Maiah Pardo
CruzID: mapardo

Design Doc: Asgn1: httpserver.c

- **main()**
    - Checks to make sure we have the correct number of arguments
    - Creates a port number (by converting a string to a 16 bit unsigned integer) using the given strtouint16() function
    - Checks port number is valid
    - Creates a listening socket given the port number (calls create_listen_socket() function)
    - Infinite loop: listens for requests and accepts them using accept() function
        - When a request has been found, calls handle_connection() with the request

- **strtouint16():** (given to us)
    - Converts a string to a 16 bits unsigned integer
    - returns 0 if the string is malformed or out of range

- **create_listen_socket():** (given to us)
    - creates a socket for listening for connections
    - Closes the program and prints an error message on error

- **handle_connection():**
    - Receives the request from the client
    - Parses through the request
    - Checks for a Content-Length value (and saves it in the case of a PUT)
    - Checks that Host name has no white space
    - Checks that all other headers are valid and contain a ":"
    - Checks that the version is HTTP/1.1
        - If the version is incorrect:
            - send a 400 bad request and close the connection
    - Checks that the filename is 19 characters long
        - If the filename is longer than 19 characters:
            - send a 400 bad request and close the connection
    - Checks that the filename is a valid by consisting of alphanumeric, '.', and '_' characters only
        - If the filename does not use valid characters send a 400 bad request and close connection
    - Remove the beginning '\' in the filename
    - Checks to make sure filename is up to 19 characters or less
        - If filename is larger than 19 characters, send bad request and close connection

- checks to make sure that we are given a PUT, GET, or HEAD request depending on the request type the client asked for
    - calls handlePut(), handleGet(), or handleHEAD() functions respectively
    - If we have a different request that is not PUT, GET, or HEAD:
        - send a 501 Not Implemented error message and close the request
- return to main (main contains the main loop where the server continues listening for a new connection)

- **handlePut():**
    - Given connection file descriptor, filename, and content-length
    - Check if file exists using access
        - If the file does exist, check if it has write permission
            - If it does not have write permission, send forbidden message and close connection
    - Try to open the file, or create it if it does not already exist
    - If file was not able to be opened:
        - Check if file was not found (if not, send file not found message and close connection)
        - Check if file cannot be accessed (if not, send forbidden message and close connection)
        - For any other reason, send internal server error and close connection
    - Receive the exact number of bytes (which is the content length) of data from the client and write it to the file
        - If recv() returns an error, possible error with connection, send Internal Server Error and close the connection
    - If file was created, send the created message and close the connection
    - If file was not created and already exists, send the OK message and close the connection
    - return back to handle_connection()

- **handleGet():**
    - Given connection file descriptor and filename
    - Check if file exists using access
        - If file does not exist, send file not found and close connection
    - Check if file has read permission
        - If file does not have read permission, send forbidden message
    - Try to open file as read only
        - If error upon opening:
            - Check if file does not exist (if so, send file not found and close connection)
            - Check if file cannot be opened for reading (if so, send forbidden message)
            - Any other reason, send internal server error message and close connection

    - If file could be opened successfully:
        - grab the file size
        - Send an OK message to the client, with a Content-Length which is the size of the file
        - send the file contents
            - If send() returns an error, error with connection, send Internal Server Error message and close connection
    - Close the connection
    - return to handle_connection()

- **handleHead():**
    - Given connection file descriptor and filename
    - Check, using access(), if file exists
        - If it does not, send a file not found message and close the connection
    - Check, using access(), if the file has read permission
        - If it does not, send a forbidden message and close the connection
    - Try to open the file
    - If the file could not be opened,
        - Check that the file exists (if not, send file not found message and close the connection)
        - Check that the file can be accessed (if not, send the forbidden message and close the connection)
        - For any other reason, send the internal server error message and close the connection
    - retrieve the size of the size
    - send an OK message to the client, with the Content-Length value as the size of the file
    - Close the connection
    - Return back to handle_connection()