Questions:

1. What happens if the file being requested is not inside the chroot? Does it matter?

   - Since we are passing the file descriptor from parent process to child process, the child process uses that file descriptor to read the contents of the file. So it does not matter if the file requested in not inside chroot.

2. Explain the mechanism you are using to pass file descriptors between parent and child processes.

   - I am passing the file descriptor as a command line argument to the child process. I have converted the data type of file descriptor to match the data type of command line arguments.

     char * exec_args[4]={"./child",(char*)&new_socket, fd ,NULL};

   - Once the file descriptor is received by child process, it converts the type to int and reads the file.

3. What happens if the file size of the disk file exceeds the size of the client's hardcoded buffer?
   Does the client have sufficient checks to ensure no buffer overruns occur?

   - client reads the value from socket as - **valread = read(sock , buffer, 1024);** Since the size of the buffer is 1024, the buffer array gets filled with the file content from socket from buffer[0] to buffer[1023]. If the size of the content sent is 1024 or more, buffer[1023] will get the 1024th character of the content and not '\0'. Thus we get unexpected characters for buffer[1023] when we print the buffer in stdout. This leads to buffer overflow. client.c does not have enough checks to overcome the same.

   - If we change the read to **valread = read(sock , buffer, 1023);** , only 1023 characters will be read from the socket even if the content sent has more than 1023 characters. This will prevent from buffer overflow.