```
Help on module netpack:

NAME
    netpack - Networking module for Linux, Developed by Sujoy Mondal (SMN), Dept of ECE, RCCIIT

CLASSES
    builtins.object
        CRC_Class
    threading.Thread(builtins.object)
        ConnectClient_thread

    class CRC_Class(builtins.object)
     |  Methods defined here:
     |
     |  __init__(self)
     |
     |  crc_execute(self, message, key)
     |      Args:
     |          - message = bits as string
     |          - key = bits as string
     |      Returns:
     |          - remainder of mod-2 division
     |
     |  decode_data(self, encoded_data, key)
     |      Args:
     |          - encoded_data = bits as string
     |          - key = bits as string
     |      Returns:
     |          - remainder, message
     |
     |  encode_data(self, data, key)
     |      Args:
     |          - data = bits as string
     |          - key = bits as string
     |      Returns:
     |          - remainder, codeword
     |
     |  xor(self, a, b)
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
```

```
 |      list of weak references to the object (if defined)

class ConnectClient_thread(threading.Thread)
 |  ConnectClient_thread(client_obj, client_addr, msg_count)
 |
 |  Method resolution order:
 |      ConnectClient_thread
 |      threading.Thread
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, client_obj, client_addr, msg_count)
 |      Initialize a ClientThread instance.
 |
 |      Args:
 |      client_obj: The client object associated with this thread.
 |      client_addr: The address of the client.
 |      msg_count (int): The count of messages.
 |
 |      The function initializes the ClientThread instance by assigning values to its attributes:
 |      - client: The client object associated with this thread.
 |      - client_addr: The address of the client.
 |      - count: The count of messages.
 |      - status: The status of the thread (initialized as True).
 |
 |      It also prints a message to indicate the creation of the client thread.
 |
 |      Returns:
 |      None
 |
 |  check_thread_status(self)
 |
 |  close_connection(self)
 |
 |  run(self)
 |      - Performs:
 |          - send client socket at 1 sec interval
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from threading.Thread:
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  getName(self)
```

```
|      Return a string used for identification purposes only.
|
|      This method is deprecated, use the name attribute instead.
|
|  isDaemon(self)
|      Return whether this thread is a daemon.
|
|      This method is deprecated, use the daemon attribute instead.
|
|  is_alive(self)
|      Return whether the thread is alive.
|
|      This method returns True just before the run() method starts until just
|      after the run() method terminates. See also the module function
|      enumerate().
|
|  join(self, timeout=None)
|      Wait until the thread terminates.
|
|      This blocks the calling thread until the thread whose join() method is
|      called terminates -- either normally or through an unhandled exception
|      or until the optional timeout occurs.
|
|      When the timeout argument is present and not None, it should be a
|      floating point number specifying a timeout for the operation in seconds
|      (or fractions thereof). As join() always returns None, you must call
|      is_alive() after join() to decide whether a timeout happened -- if the
|      thread is still alive, the join() call timed out.
|
|      When the timeout argument is not present or None, the operation will
|      block until the thread terminates.
|
|      A thread can be join()ed many times.
|
|      join() raises a RuntimeError if an attempt is made to join the current
|      thread as that would cause a deadlock. It is also an error to join() a
|      thread before it has been started and attempts to do so raises the same
|      exception.
|
|  setDaemon(self, daemonic)
|      Set whether this thread is a daemon.
|
|      This method is deprecated, use the .daemon property instead.
|
|  setName(self, name)
|      Set the name string for this thread.
```

```
|
|      This method is deprecated, use the name attribute instead.
|
|  start(self)
|      Start the thread's activity.
|
|      It must be called at most once per thread object. It arranges for the
|      object's run() method to be invoked in a separate thread of control.
|
|      This method will raise a RuntimeError if called more than once on the
|      same thread object.
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from threading.Thread:
|
|  ident
|      Thread identifier of this thread or None if it has not been started.
|
|      This is a nonzero integer. See the get_ident() function. Thread
|      identifiers may be recycled when a thread exits and another thread is
|      created. The identifier is available even after the thread has exited.
|
|  native_id
|      Native integral thread ID of this thread, or None if it has not been started.
|
|      This is a non-negative integer. See the get_native_id() function.
|      This represents the Thread ID as reported by the kernel.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from threading.Thread:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  daemon
|      A boolean value indicating whether this thread is a daemon thread.
|
|      This must be set before start() is called, otherwise RuntimeError is
|      raised. Its initial value is inherited from the creating thread; the
|      main thread is not a daemon thread and therefore all threads created in
|      the main thread default to daemon = False.
|
|      The entire Python program exits when only daemon threads are left.
```

```
     |
     |  name
     |      A string used for identification purposes only.
     |
     |      It has no semantics. Multiple threads may be given the same name. The
     |      initial name is set by the constructor.

FUNCTIONS
    accept_tcp_client(s)
        Accepts a TCP client connection on the given socket.

        Parameters:
        s (socket.socket): A socket object representing the server.

        Returns:
        tuple: A tuple containing the client socket object and its address.
               - client_socket (socket.socket): Socket object representing the client connection.
               - addr (tuple): Address information of the client (IP address, port).

        Note:
        This function blocks until a client connection is received.

    close_socket_connection(s)
        Closes the provided socket connection.

        Args:
        s (socket.socket): The socket connection to be closed.

        Returns:
        None

        Raises:
        OSError: If an error occurs while attempting to close the socket => socket closing error

    connect_to_tcp_server(s, ADDR)
        Connects a socket to a TCP server at the specified address.

        Args:
        s (socket.socket): The socket object to connect.
        ADDR (tuple): A tuple representing the server address in the form of (host, port).

        Returns:
        None

        Raises:
        OSError: If the connection to the server fails.
```

```
create_process(func_name, args_list)
    Create a process targeting a given function with arguments.

    This function creates a multiprocessing Process object that targets the provided
    function using the specified arguments.

    Args:
    func_name (function): The function to be executed in the new process.
    args_list (tuple or list): The arguments to be passed to the function. If a list is
        provided, it will be converted to a tuple before passing it as arguments.

    Returns:
    multiprocessing.Process: A Process object targeting the specified function with the
        given arguments.

create_queue()
    Create and return a new Queue instance.

    Returns:
    Queue: A new instance of the Queue class.

create_tcp_socket()
    Creates a TCP socket.

    The socket is configured with a socket option to allow reusing ports to prevent errors
    related to address already in use.

    Returns:
    A TCP socket ready for use.

create_tcp_timeoutsocket(delay=0.2)
    Create a TCP socket with a specified timeout.

    Creates a TCP socket with the option to set a timeout for socket operations.

    Args:
    delay (float, optional): The timeout value in seconds (default is 0.2).

    Returns:
    socket: A TCP socket object with the specified timeout.

create_udp_broadcast_client_timeoutsocket(client_port, delay=0.2)
    Create a UDP broadcast client socket with a timeout.

    This function creates a UDP broadcast client socket with the given client_port
```

```
    and sets a timeout for receiving data.

    Args:
    client_port (int): The port number to bind the client socket.
    delay (float, optional): The timeout value in seconds (default is 0.2).

    Returns:
    socket: A UDP broadcast client socket with the specified settings.

    Raises:
    OSError: If the socket creation or binding fails.

create_udp_broadcast_server_socket()
    Create a UDP broadcast server socket.

    This function creates a UDP socket configured for broadcasting.

    Returns:
    socket: A UDP socket configured for broadcasting.

create_udp_socket()
    Create a UDP socket.

    This function creates a UDP (User Datagram Protocol) socket.

    Returns:
    socket: A UDP socket ready for use.

create_udp_timeoutsocket(delay=0.2)
    Create a UDP socket with a timeout.

    Args:
    delay (float, optional): The timeout value in seconds (default is 0.2).

    Returns:
    socket: A UDP socket with the specified timeout.

join_process(P)
    Join a given process and wait for it to complete.

    Args:
    P (multiprocessing.Process): The process to be joined.

    Returns:
    None
```

```
my_ip()
    Get the local IP address of the current machine.

    Tries to retrieve the local IP address by getting the hostname
    and then fetching the IP address associated with it.

    Returns:
    str: The IP address of the current machine.

    Raises:
    OSError: If there's an issue in obtaining the IP address, print IP Error

read_data_from_tcp_client(s)
    Reads data from a TCP client socket.

    Args:
    s (socket.socket): The socket object representing the TCP client connection.

    Returns:
    str: The message received from the client after decoding.

    Raises:
    (socket.error, OSError): If there's an issue with receiving data from the socket.

read_data_from_tcp_server(s)
    Reads data from a TCP server socket.

    Args:
    s (socket.socket): The TCP server socket object from which data will be read.

    Returns:
    str: The received message from the server after decoding.

    Raises:
    OSError: If there's an issue with receiving or decoding the data.

read_data_from_udp_client(s)
    Receive data from a UDP client.

    This function receives data from a UDP client using a socket and returns
    the decoded data along with the address of the client.

    Parameters:
    s (socket): The socket used to receive data from the client.

    Returns:
```

```
    tuple: A tuple containing the received data (decoded) and the client's address.
        - data (str): The received data as a string after decoding.
        - addr (tuple): The address (IP address and port) of the client.

    Note:
    Ensure that the 'BUFSIZE' constant is defined and matches the expected buffer size.

read_data_from_udp_server(s)
    Receive data from a UDP server.

    Args:
    s (socket): The socket object for UDP communication.

    Returns:
    tuple: A tuple containing the received data as a string and the address of the sender.

    Raises:
    OSError: If there's an issue receiving data from the UDP server.

read_queue(q, delay=0.1)
    Reads an item from the provided queue after a specified delay.

    If the queue is not empty, this function waits for a specified delay
    (in seconds) before retrieving an item from the queue. If the queue is
    empty, it returns None.

    Args:
    q (queue.Queue): The queue object from which to read an item.
    delay (float, optional): The time delay (in seconds) before reading from the queue.
        Defaults to 0.1 seconds.

    Returns:
    any: Returns the item retrieved from the queue. Returns None if the queue is empty
    or if there's an issue during the retrieval.

send_data_to_tcp_client(s, data)
    Sends data to a TCP client socket.

    Parameters:
    s (socket): The TCP client socket.
    data (str or any): The data to be sent. Will be converted to a string if not already.

    Returns:
    None

    Note:
```

```
    The function sends the provided data to the given TCP client socket after converting it to a string.

send_data_to_tcp_server(s, data)
    Sends data to a TCP server.

    Args:
    s (socket.socket): The socket object connected to the TCP server.
    data (Any): The data to be sent. Will be converted to a string.

    Returns:
    None

    Raises:
    Any socket-related exceptions that might occur during the send operation.

send_data_to_udp_client(s, data, addr)
    Send data to a UDP client.

    Args:
    s (socket): The socket object used for sending data.
    data (any): The data to be sent. It will be converted to a string.
    addr (tuple): The address (IP, port) of the UDP client.

    Returns:
    None

    Note:
    This function converts the data to a string, encodes it using a specified format (FORMAT),
    and sends it to the provided address using the given socket (s).

send_data_to_udp_server(s, data, ADDR)
    Send data to a UDP server.

    Parameters:
    s (socket): The socket object used for communication.
    data (any): The data to be sent. It will be converted to a string before transmission.
    ADDR (tuple): A tuple containing the server address and port (e.g., ('127.0.0.1', 12345)).

    Returns:
    None

    Note:
    This function sends the provided data to the specified UDP server using the provided socket.
    Ensure the socket (s) is appropriately configured and connected before calling this function.

start_process(P)
```

```
    Starts the given process.

    Args:
    P (multiprocessing.Process): The process to be started.

    Returns:
    None

start_tcp_server(s, ADDR)
    Starts a TCP server on the provided socket and address.

    Binds the provided socket to the given address and starts listening for incoming
    connections. The server listens for a single connection.

    Parameters:
    s (socket object): The socket object used for the server.
    ADDR (tuple): A tuple containing the IP address and port number to bind the socket.

    Returns:
    None

start_udp_server(s, ADDR)
    Starts a UDP server by binding the provided socket to the given address.

    Args:
    s (socket.socket): The socket object to bind for the UDP server.
    ADDR (tuple): A tuple representing the address and port to bind the server.
                Format: (address, port)

    Returns:
    None

update_BUFSIZE(val=1024)
    Update the buffer size used in the application.

    Args:
    val (int, optional): The new buffer size value to be set. Defaults to 1024.

    Returns:
    None

    Notes:
    This function updates the global variable BUFSIZE to the provided value.
    It also prints a message indicating the updated buffer size.

write_help(func, out_file)
```

```
    write_queue(q, data, delay=0.1)
        Writes data to the provided queue after a specified delay.

        This function puts the provided data into the queue and then
        sleeps for a specified duration before returning the queue.

        Args:
        q (queue.Queue): The queue to which data will be added.
        data (any): The data to be added to the queue.
        delay (float, optional): The duration (in seconds) to wait after
            adding the data to the queue. Defaults to 0.1 seconds.

        Returns:
        queue.Queue: The queue after adding the data.

DATA
    BUFSIZE = 1024
    FORMAT = 'utf-8'
```