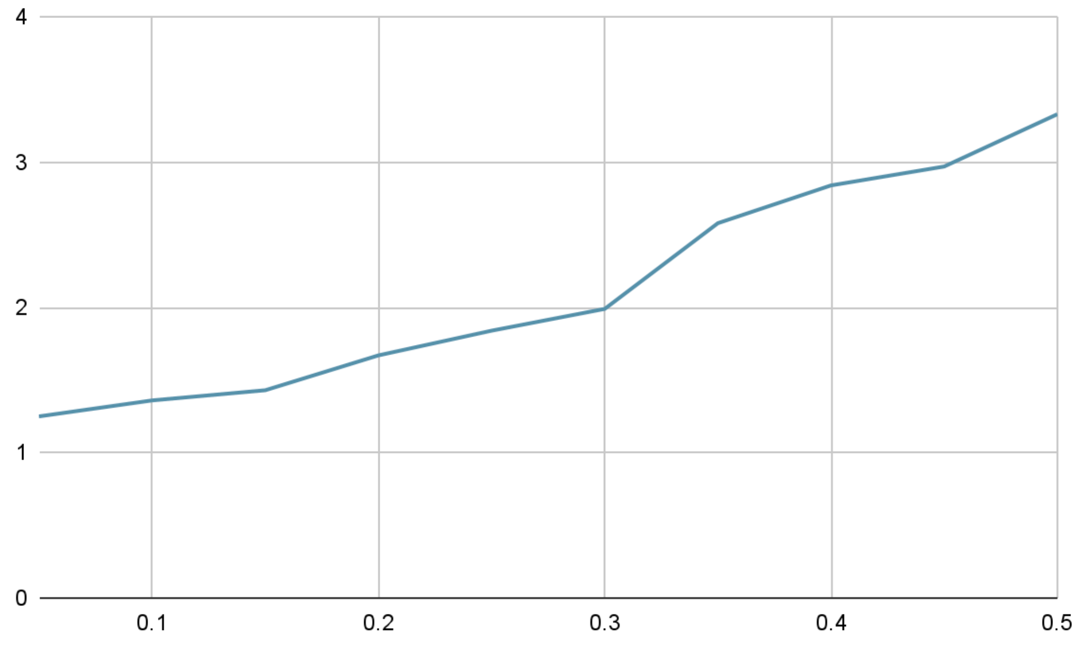
Networks Assignment 5 Documentation

Soukhin Nayek 21CS10062
Raj Pajesh Parikh 21CS30039

<u>The Average Number of Transmissions made for varying P :-</u>

Sr No.	Probability	Number of Messages Generated	Total number of transmissions	Avg No of Transmissions
1.	0.05	100	125	1.25
2.	0.1	100	136	1.36
3.	0.15	100	143	1.43
4.	0.2	100	167	1.67
5.	0.25	100	184	1.84
6.	0.3	100	199	1.99
7.	0.35	100	258	2.58
8.	0.4	100	284	2.84
9.	0.45	100	297	2.97
10.	0.5	100	333	3.33



Data Structures :-

1. **Message:** This structure is used to represent a single message that needs to be sent or received over the MTP socket.

```
typedef struct
{
    // An integer representing the sequence number of the message
    int seq_num;
    // A character array of size MESSAGE_SIZE (1024 bytes) to store the message data.
    char data[MESSAGE_SIZE];
} Message;
```

2. **Data_Packet:**This structure is used to encapsulate data or acknowledgment packets for transmission over the network.

```
typedef struct
{
    // An integer flag indicating whether the packet is an ACK packet or a data packet.
    char is_ack;
    //An integer representing the sequence number of the packet.
    char seq_num;
    union
    {
        //A character array to store the message data  if the packet is a data packet.
        char data[MESSAGE_SIZE];
        //An integer representing the receive window size if the packet is an ACK packet.
        char rwnd;
    };
} Data_Packet;
```

3. **Window:**This structure is used to manage the send and receive windows for reliable data transfer. It keeps track of the window size, base sequence number, and the status of each sequence number in the window.

```c
typedef struct
{
    // An integer representing the size of the window.
    int size;
    // for send it defines the base at the buffer
    // for recv it defines the idx for the seq_num for easy implementation
    int base;
    // An array of integers to store sequence numbers up to MAX_SEQ_NO (15).
    int seq_nums[MAX_SEQ_NO + 1];
    // for send, the seq no of the last message which was sent
    // for recv, next expected to receive seq_num from the buffer
    int last_seq_num;
    // for send, the seq no the last acknowleged message
    int last_ack_seq;
} Window;
```

4. **MTPSocketEntry:** This structure is the main representation of an MTP socket in the shared memory. It holds all the necessary information and state for each socket, including buffers, windows, addresses, and synchronization mechanisms.

```c
// Structure to represent an MTP socket entry in the shared memory
typedef struct
{
    // An integer flag indicating whether the socket entry is free or in use.
    int is_free;
    // The process ID of the process that owns the socket.
    pid_t process_id;
    // The underlying UDP socket ID.
    int udp_socket_id;
    // The remote address (struct sockaddr_in) for the socket.
    struct sockaddr_in remote_addr;
    // An array of Message structures to store messages for sending.
    Message send_buffer[MAX_SEND_BUFFER_SIZE];
    // An array of Message structures to store received messages.
    Message recv_buffer[MAX_RECV_BUFFER_SIZE];
    // A Window structure representing the send window
    Window swnd;
    // A Window structure representing the receive window.
    Window rwnd;
    // A timespec structure to store the last time a message was sent on this socket.
    struct timespec last_send_time;
} MTPSocketEntry;
```

5. **SockInfoEntry:**This structure is used during the socket creation and binding process to store and exchange information about the socket being created or bound.

```
typedef struct
{
    // An integer representing the socket ID.
    int sockid;
    // An integer representing the domain of the socket (e.g., AF_INET).
    int domain;
    // A sockaddr_in structure representing the address of the socket.
    struct sockaddr_in addr;
    //  An integer to store an error code.
    int err_no;
    // The process ID of the process that owns the socket.
    pid_t process_id;
} SockInfoEntry;
```

Functions in msocket.c :-

1. sockaddr_in_equal: A helper function that compares two sockaddr_in structures for equality.
2. lock_init: Initializes the shared memory segments, semaphores, and locks for socket management.
3. m_socket: Implements the socket system call for creating an MTP socket.
4. m_bind: Implements the bind system call for binding an MTP socket to a local and remote address.
5. m_sendto: Implements the sendto system call for sending data over an MTP socket.
6. m_recvfrom: Implements the recvfrom system call for receiving data from an MTP socket.
7. m_close: Implements the close system call for closing an MTP socket.
8. dropMessage: A function to simulate message drops based on a probability P.

Functions in initmsocket.c :-

1. r_thread: A thread function responsible for receiving messages on MTP sockets.
2. receive_message: A function that receives a message from a UDP socket and handles it accordingly.
3. handle_data_msg: A function that handles a received data message.
4. handle_ack_msg: A function that handles a received ACK message.
5. store_message: A function that stores a received message in the receive buffer.
6. send_ack: A function that sends an ACK message.
7. update_swnd: A function that updates the send window based on received ACKs.

8. s_thread: A thread function responsible for sending pending messages on MTP sockets.
9. check_timeouts: A function that checks for message timeouts and retransmits messages if needed.
10. send_pending_messages: A function that sends pending messages from the send buffer.
11. gc_thread: A thread function that acts as a garbage collector for socket entries.
12. sigint_handler: A signal handler function for the SIGINT signal (Ctrl+C).
13. main: The main function that initializes the shared memory segments, semaphores, and starts the receiver, sender, and garbage collector threads.

<u>additional descriptions :-</u>

The additional descriptions for the conventions used in the sender window, receiver window, sender buffer, and related variables. Here's a summary of the key points:

Sender Window:

seq_nums array:
1: Message in the sender window is sent, but the ACK is not yet received.
0: Message is not sent yet.
last_ack_seq: The sequence number of the last acknowledged message.
last_seq_num: The sequence number of the last message that was sent.
base: The last message that did not receive an ACK. Initially set to 0.

Sender Buffer:

seq_num:
-2: Space in the sender buffer is empty.
-1: Space is filled, but the message is not sent yet.
else: Contains a sent message with the corresponding sequence number.

Receiver Window:

last_seq_num: The next expected sequence number to receive from the buffer. Initially set to 1.
seq_nums array:
Initialization: All elements set to 1.
1: Expected to receive the corresponding sequence number.
0: The corresponding sequence number has been received and stored in the buffer.
base: The starting sequence number from which messages are expected to be received. Initially set to 1.