",getpid());

    }

    else{

        printf("\nParent Process id :");

        printf("%d",getppid());

        printf("\nChild Process with parent id :");

        printf("%d\n",getpid());

    }

    return 0;

}

/\*

Parent Process id :3062

Child Process with parent id :4425

Parent Process id :4425

Child Process with parent id :4426

\*/

**Q1.c)**

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    // Fork returns process id

    // in parent process

    pid\_t child\_pid = fork();

    // Parent process

    if (child\_pid > 0)

        sleep(50);

    // Child process

    else

        exit(0);

    return 0;

}

/\*

universe    4328  0.0  0.0      0     0 pts/2    Z+   10:37   0:00 [lab3\_2] <defunct>

\*/

//Orphan process

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    // Create a child process

    int pid = fork();

    if (pid > 0)

        printf("in parent process");

    // Note that pid is 0 in child process

    // and negative if fork() fails

    else if (pid == 0)

    {

        sleep(10);

        printf("in child process \n");

        printf("parent is id %d",getppid());

    }

    return 0;

}

/\*

in parent process universe@dell16:~/Desktop/9555$ in child process

parent is id 1

\*/

**Q1.d)**

**Fork():** It is used for creating a child process of the current process

fork()

**Getpid():**It returns the process ID of the calling process

pid\_t getppid(void);

**Getppid():** It returns the process ID of the parent of the calling function. If a process was created by forking, it returns ID of that original process, else it returns 1 which is default process ID of init process

pid\_t getppid();

**Wait():**This function is used when the we want to suspend a particular process until any one of it’s child process terminates.

pid\_t wait(int \*status);

**Waitpid():**This function stops/suspends the execution of the calling process until one specific child process terminates. It is specified by the process ID passed

pid\_t waitpid(pid\_t pid, int \*status, int options);

Q2.)

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#define BUFFER\_SIZE 1024

int main()

{

    int source\_file, destination\_file, n;

    char buffer[BUFFER\_SIZE];

    source\_file = open("file.txt", O\_RDONLY);

    if (source\_file == -1) {

        printf("Unable to open source file.");

        exit(1);

    }

    destination\_file = open("new.txt", O\_WRONLY | O\_CREAT, 0644);

    if (destination\_file == -1) {

        printf("Unable to open destination file.");

        exit(1);

    }

    while ((n = read(source\_file, buffer, BUFFER\_SIZE)) > 0) {

        if (write(destination\_file, buffer, n) != n) {

            printf("Error writing to destination file.");

            exit(1);

        }

    }

    close(source\_file);

    close(destination\_file);

    destination\_file = open("new.txt", O\_RDONLY);

    if (destination\_file == -1) {

        printf("Unable to open destination file.");

        exit(1);

    }

    while ((n = read(destination\_file, buffer, BUFFER\_SIZE)) > 0) {

        if (write(STDOUT\_FILENO, buffer, n) != n) {

            printf("Error writing to standard output.");

            exit(1);

        }

    }

    close(destination\_file);

    return 0;

}

