```
#!/usr/bin/env python3

# See https://docs.python.org/3.x/library/socket.html
# for a description of python socket and its parameters
#
# Copyright 2019, Shaden Smith, Koorosh Vaziri,
# Niranjan Tulajapure, Ambuj Nayab,
# Akash Kulkarni, Ruofeng Liu and Daniel J. Challou
# for use by students enrolled in Csci 4131 at the University of
# Minnesota-Twin Cities only. Do not reuse or redistribute further
# without the express written consent of the authors.
#
import socket

#add the following
import socket
import os
import stat
import sys
import urllib.parse
import datetime

from threading import Thread
from argparse import ArgumentParser



BUFSIZE = 4096
#add the following
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)
NOT_FOUND = 'HTTP/1.1 404 NOT FOUND{}Connection:
close{}{}'.format(CRLF, CRLF, CRLF)
FORBIDDEN = 'HTTP/1.1 403 FORBIDDEN{}Connection:
close{}{}'.format(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):
    """Returns True if resource has read permissions set on
'others'"""
    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:
      print(data.decode('utf-8'))
      data = client_sock.recv(BUFSIZE)

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

class HTTP_HeadServer:  #A re-worked version of 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=client_talk, args=(client, address))
      th = Thread(target=self.accept_request, args=(client, address))
      th.start()

  # here, we add a function belonging to the class to accept
  # and process a request
  def accept_request(self, client_sock, client_addr):
    print("accept request")
    data = client_sock.recv(BUFSIZE)
    req = data.decode('utf-8') #returns a string
    response=self.process_request(req) #returns a string
    #once we get a response, we chop it into utf encoded bytes
    #and send it (like EchoClient)
    client_sock.send(bytes(response,'utf-8'))

    #clean up the connection to the client
    #but leave the server socket for recieving requests open
    client_sock.shutdown(1)
    client_sock.close()
  #added method to process requests, only head is handled in this code
  def process_request(self, request):
    print('######\nREQUEST:\n{}######'.format(request))
    linelist = request.strip().split(CRLF)
    reqline = linelist[0]
    rlwords = reqline.split()
    if len(rlwords) == 0:
        return ''

    if rlwords[0] == 'HEAD':
        resource = rlwords[1][1:] # skip beginning /
        return self.head_request(resource)
    else: #add ELIF checks for GET and POST before this else..
        return METHOD_NOT_ALLOWED

  def head_request(self, resource):
    """Handles HEAD requests."""
    path = os.path.join('.', resource) #look in directory where server
is running
    if not os.path.exists(resource):
      ret = NOT_FOUND
    elif not check_perms(resource):
      ret = FORBIDDEN
    else:
      ret = OK
    return ret

#to do a get request, read resource contents and append to ret value.
#(you should check types of accept lines before doing so)
# You figure out the rest

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()
  HTTP_HeadServer(host, port) #Formerly EchoServer
```