Server Design:
Vincent Taylor
Harsh Patel

The first thing we focused on was the how we were going to be sending the data from the CSV on the client side to the server. We decided to use a particular format of header metadata and the CSV data. So it looks like sizeofbuffer@CSVdata. This was an important thing to do because this way both the client and the server knew how to encode and decode given data. My partner and I were tasked with creating the server. We set up all the necessary structs and were listening on a port till a client is connected. On accepting the connection the server sends them an acknowledgement message. After, the client starts feeding us data. For each line we are sent there are functions to decode the line received so we are left with raw data. Then another function parses the line which is basically a row from the CSV and separates all the token and add it to a struct. Server keeps adding data into the file till the client sends the record parameter. When the client sends a sort request, the server asks for a sorting field id which is basically an int corresponding to the column to be sorted. The server then sorts the struct and saves the sorted fields to a file. The filename is saved in another global struct. On a "return" request the server sends all the data from the struct back to the client encoding the same way it was received.

Client Design:
Rajan Patel
Edward Patrick Schwerdtfeger
Using the gnu11 C standard is my preference, but I don't know if it would compile under older or ISO dialects. The size of the threadpool can be changed by changing the gcc command in the makefile.

Thread pool size should be decided based on the number of physical threads available. The exact relationship depends on how CPU vs memory heavy the program is.

How to use our code:
Server: There is a makefile. Use the command make to compile the files and ./server to run.
Client: There is a makefile. Run 'make' to build the project. ./sorter_client to run

Files:
1. Server:
    a. Makefile: for compiling the files necessary.
    b. Sorter_server.c: contains all the code for creating the server. Also contains functions for encoding, decoding, and parsing.
    c. Sorter_server.h: contains the struct definition for holding Movie data and another struct for the filenames that we create.
    d. Sorter.c: contains all the sorting function.

2. Client:
    a. Makefile
    b. sorter.c
    c. sorter.h

API:

1. Server:
   int headerDigitCount(int c_s);
   void getRecord(char* record, int c_s, int length);
   int byteCount(int c_s, int digitCount);
   char* checkIf(char * p);
   Checks for space

   int isspace(int c);
   Needed for checkIf

   void Print(struct mData records[], int size);
   A debugging function for printing out all the field from the struct.

   int parse_line(char * line);
   Parses a given line and save to a struct.

   Void print2file(FILE *nf, struct mData records[], int size);
   Function to print data to a file.

2. Client.

   void get_str(char**, char**);
   char* trim(char* str);
   void* newdir(void* pathin);
   void* newfile(void* pathin);
   int read_sock(int sockfd, char* buf, int len, int flags, fd_set *socks)