**Struct Definitions:**

1. **sendwindow**:
   * **cnt**: Tracks the number of packets in the sender buffer.
   * **front**: Points to the index where the buffer is empty for packet insertion.
   * **size**: Specifies the buffer size (fixed at 10 packets).
   * **lastsent**: Records the time of the last sent message.
   * **seqno**: Holds the sequence number of the message being inserted.
   * **winstart**: Marks the starting index of the sender window.
   * **winsize**: Indicates the size of the sender window.
   * **winsent**: Tracks the index of the last sent message within the window.
   * **winseq**: Stores the sequence number of the first message in the window.
2. **recvwindow**:
   * **cnt**: Represents the current count of packets in the receive buffer.
   * **front**: Indicates the front (or starting) index of the receive buffer.
   * **rear**: Indicates the rear (or ending) index of the receive buffer.
   * **size**: Specifies the size of the receive buffer, set to 5 in this case.
   * **nospace**: A flag that is set when there is no space left in the receive buffer for incoming packets.
   * **seqno**: Holds the sequence number that the receiver is expecting to receive next.
   * **winstart**: Represents the starting index of the receiver window.
   * **winsize**: Specifies the size of the receiver window.
3. **SM**:
   * **smvalid**: Indicates whether the struct is valid or not (possibly used for error checking or initialization).
   * **pid**: Process ID associated with the socket or communication.
   * **sockid**: Socket ID or descriptor related to the communication.
   * **sip**: Source IP address stored as a string.
   * **sport**: Source port number.
   * **dip**: Destination IP address stored as a string.
   * **dport**: Destination port number.
   * **s**: Array of strings representing sent messages (10 strings of maximum length 1024 characters each).
   * **r**: Array of strings representing received messages (5 strings of maximum length 1024 characters each).
   * **seqs**: Array storing sequence numbers for sent messages (corresponding to the 's' array).
   * **valid**: Array indicating the validity of received messages (corresponding to the 'r' array).
   * **swnd**: Struct containing information about the sender's window (potentially related to flow control and packet buffering for sending).
   * **rwnd**: Struct containing information about the receiver's window (related to receiving and managing incoming packets).
   * **isclosed**: Flag indicating whether the communication or socket is closed.
   * **no\_pacs\_sent**: Number of packets sent during the communication.
4. **sockinfo**:
   * **sockid**: An integer representing the socket ID or descriptor.
   * **ip**: A character array storing the IP address associated with the socket (limited by INET\_ADDRSTRLEN characters).
   * **port**: An integer indicating the port number used by the socket.
   * **err**: An integer field that typically stores error codes or status information related to socket creation and binding.

**Functions in msocket.c:**

1. **M\_socket**:
   * m\_socket function creates a socket and manages related information in shared memory (SM struct).
   * Utilizes two semaphores (sem1 and sem2) for synchronization.
     1. Signals sem1 to indicate search for empty space in sm for the socket.
     2. Waits on sem2 for main thread of init process to create socket and signal back.
   * Communicates through shared memory to update SM struct with socket-related information.
     1. Finds empty space in sm for new socket and sets its validity.
     2. Sets source IP and socket ID in sm if no errors during socket creation.
   * Checks for socket type (SOCK\_MTP) and returns error if not matched.
   * Handles errors related to shared memory operations (e.g., finding empty space in sm).
   * Checks errno after main thread of init process creates socket and sets source IP and socket ID in sm.
   * Returns index in sm where socket information is stored if successful.
   * Returns error code (negative value) if socket creation or shared memory operations fail.
2. **M\_bind**:
   * m\_bind function populates si (sockinfo) with the entries from sm[sockfd] and signals the main thread of the init process to bind the socket with the provided address and port.
   * Upon successful binding by the main thread, it signals back to m\_bind using sem2.
   * m\_bind then fills sm[sockfd] with the information of the destination address and port, initializes all other relevant values, and returns 0 on success.
   * If the binding process fails or encounters errors, m\_bind returns -1 to indicate failure.
3. **M\_sendto**:
   * m\_sendto function first checks if the process ID matches and the socket is still open in sm[sockfd].
   * If the conditions are met, it proceeds to check if the provided destination matches the destination in sm[sockfd].
   * If there's a match, m\_sendto searches for an empty space in the send buffer of sm[sockfd].
   * If an empty space is found, it copies the provided message to the buffer and returns the length of the message.
   * If no empty space is available, it returns -1 to indicate that the message could not be sent.
4. **M\_recvfrom**:
   * m\_recvfrom begins by checking if the process ID matches and if the socket is still open in sm[sockfd].
   * If both conditions are met, it proceeds to check if the rear pointer in the receive window (rwnd) of sm[sockfd] points to a valid message that should be given to the user.
   * If the message pointed to by the rear pointer is valid, m\_recvfrom copies that message to the buffer provided by the user.
   * Finally, m\_recvfrom returns the length of the message copied to the buffer. If no valid message is available, it returns -1 to indicate that no message could be received.
5. **M\_close**:
   * It checks if the process ID matches and the socket is still open in sm[sockfd].
   * If both conditions are met, it sets the isclosed flag to indicate that the socket is closed.
   * Finally, it returns 0 to indicate successful closure of the socket. If the conditions are not met, it returns -1 to indicate a failure in closing the socket.
6. **dropMessage**:
   * Determines whether to drop a message based on a specified probability.

**Functions in initsocket.c:**

1. **sighandler**:
   * Purpose: Handles the SIGINT signal, which is typically sent by the operating system when the user presses Ctrl+C to stop the server.
   * Actions:
     1. Sets the stop flag to 1 to indicate that the server should stop.
     2. Uses semop to perform semaphore operations, specifically releasing sem1.
   * Note: This function is triggered when the server receives a SIGINT signal, usually during shutdown.
2. **G\_Collector**:
   * Purpose: Monitors and cleans up resources related to open sockets.
   * Actions:
     1. Periodically checks the status of sockets and their associated resources.
     2. If the server is stopping (stop flag is set), closes sockets, terminates associated processes, and cleans up shared memory and semaphores.
     3. Handles cases where sockets are no longer valid or need to be closed due to certain conditions.
   * Note: This function ensures proper cleanup of resources related to sockets and associated processes.
3. **R\_routine**:
   * Purpose: Handles receiving data packets from client sockets.
   * Actions:
     1. Uses select to monitor sockets for incoming data packets.
     2. Processes incoming packets, checks for message dropping based on probability, handles out-of-order packets, and sends acknowledgment messages.
     3. Manages the receive window, updates status based on received packets, and handles duplication acknowledgment messages.
   * Note: This function is responsible for managing the reception of data packets and acknowledgments from client sockets.
4. **S\_routine**:
   * Purpose: Handles sending data packets to client sockets.
   * Actions:
     1. Periodically sends data packets from the send buffer to client sockets.
     2. Manages the send window, handles timing for packet transmission, and updates status based on sent packets.
   * Note: This function is responsible for managing the transmission of data packets to client sockets.
5. **main**:
   * Purpose: Entry point of the server program.
   * Actions:
     1. Initializes shared memory, semaphores, and other necessary resources.
     2. Creates and manages threads (R\_routine, S\_routine, G\_Collector) for handling socket operations and resource cleanup.
     3. Handles signals, such as SIGINT for graceful shutdown of the server.
     4. Manages the server's socket creation, binding, and cleanup operations.
     5. Ensures proper termination and cleanup of resources upon server shutdown.

|  |  |  |
| --- | --- | --- |
| P | Number of Messages (KB) | Number of packets sent |
| 0.05 | 430 | 487 |
| 0.1 | 430 | 526 |
| 0.15 | 430 | 615 |
| 0.2 | 430 | 645 |
| 0.25 | 430 | 765 |
| 0.3 | 430 | 895 |
| 0.35 | 430 | 976 |
| 0.4 | 430 | 1101 |
| 0.45 | 430 | 1225 |
| 0.5 | 430 | 1378 |