Assignment-2

Date: 31/01/2025
Author: Harsh Kumar (2K22/SE/71)

Problem Statement:

- Objective to communicate between two processes using message queues (Beej's guide) using Framing
- Send the file contents as frames.
- A message can contain multiple frames. Each frame follows framing .
- Do for character count and byte stuffing

Sol:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <fcntl.h>
#include <unistd.h>

#define FLAG_BYTE '@'
#define ESCAPE_BYTE '!'
#define MAX_BUFFER_SIZE 200

// Message structure for the queue
struct message_buffer {
   long message_type;
   char message_text[MAX_BUFFER_SIZE];
};

// Function to add byte stuffing
void add_byte_stuffing(const char *input, char *output, int *output_length)
{
   int j = 0;
   output[j++] = FLAG_BYTE; // Start with flag byte
```

```
    for (int i = 0; input[i] != '\0'; i++) {
        if (input[i] == FLAG_BYTE || input[i] == ESCAPE_BYTE) {
            output[j++] = ESCAPE_BYTE; // Escape special bytes
        }
        output[j++] = input[i];
    }
    output[j++] = FLAG_BYTE; // End with flag byte
    output[j] = '\0'; // Null-terminate
    *output_length = j;
}

// Function to remove byte stuffing
void remove_byte_stuffing(const char *input, char *output, int
*output_length) {
    int j = 0;
    int input_length = strlen(input);
    for (int i = 1; i < input_length - 1; i++) { // Skip start and end flags
        if (input[i] == ESCAPE_BYTE && i + 1 < input_length) {
            i++;  // Skip escape byte
        }
        output[j++] = input[i];
    }
    output[j] = '\0';  // Ensure null termination
    *output_length = j;
}

// Server function to send file data
void server_process(const char *file_name, int queue_id) {
    struct message_buffer msg;

    FILE *file_ptr = fopen(file_name, "r");
    if (!file_ptr) {
        perror("Failed to open file");
        exit(1);
    }

    char data_buffer[MAX_BUFFER_SIZE];
    while (fgets(data_buffer, MAX_BUFFER_SIZE, file_ptr)) {
        int data_length = strlen(data_buffer);

        // Ensure proper termination
        if (data_buffer[data_length - 1] == '\n') {
            data_buffer[data_length - 1] = '\0';
```

```
            data_length--; // Remove newline from count
        }

        // Character count framing
        msg.message_type = 1; // Message type for character count
        msg.message_text[0] = data_length;
        memcpy(msg.message_text + 1, data_buffer, data_length);
        msg.message_text[data_length + 1] = '\0';  // Ensure termination

        // Send the message
        if (msgsnd(queue_id, &msg, data_length + 1, 0) == -1) {
            perror("Message send failed");
            exit(1);
        }

        // Byte stuffing framing
        msg.message_type = 2; // Message type for byte stuffing
        char stuffed_data[MAX_BUFFER_SIZE * 2];
        int stuffed_length;
        add_byte_stuffing(data_buffer, stuffed_data, &stuffed_length);

        memcpy(msg.message_text, stuffed_data, stuffed_length);
        msg.message_text[stuffed_length] = '\0'; // Ensure null termination

        // Send the stuffed message
        if (msgsnd(queue_id, &msg, stuffed_length, 0) == -1) {
            perror("Message send failed");
            exit(1);
        }
    }

    fclose(file_ptr);
    printf("Server: File data sent successfully.\n");
}

// Client function to receive file data
void client_process(int queue_id) {
    struct message_buffer msg;

    printf("Client: Receiving file data...\n");

    while (1) {
        // Receive a message
```

```
        if (msgrcv(queue_id, &msg, MAX_BUFFER_SIZE, 0, 0) == -1) {
            perror("Message receive failed");
            break;
        }

        // Process based on message type
        if (msg.message_type == 1) {
            // Character count framing
            int data_length = msg.message_text[0];
            printf("Character Count Frame: %.*s\n", data_length,
msg.message_text + 1);
        } else if (msg.message_type == 2) {
            // Byte stuffing framing
            char unstuffed_data[MAX_BUFFER_SIZE];
            int unstuffed_length;
            remove_byte_stuffing(msg.message_text, unstuffed_data,
&unstuffed_length);
            printf("Byte Stuffing Frame: %s\n", unstuffed_data);
        }
    }

    printf("Client: File data received successfully.\n");
}

int main() {
    key_t queue_key = ftok("queue_key_file", 65);
    int queue_id = msgget(queue_key, 0666 | IPC_CREAT);
    if (queue_id == -1) {
        perror("Message queue creation failed");
        exit(1);
    }

    if (fork() == 0) {
        // Server process
        server_process("input.txt", queue_id);
    } else {
        // Client process
        client_process(queue_id);

        // Clean up the message queue
        msgctl(queue_id, IPC_RMID, NULL);
    }
```

```c
    return 0;
}
```

```
❯ desktop/LABS/"CN Lab"/Assignment_3
❯ ls
assign_3   assign_3.c input.txt
❯ gcc assign_3.c -o assign_3
❯ ./assign_3
Client: Receiving file data...
Character Count Frame: Author: Harsh Kumar
Byte Stuffing Frame: Author: Harsh Kumar
Character Count Frame: Roll No: 2K22/SE/71
Byte Stuffing Frame: Roll No: 2K22/SE/71@
Server: File data sent successfully.
Character Count Frame: Context: This is a demo file
Byte Stuffing Frame: Context: This is a demo file
```