Operating Systems

Lab Manual # 6

Inter-Process Communication using Shared File

Objectives:

- To understand and implement interprocess communication through a shared file using the C programming language.
- Learn how to use the fork system call to create a new process and child process.
- Observe and trace the interaction between parent and child processes.

Pre-Requisites:

- Basic understanding of GCC and C language in Ubuntu.
- Completion of Lab 5.
- Ubuntu operating system (either installed on your computer or in a virtual machine).

Lab Tasks # 1:

Exercise 1: Writing and Reading from a Shared File

- In this exercise, you will create a parent process that writes data to a shared file, and a child process that reads data from the same file.

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

#define FILE_PATH "shared_file.txt"

int main() {
    // Variables
    FILE *sharedFile;
    int dataFromParent = 42;
    int dataFromChild = 0;

    // Create or open the shared file
    sharedFile = fopen(FILE_PATH, "w+");
    if (sharedFile == NULL) {
        perror("Error opening the file");
        return 1;
```

```
    }

    pid_t childPid = fork();

    if (childPid == 0) {
        // Child process
        printf("Child Process: Reading data from the file...\n");
        fseek(sharedFile, 0, SEEK_SET);
        fscanf(sharedFile, "%d", &dataFromChild);
        printf("Child Process: Data read from the file: %d\n",
dataFromChild);
    } else if (childPid > 0) {
        // Parent process
        printf("Parent Process: Writing data to the file...\n");
        fseek(sharedFile, 0, SEEK_SET);
        fprintf(sharedFile, "%d", dataFromParent);
        wait(NULL); // Wait for the child process to complete

        // Read data from the file to confirm the child's read
        fseek(sharedFile, 0, SEEK_SET);
        fscanf(sharedFile, "%d", &dataFromChild);
        printf("Parent Process: Data read by the child from the
file: %d\n", dataFromChild);
    } else {
        // Fork failed
        perror("Error in fork");
        return 1;
    }

    // Close the file
    fclose(sharedFile);

    return 0;
```

- }

Instructions to Run

- Compile and run the program.
- Observe how the parent process writes data to the shared file.
- Observe how the child process reads data from the shared file.
- Understand the use of `fork()` for creating child processes.
- Analyze the role of file operations in interprocess communication.
- Open a terminal and navigate to the directory where you saved the program.
- Compile the program using the following command:

```
$ gcc process_creation.c -o process_creation
```

- Execute the program:

```
$ ./process_creation
```

Explanation:

- `sharedFile`: File pointer for the shared file.
- `dataFromParent`: Initial data to be written by the parent.
- `dataFromChild`: Variable to store data read from the file by the child.
- The program opens or creates a file named shared_file_bidirectional.txt in read and write mode `(w+)`. If there's an error opening the file, an error message is printed, and the program exits.

Sample Output:

```
Parent Process: Writing data to the file...
Child Process: Reading data from the file...
Child Process: Data read from the file: 42
Parent Process: Data read by the child from the file: 42
```

Exercise 2: Modify and Experiment

- Modify the program to perform additional actions such as writing an array of integers to the file and reading the array from the file. Ensure that both processes can successfully communicate and share the data.
- Re-compile the program and observe the changes in the output.
- Experiment with different commands in the child process and analyze the results.

Instructions to Run

- Enhance the program to write an array of integers to the shared file in the parent process.
- Modify the child process to read the array from the shared file and display it.
- Ensure proper synchronization using appropriate system calls.

- Open the program in a text editor and make the necessary modifications.
- Save the changes and re-compile the program.

```
$ gcc process_creation.c -o process_creation
```

- Execute the modified program:

```
$ ./process_creation
```

Explanations:

- Through this lab, you have gained hands-on experience in implementing interprocess communication using shared files. This knowledge can be extended to more complex scenarios and different communication mechanisms in real-world applications.

Exercise 3: Home Exploration Task

- Explore additional interprocess communication mechanisms, such as using shared memory or pipes, to achieve communication between parent and child processes.
- Observe how the program handles the error.

Instructions

- Research and implement shared memory or pipe-based communication between parent and child processes.
- Compare and contrast the different interprocess communication mechanisms used in the exercises.
- Identify advantages and disadvantages of each mechanism.
- Document your findings in the lab report.
- Save the changes and re-compile the program.
- Execute the modified program
- Observe the error message and understand how the program responds to the error.

Explanations:

- Give explanations to the above changes.

Lab Tasks # 2: Exercise 1: Bidirectional Communication

- Modify the existing program to enable bidirectional communication between the parent and child processes. In this exercise, both processes will take turns writing and reading data to and from the shared file.

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

#define FILE_PATH "shared_file_bidirectional.txt"

int main() {
    // Variables
    FILE *sharedFile;
    int dataFromParent = 42;
    int dataFromChild = 0;

    // Create or open the shared file
    sharedFile = fopen(FILE_PATH, "w+");
    if (sharedFile == NULL) {
        perror("Error opening the file");
        return 1;
    }

    pid_t childPid = fork();

    if (childPid == 0) {
        // Child process
        for (int i = 0; i < 3; ++i) {
            // Read data from the file
            printf("Child Process: Reading data from the file...\n");
            fseek(sharedFile, 0, SEEK_SET);
            fscanf(sharedFile, "%d", &dataFromChild);
            printf("Child Process: Data read from the file: %d\n",
dataFromChild);

            // Increment and write data to the file
            dataFromChild++;
            printf("Child Process: Writing data to the file...\n");
            fseek(sharedFile, 0, SEEK_SET);
            fprintf(sharedFile, "%d", dataFromChild);
        }
    } else if (childPid > 0) {
        // Parent process
        for (int i = 0; i < 3; ++i) {
            // Write data to the file
            printf("Parent Process: Writing data to the file...\n");
            fseek(sharedFile, 0, SEEK_SET);
            fprintf(sharedFile, "%d", dataFromParent);
```

```
            // Wait for the child process to read and increment data
            wait(NULL);

            // Read and display data from the file
            printf("Parent Process: Reading and displaying data from
the file...\n");
            fseek(sharedFile, 0, SEEK_SET);
            fscanf(sharedFile, "%d", &dataFromChild);
            printf("Parent Process: Data read from the file: %d\n",
dataFromChild);
        }
    } else {
        // Fork failed
        perror("Error in fork");
        return 1;
    }

    // Close the file
    fclose(sharedFile);

    return 0;
}
```

Instructions to Run

- Open a terminal and navigate to the directory where you saved the program.
- Compile the program using the following command:

```
$ gcc process_creation.c -o process_creation
```

- Execute the program:

```
$ ./process_creation
```

Explanation:

- Examine the modified program, which introduces a loop in both the parent and child processes to perform bidirectional communication.
- Compile and run the program to observe the bidirectional communication between the parent and child processes.
- Understand the synchronization and sequencing of actions between the processes.
- Document your observations in the lab report.

Sample Output:

```
Parent Process: Writing data to the file...
Child Process: Reading data from the file...
Child Process: Data read from the file: 42
Child Process: Writing data to the file...
Parent Process: Reading and displaying data from the file...
Parent Process: Data read from the file: 43
...
```

Modify and Experiment

- Explore and implement shared memory as an alternative method for interprocess communication. Modify the program to utilize shared memory segments for data exchange.
- Re-compile the program and observe the changes in the output.
- Experiment with different commands in the child process and analyze the results.

Instructions to Run

- Open the program in a text editor and make the necessary modifications.
- Save the changes and re-compile the program.
- Execute the modified program:

```
$ ./process_creation
```

Handling Errors

- To understand error handling in process creation.
- Intentionally introduce an error in the factorial calculations in the child process (e.g., provide an incorrect integer data value).
- Observe how the program handles the error.

Conclusions

Congratulations! You've successfully completed this lab. Explore and implement shared memory as an alternative method for interprocess communication. Modify the program to utilize shared memory segments for data exchange. This lab equipped students with practical experience in implementing interprocess communication through shared files. The exercises provided a foundation for understanding the principles of IPC, which can be extended to more complex scenarios and diverse communication mechanisms in real-world software development.