Assignment-5

Date: 04/04/2025
Author: Harshit Grover (2K22/SE/73)

Problem Statement:

- Objective to develop a client and server program for transferring file contents from server to client using Stop and Wait sliding window protocol.
- Implement Stop and Wait sliding window protocol (Each frame is acknowledged before sending the next frame)
- Send a file from the server, breaking into the frames, sending over stop and wait protocol (single window). Assume it's an error free channel and no acknowledgements are lost.
- Print the frame number sent and received.
- Display the content of the reassembled file.

Note:

1. *Underlying IPC mechanisms can be chosen as messages queues or pipes.*
2. *The data link layer must take data of any size from the higher application layer and fragment it into appropriate frames.*

Sol:

[assign_5_server.c]

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

#define MAX_FRAME_SIZE 1024

struct msg_buffer {
    long mtype;
    int frame_num;
```

```
    int data_size;
    char data[MAX_FRAME_SIZE];
};

struct ack_msg {
    long mtype;
    int frame_num;
};

int main() {
    key_t data_key = 1234;
    key_t ack_key = 5678;

    int data_qid = msgget(data_key, 0666 | IPC_CREAT);
    int ack_qid = msgget(ack_key, 0666 | IPC_CREAT);

    FILE *file = fopen("input.txt", "rb");
    if (!file) {
        perror("File open error");
        exit(1);
    }

    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    fseek(file, 0, SEEK_SET);

    char *buffer = malloc(file_size);
    fread(buffer, 1, file_size, file);
    fclose(file);

    int total_frames = (file_size + MAX_FRAME_SIZE - 1) /
MAX_FRAME_SIZE;
    struct msg_buffer msg;
    msg.mtype = 1; // Data message type

    for (int i = 0; i < total_frames; i++) {
        int offset = i * MAX_FRAME_SIZE;
        int size = (i == total_frames - 1) ? (file_size - offset) :
MAX_FRAME_SIZE;
```

```
        msg.frame_num = i;
        msg.data_size = size;
        memcpy(msg.data, buffer + offset, size);

        msgsnd(data_qid, &msg, sizeof(msg) - sizeof(long), 0);
        printf("Server: Sent frame %d (Size: %d bytes)\n", i, size);

        struct ack_msg ack;
        msgrcv(ack_qid, &ack, sizeof(ack.frame_num), 2, 0);
        printf("Server: Received ACK for frame %d\n", ack.frame_num);
    }

    // Send termination signal
    msg.frame_num = -1;
    msgsnd(data_qid, &msg, sizeof(msg) - sizeof(long), 0);

    free(buffer);
    msgctl(data_qid, IPC_RMID, NULL);
    msgctl(ack_qid, IPC_RMID, NULL);
    return 0;
}
```

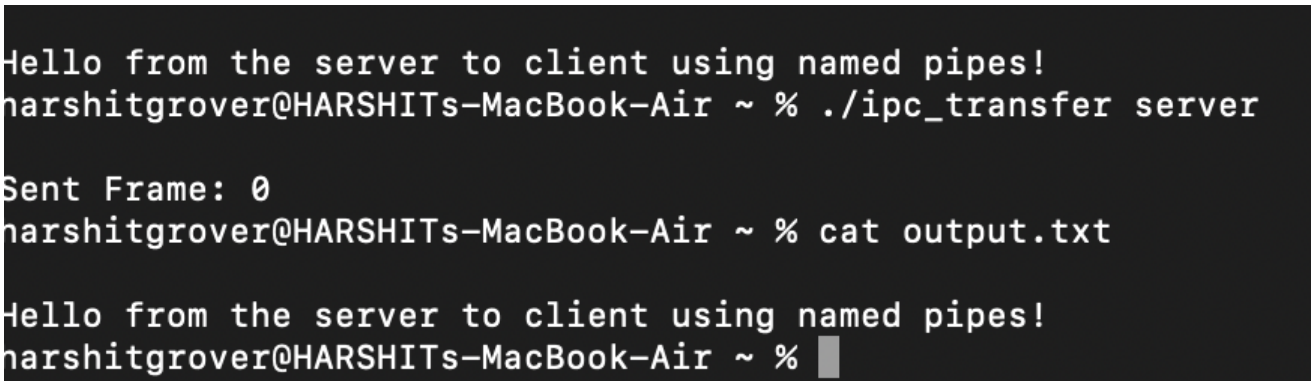
[assign_5_client.c]

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

#define MAX_FRAME_SIZE 1024

struct msg_buffer {
    long mtype;
```

```
    int frame_num;
    int data_size;
    char data[MAX_FRAME_SIZE];
};

struct ack_msg {
    long mtype;
    int frame_num;
};

int main() {
    key_t data_key = 1234;
    key_t ack_key = 5678;

    int data_qid = msgget(data_key, 0666);
    int ack_qid = msgget(ack_key, 0666);

    if (data_qid == -1 || ack_qid == -1) {
        perror("Error accessing message queues");
        exit(1);
    }

    FILE *file = fopen("output.txt", "wb");
    if (!file) {
        perror("File open error");
        exit(1);
    }

    struct msg_buffer msg;
    struct ack_msg ack;
    ack.mtype = 2; // ACK message type

    while (1) {
        // Receive message with proper error checking
        if (msgrcv(data_qid, &msg, sizeof(msg) - sizeof(long), 1,
0) == -1) {
```

```
            perror("Error receiving message");
            break;
        }

        // Check for termination signal
        if (msg.frame_num == -1) {
            printf("Client: Received termination signal\n");
            break;
        }

        // Process the frame
        fwrite(msg.data, 1, msg.data_size, file);
        printf("Client: Received frame %d (Size: %d bytes)\n",
msg.frame_num, msg.data_size);

        // Send acknowledgment
        ack.frame_num = msg.frame_num;
        if (msgsnd(ack_qid, &ack, sizeof(ack.frame_num), 0) ==
-1) {
            perror("Error sending acknowledgment");
            break;
        }
        printf("Client: Sent ACK for frame %d\n", msg.frame_num);
    }

    fclose(file);
    printf("Client: File transfer complete\n");
    return 0;
}
```

server side



client side