FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ

Student:

Name: Furkan

Surname: AK

Department: Computer Engineering

Project:

Topic: Distributed Calculator Project

Content

Pages

FIGURES

Pages

1- Project Topic

The aim of this project is to develop a calculator system that performs mathematical operations. The system provides a continuously running structure using Named Pipes (FIFO) for communication between the parent process and child processes. The project achieves the following objectives:

This project focuses on the following topics:

1- **Mathematical Operations:** Addition, subtraction, multiplication, and division are performed by separate child processes.
2- **Inter-Process Communication (IPC):** Data transfer between the parent process and child processes is carried out using FIFOs.
3- **Continuously Running Child Processes:** Child processes are started once and remain active, waiting for data until the program terminates.
4- **Result Logging:** Results from the child processes are sent to the saver program and logged in a file.
5- **Process Management:** Fork and execlp are used to correctly create and manage child processes.

2- Tasks Completed During the Project

2.1 Parent Process (calculator.c)

Tasks in the Parent Process:

- Accepts user input for operation selection and numbers.
- Sends the entered numbers to the corresponding child process through FIFOs.
- Receives results from the child processes via FIFOs and displays them.
- Cleans up all FIFOs when the program exits.

Key Features:

- **Menu Management:** The user selects an operation, and the system sends data to the corresponding child process.
- **Communication with Continuously Running Child Processes:** Each mathematical operation uses a separate FIFO for data transfer.
- **FIFO Management:** Separate FIFOs (calc_fifo_addition, calc_fifo_subtraction, etc.) are used for each operation, and a result FIFO (calc_fifo_result) is used to receive results.

Code Workflow:

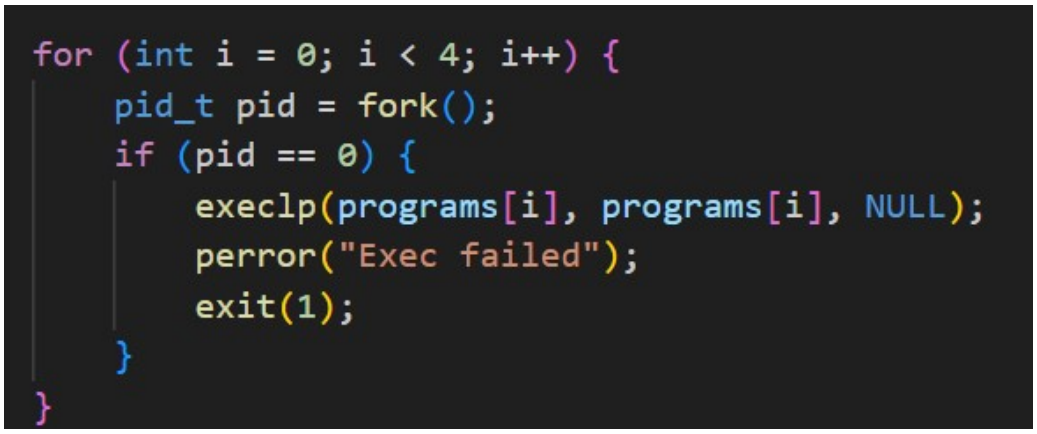
1- **Menu Selection and Input:**
   - The user selects an operation and enters two numbers.
   - These numbers are sent to the corresponding FIFO.

2- **Result Reading:**
   - The child process reads the data, performs the operation, and sends the result back through the result FIFO.

3- **FIFO Cleanup:**
   - All FIFO files are unlinked when the program exits.

Continuously Running Structure:

- The parent process creates child processes only once using fork.
- Child processes remain active, waiting for input data after being started with execlp.



```c
for (int i = 0; i < 4; i++) {
    pid_t pid = fork();
    if (pid == 0) {
        execlp(programs[i], programs[i], NULL);
        perror("Exec failed");
        exit(1);
    }
}
```

Figure 1 Creating child processes with fork and synchronizing parent-child processes using wait was learned.

2.2 Child Process

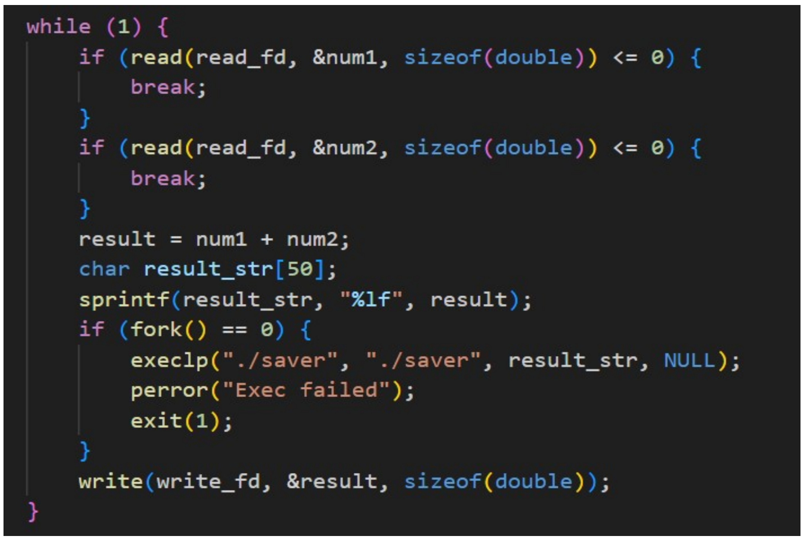
Each mathematical operation is handled by a separate child process (addition.c, subtraction.c, multiplication.c, division.c).

Tasks in Child Processes:

- Each child process reads two numbers from its specific FIFO (calc_fifo_addition, etc.).
- It performs the mathematical operation and writes the result to the calc_fifo_result.
- The result is also sent to the saver program as a parameter for logging.

Workflow:

- The child process reads input data from the FIFO.
- It performs the operation (e.g., addition).
- The result is sent to the saver program as a parameter and written to the result FIFO.
- The child process continues waiting for the next input without terminating.



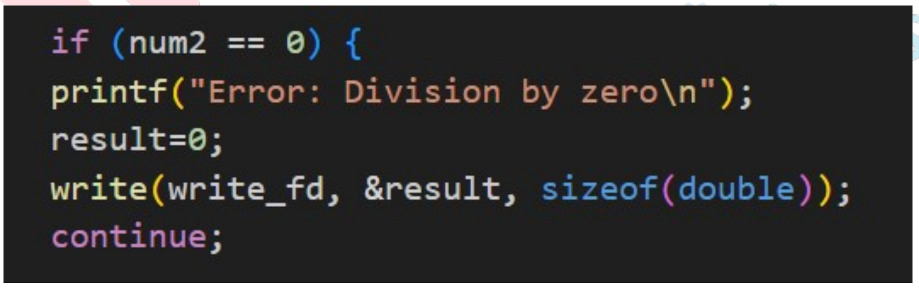
```
while (1) {
    if (read(read_fd, &num1, sizeof(double)) <= 0) {
        break;
    }
    if (read(read_fd, &num2, sizeof(double)) <= 0) {
        break;
    }
    result = num1 + num2;
    char result_str[50];
    sprintf(result_str, "%lf", result);
    if (fork() == 0) {
        execlp("./saver", "./saver", result_str, NULL);
        perror("Exec failed");
        exit(1);
    }
    write(write_fd, &result, sizeof(double));
}
```

Figure 2 Mathematical Operations in Subprocesses and Writing to the Saver Program

Special Cases:

Division (division.c):

- If the divisor is 0, an error message is displayed, and the result is set to 0.
- The program continues without blocking.



```
if (num2 == 0) {
printf("Error: Division by zero\n");
result=0;
write(write_fd, &result, sizeof(double));
continue;
```

Figure 3 Division by Zero Error Handling

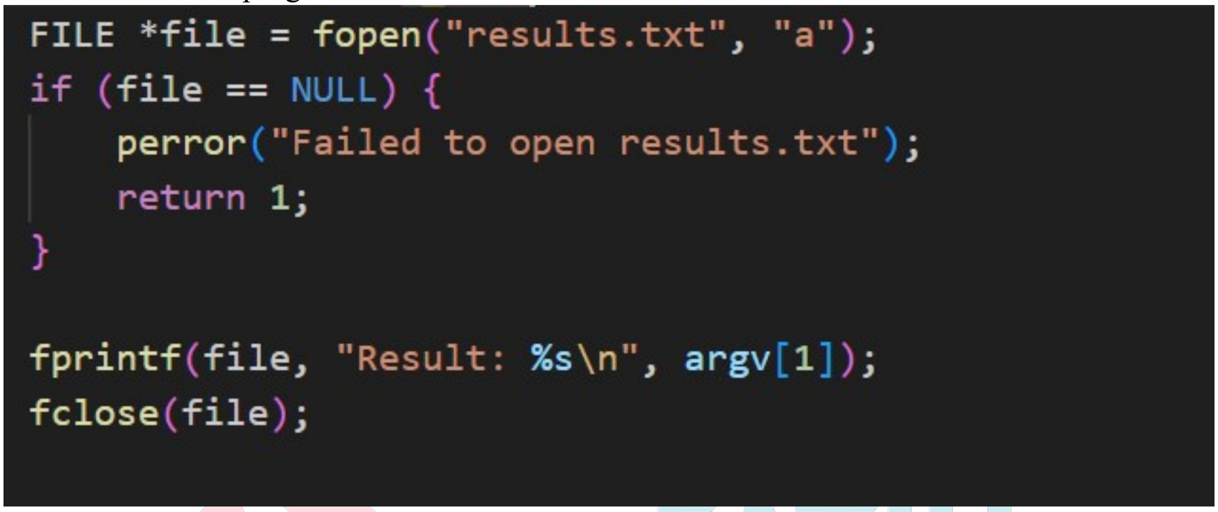
2.3 Saver

Responsibilities of the Saver Program:

- Logs the results from child processes into results.txt.
- Takes the result as a parameter and writes it to the file.

Workflow:

- The child process calls the saver program using execlp.
- The saver program writes the received result into results.txt.



```
FILE *file = fopen("results.txt", "a");
if (file == NULL) {
    perror("Failed to open results.txt");
    return 1;
}

fprintf(file, "Result: %s\n", argv[1]);
fclose(file);
```

Figure 4 Creating results.txt and Appending Results

Error Handling:

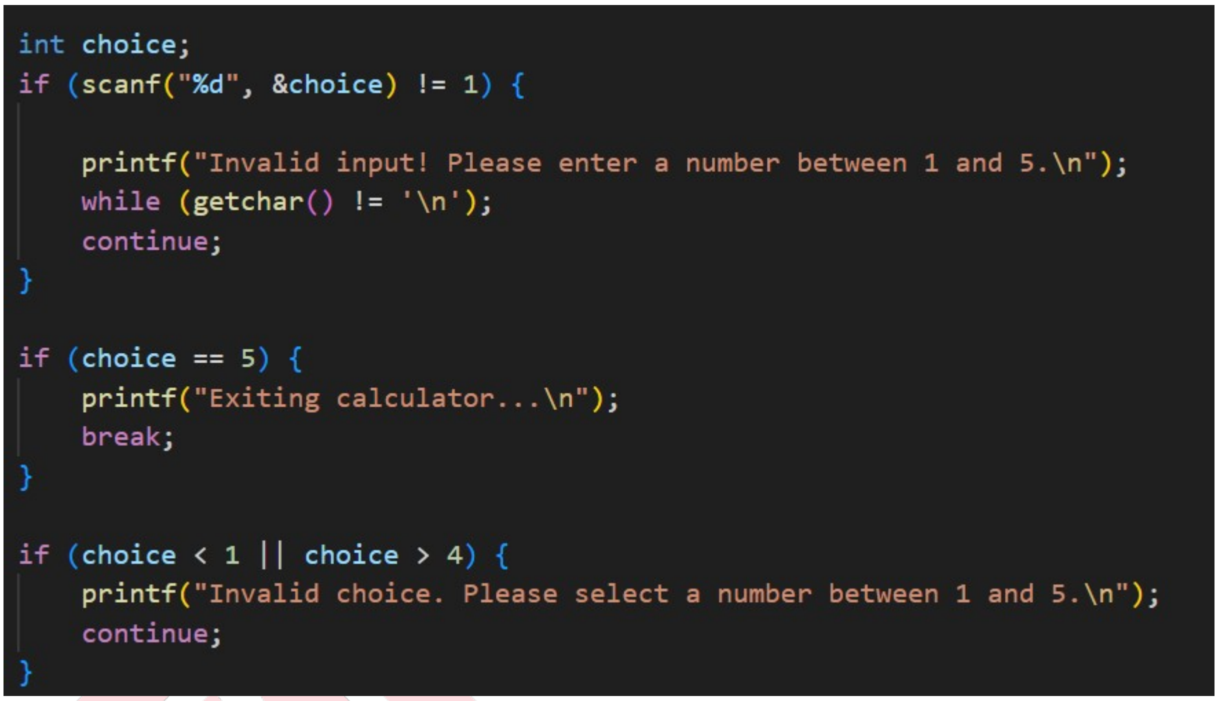
- If the file cannot be opened, an error message is displayed, and the process terminates.

2.4 Makefile

A Makefile has been created to automate the compilation of the programs. After the make command, the program can be executed using make run, which compiles the code and runs the ./calculator executable. Additionally, the make clean command can be used to delete the results.txt file or any compiled programs.

2.5 Error Handling

In the Calculator program, the user's input in the terminal menu is checked to determine whether it is a number or a letter. If a letter is entered instead of the numbers in the menu, a warning message is displayed, and the program returns to the menu. Similarly, if the required numbers for the operation are entered as letter characters, a warning is displayed.



```c
int choice;
if (scanf("%d", &choice) != 1) {

    printf("Invalid input! Please enter a number between 1 and 5.\n");
    while (getchar() != '\n');
    continue;
}

if (choice == 5) {
    printf("Exiting calculator...\n");
    break;
}

if (choice < 1 || choice > 4) {
    printf("Invalid choice. Please select a number between 1 and 5.\n");
    continue;
}
```

Figure 5 Calculator Error Handling

3- Additional Notes

3.1 Challenges Encountered

- FIFO Blocking: Issues arose when one end of the FIFO was not opened in time. Proper synchronization was added to avoid waiting problems.
- Continuously Running Structure: Child processes were restructured to run continuously without reinitializing for each operation.
- Division by Zero: Blocking issues were resolved by displaying an error message and continuing with the program.

3.2 Key Learnings

- Proper management of FIFOs and synchronization between processes.

- Implementing a continuously running structure with minimal overhead.
- Handling edge cases like division by zero and ensuring the program continues functioning smoothly.

4- References

1- Sadi Evren Şeker Operating Systems playlist.

2- Linux Manual Pages: man 2 open, man 2 read, man 2 write, man 2 fork, man 3 execlp

3- TutorialsPoint: Inter-Process Communication in C

4- GeeksforGeeks: File Handling and Named Pipes in C