Week-7

1. Write a C program to
   a. To create a child process.
   b. Child should execute an interpreter file by passing few arguments and some environment variables.
   c. Parent should execute an interpreter file by passing few arguments
   d. Create an interpreter file which has the path of echoall.c file
   e. Create echoall.c file which prints the arguments and environment variables received through parent and child process.

//Create the program.c file

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

int main() {
   pid_t pid;
   int status;

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

   if (pid == -1) {
      // Fork failed
      perror("fork");
      exit(EXIT_FAILURE);
   } else if (pid == 0) {
      // Child process
      printf("Child process executing...\n");

      // Arguments for the interpreter file
      char *args[] = {"./interpreter", "arg1", "arg2", NULL};

      // Environment variables for the interpreter file
      char *envp[] = {"VAR1=value1", "VAR2=value2", NULL};

      // Execute the interpreter file with arguments and environment variables
      execve(args[0], args, envp);
```

```
        // If execve fails
        perror("execve");
        exit(EXIT_FAILURE);
    } else {
        // Parent process
        waitpid(pid, &status, 0); // Wait for the child to finish

        printf("Parent process executing...\n");

        // Arguments for the interpreter file
        char *args[] = {"./interpreter", "arg3", "arg4", NULL};

        // Execute the interpreter file with arguments
        execvp(args[0], args);

        // If execvp fails
        perror("execvp");
        exit(EXIT_FAILURE);
    }

    return 0;
}s
```

//Create the interpreter file 'interpreter':

i) Open a text editor:
gedit echo_interpreter

ii) Write the path of the echoall.c file in the interpreter file. Let's say the echoall.c file is located in the same directory as the interpreter file. If so, the interpreter file content would be:

```
#!/bin/bash

gcc -o echoall echoall.c
```

iii) Save the file by pressing Ctrl + O, then press Enter to confirm the filename, and exit the text editor by pressing Ctrl + X.

iv) Make the file executable:
chmod +x echo_interpreter

Now you have created an interpreter file named echo_interpreter that contains the command to compile the echoall.c file. You can execute it to compile echoall.c. After compiling, you'll have an executable named echoall in the same directory.

//Create the echoall.c file

```
#include <stdio.h>
```

```
#include <stdlib.h>

int main(int argc, char *argv[], char *envp[]) {
   printf("Received arguments:\n");
   for (int i = 0; i < argc; i++) {
      printf("arg[%d]: %s\n", i, argv[i]);
   }

   printf("Received environment variables:\n");
   for (char **env = envp; *env != NULL; env++) {
      printf("%s\n", *env);
   }

   return 0;
}
```

Compile all three files:

```
gcc -o program program.c
gcc -o interpreter interpreter.c
gcc -o echoall echoall.c
```

Now, when you run ./program, it will create a child process that executes the interpreter file with some arguments and environment variables, and the parent process will execute the interpreter file with different arguments. Both processes will print the arguments and environment variables they receive.

2. Write a C program to perform the following operations
   a. To create a child process
   b. Child process should execute a program to show the use of the access function
   c. Parent process should wait for the child process to exit
   d. Also print the necessary process IDs

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

int main() {
   pid_t pid;
   int status;
```

```
    // Get the process ID of the parent process
    pid_t parent_pid = getpid();

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

    if (pid == -1) {
        // Fork failed
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process

        // Get the process ID of the child process
        pid_t child_pid = getpid();

        printf("Child process (PID: %d) executing...\n", child_pid);

        // Execute a program to demonstrate the access function
        if (access("example.txt", F_OK) == 0) {
            printf("Child process: File exists and can be accessed.\n");
        } else {
            printf("Child process: File does not exist or cannot be accessed.\n");
        }

        // Exit the child process
        exit(EXIT_SUCCESS);
    } else {
        // Parent process

        // Get the process ID of the parent process
        pid_t parent_pid = getpid();

        printf("Parent process (PID: %d) executing...\n", parent_pid);

        // Wait for the child process to exit
        waitpid(pid, &status, 0);

        printf("Parent process: Child process (PID: %d) has exited.\n", pid);
    }

    return 0;
}
```

The program generates a child process, and the child process uses the access function to determine whether the file "example.txt" is actually exist and

accessible. After waiting for the child process to terminate, the parent process provides the required process IDs.