GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING FACULTY

CSE 344 SYSTEM PROGRAMMING

HOMEWORK 2 REPORT

ENES AYSU
1901042671

Program Description

This homework assignment involves implementing inter-process communication (IPC) between a parent process and two child processes using FIFOs (named pipes). The parent process creates two FIFOs and sends an array of random numbers along with a command to the child processes. Two child processes are created using the fork() system call, each assigned to one of the FIFOs. The first child process reads the random numbers from the first FIFO, calculates the sum, and writes the result to the second FIFO. The second child process reads the command from the second FIFO, performs the multiplication operation if the command is "multiply", and prints the sum of the results of all child processes on the screen. The parent process sets a signal handler for “SIGCHLD” to handle child process termination and enters a loop, printing a message every two seconds. The signal handler reaps terminated child processes, increments a counter, and exits the program when the counter reaches the number of children originally spawned. Error handling is implemented throughout the program to handle scenarios such as failure to create FIFOs, errors in data/command transmission, and unexpected errors in child processes. Overall, this assignment provides a practical exercise in IPC using FIFOs and demonstrates error handling and process management techniques in a multi-process environment.

Function and Processes

cleanup():

- Purpose: This function is responsible for cleaning up the FIFOs by unlinking them from the filesystem.
- Functionality: It calls `unlink()` for both FIFO1 and FIFO2, removing them from the filesystem.

error_exit(char *msg):

- Purpose: This function prints an error message along with the corresponding error message from `perror()` and exits the program with a failure status.
- Functionality: It prints the provided error message using `perror()` and calls `cleanup()` to clean up resources before exiting with `EXIT_FAILURE`.

sigchld_handler():

- Purpose: This function is a signal handler for the `SIGCHLD` signal, which is sent when a child process terminates.
- Functionality: It reaps terminated child processes using `waitpid()` in a loop. For each terminated child, it prints its process ID and exit status if it exited normally. It increments the `counter` variable to keep track of the number of terminated children.

send_random_numbers(int fd, int *array, int num):

- Purpose: This function sends an array of random numbers to a FIFO.
- Functionality: It writes the contents of the `array` to the FIFO specified by the file descriptor `fd`, with the size of `num` integers multiplied by the size of an integer.

send_command(int fd, const char *command):

- Purpose: This function sends a command to a FIFO2.
- Functionality: It writes the provided command string to the FIFO2 specified by the file descriptor `fd`, including the null terminator.

child_process_1():

- Purpose: This function represents the behavior of the first child process.
- Functionality: It opens FIFO1 for reading, reads random numbers from it, calculates their sum, and writes the sum to FIFO2. Then, it closes both FIFOs and exits.

child_process_2(int size):

- Purpose: This function represents the behavior of the second child process.
- Functionality: It opens FIFO2 for reading, sleeps 2 seconds to allow the child process to write its total result to FIFO2, reads a command from it, checks if the command is "multiply", reads random numbers and summarization result from FIFO2, performs multiplication, prints the summation and multiplication results, and exits.

main(int argc, char *argv[]):

- Purpose: This is the main function of the program.
- Functionality: It parses the command-line argument to determine the size of the random number array. Then, it initializes resources, sets up the `SIGCHLD` signal handler, creates FIFOs, generates random numbers, forks child processes, opens FIFOs for writing, sends data and commands to child processes, waits for child processes to terminate, and cleans up resources before exiting. It also prints "proceeding" messages while waiting for child processes to terminate.

Execution Steps of the Program

- The program starts by parsing command-line arguments to determine the size of the random number array.
- It initializes necessary variables and sets up signal handling for the SIGCHLD signal.
- The parent process creates two FIFOs (named pipes) named FIFO1 and FIFO2 using the mkfifo() function. These FIFOs serve as communication channels between the parent and child processes.
- The parent process generates an array of random numbers of the specified size. These random numbers represent the data that will be transmitted to the child processes.
- Using the fork() system call, the parent process spawns two child processes. Each child process is assigned to one of the FIFOs for communication.
- The parent process sends the array of random numbers to Child Process 1 via FIFO1 and sends the command ("multiply") to Child Process 2 via FIFO2.
- Each child process sleeps for 10 seconds to simulate work.
- Child Process 1 reads the random numbers from FIFO1, calculates their sum, and writes the sum to FIFO2.
- Child Process 2 reads the command from FIFO2, performs the multiplication operation, and prints the result.
- The parent process sets up a signal handler for the SIGCHLD signal to handle the termination of child processes.
- When a child process terminates, the signal handler is triggered, and the parent process reaps the terminated child process using the waitpid() function.
- The parent process enters a loop, periodically printing a message indicating that it is "proceeding."
- Once all child processes have terminated and been reaped, the parent process performs cleanup by removing the FIFOs.

Tests

For Compilation

$ make

$ ./hw2 <desired array size>

All Executions

```
FIFOs created successfully
4 0 8 9 1 7 8 1 5 5
Child process 1 created successfully
Child process 1 started
Child process 2 created successfully
Child process 2 started
FIFOs opened successfully
Random number is sent to 3
Multiply command is sent to 4
Random number is sent to 4
Proceeding...
Summation result is written into FIFO2
Child process 1 completed
Child process 29775 exited with status 0
Proceeding...
Command: multiply
Summation result: 48
Multiplication result: 0
Child process 2 completed
Child process 29776 exited with status 0
Parent process completed successfully
```

Parent Process Creates FIFOs and Child Processes

```
FIFOs created successfully
4 0 8 9 1 7 8 1 5 5
Child process 1 created successfully
Child process 1 started
Child process 2 created successfully
Child process 2 started
```

Generated Random Numbers is Sent Into Both FIFOs

```
Random number is sent to 3
```

```
Multiply command is sent to 4
Random number is sent to 4
```

Child1 Does Its Operation and Exits

```
Summation result is written into FIFO2
Child process 1 completed
Child process 29775 exited with status 0
```

Child2 Does Its Operation and Exits

```
Command: multiply
Summation result: 48
Multiplication result: 0
Child process 2 completed
Child process 29776 exited with status 0
```

Summary

This project showcases a practical implementation of IPC using FIFOs in a multi-process environment. By demonstrating the creation of FIFOs, forking of child processes, data transmission, signal handling, and error management, the project provides valuable insights into IPC concepts and techniques.

Overall, the project serves as an educational resource for understanding and implementing IPC mechanisms using FIFOs, while also addressing advanced topics such as zombie process protection and exit status reporting.