GEBZE TECHNICAL UNIVERSITY

CSE 344
SYSTEM PROGRAMMING
Homework #2 Report

BARIŞ YURDAKUL
1801042620

1.Problem Definition

This project aims to develop a communication protocol using FIFOs (named pipes) to facilitate inter-process communication (IPC) between two distinct processes. The communication protocol is designed to manage data exchange between a parent process and two child processes.

The parent process will take an integer argument and perform operations based on this argument. Two FIFOs will be created, each serving a different purpose. The first FIFO will receive a random number array and transmit it from one child process to another. The second FIFO will transmit a request for a multiplication operation between child processes.

The two child processes will perform different operations based on the data and commands received from the parent process. The first child process will sum the received random numbers and send the result to the second child process. The second child process will perform the multiplication operation specified by the command received from the parent process and print the result to the screen.

The parent process will wait for all child processes to complete and manage their completion using a signal handler. Additionally, it will check the exit statuses of all child processes to determine the success of the program execution and report whether the program completed successfully or encountered errors.

2.Implementation

The architecture considered for this assignment will be as follows.

This code implements a system using multiple processes to calculate the sum and multiplication of an array. It utilizes named pipes (FIFOs) for communication between the parent process and child processes.

Key Components:

Header Files:

- <stdio.h>: Standard input/output operations (printf, write, etc.)
- <stdlib.h>: General utility functions (exit, rand, etc.)
- <unistd.h>: Unix-like standard functions (fork, sleep, etc.)
- <sys/types.h>: System data types (pid_t)
- <sys/wait.h>: Process waiting primitives (waitpid)
- <string.h>: String manipulation functions (strcmp, strcpy, etc.)
- <fcntl.h>: File control operations (open)
- <signal.h>: Signal handling (sigaction, sigchld_handler)
- <sys/stat.h>: File system statistics (mkfifo)
- <time.h>: Time-related functions (time, srand)
- <errno.h>: System error codes (perror)

Constants:

- FIFO_1, FIFO_2: Names of the FIFOs used for communication
- MAX_SIZE: Maximum size for data written to FIFOs.*I used static size for all arrays.*
- MULTIPLY: Command string used to indicate multiplication operation(“multiply”)
- command_size: Size of the command string

Functions:

- generate_random_number(): Generates a random number between 10000 and 90000 (unused in final implementation because it is decided to use fifo1 and fifo2 names)
- sigchld_handler(int signal): Handles SIGCHLD and SIGINT signals.

For SIGCHLD, it waits for child processes to terminate, keeps track of exited children, and closes FIFOs when all children exit.

For SIGINT, it terminates the program and cleans up FIFOs.It is used for CTRL+C interruption.

```c
// Function to handle SIGCHLD an SIGINT signal
void sigchld_handler(int signal) {
    pid_t pid;
    int status;
    if(signal == SIGCHLD){
        while ((pid = waitpid( pid: -1,  stat_loc: &status,  options: WNOHANG)) > 0) {
            printf( format: "SIGCHLD received. pid: %d\n", pid);
            exited_children++;
            child_count--;
            printf( format: "Exited Children : %d\n", exited_children);
            char msg[100];
            sprintf( s: msg,  format: "Child process %d exited with status %d\n", pid, status);
            write( fd: STDOUT_FILENO,  buf: msg,  n: strlen( s: msg));

        }
        if(exited_children == 2){
            // Close FIFOs
            unlink( name: FIFO_1);
            unlink( name: FIFO_2);

            exit( status: EXIT_SUCCESS);
        }
    }
    if(signal == SIGINT){
        printf( format: "\nKilled PID : %d", getpid());
        unlink( name: FIFO_1);
        unlink( name: FIFO_2);
        exit( status: EXIT_SUCCESS);
    }
```

Main Function:

- Signal Handling:

Sets up a signal handler (sigchld_handler) to handle SIGCHLD (child process termination) and SIGINT (Ctrl+C interruption).

```
struct sigaction sa;
sa.sa_handler = sigchld_handler;
sa.sa_flags = 0;
sigfillset( set: &sa.sa_mask);
const int TERM_SIGNALS[2] = { [0]: SIGCHLD, [1]: SIGINT};

for (int i = 0; i < 2; i++) {
    if (sigaction( sig: TERM_SIGNALS[i], act: &sa, oact: NULL) == -1) {
        perror( s: "Error registering signal handler");
        exit( status: EXIT_FAILURE);
    }
}
```

This code snippet sets up a signal handler function (sigchld_handler) to be called whenever a SIGCHLD (child process termination) or SIGINT (Ctrl+C) signal is received. It also blocks all other signals while the handler is executing.

- Argument Validation:

Checks if exactly one command-line argument is provided. If not, it displays usage information and exits.

- Array Size:

Converts the command-line argument to an integer representing the size of the array.

```
// Create FIFOs if they don't exist
if (mkfifo( path: FIFO_1,  mode: 0666) < 0 && errno != EEXIST) {
    perror( s: "mkfifo");
    exit( status: 1);
}
if (mkfifo( path: FIFO_2,  mode: 0666) < 0 && errno != EEXIST) {
    perror( s: "mkfifo");
    exit( status: 1);
}
```

- FIFO’s

Creating Fifos using mkfifo, error check if it exists.

- Fork Child Processes:

Creates two child processes using fork().

In the child process:

Prints a message indicating the child process ID (PID).

Child process 1 (Summation):

- Sleeps for 10 seconds (can be removed for faster execution).
- Reads the random array from FIFO_1.

```
int fifo1_w_parent = open( file: FIFO_1,  oflag: O_WRONLY);
char array msg[200];
```

It would not impossible if it opened on parent process in write mode.

For using every fifo, you should open both sides.Otherwise data transmission can not happen.

- Calculates the sum of the array elements.
- Writes the sum to FIFO_2.

Child process 2 (Multiplication):

- Sleeps for 10 seconds.
- Reads the sum and command string from FIFO_2.
- Verifies the command is "MULTIPLY".

- Reads the random array from the combined string received in FIFO_2.
- Calculates the product of the array elements.
- Prints the sum, product, and their sum (total result).

In the parent process:

Sleeps for a while (can be removed for faster execution).

Prints the initialized random array.

Writes the random array to FIFO_1.

Creates a combined string containing the "multiply" command followed by the elements of the random array.

```
int fifo2_w_parent = open( file: FIFO_2,  oflag: O_WRONLY);
char random_number_string[ARRAY_SIZE + 1];
char new_string[command_size + ARRAY_SIZE + 1];

strcpy( dest: new_string,  src: MULTIPLY);
//printf("new string : %s\n", new_string);
for (int i = 0; i < ARRAY_SIZE; i++) {
    snprintf( s: random_number_string,  maxlen: ARRAY_SIZE + 1,  format: "%1d", random_array[i]);
    strcat( dest: new_string,  src: random_number_string);
}
//printf("combined string in parent : %s\n", new_string);


if (write( fd: fifo2_w_parent,  buf: new_string,  n: sizeof(new_string)) < 0) {
    perror( s: "Not Writen Fifo2 In Parent");
    exit( status: 1);
}
```

Writes the combined string to FIFO_2.

- Wait for Child Processes:

The parent process waits for all child processes to terminate using waitpid.

3.Results and Test Cases

>./main 10

```
baris@baris-Ubuntu:~/Desktop/system_hw2$ ./main 10
FIFO Name 1: fifo_1
FIFO Name 2: fifo_2
Child Process (PID: 38469): Created
Child Process (PID: 38470): Created
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Array initialized: [1, 4, 9, 0, 5, 6, 3, 7, 3, 9,]
Sum of Array : 47
Multiply of Array : 0
Total Result : 47
SIGCHLD received. pid: 38469
Exited Children : 1
Child process 38469 exited with status 0
SIGCHLD received. pid: 38470
Exited Children : 2
Child process 38470 exited with status 0
```

Completed in 10 seconds as we expected.We saw 5 proceeding printing.Parent and 2 childs syncronized perfectly and works at the same time.

Our handler works fine and kills all children.

```
baris@baris-Ubuntu:~/Desktop/system_hw2$ ./main 10
FIFO Name 1: fifo_1
FIFO Name 2: fifo_2
Child Process (PID: 38633): Created
Child Process (PID: 38634): Created
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Array initialized: [5, 6, 3, 2, 3, 6, 8, 3, 2, 5,]
Sum of Array : 43
Multiply of Array : 777600
Total Result : 777643
SIGCHLD received. pid: 38633
Exited Children : 1
Child process 38633 exited with status 0
SIGCHLD received. pid: 38634
Exited Children : 2
Child process 38634 exited with status 0
```

There is no problem if array does not contain zero.

>./main 0

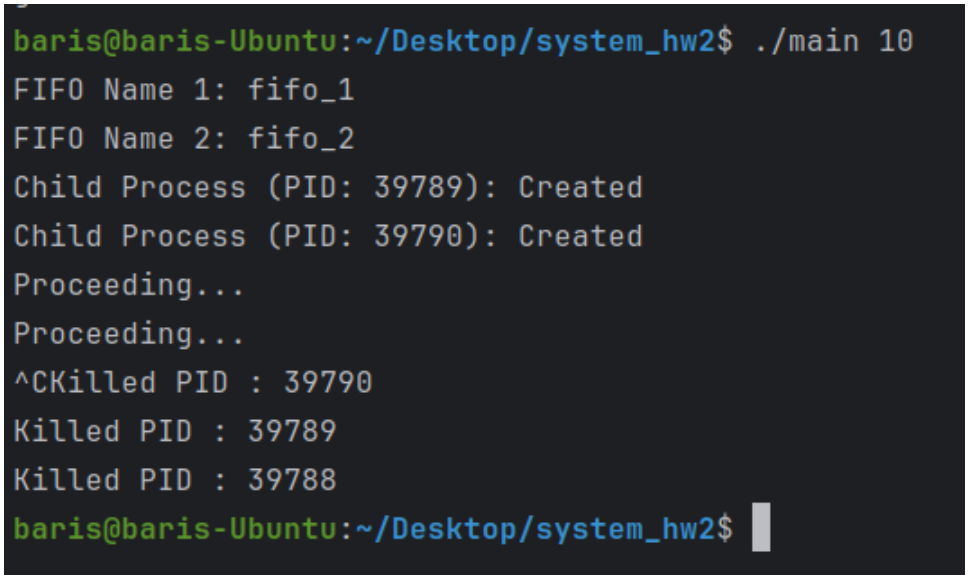
```
baris@baris-Ubuntu:~/Desktop/system_hw2$ ./main 0
FIFO Name 1: fifo_1
FIFO Name 2: fifo_2
Child Process (PID: 39106): Created
Child Process (PID: 39105): Created
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Array initialized: ]
Command is not 'multiply'
: Success
SIGCHLD received. pid: 39105
Exited Children : 1
Child process 39105 exited with status 0
SIGCHLD received. pid: 39106
Exited Children : 2
Child process 39106 exited with status 256
```

Our program does not work fine if we give array size 0 as we see.

> If command is “1multiply”

```
baris@baris-Ubuntu:~/Desktop/system_hw2$ ./main 10
FIFO Name 1: fifo_1
FIFO Name 2: fifo_2
Child Process (PID: 39343): Created
Child Process (PID: 39344): Created
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Proceeding...
Array initialized: [5, 9, 6, 8, 6, 5, 6, 6, 9, 1,]
Command is not 'multiply'
: Success
SIGCHLD received. pid: 39343
Exited Children : 1
Child process 39343 exited with status 0
SIGCHLD received. pid: 39344
Exited Children : 2
Child process 39344 exited with status 256
```

> Ctrl+C situation



Our handler catch SIGINT signal and killed our parent and 2 child.Also unlinks our fifo’s.