NETWORKS ASSIGNMENT-5 REPORT

CHITYALA RAVITEJA - 21CS30016
SUMANTH TADIGOPPALA - 21CS30053

Table for different p values:

Probability(p values)	**No. of transmissions made to send**	**No. of messages sent (to send 100kb file)**	**Avg no.of transmissions per msg**
0.05	101	94	1.106
0.10	130	101	1.287
0.15	133	95	1.400
0.20	134	92	1.456
0.25	164	106	1.547
0.30	155	99	1.565
0.35	173	98	1.765
0.40	215	101	2.128
0.45	211	99	2.131
0.50	249	104	2.394

Note:
These are the values we got while running the code (which was initially implemented without use of few semaphores(for ensuring mutual exclusion) when we added semaphores , we got a error called stack crashed (which we were unable to debug on the last half day, so we submitted with that but functionalities were implemented completely)

TO run: in separate terminals

1. **make init_msocket**
2. **Make user1**

File: msocket.h
Header file for MTP (Message Transfer Protocol) socket implementation.

Structs:
1. sender_buffer_element
- Represents a buffer element for sending messages.
- Members:
  - in_use: Indicates if the buffer element is in use.
  - message: Array to store the message content.
  - seqnum: Sequence number of the message.
  - timer: Timer value for the message.
  - ack: Acknowledgment status of the message.
  - packet_sent: Indicates if the packet is sent or not.

2. receiver_buffer_element
- Represents a buffer element for receiving messages.
- Members:
  - in_use: Indicates if the buffer element is in use.
  - message: Array to store the received message content.
  - seqnum: Sequence number of the received message.
  - ack: Acknowledgment status of the received message.

3. swnd (Sender Window)
- Represents window information for the sender side.
- Members:
  - base: Base sequence number of the sender window.
  - nextseqnum: Next sequence number to be sent.
  - window_size: Size of the sender window.

4. rwnd (Receiver Window)
- Represents window information for the receiver side.
- Members:
  - window_size: Size of the receiver window.
  - nospace: Indicates if there is space in the receiver window.
  - last_delivered_seqnum: Last delivered sequence number.

5. MTP_Socket_Info
- Represents information about an MTP socket.
- Members:
  - is_free: Indicates if the socket is free.
  - creator_pid: Process ID of the socket creator.
  - udp_socket_id: UDP socket ID.

  - other_end_IP: IP address of the other end.
  - other_end_port: Port of the other end.
  - send_buffer: Array of sender buffer elements.
  - last_free_sender_buffer_index: Index of the last free sender buffer.
  - sender_window: Sender window information.
  - receiver_window: Receiver window information.
  - receive_buffer: Array of receiver buffer elements.

6. SOCK_INFO
- Represents information about a socket.
- Members:
  - sock_id: Socket ID.
  - src_ip: Source IP address.
  - dst_ip: Destination IP address.
  - src_port: Source port.
  - dst_port: Destination port.
  - error_number: Error number.
  - creator_pid: Process ID of the socket creator.

Functions:

1. int m_socket(int domain, int type, int protocol)
- Creates a new MTP socket.
- Parameters: domain, type, protocol
- Returns: Socket file descriptor or -1 on failure.

2. int m_bind(int sockfd, const char *src_ip, int src_port, const char *dst_ip, int dst_port)
- Binds a local IP address and port to an MTP socket.
- Parameters: sockfd, src_ip, src_port, dst_ip, dst_port
- Returns: 1 on success, -1 on failure.

3. ssize_t m_sendto(int sockfd, const void *buf, size_t len, int flags, const char *dst_ip, int dst_port)
- Sends data to a specific destination IP address and port.
- Parameters: sockfd, buf, len, flags, dst_ip, dst_port
- Returns: Number of bytes sent or -1 on failure.

4. ssize_t m_recvfrom(int sockfd, void *buf, size_t len, int flags, char *src_ip, int *src_port)
- Receives data from a specific source IP address and port.
- Parameters: sockfd, buf, len, flags, src_ip, src_port
- Returns: Number of bytes received or -1 on failure.

5. int m_close(int sockfd)

- Closes an MTP socket.
- Parameters: sockfd
- Returns: 0 on success, -1 on failure.

6. void initialize_semaphores()
- Initializes semaphores.

7. void destroy_semaphores()
- Destroys semaphores.

8. void semaphore_wait(int s)
- Performs semaphore wait operation.

9. void semaphore_signal(int s)
- Performs semaphore signal operation.

10. MTP_Socket_Info* access_shared_memory()
- Accesses shared memory for MTP socket information.

11. void print_SM_table()
- Prints the MTP socket table.

12. void print_sender_buffer_element(const sender_buffer_element *sender_buffer)
- Prints a sender buffer element.

13. void print_receiver_buffer_element(const receiver_buffer_element *receiver_buffer)
- Prints a receiver buffer element.

14. void print_sender_receiver_buffers(const MTP_Socket_Info *SM, int index)
- Prints sender and receiver buffers for a given index.

15. void detach_shared_memory(MTP_Socket_Info *SM)
- Detaches shared memory for MTP socket information.

16. void detach_sockinfo_memory(SOCK_INFO *sock_info)
- Detaches shared memory for socket information.

17. SOCK_INFO* access_sockinfo_memory()
- Accesses shared memory for socket information.

18. void signal_sem1_wait_sem2()
- Signals semaphore 1 and waits on semaphore 2.

19. void wait_sem1_signal_sem2()
- Waits on semaphore 1 and signals semaphore 2.

20. extern pthread_mutex_t mutex
- External declaration for mutex used for mutual exclusion of shared memory segments sockinfo and SM table.

21. extern int m_errno
- External declaration for global error variable.

File: msocket.c

Implemented functionalities

1.m_socket():

Purpose:

This function creates an MTP (Message Transfer Protocol) socket.

Functionality:

- It initializes necessary structures and resources for the MTP socket.
- It finds a free entry in the MTP socket table and marks it as used.
- It signals semaphore 1 and waits on semaphore 2 for synchronization.
- It retrieves the UDP socket ID from the shared memory segment and assigns it to the MTP socket entry.
- Finally, it returns the socket descriptor (index in the MTP socket table) to the caller.

2.m_bind():

Purpose:

This function binds the MTP socket to a specific IP address and port.

Functionality:

- It retrieves the UDP socket ID associated with the given MTP socket descriptor.
- It populates the SOCK_INFO structure with the provided IP address and port.
- It signals semaphore 1 and waits on semaphore 2 for synchronization.
- If successful, it updates the MTP socket table with the assigned UDP socket ID and other information.

- Finally, it returns 1 upon successful binding.

3. m_sendto():
Purpose: This function sends data over the MTP socket by writing to the sender buffer.
Functionality:
- It retrieves the sender buffer associated with the given MTP socket descriptor.
- It writes the data to the next available slot in the sender buffer.
- If the sender buffer is full, it returns an error. Otherwise, it returns the length of the data sent.

4. m_recvfrom():
Purpose: This function receives data from the MTP socket by reading from the receiver buffer.
Functionality:
- It retrieves the receiver buffer associated with the given MTP socket descriptor.
- It reads data from the receiver buffer and copies it to the provided buffer.
- If the receiver buffer is empty, it returns an error. Otherwise, it returns the length of the data received.

5.m_close():
Purpose: This function closes the MTP socket and releases associated resources.
Functionality:
- It marks the corresponding entry in the MTP socket table as free.
- It resets various fields in the MTP socket entry.
- It releases any shared memory segments associated with the MTP socket.
- Finally, it returns 0 upon successful closure.

File: init_msocket.c
multi-threaded program that implements a My Transport Protocol (MTP) using shared memory and UDP sockets.

It continuously checks if there are any pending messages to send and any messages yet to receive or close the appropriate sockets, It maintains the socket table, sock info shared memory segments to implement MTP.

Thread S Function:

- Periodically wakes up to perform tasks related to sending messages.
- Sleeps for a specific duration, less than T/2.
- Checks for message timeouts and retransmits messages within the current send window if necessary.
- Sends pending messages from the sender-side buffer to the receiver.
- Updates send timestamps for transmitted messages.

Thread R Function:

- Waits for incoming messages on UDP sockets.
- Processes ACKs and data messages received from the sender.
- Sends duplicate ACKs when timeouts occur to indicate receipt of messages.
- Updates the receiver buffer based on received messages and maintains the receiver window.
- Delivers in-order messages to the user from the receiver buffer.

Garbage Collector Thread:

- Periodically checks for terminated processes associated with MTP sockets.
- Closes sockets and frees resources for terminated processes to prevent resource leakage.
- Ensures proper cleanup and resource management by closing sockets when processes terminate.
- Helps maintain system stability and prevents resource exhaustion by reclaiming resources from terminated processes.

is_timeout(int last_time, int curr_time):

- Determines whether the timeout period for a message has elapsed.
- Takes the last time the message was sent and the current time as input.
- Returns true if the difference between the current time and the last time is greater than T (timeout period), indicating a timeout.

retransmit(int i):

- Retransmits all messages within the current send window for a specified MTP socket.
- Uses semaphores (Sem1 and Sem2) to synchronize access to shared resources.
- Accesses shared memory structures (MTP_Socket_Info and SOCK_INFO) to retrieve information about the socket and messages.
- Constructs a destination address (dest_addr) for sending messages.
- Iterates through messages within the send window, resending them via UDP using sendto().

cleanup():

- Cleans up resources used by the program.
- Closes all socket file descriptors.
- Removes the shared memory segment.
- Called upon program termination or by the signal handler for SIGINT (Ctrl+C).

sigint_handler(int signum):

- Signal handler function for SIGINT (Ctrl+C).
- Invokes the cleanup() function to perform cleanup operations before terminating the program.