PZIP Design Implementation

General idea of “parallelizing” the zip function:

1. Have multiple threads reading the file while one thread will write our the results and one will serve as our task manager thread
    a. Read threads: all available threads
    b. Assign 1 thread for writing, and one for managing
2. Writing will be essentially serialized as the printing as to be in order
    a. A bottleneck in the runtime
3. Each reading thread has a 4k chunk of the file to read
    a. Threads should read in parallel and write the number of occurrences of a character and the character itself to their own buffer.
    b. There will be a combo buffer that holds how many combinations of integer+char sequences we have seen
    c. Thread will signal when it is done reading
4. New structures will be used to store information
    a. Thread_control struct - holds information about specific thread being run such as a thread number + file
    b. File struct - holds information about the file as well as locks, conditions, number of read threads

Manager thread:

In the manageThread function, we are going to assign the number of reading threads, write thread, and the chunk size which will depend our chosen buffer size. The buffer and combo buffers will be allocated the proper amount of memory. The loop size will also be calculated by using the file size and the chunk size. There will be two buffers to keep track if the read is complete and if one of the threads has the end of the file. For the number of loops it takes to get through the entire file, we will create a read thread for each one.

Creating Read Threads:

In the create_read_thread function, we will iterate through the number of threads needed to be made and every thread will have its own control thread that has its thread id and the file. It will create a thread with a real pthread id with our read file and the control info thread.

Read File function:

In the read_file function, we will first create a thread_control and a file struct that will store all of our thread-specific and file information respectively. the starting position in the file found by multiplying how many groups of threads running by the number of read threads and then adding thread id...and then multiplying by the file's chunk size that we decide to split it by in the manager thread. The end position is found by adding a chunk size to the starting position. We will also be keeping track of which thread has the end of the file and if runs are completing a whole combination. Depending on which condition it is, we will place the number of occurrences and the character into the buffer. We will iterate through the file, and if we see the character again, we will increment the number of the occurrences. If the occurrences is greater than 0, put that into the file buffer at the combo's position, increment the combo position by four (size of int), then put the character in the combo position. We will then use the lock to signal that we are done reading.

Pzip function:

We will acquire the lock in the beginning of the function. Iterate through the number of read threads, and while the file has not been read, wait for that condition to be completed before moving on. We will be checking if the thread has the end of file, and we will be setting the starting position and ending position as well as the starting char and starting integer if the thread is not the end of the file. Then, we will be write the contents of the buffer into an output file. If the file does have the end of the file, then we will print out the last character saved with its number and character from the ending position in the buffer. We will also be updating the last char and int to whatever was at the end of the buffer.

These functions should work alongside each other to read the file in multiple chunks, process those chunks, and then send the results to one write thread.