```python
#!/usr/bin/env python3
# See https://docs.python.org/3.2/library/socket.html
# for a decscription of python socket and its parameters
import socket

import os
import stat
import sys
import urllib.parse
import datetime

from threading import Thread
from argparse import ArgumentParser

BUFSIZE = 4096
CRLF = '\r\n'
METHOD_NOT_ALLOWED = 'HTTP/1.1 405 METHOD NOT ALLOWED{}Allow: GET, HEAD, POST{}Connection: close{}{}'.format(CRLF, CRLF, CRLF, CRLF)
OK = 'HTTP/1.1 200 OK{}{}{}'.format(CRLF, CRLF, CRLF, CRLF)
NOT_FOUND = 'HTTP/1.1 404 NOT FOUND{}Connection: close{}{}'.format(CRLF, CRLF, CRLF, CRLF)
FORBBIDDEN = 'HTTP/1.1 403 FORBIDDEN{}Connection: close{}{}'.format(CRLF, CRLF, CRLF, CRLF)
MOVED_PERMANENTLY = 'HTTP/1.1 301 MOVED PERMANENTLY{}Location: https://www.cs.umn.edu/{}Connection:close{}{}'.format(CRLF, CRLF, CRLF, CRLF)

def get_contents(fname):
    with open(fname, 'r') as f:
        return f.read()

def check_perms(resource):
    stmode = os.stat(resource).st_mode
    return (getattr(stat, 'S_IROTH') & stmode) > 0

def client_talk(client_sock, client_addr):
    print('talking to {}'.format(client_addr))
    data = client_sock.recv(BUFSIZE)
    while data:
      filename = data.split()[1]
      f = open('MyContacts.html')
      outputdata = f.read()
      # print(data.decode('utf-8'))
      client_sock.send(bytes('HTTP/1.0 200 OK\n', 'utf-8'))
      client_sock.send(bytes('Content-Type: text/html\n', 'utf-8'))
      client_sock.send(bytes('\n', 'utf-8')) # header and body should b
      for i in range(0, len(outputdata)):
          client_sock.send(bytes(outputdata[i], 'utf-8'))
      client_sock.shutdown(1)
      client_sock.close()
      break
      # data = client_sock.recv(BUFSIZE)

    # clean up
    # client_sock.shutdown(1)
    # client_sock.close()
    # print('connection closed.')

class EchoServer:
  def __init__(self, host, port):
    print('listening on port {}'.format(port))
    self.host = host
    self.port = port

    self.setup_socket()

    self.accept()

    self.sock.shutdown()
    self.sock.close()

  def setup_socket(self):
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sock.bind((self.host, self.port))
    self.sock.listen(128)

  def accept(self):
    while True:
      (client, address) = self.sock.accept()
      th = Thread(target=self.accept_request, args=(client, address))
      th.start()

  def accept_request(self, client_sock, client_addr):
      data = client_sock.recv(BUFSIZE)
      req = data.decode('utf-8')
      response = self.process_request(req)
      client_sock.send(bytes(response, "utf-8"))
      client_sock.shutdown(1)
      client_sock.close()
      print('connection closed.')

  def process_request(self, request):
      linelist = request.strip().split(CRLF)
      reqline = linelist[0]
      rlwords = reqline.split() #Method, URL, HTTP
      if len(rlwords) == 0:
          return ''
      if rlwords[0] == 'HEAD':
          resource = rlwords[1][1:] #Skip beginning /
          return self.head_request(resource)
      elif rlwords[0] == 'GET':
          resource = rlwords[1][1:] #Skip beginning /
          return self.get_request(resource)
      else: #ad elif for get, post
         return METHOD_NOT_ALLOWED

  def head_request(self, resource):
      path = os.path.join('.', resource)
      if not os.path.exists(resource):
          ret = NOT_FOUND
      elif not check_perms(resource):
          ret = FORBIDDEN
      else:
          ret = OK
      return ret

  def get_request(self, resource):
      path = os.path.join('.', resource)
      if not os.path.exists(resource):
          ret = NOT_FOUND
      elif not check_perms(resource):
          ret = FORBIDDEN
      else:
          if (path.endswith(".jpg")):
            filetype = 'image/*'
          if (path.endswith(".png")):
             filetype = 'image/*'
          elif (path.endswith(".css")):
            filetype = 'text/css'
          else:
            filetype = 'text/html'
          file_contents = get_contents(path)
          # print(data.decode('utf-8'))
          ret = 'HTTP/1.0 200 OK\nContent-Type: ' + str(filetype) + '\n\n'
          ret += file_contents
      return ret


def parse_args():
  parser = ArgumentParser()
  parser.add_argument('--host', type=str, default='localhost',
                      help='specify a host to operate on (default: localhost)')
  parser.add_argument('-p', '--port', type=int, default=9001,
                      help='specify a port to operate on (default: 9001)')
  args = parser.parse_args()
  return (args.host, args.port)


if __name__ == '__main__':
  (host, port) = parse_args()
  EchoServer(host, port)
```