Assignment Number 03

Name: Mihir Unmesh Patil

Roll NO: TYCOC213

Batch: C/C-3

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void process(int id, int read_fd, int write_fd) {
    char buffer[100];
    while (1) {
        // Read message
        read(read_fd, buffer, sizeof(buffer));
        if (strncmp(buffer, "exit", 4) == 0 || strncmp(buffer, "stop", 4) == 0) {
            printf("Process %d received exit command.\n", id);
            break;
        }
        printf("Process %d received: %s\n", id, buffer);

        // Send message
        printf("Process %d, enter message: ", id);
        fgets(buffer, sizeof(buffer), stdin);
        write(write_fd, buffer, strlen(buffer) + 1);
        if (strncmp(buffer, "exit", 4) == 0 || strncmp(buffer, "stop", 4) == 0) {
            printf("Process %d sent exit command.\n", id);
            break;
        }
    }
    close(read_fd);
    close(write_fd);
    exit(0);
}

void communication() {
    int pipe_fd1[2], pipe_fd2[2];
    char buffer[100];

    if (pipe(pipe_fd1) == -1 || pipe(pipe_fd2) == -1) {
        perror("Pipe failed");
        exit(1);
    }

    pid_t pid = fork();

    if (pid > 0) {
        // Process 1
        close(pipe_fd1[0]); // Close unused read end of pipe1
        close(pipe_fd2[1]); // Close unused write end of pipe2

        while (1) {
```

```
            printf("Process 1, enter message: ");
            fgets(buffer, sizeof(buffer), stdin);
            write(pipe_fd1[1], buffer, strlen(buffer)
+ 1);
            if (strncmp(buffer, "exit", 4) == 0 ||
strncmp(buffer, "stop", 4) == 0) {
                printf("Process 1 sent exit
command.\n");
                break;
            }
            read(pipe_fd2[0], buffer,
sizeof(buffer));
            if (strncmp(buffer, "exit", 4) == 0 ||
strncmp(buffer, "stop", 4) == 0) {
                printf("Process 1 received exit
command.\n");
                break;
            }
            printf("Process 1 received: %s\n",
buffer);
        }
        close(pipe_fd1[1]);
        close(pipe_fd2[0]);
    } else if (pid == 0) {
        // Process 2
        close(pipe_fd1[1]); // Close unused write
end of pipe1
        close(pipe_fd2[0]); // Close unused read
end of pipe2
        process(2, pipe_fd1[0], pipe_fd2[1]);
    } else {
        perror("Fork failed");
        exit(1);
    }
}

int main() {
    communication();
    return 0;
}
```

Output: