LAB ASSIGNMENT NO -04

NAME- AVISEK MANDAL

ROLL NO - 102203700

2CO35

1. Write a program to implement fork () system call.

fork()-The Fork system call is used for creating a new process in Linux, and Unix systems, which is called the child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call.

The child process uses the same pc(program counter), same CPU registers, and same open files which use in the parent process. It takes no parameters and returns an integer value.

~/FORK.c - Mousepad

File Edit Search View Document Help

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    // Fork a child process
    pid_t child_pid = fork();

    if (child_pid == -1) {
        // Fork failed
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (child_pid == 0) {
        // This code is executed by the child process
        printf("Child process (PID=%d) is running.\n", getpid());
        // Add child process logic here
    } else {
        // This code is executed by the parent process
        printf("Parent process (PID=%d) created a child process with PID=%d.\n", getpid(), child_pid);
        // Add parent process logic here
    }

    return 0;
}
```

```
┌──(avisek㉿kali)-[~]
└─$ ls
 Avisek     Documents   Downloads   FORK.c  'my directory'          new.txt                  Pictures   Templates   vscode
 Desktop    download    file2.txt   Music   'my vs code directory'  packages.microsoft.gpg   Public     Videos     'vs directory'

┌──(avisek㉿kali)-[~]
└─$ gcc -0 FORK FORK.c
gcc: error: unrecognized command-line option '-0'

┌──(avisek㉿kali)-[~]
└─$ gcc -o FORK FORK.c

┌──(avisek㉿kali)-[~]
└─$ ./FORK
Parent process (PID=7967) created a child process with PID=7968.
Child process (PID=7968) is running.

┌──(avisek㉿kali)-[~]
└─$
```

2.Write a program to implement wait () and exit () System Calls.

ans-
wait()-A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.

exit()-The exit() function in C is a standard library function that is used to terminate the calling process. When exit() is called, any open file descriptors belonging to the process are closed and any children of the process are inherited by process 1, init, and the process parent is sent a SIGCHLD signal.
It is defined inside the <stdlib.h> header file.

File Edit Search View Document Help

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t child_pid;

    // Fork a child process
    child_pid = fork();

    if (child_pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (child_pid == 0) {
        // This code is executed by the child process
        printf("Child process (PID=%d) is running.\n", getpid());

        // Add child process logic here

        // Terminate the child process
        exit(EXIT_SUCCESS);
    } else {
        // This code is executed by the parent process
        printf("Parent process (PID=%d) created a child process with PID=%d.\n", getpid(), child_pid);

        // Wait for the child to complete
        int status;
        pid_t terminated_pid = wait(&status);

        if (terminated_pid == -1) {
            perror("wait");
            exit(EXIT_FAILURE);
        }

        if (WIFEXITED(status)) {
            printf("Child process (PID=%d) exited with status %d.\n", terminated_pid, WEXITSTATUS(status));
        } else {
            printf("Child process (PID=%d) did not exit normally.\n", terminated_pid);
        }

        // Add parent process logic here
```

File Actions Edit View Help

```
┌──(avisek㉿kali)-[~]
└─$ gcc -o wait wait.c

┌──(avisek㉿kali)-[~]
└─$ ./wait
Parent process (PID=23974) created a child process with PID=23975.
Child process (PID=23975) is running.
Child process (PID=23975) exited with status 0.

┌──(avisek㉿kali)-[~]
└─$
```

3.Write a program to implement execv() system call

ANS-execv()- The execv function executes the file named by filename as a new process image. The argv argument is an array of null-terminated strings that is used to provide a value for the argv argument to the main function of the program to be executed. The last element of this array must be a null pointer.

File Edit Search View Document Help

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {

    char *programPath = "/bin/ls";
    char *args[] = {programPath, "-l", NULL};

    // Execute the new program using execv()
    if (execv(programPath, args) == -1) {
        perror("execv"); // Print an error message if execv fails
        exit(EXIT_FAILURE);
    }

    // This code will only execute if execv fails
    printf("This line should not be reached.\n");

    return 0;
}
```

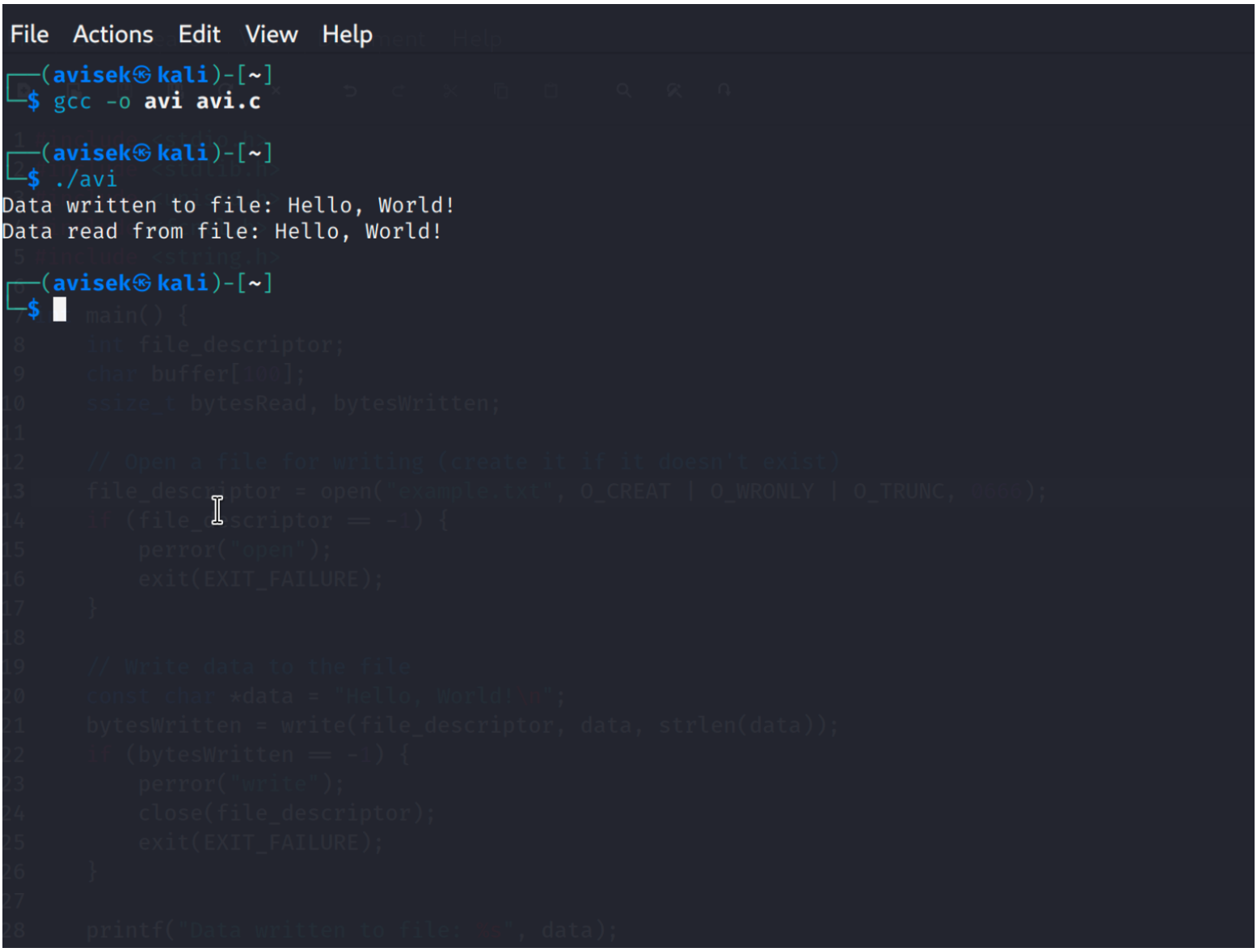
File Actions Edit View Help

```
┌──(avisek㉿kali)-[~]
└─$ ls -1
Avisek
Desktop
Documents
download
Downloads
execv.c
file2.txt
FORK
FORK.c
Music
'my directory'
'my vs code directory'
new.txt
packages.microsoft.gpg
Pictures
Public
Templates
Videos
vscode
'vs directory'
wait
wait.c
'waitexit().c'

┌──(avisek㉿kali)-[~]
└─$
```

4.Write a program to implement the system calls open (), read (), write () & close ()

TEXT EDITOR

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

int main() {
    int file_descriptor;
    char buffer[100];
    ssize_t bytesRead, bytesWritten;

    // Open a file for writing (create it if it doesn't exist)
    file_descriptor = open("example.txt", O_CREAT | O_WRONLY | O_TRUNC, 0666);
    if (file_descriptor == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    // Write data to the file
    const char *data = "Hello, World!\n";
    bytesWritten = write(file_descriptor, data, strlen(data));
    if (bytesWritten == -1) {
        perror("write");
        close(file_descriptor);
        exit(EXIT_FAILURE);
    }

    printf("Data written to file: %s", data);

    // Close the file
    if (close(file_descriptor) == -1) {
        perror("close");
        exit(EXIT_FAILURE);
    }

    // Open the same file for reading
    file_descriptor = open("example.txt", O_RDONLY);
    if (file_descriptor == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }


    // Read data from the file
    bytesRead = read(file_descriptor, buffer, sizeof(buffer));
    if (bytesRead == -1) {
        perror("read");
        close(file_descriptor);
        exit(EXIT_FAILURE);
    }

    buffer[bytesRead] = '\0'; // Null-terminate the string

    printf("Data read from file: %s", buffer);

    // Close the file again
    if (close(file_descriptor) == -1) {
        perror("close");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```


File Actions Edit View Help
(avisek㉿kali)-[~]
$ gcc -o avi avi.c
(avisek㉿kali)-[~]
$ ./avi
Data written to file: Hello, World!
Data read from file: Hello, World!
(avisek㉿kali)-[~]
$