Gebze Technical University

Department of Computer Engineering

CSE344 System Programming

Homework 2 Report

Burak Yıldırım
1901042609

Contents

1 Introduction

1.1 Project Description

The project focuses on interprocess communication (IPC) in C, emphasizing the use of **FIFOs** to facilitate robust interaction between a parent process and two child processes. It underscores process creation using the **fork()** system call, where the parent process initiates two FIFOs for sending data and commands to the child processes, which perform arithmetic operations and communicate results. The system is designed with thorough error handling and incorporates a signal handler to manage child process terminations effectively.

1.2 Compilation

```
CC = gcc
CFLAGS = -w
DFLAGS = -g
INCDIR = include
SRCDIR = src
TARGET = main
SOURCES = $(wildcard $(SRCDIR)/*.c) $(wildcard $(SRCDIR)/**/*.c)
HEADERS = $(wildcard $(INCDIR)/*.h) $(wildcard $(INCDIR)/**/*.h)
VALGRIND_OPTIONS = --leak-check=full --show-leak-kinds=all --track-origins=yes

all: $(TARGET)

$(TARGET): $(SOURCES) $(HEADERS)
        $(CC) $(CFLAGS) -o $@ $(SOURCES) -I$(INCDIR)

debug: CFLAGS += $(DFLAGS)
debug: $(TARGET)

clean:
        rm -f $(TARGET) fifo_*

valgrind: debug
        valgrind $(VALGRIND_OPTIONS) ./$(TARGET)
```

The provided Makefile automates the build process: executing **make** compiles the project, **make clean** removes the main executable and the FIFOs, and **make valgrind** launches the program in Valgrind for memory leak checks, streamlining development and debugging.

2 Implementation

2.1 Utility

2.1.1 I/O

2.1.1.1 Macros

- `FILE_PERMS`: Sets the default file permissions to 0666 when creating a new file, allowing read and write operations for the user, group, and others.
- `MAX_OUTPUT`: Sets the maximum output size to 256 characters, establishing the buffer size for output operations.

2.1.1.2 Functions

my_fgets

```
/**
 * Reads a line from a file descriptor into a buffer, handling files and stdin.
 * Prints an error and sends a SIGUSR2 signal to the parent process on error.
 *
 * @param buffer The buffer to store the read data.
 * @param size The buffer size including null terminator.
 * @param fd File descriptor for reading.
 * @return Pointer to the buffer with the read line, or NULL on EOF.
 */
char *my_fgets(char *buffer, size_t size, int fd);
```

my_open

```
/**
 * Opens a file with specified flags and mode. Prints an error and sends a SIGUSR2
 * signal to the parent process on error.
 *
 * @param filename The name of the file to open.
 * @param flags Flags to control how the file is opened.
 * @param mode Permissions to set if a new file is created.
 * @return The file descriptor for the opened file.
 */
int my_open(const char *filename, int flags, mode_t mode);
```

my_printf

```
/**
 * Formats a string and writes it to the stdout. Prints an error and sends a SIGUSR2
 * signal to the parent process on error.
 *
 * @param format The format string.
 * @param ... Additional arguments for the format string.
 * @return The number of bytes written.
 */
ssize_t my_printf(const char *format, ...);
```

my_read

```
/**
 * Reads data from a file descriptor into a buffer. Prints an error and sends a SIGUSR2
 * signal to the parent process on error.
 *
 * @param fd The file descriptor from which to read.
 * @param buffer The buffer into which the data will be read.
 * @param size The maximum number of bytes to read.
 * @return The number of bytes actually read, or -1 if an error occurs.
 */
ssize_t my_read(int fd, void *buffer, size_t size);
```

my_vprintf

```
/**
 * Formats a string using a va_list and writes it to the stdout. Prints an error
 * and sends a SIGUSR2 signal to the parent process on error.
 *
 * @param format The format string.
 * @param args A va_list of arguments for the format string.
 * @return The number of bytes written.
 */
ssize_t my_vprintf(const char *format, va_list args);
```

my_write

```
/**
 * Writes data from a buffer to a file descriptor. Prints an error and sends a SIGUSR2
 * signal to the parent process on error.
 *
 * @param fd The file descriptor to which the data will be written.
 * @param buffer The buffer containing the data to write.
 * @param size The number of bytes to write.
 * @return The number of bytes actually written.
 */
ssize_t my_write(int fd, const void *buffer, size_t size);
```

my_close

```
/**
 * Closes a file descriptor. Prints an error and sends a SIGUSR2 signal to the parent
 * process on error.
 *
 * @param fd The file descriptor to close.
 */
void my_close(int fd);
```

2.1.2 Type

2.1.2.1 Macros

- `bool`: Defines an alias for the **`int`** type. This macro introduces a boolean type into the program.
- `true`: Sets the boolean value **`true`** to the integer **1**.
- `false`: Sets the boolean value **`false`** to the integer **0**.

2.1.3 IPC

2.1.3.1 Macros

- `FIFO_PERMS`: Sets the default fifo permissions to 0666 when creating a new fifo, allowing read and write operations for the user, group, and others.

2.1.3.2 Functions

my_mkfifo

```
/**
 * Creates a FIFO at the specified path with the given mode. Prints an error and
 * exits on error.
 *
 * @param path The pathname for the FIFO file.
 * @param mode The permissions to be set on the FIFO file.
 */
 void my_mkfifo(const char *path, mode_t mode);
```

2.1.4 String

2.1.4.1 Functions

is_integer

```
/**
 * Checks if a string represents a valid integer.
 *
 * @param str The string to check.
 * @return 1 if the string is an integer, 0 otherwise.
 */
int is_integer(const char *str);
```

my_strcmp

```
/**
 * Compares two strings lexicographically.
 *
 * @param str1 The first string to compare.
 * @param str2 The second string to compare.
 * @return An integer less than, equal to, or greater than zero if str1 is found,
 *         respectively, to be less than, to match, or be greater than str2.
 */
int my_strcmp(const char *str1, const char *str2);
```

parse_int

```c
/**
 * Parses a string and converts it to an integer.
 *
 * @param str The string to convert to an integer.
 * @return The converted integer or INT_MIN if the conversion is not possible.
 */
int parse_int(const char *str);
```

my_strlen_utf8

```c
/**
 * Calculates the length of a UTF-8 encoded string, counting each multi-byte
 * character as a single character.
 *
 * @param str The UTF-8 encoded string to calculate the length of.
 * @return The length of the string in terms of UTF-8 characters.
 */
size_t my_strlen_utf8(const char *str);
```

my_vsprintf

```c
/**
 * Formats a string and stores it in a buffer using a va_list.
 *
 * @param buffer The buffer to store the formatted string.
 * @param size The size of the buffer.
 * @param format The format string.
 * @param args A va_list of arguments to format.
 * @return The number of bytes written to the buffer.
 */
size_t my_vsprintf(char *buffer, size_t size, char *format, va_list args);
```

2.1.5 Array

2.1.5.1 Functions

indexof_char

```
/**
 * Searches for a character value in an array and returns its index.
 *
 * @param arr The array to search.
 * @param size The size of the array.
 * @param value The character value to search for.
 * @return The index of the first occurrence of the value in the array, or -1 if
 *         not found.
 */
int indexof_char(char *arr, size_t size, char value);
```

2.2 Main

2.2.1 Macros

- `CMD_SIZE`: Defines the size of the command, set to 9.
- `MAX_INPUT_LEN`: Defines the maximum length of the input string, set to 16.
- `MAX_INPUT_VALUE`: Defines maximum integer value of the input, set to 10.
- `SLEEP_TIME`: Defines the sleep time of the child processes, set to 10 seconds.
- `TERM_SIGS_NUM`: Represents the number of termination signals, set to 6.

2.2.2 Constraints

The length of the random numbers array is limited to 10, as defined by **`MAX_INPUT_VALUE`**, due to the size limits of the **`long`** data type. The numbers in the array are generated using modulus 100. If by any chance the numbers end up closer to 100, the result of the multiplication exceeds the maximum size of the **`long`** type, resulting in overflow. Data transmissions between the parent process and the child processes are done before the child processes sleep for two reasons:

- If the parent opens the FIFOs with **`O_WRONLY`**, it would be blocked for the duration of the sleep because the read ends of the FIFOs would not be open until after the sleep.
- If the parent opens the FIFOs with **`O_RDWR`** to prevent blocking, the child processes would not read the data sent by the parent because the parent and the children would not be synchronized. The parent would write data to the FIFOs before the child processes are ready to read it. After the sleep, the data streams would not be accessible, causing the child processes to be blocked indefinitely.

If an interrupt happens before all the data transmissions are done, a **SIGUSR1** signal is sent to the child processes by the parent process and all the processes are terminated. If a terminating error happens in one child process, it sends a **SIGUSR2** signal to the parent process which sends a **SIGUSR2** signal to the child processes and all the processes are terminated.

2.2.3 Structs

- `FifoData`: A structure representing a FIFO data, which includes:
  - `int value[MAX_INPUT_VALUE]`: An integer array containing the random numbers.
  - `int length`: An integer representing the number of elements in the value array.

2.2.4 Functions

child_1

```
/**
 * This function models the child process 1. It reads the random numbers from FIFO 1,
 * sleeps for 10 seconds, sums the numbers, and writes the result to FIFO 2.
 */
 void child_1();
```

child_2

```
/**
 * This function models the child process 2. It reads the "multiply" command and the
 * random numbers from FIFO 2, sleeps for 10 seconds, multiplies the numbers, reads
 * the sum result from child 1 from FIFO 2, then sums and prints the final result.
 */
 void child_2();
```

cleanup

```
/**
 * Performs cleanup operations by removing FIFOs if the current process is the parent.
 * This ensures that any FIFOs created during the process are deleted at the end of
 * the process' lifecycle.
 */
 void cleanup();
```

sigchld_handler

```
/**
 * Handles SIGCHLD signals to manage child process terminations. This function
 * continuously checks for terminated child processes, prints their exit status,
 * and terminates the parent process after handling two child terminations.
 *
 * @param signo The signal number, expected to be SIGCHLD.
 */
void sigchld_handler(int signo);
```

sigusr1_handler

```
/**
 * Handles SIGUSR1 signals by terminating the process that invokes this handler.
 * This function prints a message indicating that the process is terminating
 * due to incomplete data transmission and exits successfully.
 *
 * @param signo The signal number, expected to be SIGUSR1.
 */
void sigusr1_handler(int signo);
```

sigusr2_handler

```
/**
 * Handles SIGUSR2 signals by sending SIGUSR2 signal to child processes in parent
 * process and terminating the process that invokes this handler in child processes.
 * This function prints a message indicating that the process is terminating
 * due to an error and exits successfully.
 *
 * @param signo The signal number, expected to be SIGUSR2.
 */
void sigusr2_handler(int signo);
```

term_sigs_handler

```
/**
 * Handles termination signals by either waiting for child processes to finish or
 * sending a SIGUSR1 signal to child processes if any FIFO data transmissions are
 * incomplete. The function then exits successfully.
 *
 * @param signo The signal number received.
 */
void term_sigs_handler(int signo);
```

main

```
/**
 * The main function initializes signal handling, manages child processes, and
 * handles inter-process communications via FIFOs. It sets up a command-line
 * interface allowing user input to specify a number, n, validates this number, and
 * generates an integer array of random numbers of length n. This array is transmitted
 * to the child processes using FIFOs. The function also sets up a loop that prints
 * "proceeding" every 2 seconds and ensures proper cleanup at program termination.
 *
 * - Signal Handling Setup: Registers handlers for termination signals, SIGCHLD,
 *                          SIGUSR1, and SIGUSR2.
 * - Cleanup: Registers a cleanup function with atexit for FIFO deletion.
 */
 int main();
```

3 Testing

3.1 Expected Behavior

```
Enter a number: 10
Child 1 with PID 9018 is started.
Child 2 with PID 9019 is started.
Numbers: 15, 74, 39, 35, 38, 45, 86, 80, 77, 63
proceeding
proceeding
proceeding
proceeding
proceeding
Result: 8647119352872055 2
Child with PID 9018 has terminated with status 0.
proceeding
Child with PID 9019 has terminated with status 0.
```

3.2 Interrupt Signal Handling

```
Enter a number: 10
Child 2 with PID 9022 is started.
Child 1 with PID 9021 is started.
Numbers: 73, 2, 2, 65, 60, 48, 49, 33, 47, 8
proceeding
^CResult: 33234301901187
Child with PID 9021 has terminated with status 0.
Child with PID 9022 has terminated with status 0.
```

3.3 Incomplete Data Transmission

```
Enter a number: 10
Child 1 with PID 4169 is started.
Child 2 with PID 4170 is started.
Child with PID 4169 has terminated with status 0 due to incomplete data transmission.
Child with PID 4170 has terminated with status 0 due to incomplete data transmission.
```

3.4 A Child Encountering a Terminating Error

```
Enter a number: 10
Child 2 with PID 4360 is started.
Child 1 with PID 4359 is started.
Numbers: 86, 43, 38, 18, 76, 6, 75, 84, 94, 65
proceeding
Child with PID 4359 has terminated abnormally with status 1 due to an error.
Child with PID 4360 has terminated abnormally with status 1 due to an error.
```

3.5 FIFO Exists

```
burakyildirim@ubuntu:~/Desktop/Hw2$ mkfifo fifo_1
burakyildirim@ubuntu:~/Desktop/Hw2$ make
gcc -w -o main src/main.c src/utils/io.c src/utils/string.c src/utils/ipc.c src/utils/array.c -Iinclude
burakyildirim@ubuntu:~/Desktop/Hw2$ ./main
Enter a number: 2
FIFO creation failed: File exists
```

3.6 Non-Integer Input

```
Enter a number: deneme
Invalid input. Please enter a valid number.
```

3.7 Out-of-Range Negative Input

```
Enter a number: -1
Invalid input. Please enter a number between 1 and 10.
```

3.8 Out-of-Range Positive Input

```
Enter a number: 100
Invalid input. Please enter a number between 1 and 10.
```