|  |  |  |  |
| --- | --- | --- | --- |
| **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.

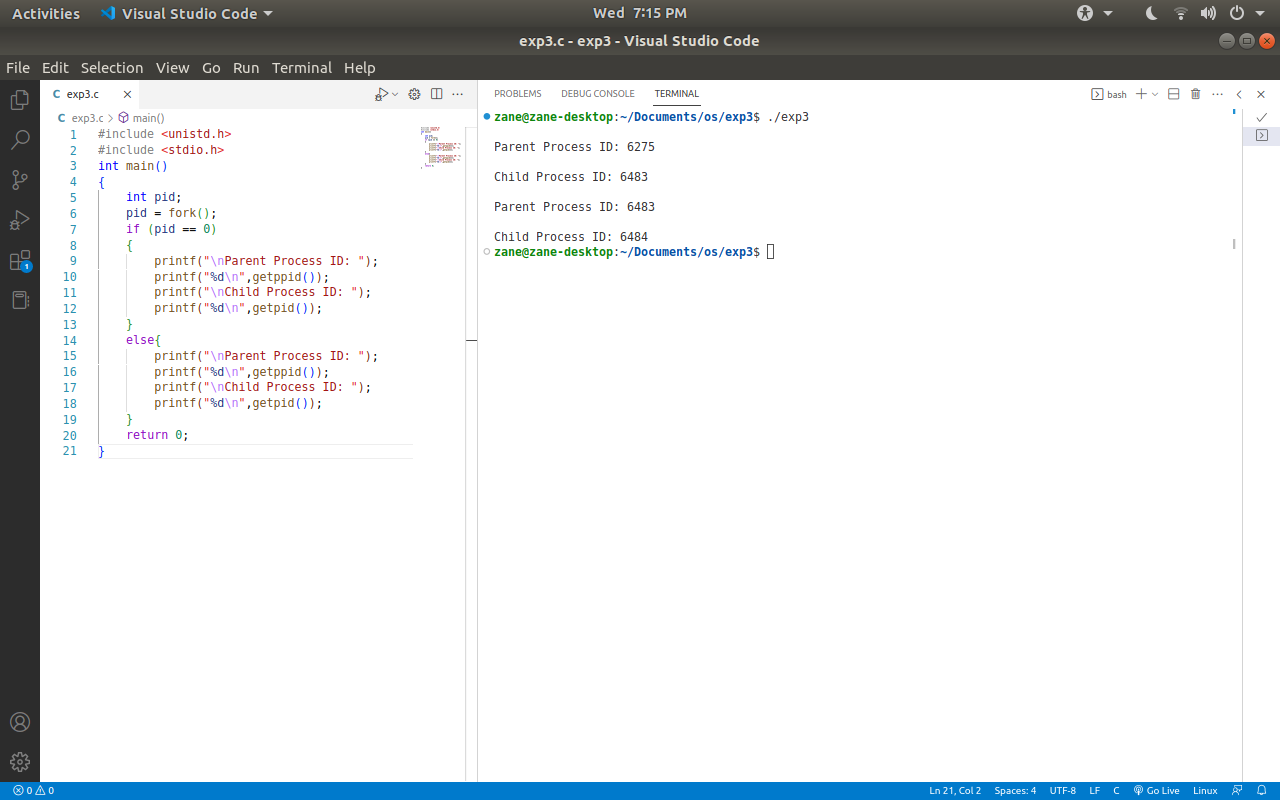
(2) File related system calls

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

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

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

1. **a)**

****

**1.b)**

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

**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

**1.c ) PS Command**

The ps command command stands for “process status”. It is used for displaying the currently running processes along with important information like PID,TTY,CPU% etc.

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

san 9 0.1 0.0 14216 3804 tty1 S 00:06 0:00 -bash

san 42 0.0 0.0 15520 1928 tty1 R 00:06 0:00 ps ux

PID: Process ID

PPID: Parent Process ID

%CPU/%MEM: CPU or Memory Utilization

PRI: Process Priority Number

RSS: Real Memory Usage

VSZ: Virtual Memory Usage

TTY: Terminal associated with the process

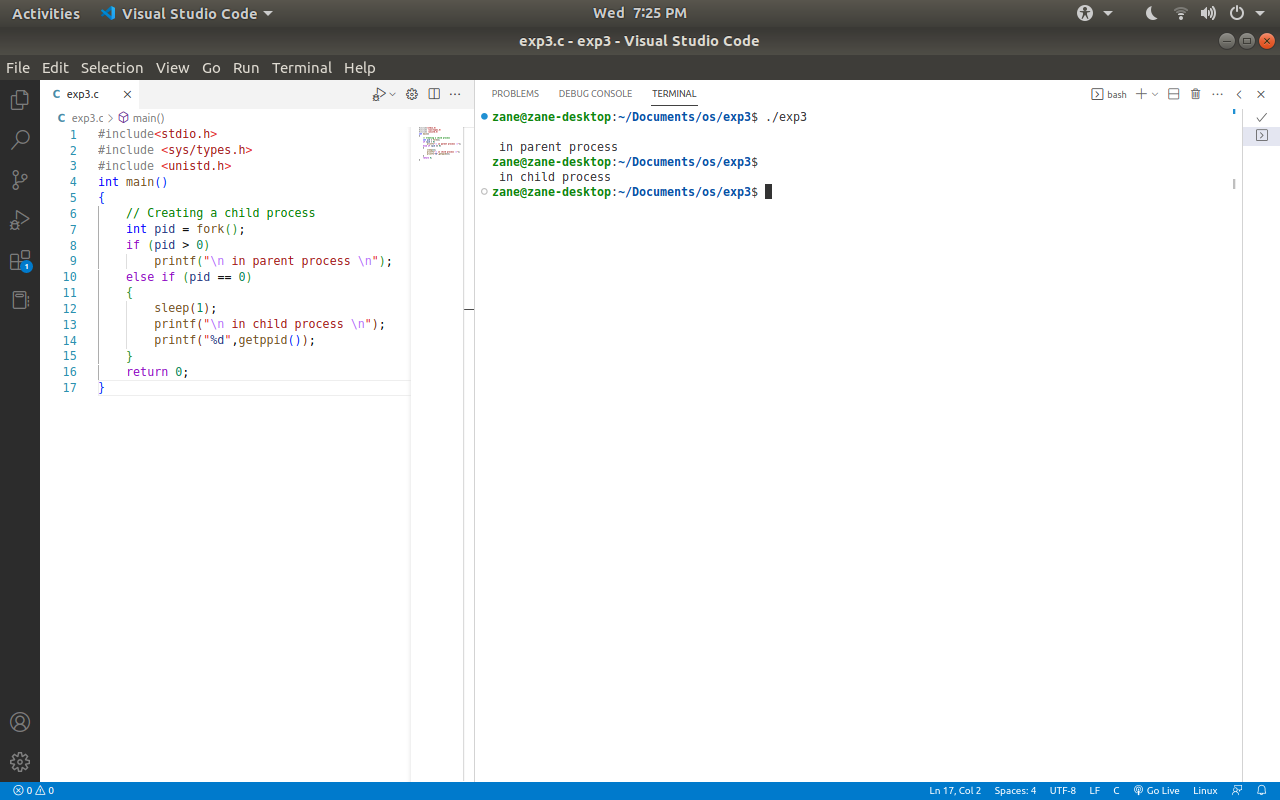
STAT: Process Status Code

START: Process Start Time

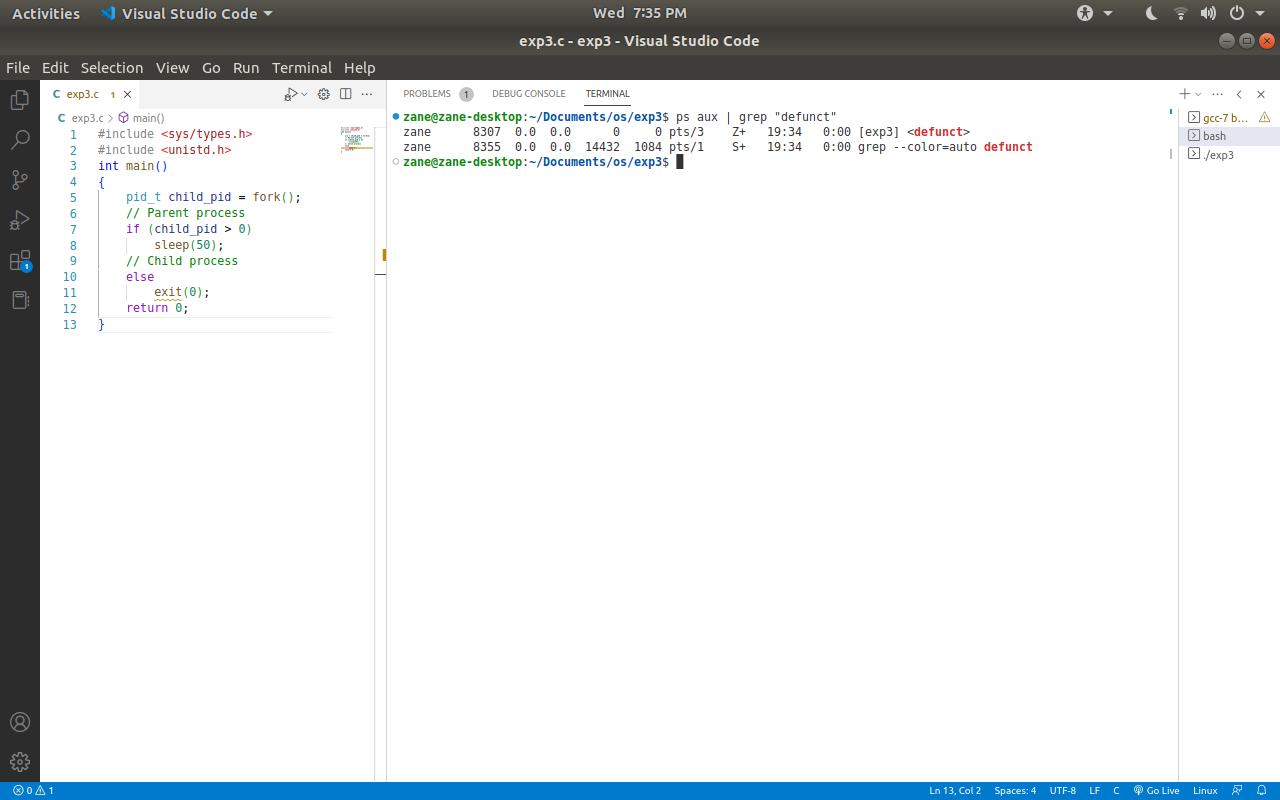
COMMAND: Name of the process

ADDR: Memory Address of the process

**Orphan Process**

****

**1.c Zombie Process**



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

**1.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);

**2.a)**

#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); // Opening Source File

if (source\_file == -1)

{

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

exit(1);

}

// Opening Destination File Or Creating it

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) //Copying the contents

{

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

{

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

exit(1);

}

}

close(source\_file);// Close both files

close(destination\_file);

// Open the destination file in read-only mode

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

if (destination\_file == -1)

{

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

exit(1);

}

// Display the contents of the destination file

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;

}

**Output:**

zane@zane-desktop:~/Documents/os/exp3$ ./exp3

Welcome to my Hell!

zane@zane-desktop:~/Documents/os/exp3$

**2.b)**

All of these functions are stored inside the fcntl.h header file.

**Create():** It is used to create a new file.

Syntax:

int create(char\*filename, mode\_t mode)

mode->permissions of the new file

It returns first unused file descriptor

Returns -1 when error occurs

**Create():** It is used open a file for reading and/or writing

Syntax:

int open(const char\* Path, int flags[,int mode]);

flags->

**O\_RDONLY**: read only

**O\_WRONLY**: write only

**O\_RDWR**: read and write

**O\_CREAT**: create file if it doesn’t exist

**O\_EXCL**: prevent creation if it already exists

It returns first unused file descriptor

Returns -1 when error occurs

**Close():** It is used to close a file using its file descriptor got by using the create(), open() functions

Syntax:

int close(int fd);

Sets element fd of file descriptor table to NULL

Return 0 on success and -1 on error

**Read():** It is used to read a file. It reads cnt bytes of input into the memory indicated by buf.

Syntax:

size\_t read (int fd, void\* buf, size\_t cnt);

buf->buffer to read data from

cnt->length of buffer

It returns;

* Number of bytes on Success
* 0 on reaching End Of File
* -1 on error
* -1 on signal interrupt

**Write():** It is used to write into file. It writes cnt bytes from buf to the file or socket associated with fd.

Cnt should not be greater than INT\_MAX

If cnt is zero, write() simply returns 0 without attempting any other action

Syntax:

size\_t write (int fd, void\* buf, size\_t cnt);

It returns;

* Number of bytes on Success
* 0 on reaching End Of File
* -1 on error
* -1 on signal interrupt