Operating Systems

Lab Manual # 4

UNIX System Calls Programming

Objectives:

To write C programs using the following system calls of the UNIX operating system: `fork`, `exec`, `getpid`, `exit`, `wait`, `close`, `stat`, `opendir`, `readdir`.

Pre-requisites:

- Basic understanding of GCC and C language in Ubuntu.
- Completion of Lab 3.
- Ubuntu operating system (either installed on your computer or in a virtual machine).

Lab Tasks:

Exercise 1: Directory Operations (`opendir, readdir, closedir`)

1. **Start the Program:**
   - The program begins its execution.
2. **Create `struct dirent`:**
   - A structure, `struct dirent`, is defined for representing directory entries.
3. **Declare Variables:**
   - Variables, such as buff (for storing directory name) and a pointer `dptr` of type `struct dirent`, are declared.
4. **Get the Directory Name:**
   - The program prompts the user to input the directory name and stores it in the `buff` variable.
5. **Open the Directory:**
   - Using the `opendir` system call, the program attempts to open the specified directory.
   - If the directory does not exist, an error message is displayed, and the program exits.
6. **Read and Print Directory Contents:**
   - Utilizing the `readdir` system call within a loop, the program reads the contents of the opened directory and prints each entry's name.
7. **Close the Directory:**
   - The program uses the `closedir` system call to close the directory.

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

struct dirent *dptr;

int main(int argc, char *argv[]) {
    char buff[100];
    DIR *dirp;

    printf("\n\n ENTER DIRECTORY NAME: ");
    scanf("%s", buff);

    if ((dirp = opendir(buff)) == NULL) {
        printf("The given directory does not exist");
        exit(1);
    }

    while ((dptr = readdir(dirp)) != NULL) {
        printf("%s\n", dptr->d_name);
    }

    closedir(dirp);
}
```

Exercise 2: Instructions to Run

- Compile the program:

```
$ gcc directory_operations.c -o directory_operations
```

- Execute the program.

```
$ ./directory_operations
```

Explanation:

- This program uses the `opendir` function to open a directory specified by the user.
- It then reads the contents of the directory using the `readdir` function and prints each entry.
- Finally, the program closes the directory using the `closedir` function.

Sample Output:

```
ENTER DIRECTORY NAME: /path/to/directory
file1.txt
file2.txt
subdirectory
...
```

Exercise 2: File Information using `stat`

- To modify the existing program to display detailed information about files using the `stat` system call.
- Include the necessary headers `(#include <sys/stat.h> and #include <unistd.h>)`
- Create a structure of `type struct stat` to store file information.
- For each file in the directory, use the `stat` system call to obtain and print detailed information such as file size, permissions, owner, etc.

c

```c
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include <unistd.h>

struct dirent *dptr;

int main(int argc, char *argv[]) {
    char buff[100];
    DIR *dirp;

    printf("\n\n ENTER DIRECTORY NAME: ");
    scanf("%s", buff);

    if ((dirp = opendir(buff)) == NULL) {
        printf("The given directory does not exist\n");
        exit(1);
    }

    while ((dptr = readdir(dirp)) != NULL) {
        struct stat fileStat;

        // Construct the full file path
        char filePath[200];
        sprintf(filePath, "%s/%s", buff, dptr->d_name);

        // Use stat to obtain file information
        if (stat(filePath, &fileStat) < 0) {
            perror("Error in stat");
            exit(1);
        }

        // Print file information
        printf("File: %s\n", dptr->d_name);
        printf("Size: %lld bytes\n", (long long)fileStat.st_size);
        printf("Permissions: %o\n", fileStat.st_mode & 0777);
        printf("Owner UID: %d\n", fileStat.st_uid);
        printf("Last accessed: %s", ctime(&fileStat.st_atime));
        printf("-------------------------------\n");
    }

    closedir(dirp);
}
```

Exercise 3: File Copy using open, read, write, and close

- To create a program that copies the contents of one file to another using system calls.
- Open the source file for reading using the `open` system call.
- Open or create the destination file for writing using the `open` system call.
- Use `read` and `write` system calls to read from the source file and write to the destination file.
- Close both files using the `close` system call.

c

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

#define BUFFER_SIZE 4096

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s <source_file> <destination_file>\n", argv[0]);
        return 1;
    }

    int sourceFile, destFile;
    char buffer[BUFFER_SIZE];
    ssize_t bytesRead;
    // Open the source file for reading
    sourceFile = open(argv[1], O_RDONLY);
    if (sourceFile < 0) {
        perror("Error opening source file");
        return 1;
    }

    // Open or create the destination file for writing
    destFile = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
    if (destFile < 0) {
        perror("Error opening destination file");
        close(sourceFile);
        return 1;
    }

    // Read from source and write to destination
    while ((bytesRead = read(sourceFile, buffer, BUFFER_SIZE)) > 0) {
        if (write(destFile, buffer, bytesRead) != bytesRead) {
            perror("Error writing to destination file");
            close(sourceFile);
            close(destFile);
            return 1;
        }
    }

    // Close both files
    close(sourceFile);
    close(destFile);

    printf("File copy successful.\n");
    return 0;
}
```

Conclusions:

Congratulations! You've successfully completed this lab, feel free to incorporate these exercises into your lab manual, and encourage students to modify and experiment with the code to deepen their understanding of UNIX system calls and operating system concepts.