>>>>>>>>>>>>>>>>>>>>>>>>>>>> INSTRUCTION TO RUN MAKEFILE <<<<<<<<<<<<<<<<<<<<<<<

- To generate librsocket.a , run 'makefile1' using the following command:

    make -f makefile1

- To generate the executables from the user1.c and user2.c , please use 'makefile2' using the following command:

    make -f makefile2

- Now two executable files will be created in the same directory - user1, user2

- Run them using the following command in two separate command-lines :

    ./user1
    ./user2

>>>

>>>>>>>>>>>>>>>>>>>>>>>>>>>> EXPERIMENT <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    String sent: "abcdefghijklmnopqrstuvwxyz"
    Length = 26

p	Total No. of Transmissions	Avg No. of Transmissions
0.05	27	1.038462
0.10	27	1.038462
0.15	31	1.192308
0.20	33	1.269231
0.25	35	1.346154
0.30	37	1.423077
0.35	38	1.461538
0.40	41	1.576923
0.45	42	1.615385
0.50	47	1.807692

>>>

>>>>>>>>>>>>>>>>>>>>>>> DATA STRUCTURES <<<<<<<<<<<<<<<<<<<<<<<<<

1. unack_msg : A Structure to store all relevant information about an unacknowledged message

Fields	Data type	Description
seq_num	int	Stores sequence number of this message
dest_addr	struct sockaddr *	Stores information about destination address of this message
dest_len	socklen_t	Destination Address length
flags	int	Stores the flags which will be used during sendto(...) call
msg_len	int	Length of the message including its header
msg	char *	Actual Message String prefixed with its header.
msg_time	time_t	Time when the message was last sent to the receiver
next	unack_msg *	Pointer to a next entry of this structure
prev	unack_msg *	Pointer to a previous entry of this structure

2. unack_msg_table_t : A Table containing all unacknowledged messages. It stores the messages in a FIFO Queue based implementation as doubly linked list.

Fields	Data type	Description
next_seq_num	int	Stores the next sequence number for the message entry which will be inserted into the table in future.
table	unack_msg *	Stores the list of unacknowledged messages as a doubly linked list
tail	unack_msg *	Stores the tail pointer of the message table for maintaining it as a doubly linked list
size	int	Stores the Size of the message table

3. recv_msg : A structure to hold information about a received message.

Fields	Data type	Description
msg	char *	Stores the received message string prefixed with its header.
msg_len	int	Stores the length of the received message string
src_addr	struct sockaddr *	Stores the address of the source from which this message is received
src_len	socklen_t *	Stores the length of the source address

next	recv_msg *	Pointer to a next entry of this structure

4. recv_msg_table_t : A Table containing all the received messages. It is implemented as a FIFO queue using linked list

Fields	Data type	Description
table	recv_msg *	Stores all the received messages as a linked list
size	int	Stores size of the received message table
msg_in	recv_msg *	Stores the pointer to the last entry of the table(received messages are inserted using this)
msg_out	recv_msg *	Stores the pointer to the first entry of the table(received messages are extracted using this)

5. thread_data : This structure holds data for passing into a thread_data

Fields	Data type	Description
sockfd	int	Stores the socket file descriptor of the sender and receiver for different thread handlers

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

>>>>>>>>>>>>>>>>>>>>>>>>>> FUNCTIONS <<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

1. insert_unack_entry:
    - This function takes a buffer, message length, sequence number, destination address, destination length and flags as its parameters.
    - Then it creates a new unacknowledged message entry using malloc(...) and fills the message entry with appropriate information.
    - It updates the pointers maintained in the unack_message_table after inserting a new message and increases its size by 1

2. delete_unack_entry:
    - This function takes a sequence number as a parameter
    - It finds the pointer pointing to the message entry in the unacknowledged_message_table having that sequence number.
    - If the message having this sequence number does not exist in the table, it does nothing.
    - Otherwise, It releases the memory occupied by that unacknowledged message entry and deletes that entry from the table
    - Then reduces the size of the table by 1

3. insert_recv_entry
    - It takes a message, message length, source address and source address length as its parameters
    - It creates a new entry for a received message in the received_message_table.
    - It fills up the relevant information in the structure and updates the pointers for maintaining the linked list
    - It updates the size of the table by increasing it by 1

4. delete_recv_entry
    - This function removes the received message from the received_message_table
    - It removes the message like a FIFO queue based pop operation
    - Finally it releases the memory occupied by the received message and reduces the size of the table by 1

5. r_socket
    - This function takes the same parameters as socket(...) call but takes SOCK_MRP as socket type
    - It creates the underlying UDP socket, creates thread 'R' and thread 'S' and allocates memory to the message tables
    - It initializes the values to empty in the tables and thread data for each thread
    - It returns the socket file descriptor of the created socket

6. r_bind
    - This function binds the socket to the given address and port as parameter using bind(...) call

7. r_sendto
    - This function takes the same parameters as sendto(...) call
    - It prefixes the message to be sent using a special header
    - the header is of length 3 bytes with the first byte being 'STX' which denotes start of the text/message.
    - The next 2 bytes store the size of the message to be sent.
    - Finally it inserts the message using insert_unack_entry(...) method.
    - Then it uses sendto(...) call for sending the message and returns the length of the message sent.

8. r_recvfrom
    - This function takes the same parameters as recvfrom(...) call.
    - It uses pthread_mutex_lock and unlock mechanism for safely extracting the size of the received message table
    - If the table is empty, it sleeps for 2 seconds using nanosleep(...) and checks again for any incoming message
    - If there is a message, it uses pthread mutex locks to extract the message is copied into the buffer address provided as the parameter
    - Then it deletes the received message entry from the received_message_table

9. r_close
    - This function kills the threads and deletes all messages from unacknowledged_message_table and received_message_table
    - It releases the memory occupied by each message entry
    - Finally it closes the socket file descriptor which was created during r_socket(...) call

10. s_thread_handler
    - This is the handler function for the S thread
    - This function makes the thread S sleep for some time T (T=2 sec) and wakes up after that.
    - Then it iterates through the unacknowledged_message_table
    - It checks if any message has passed its timeout period(2*T) and if so, then it re-transmits that message and updates its time in the table
    - It uses pthread_mutex_lock for avoiding data race while scanning the unacknowledged_message_table.

11. r_thread_handler
    - This is the handler function for the R thread
    - It keeps receiving messages using recvfrom(...) call
    - If there is a message, it uses the dropMessage(...) method and based on its return value, it accepts/discards the message
    - If the message is not discarded, then it is checked whether it is a data message or an ACK message.
    - If it is a data message, corresponding entry is created in the received_message_table using insert_recv_entry(...) method.
    - Otherwise, the sequence number of the ACK message is decoded and used to remove the entry from the unacknowledged_message_table

12. dropMessage
    - This function takes a floating point number, p as its parameter
    - It generates a random decimal number in the range [0,1]
    - If the generated_number < p, it returns 1 else it returns 0

>><<<<<<<<<<<<<<<<<<<<<<<<<<<<