Linux system calls execl(), execlp(), dup(), and dup2() are low-level functions available in the Linux operating system that provide access to certain functionalities of the kernel. These system calls allow a program to interact with the underlying operating system to perform specific operations.

execl(): The execl() system call is used to replace the current process with a new executable file. It requires the full path of the executable file to be specified along with the command-line arguments. Once execl() is called, the current process is terminated, and a new process starts executing the specified executable.

execlp(): The execlp() system call is similar to execl(), but it allows the command to be specified without the need for a full path. It searches for the executable file in the directories listed in the PATH environment variable. This means you can directly specify the command name instead of the full path.

dup(): The dup() system call is used to duplicate a file descriptor. It takes an existing file descriptor as input and returns a new file descriptor that refers to the same open file. The new file descriptor is assigned the lowest available file descriptor number greater than or equal to the value specified.

dup2(): The dup2() system call is similar to dup(), but it allows the duplication of a file descriptor to a specific file descriptor number. If the specified file descriptor number is already open, it is closed first before duplicating the file descriptor.

These system calls are used for various purposes, such as executing external programs, redirecting input/output, and managing file descriptors in a program. They provide powerful functionality for interacting with the underlying operating system and managing processes, files, and input/output streams in a low-level manner.

Program 1: Executing a Program Using execl()

#include <unistd.h>

#include <stdio.h>

int main() {

 // Execute the ls command

 execl("/bin/ls", "ls", "-l", NULL);

 // If execl() is successful, the following code won't be executed

 perror("Error executing the command");

 return 1;

}

Program 2: Executing a Program Using execlp()

#include <unistd.h>

#include <stdio.h>

int main() {

 // Execute the ls command

 execlp("ls", "ls", "-l", NULL);

 // If execlp() is successful, the following code won't be executed

 perror("Error executing the command");

 return 1;

}

Program 3: Duplicating a File Descriptor Using dup()

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main() {

 int originalFD, newFD;

 // Open a file for reading

 originalFD = open("example.txt", O\_RDONLY);

 if (originalFD == -1) {

 perror("Error opening the file");

 return 1;

 }

 // Duplicate the file descriptor

 newFD = dup(originalFD);

 if (newFD == -1) {

 perror("Error duplicating the file descriptor");

 close(originalFD);

 return 1;

 }

 printf("Original file descriptor: %d\n", originalFD);

 printf("Duplicated file descriptor: %d\n", newFD);

 // Close the file descriptors

 close(originalFD);

 close(newFD);

 return 0;

}

Program 4: Duplicating a File Descriptor Using dup2()

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main() {

 int originalFD, newFD;

 // Open a file for reading

 originalFD = open("example.txt", O\_RDONLY);

 if (originalFD == -1) {

 perror("Error opening the file");

 return 1;

 }

 // Duplicate the file descriptor using dup2()

 newFD = dup2(originalFD, STDOUT\_FILENO);

 if (newFD == -1) {

 perror("Error duplicating the file descriptor");

 close(originalFD);

 return 1;

 }

 printf("Original file descriptor: %d\n", originalFD);

 printf("Duplicated file descriptor: %d\n", newFD);

 // Close the file descriptors

 close(originalFD);

 close(newFD);

 return 0;

}

These examples showcase different scenarios involving the execl(), execlp(), dup(), and dup2() system calls. Adapt and modify these programs as needed to suit your specific requirements. Ensure proper error handling and check the return values of these system calls for successful execution.