

CS 39006: Assignment 5
File Transfer in Blocks
Assignment Date: 07-Feb-2019
Deadline: 14-Feb-2019 2:00 PM

Objective:

The objective of this assignment will be transfer files between two hosts with block based transfer. We'll learn a special flag in `recv()` call named `MSG_WAITALL`.

Problem Statement:

The overall protocol is mostly similar to the one that you did in Assignment 2 for file transfer over a stream socket, except that here the transfer will be through fixed sized blocks. Your task will be to write two programs - one program corresponding to the server process and another program corresponding to the client process. The client process requests for the content of a file (by providing the file name) and the server process sends the contents of that file to the client. The file is a text file of arbitrary size. The transfer of the contents of the file works using a communication protocol as follows.

1. The client establishes a connection to the server using the `connect()` call.
2. The client reads the filename from the user (keyboard).
3. The client sends the file name to the server.
4. The server looks for the file in the local directory, if the file is not there it sends an error message "E" (a single character message) to the client and closes the connection. The client receives this message, prints an error message over the screen and exits after closing the socket.
5. If the file is present, the server first sends a message "L" followed by an integer `FSIZE` indicating the size of the file. The client receives this message and waits for the file data to be received in blocks.
6. The server divides the file in a predefined block size `B` which is an integer. `B` is known to both the server and the client a priori. Assume, $B = 20$ bytes. Note that if the file size is `FSIZE`, then there will be $(FSIZE/B)$ (integer division) number of blocks of size `B` and the last block will be of size $(FSIZE \% B)$ (remainder operation) bytes. Every `send()` call for file transfer sends data not more than that of a single block to the client.
7. With every `recv()` call, the client receives the message block exactly of size `B` bytes except for the last block. With reception of every blocks, it maintains a running count of the number of blocks received, and accordingly calculates the size of the last block. For the last block, it waits for exactly $(FSIZE \% B)$ bytes.
8. As the client receives data in blocks, it copies the same into a new file in the local directory. Once the client has received all the blocks, it closes the connection and exits.
9. Note that no special end of file marker is sent to the client. Client understands the end of file transfer from the number of bytes received.

10. After the entire file is transferred, the client prints a message "The file transfer is successful. Total number of blocks received = X, Last block size = Y", where X is the total number of blocks received and Y is the size of the last block in bytes.

You need to ensure the followings in your code:

1. The `recv()` call for receiving the message code "E" and "L", along with the file size parameter, should receive exactly the required number of bytes. So, one `recv()` call will receive "E" or "L"; if the received code is "L", then the next `recv()` call will receive exactly the bytes corresponding to the file size parameter.
2. A single `recv()` call for receiving a block of data should receive all the bytes for a single block only; it should not receive any bytes from the previous block or from the next block.
3. You cannot use `fopen/fscanf/fprintf` functions. You must use `open/read/write` functions to read/write from/to the file.

Submission Instruction:

You should write two C programs corresponding to the server and the client. Keep these two files in a single compressed folder (zip or tar.gz) having the name <roll number>_Assignment5.zip or <roll number>_Assignment5.tar.gz. Upload this compressed folder at Moodle course page by the deadline (14 Feb 2019 2:00 PM).