Assignment 2B

Question

Solution

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>

#define FILE_SIZE (1 * 1024 * 1024 * 1024) /* 1GB */

int main() {
    int fd_fifo1, fd_fifo2;
    pid_t pid;
    struct timeval start_time, end_time;

    /* Create FIFOs */
    if (mkfifo("team1_fifo1", 0666) == -1 || mkfifo("team1_fifo2", 0666) == -1) {
        perror("mkfifo already exists");
        return 1;
    }

    /* Fork a child process */
    pid = fork();

    if (pid == 0) {
        /* Child process (Process 1) */
        gettimeofday(&start_time, NULL);

        /* Open FIFOs */
        fd_fifo1 = open("team1_fifo1", O_RDONLY);
        fd_fifo2 = open("team1_fifo2", O_WRONLY);

        /* Transfer file from Process 1 to Process 2 */
        char buffer[4096];
        ssize_t bytes_read, bytes_written;
        int file_fd = open("/tmp/team1_large_file.txt", O_RDONLY); /* Updated file
path */
        while ((bytes_read = read(file_fd, buffer, sizeof(buffer))) > 0) {
            bytes_written = write(fd_fifo2, buffer, bytes_read);
            if (bytes_written != bytes_read) {
                perror("write");
                return 1;
            }
        }
```

```
        gettimeofday(&end_time, NULL);
        long long elapsed_time = (end_time.tv_sec - start_time.tv_sec) * 1000000LL +
(end_time.tv_usec - start_time.tv_usec);

        printf("Process 1 to Process 2 Transfer Time: %lld microseconds\n",
elapsed_time);

        close(file_fd);
        close(fd_fifo1);
        close(fd_fifo2);
    }
    else if (pid > 0) {
        /* Parent process (Process 2) */
        gettimeofday(&start_time, NULL);

        /* Open FIFOs */
        fd_fifo1 = open("team1_fifo1", O_WRONLY);
        fd_fifo2 = open("team1_fifo2", O_RDONLY);

        /* Transfer file from Process 2 to Process 1 */
        char buffer[4096];
        ssize_t bytes_read, bytes_written;
        int file_fd = open("/tmp/team1_received_file.txt", O_WRONLY | O_CREAT, 0666);
/* Updated file path */
        while ((bytes_read = read(fd_fifo2, buffer, sizeof(buffer))) > 0) {
            bytes_written = write(file_fd, buffer, bytes_read);
            if (bytes_written != bytes_read) {
                perror("write");
                return 1;
            }
        }

        gettimeofday(&end_time, NULL);
        long long elapsed_time = (end_time.tv_sec - start_time.tv_sec) * 1000000LL +
(end_time.tv_usec - start_time.tv_usec);

        printf("Process 2 to Process 1 Transfer Time: %lld microseconds\n",
elapsed_time);

        close(file_fd);
        close(fd_fifo1);
        close(fd_fifo2);

        int compare_result = system("diff -q /tmp/team1_large_file.txt
/tmp/team1_received_file.txt");
        if (compare_result == 0) {
            printf("Sent and received files are the same!\n");
        } else {
            printf("Sent and received files are different.\n");
        }

        /* Clean up FIFOs */
```

```
        /* unlink("team1_fifo1"); */
        /* unlink("team1_fifo2"); */
    }
    else {
        perror("fork");
        return 1;
    }

    return 0;
}
```

Explaination

This C program demonstrates interprocess communication between two processes (Process 1 and Process 2) using named pipes (FIFOs). The program transfers a large file from Process 1 to Process 2 and then from Process 2 back to Process 1, measuring the transfer time and checking if the sent and received files are the same. Let's break down the code and explain it step by step:

1. Include Headers:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>
```

- `<unistd.h>`: This header provides access to various POSIX (Portable Operating System Interface) system calls and constants. It's often used for working with file descriptors, process control, and system-related functions such as `fork`, `exec`, and `pipe`.
- `<sys/types.h>`: This header defines various data types used in system calls and library functions. It includes important types like `pid_t` for process IDs and `ssize_t` for signed sizes.
- `<sys/stat.h>`: This header provides data structures and constants for working with file status and file permissions. It includes constants like **`S_IRUSR`** (read permission for the owner) and **`S_IWGRP`** (write permission for the group).
- `<fcntl.h>`: This header is used for manipulating file descriptors and provides constants for various file control options. It's often used in conjunction with functions like `open`, `fcntl`, and `read`.
- `<sys/time.h>`: This header provides data structures and functions for working with time-related operations. In our code, it's used to measure the elapsed time using the `gettimeofday` function.

2. Define Constants:

```
#define FILE_SIZE (1 * 1024 * 1024 * 1024) /* 1GB */
```

This constant defines the size of the file to be transferred, which is set to 1 gigabyte (1GB).

3. Create FIFOs:

```
    if (mkfifo("team1_fifo1", 0666) == -1 || mkfifo("team1_fifo2", 0666) == -1) {
        perror("mkfifo already exists");
        return 1;
    }
```

This code creates two named pipes (FIFOs) named "team1_fifo1" and "team1_fifo2." If these FIFOs already exist, an error message is printed using `perror()`.

4. Fork a Child Process:

```
    pid = fork();
```

This line of code forks the program into two processes: the parent process (Process 2) and the child process (Process 1). The `fork()` function returns `0` in the child process, the child's PID in the parent process, and `-1` on error.

5. Child Process (Process 1):

```
    if (pid == 0) {
        /* Child process (Process 1) */
        gettimeofday(&start_time, NULL);

        /* Open FIFOs */
        fd_fifo1 = open("team1_fifo1", O_RDONLY);
        fd_fifo2 = open("team1_fifo2", O_WRONLY);

        /* Transfer file from Process 1 to Process 2 */
        char buffer[4096];
        ssize_t bytes_read, bytes_written;
        int file_fd = open("/tmp/team1_large_file.txt", O_RDONLY); /* Updated file
path */
        while ((bytes_read = read(file_fd, buffer, sizeof(buffer))) > 0) {
            bytes_written = write(fd_fifo2, buffer, bytes_read);
            if (bytes_written != bytes_read) {
                perror("write");
                return 1;
            }
        }

        gettimeofday(&end_time, NULL);
        long long elapsed_time = (end_time.tv_sec - start_time.tv_sec) * 1000000LL +
(end_time.tv_usec - start_time.tv_usec);

        printf("Process 1 to Process 2 Transfer Time: %lld microseconds\n",
elapsed_time);

        close(file_fd);
```

```
        close(fd_fifo1);
        close(fd_fifo2);
    }
```

Process 1 is responsible for transferring a large file from Process 1 to Process 2 using named pipes (FIFOs). It measures the transfer time and sends data from the file in manageable chunks.

1. **Start Time Measurement:**

   ```
   gettimeofday(&start_time, NULL);
   ```

   Process 1 begins by measuring the start time using `gettimeofday()`. This is done to calculate the elapsed time for the transfer.

2. **Opening FIFOs:**

   ```
   fd_fifo1 = open("team1_fifo1", O_RDONLY);
   fd_fifo2 = open("team1_fifo2", O_WRONLY);
   ```

   - `fd_fifo1` is opened for reading (`O_RDONLY`) and corresponds to "team1_fifo1."
   - `fd_fifo2` is opened for writing (`O_WRONLY`) and corresponds to "team1_fifo2."

   These file descriptors allow Process 1 to communicate with Process 2 through the named pipes.

3. **Reading and Sending Data:**

   ```
   char buffer[4096];
   ssize_t bytes_read, bytes_written;
   int file_fd = open("/tmp/team1_large_file.txt", O_RDONLY);
   ```

   - `buffer` is declared as an array of 4096 bytes. This serves as a temporary storage location for data read from the file and sent to Process 2 in chunks.
   - `bytes_read` and `bytes_written` are used to track the number of bytes read from the file and written to the FIFO, respectively.
   - A file descriptor `file_fd` is opened for reading (`O_RDONLY`) the file "/tmp/team1_large_file.txt."

4. **Transfer Loop:**

   ```
   while ((bytes_read = read(file_fd, buffer, sizeof(buffer))) > 0) {
       bytes_written = write(fd_fifo2, buffer, bytes_read);
       if (bytes_written != bytes_read) {
           perror("write");
           return 1;
       }
   }
   ```

   - Process 1 enters a loop to transfer data from the file to Process 2.
   - Inside the loop, it reads data from the file into the `buffer` in chunks of up to the size of `buffer` using the `read()` function. The number of

bytes read is stored in `bytes_read`.
    - It then writes the data from the `buffer` to "team1_fifo2" using the `write()` function. The number of bytes written is stored in `bytes_written`.
    - It checks whether the number of bytes written matches the number of bytes read. If they don't match, it prints an error message using `perror()` and returns with an error code.

5. **End Time Measurement:**

```
gettimeofday(&end_time, NULL);
```

   After the loop completes, Process 1 measures the end time using `gettimeofday()`.

6. **Calculating Elapsed Time:**

```
long long elapsed_time = (end_time.tv_sec - start_time.tv_sec) * 1000000LL + (end_time.tv_usec - start_time.tv_usec);
```

   It calculates the elapsed time in microseconds by subtracting the start time from the end time.

7. **Cleanup and Close File Descriptors:**

```
close(file_fd);
close(fd_fifo1);
close(fd_fifo2);
```

   Process 1 closes the file descriptor for the input file and the FIFOs to release system resources.

6. Parent Process (Process 2):

```
else if (pid > 0) {
        /* Parent process (Process 2) */
        gettimeofday(&start_time, NULL);

        /* Open FIFOs */
        fd_fifo1 = open("team1_fifo1", O_WRONLY);
        fd_fifo2 = open("team1_fifo2", O_RDONLY);

        /* Transfer file from Process 2 to Process 1 */
        char buffer[4096];
        ssize_t bytes_read, bytes_written;
        int file_fd = open("/tmp/team1_received_file.txt", O_WRONLY | O_CREAT, 0666);
/* Updated file path */
        while ((bytes_read = read(fd_fifo2, buffer, sizeof(buffer))) > 0) {
            bytes_written = write(file_fd, buffer, bytes_read);
            if (bytes_written != bytes_read) {
                perror("write");
                return 1;
            }
        }
```

```
        gettimeofday(&end_time, NULL);
        long long elapsed_time = (end_time.tv_sec - start_time.tv_sec) * 1000000LL +
(end_time.tv_usec - start_time.tv_usec);

        printf("Process 2 to Process 1 Transfer Time: %lld microseconds\n",
elapsed_time);

        close(file_fd);
        close(fd_fifo1);
        close(fd_fifo2);

        int compare_result = system("diff -q /tmp/team1_large_file.txt
/tmp/team1_received_file.txt");
        if (compare_result == 0) {
            printf("Sent and received files are the same!\n");
        } else {
            printf("Sent and received files are different.\n");
        }

        /* Clean up FIFOs */
        /* unlink("team1_fifo1"); */
        /* unlink("team1_fifo2"); */
    }
```

This section of the code corresponds to the parent process (Process 2). It receives data from Process 1, writes it to a file, measures the transfer time, checks the integrity of the received file, and performs cleanup. Let's break down this part of the code step by step:

1. **Start Time Measurement:**

   ```
   gettimeofday(&start_time, NULL);
   ```

   - Similar to Process 1, Process 2 measures the start time using `gettimeofday()` to calculate the transfer time.

2. **Opening FIFOs:**

   ```
   fd_fifo1 = open("team1_fifo1", O_WRONLY);
   fd_fifo2 = open("team1_fifo2", O_RDONLY);
   ```

   - `fd_fifo1` is opened for writing (`O_WRONLY`) and corresponds to "team1_fifo1."
   - `fd_fifo2` is opened for reading (`O_RDONLY`) and corresponds to "team1_fifo2."
   - These file descriptors allow Process 2 to communicate with Process 1 through the named pipes.

3. **Transfer Data from Process 2 to Process 1:**

   ```
   char buffer[4096];
   ssize_t bytes_read, bytes_written;
   int file_fd = open("/tmp/team1_received_file.txt", O_WRONLY | O_CREAT, 0666);
   ```

- A new `buffer` array is declared to read data from "team1_fifo2" before writing it to the output file.
- `bytes_read` and `bytes_written` are used to track the number of bytes read from the FIFO and written to the output file, respectively.
- A file descriptor `file_fd` is opened for writing (`O_WRONLY`) and creating (`O_CREAT`) the file "/tmp/team1_received_file.txt." The permissions `0666` allow read and write access to everyone.

4. **Transfer Loop:**

```
while ((bytes_read = read(fd_fifo2, buffer, sizeof(buffer))) > 0) {
    bytes_written = write(file_fd, buffer, bytes_read);
    if (bytes_written != bytes_read) {
        perror("write");
        return 1;
    }
}
```

- Process 2 enters a loop to transfer data from "team1_fifo2" to the output file.
- Inside the loop, it reads data from the named pipe "team1_fifo2" into the `buffer` in chunks of up to the size of `buffer` using the `read()` function. The number of bytes read is stored in `bytes_read` .
- It then writes the data from the `buffer` to the output file using the `write()` function. It checks if the number of bytes written matches the number of bytes read to ensure a complete transfer.

5. **End Time Measurement:**

```
gettimeofday(&end_time, NULL);
```

- After the loop completes, Process 2 measures the end time using `gettimeofday()` .

6. **Calculating Elapsed Time:**

```
long long elapsed_time = (end_time.tv_sec - start_time.tv_sec) * 1000000LL +
(end_time.tv_usec - start_time.tv_usec);
```

- It calculates the elapsed time in microseconds by subtracting the start time from the end time.

7. **Close File Descriptors:**

```
close(file_fd);
close(fd_fifo1);
close(fd_fifo2);
```

- Process 2 closes the file descriptor for the output file and the FIFOs to release system resources.

8. **Compare Received and Original Files:**

```
int compare_result = system("diff -q /tmp/team1_large_file.txt
/tmp/team1_received_file.txt");
if (compare_result == 0) {
```

```
    printf("Sent and received files are the same!\n");
} else {
    printf("Sent and received files are different.\n");
}
```

- Process 2 uses the `system()` function to execute the `diff` command and compare the received file ("/tmp/team1_received_file.txt") with the original file ("/tmp/team1_large_file.txt").
- It checks the result of the `diff` command, and if the result is `0`, it prints a message indicating that the sent and received files are the same. Otherwise, it prints a message indicating that they are different.

9. **Cleanup (Optional):**

```
/* Clean up FIFOs */
/* unlink("team1_fifo1"); */
/* unlink("team1_fifo2"); */
```

- These lines are commented out, but they would be used to unlink (delete) the FIFOs "team1_fifo1" and "team1_fifo2" if you decide to clean up the FIFOs after the program completes. Currently, they are not active.

In summary, Process 2 receives data from Process 1 through "team1_fifo2," writes it to an output file, measures the transfer time, checks the integrity of the received file, and can optionally clean up the FIFOs. This completes the data transfer between

the two processes.

7. Cleaning Up:

```
        /* Clean up FIFOs */
        /* unlink("team1_fifo1"); */
        /* unlink("team1_fifo2"); */
```

The code includes commented-out lines to unlink (delete) the FIFOs. These lines can be uncommented if you want to remove the FIFOs after the program finishes executing.

8. Return from `main()`:

```
    return 0;
```

The program ends, returning 0 to indicate successful execution.

In summary, this program demonstrates how to use named pipes (FIFOs) for interprocess communication, measuring the transfer time of a large file between two processes, and checking if the received file matches the original. It creates FIFOs, forks into two processes, and handles the file transfer and checking in a coordinated manner.