**I/O Linux System calls- close() and ulink()**

Program 1: Closing a File Descriptor

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main() {

char filename[] = "example.txt";

int fileDescriptor;

// Open the file for reading

fileDescriptor = open(filename, O\_RDONLY);

if (fileDescriptor == -1) {

perror("Error opening the file");

return 1;

}

printf("File opened successfully: %s\n", filename);

// Close the file descriptor

int closeResult = close(fileDescriptor);

if (closeResult == -1) {

perror("Error closing the file");

return 1;

}

printf("File closed successfully\n");

return 0;

}

Program 2: Deleting a File

#include <unistd.h>

#include <stdio.h>

int main() {

char filename[] = "example.txt";

// Delete the file

int unlinkResult = unlink(filename);

if (unlinkResult == -1) {

perror("Error deleting the file");

return 1;

}

printf("File deleted successfully: %s\n", filename);

return 0;

}

In the first program, the close() system call is used to close a file descriptor after opening a file for reading. The program checks for any errors during the opening and closing operations.

In the second program, the unlink() system call is used to delete a file specified by its name. The program checks for any errors during the deletion process.

Please note that deleting a file using unlink() permanently removes it from the filesystem, so exercise caution when using this system call.

Program 3: Closing Multiple File Descriptors

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

int main() {

int fd1, fd2;

// Open two files for reading

fd1 = open("file1.txt", O\_RDONLY);

fd2 = open("file2.txt", O\_RDONLY);

if (fd1 == -1 || fd2 == -1) {

perror("Error opening the files");

return 1;

}

printf("Files opened successfully\n");

// Close the file descriptors

close(fd1);

close(fd2);

printf("Files closed successfully\n");

return 0;

}

Program 4: Deleting a Directory

#include <unistd.h>

#include <stdio.h>

int main() {

char dirname[] = "example\_dir";

// Delete the directory

int unlinkResult = rmdir(dirname);

if (unlinkResult == -1) {

perror("Error deleting the directory");

return 1;

}

printf("Directory deleted successfully: %s\n", dirname);

return 0;

}

Program 3: Closing Standard Input, Output, and Error

#include <unistd.h>

#include <stdio.h>

int main() {

// Close standard input (stdin)

int closeInputResult = close(STDIN\_FILENO);

if (closeInputResult == -1) {

perror("Error closing standard input");

return 1;

}

// Close standard output (stdout)

int closeOutputResult = close(STDOUT\_FILENO);

if (closeOutputResult == -1) {

perror("Error closing standard output");

return 1;

}

// Close standard error (stderr)

int closeErrorResult = close(STDERR\_FILENO);

if (closeErrorResult == -1) {

perror("Error closing standard error");

return 1;

}

printf("Standard input, output, and error closed successfully\n");

return 0;

}

Program 4: Deleting a Symbolic Link

#include <unistd.h>

#include <stdio.h>

int main() {

char symlinkName[] = "example\_symlink";

// Delete the symbolic link

int unlinkResult = unlink(symlinkName);

if (unlinkResult == -1) {

perror("Error deleting the symbolic link");

return 1;

}

printf("Symbolic link deleted successfully: %s\n", symlinkName);

return 0;

}

These examples showcase different scenarios involving the close() and unlink() system calls. You can adapt and modify these programs to suit your specific needs. Remember to handle errors appropriately and check the return values of these system calls for successful execution.