# Intermediate Submission Report

## *(21CS30024, 21CS30040)*

## *Overview*

The intermediate submission of our project involves the implementation of essential functionalities for the Message Transfer Protocol (MTP) library. We have completed the initial set of functions including m_socket, m_bind, and m_close. Additionally, we have partially implemented m_sendto with stop-and-wait flow control. The current code works well for two sockets.

## *Implemented Functions*

### ➢ 1. m_socket :

1. Functionality:
   Creates an MTP socket and initializes necessary data structures.
2. Parameters:
   a. domain: Specifies the communication domain; for MTP, typically AF_INET.
   b. type: Specifies the socket type; for MTP, predefined macro SOCK_MTP.
   c. protocol: Specifies the flag.
3. Implementation Details:
   a. Shared Memory: Used to store socket information and state.
   b. Semaphores: Employed for synchronization during socket creation and access to shared memory.
   c. Synchronization Mechanism: Ensured exclusive access to shared memory segments using semaphores. Semaphore operations (sem_wait, sem_post) were used to synchronize critical sections where multiple processes/threads could access shared data simultaneously.

### ➢ 2. m_bind :

1. Functionality:
   a. Binds an MTP socket to another specific MTP socket.
2. Parameters:
   a. sockfd: The socket descriptor returned by m_socket.
   b. src_addr: The source IP address to bind to.
   c. src_port: The source port to bind to.

    d. dest_addr: The destination IP address to bind to.
    e. dest_port: The destination port to bind to.
3. Implementation Details:
    a. Shared Memory: Utilized for storing socket binding information. Semaphores: Used to synchronize access to shared memory segments. Synchronization Mechanism: Ensured mutual exclusion while updating socket binding information using semaphores. Semaphore operations were used to acquire and release locks to prevent concurrent modifications to shared data.

## ➢ 3. m_close :

1. Functionality: Closes an MTP socket and releases associated resources.
2. Parameters:
    a. sockfd: The socket descriptor of the socket to be closed.
3. Implementation Details:
    a. Shared Memory: Accessed to reset socket states and release resources. Semaphores: Used for synchronization during socket closure. Synchronization Mechanism: Ensured exclusive access to shared memory segments using semaphores. Semaphore operations were employed to synchronize socket closure operations and avoid conflicts with other socket operations.

## ➢ 4. m_sendto :

Sends data to a specific destination using MTP with stop-and-wait flow control.

1. Parameters:
    a. sockfd: The socket descriptor returned by m_socket.
    b. buf: Pointer to the data buffer containing the message to be
    c. sent. len: Length of the message in bytes.
    d. dest_addr: The destination IP address.
    e. dest_port: The destination port.
2. Implementation Details:
    a. Shared Memory: Used for synchronization between sender and receiver threads.
    b. Semaphores: Employed for thread synchronization and communication.
    c. Synchronization Mechanism: Implemented stop-and-wait flow control using semaphores. Sender thread waits for acknowledgement from the receiver before sending the next packet. Semaphores were used to signal the sender thread to send data and the receiver thread to acknowledge receipt.

# ***Defined Structures and their uses***

1. **SOCK_INFO:**
    a. Purpose: Represents temporary information about a socket.
    b. Fields:
        i. sock_id: Integer representing the socket ID.
        ii. IP: Array of characters representing the IP address.
        iii. port: Integer representing the port number.
        iv. errorno: Integer representing any error number associated with the socket.
2. **MTPSocket:**
        i. Purpose: Represents shared memory data structure for the MTP (Message Transfer Protocol).
        ii. Fields:
            1. free: Boolean indicating whether the socket is free or in use.
            2. pid: Process ID of the process that owns the socket.
            3. UDPsocID: file descriptor of the corresponding UDP socket.
            4. des_IP: Array of characters representing the destination IP address.
            5. nospace: Boolean indicating if there is no space in the receive buffer.
            6. sendNospace: Boolean indicating if there is no space in the send buffer.
            7. des_port: Integer representing the destination port for sending messages.
            8. sbuf: 2D array representing the send buffer for holding messages waiting to be sent.
            9. rbuf: 2D array representing the receive buffer for holding received messages.
            10. source_IP: Array of characters representing the source IP address.
            11. current_send: Integer representing the current size of the send buffer.
            12. source_port: Integer representing the source port.
            13. l_send: Integer representing the index of the slot where the next message will be written in the send buffer.
            14. f_send: Integer representing the index of the slot from where the next message will be read in the send buffer.
            15. ack_num: Integer representing the acknowledgment number for received messages.

## ***Makefiles and Running Instructions***

1. Makefiles :Three Makefiles are provided:
    a. Makefile_lib.mak:
        i. Purpose: Compiles the msocket library.
        ii. Creates: libmsocket.a library.
        iii. Usage: No direct execution. Used for building the msocket library.
    b. Makefile_init.mak:
        i. Purpose: Compiles the initmsocket executable.
        ii. Creates: initmsocket executable.
        iii. Usage: Run initmsocket in a terminal to initialize and manage the msocket environment.
    c. Makefile_app.mak:
        i. Purpose: Compiles user process executables.
        ii. Creates: Executables for user processes.
        iii. Usage: Run the generated executables for user processes in separate terminals.
2. Running Instructions :Follow these steps to run the code.
    a. Compile the Code: Navigate to the directory containing the code files. Run ‘make -f Makefile_lib.mak’, ‘make -f Makefile_app.mak’, ‘make -f Makefile_init.mak’ to compile the code. This will generate the necessary executables and libraries.
    b. Run the Initmsocket Process: Open a terminal. Navigate to the directory containing the compiled files. Run ./initmsocket to start the initialization process. This process continuously initializes and manages the msocket environment.
    c. Run User Processes: Open two additional terminals. In each terminal, navigate to the directory containing the compiled files. Run your desired user processes using the provided executables. These processes interact with the msocket environment.

## ***Design Considerations***

- ➢ Utilized shared memory segments and semaphores for inter-process communication and synchronization.
- ➢ Ensured proper error handling and cleanup mechanisms in each function.
- ➢ Implemented basic error checking and validation for function arguments and socket states.

## ***Challenges***

Ensuring proper synchronization between sender and receiver threads in the m_sendto function. Handling errors and edge cases effectively to maintain the stability and reliability of the library.

## ***Next Steps***

Complete the implementation of m_sendto, including timeout handling and retransmission.
Implement the m_recvfrom function to receive data from MTP sockets.
Test the implemented functionalities thoroughly to ensure correctness and reliability.
Refactor and optimize code where necessary for better performance and readability.

## ***Conclusion***

In this intermediate submission, we have made significant progress in implementing the core functionalities of the MTP library. While there are still some incomplete parts and challenges to address, we are confident in our approach and look forward to completing the remaining tasks in the final submission.