Mert Coşkuner-29120
I wrote my code in C language standards.

Follow-up structures:
I used an array-based structure to keep track of the processes and the threads.

Process Array: To keep track of the process I created a struct that includes an integer array and size variable. This will help me with the push operations through this array.

Thread Array: To keep track of the Threads I created a struct that includes a thread array and size variable. This will help me with the push operations through this array.

pushBack_f: This is the pushback function of the process array this is an operation similar to the push_back function in C++. It checks if the array has reached its maximum value and adds to the end of the array.

pushBack_t: This is the pushback function of the thread array this is an operation similar to the push_back function in C++. It checks if the array has reached its maximum value and adds to the end of the array.

Thread Function Structure:

After creating the thread, the thread function will be reading the values from the read end of the pipe. To enable this operation we are passing the read end of the pipe to the thread function as an argument. Furthermore, in the function, I implemented a lock-based structure to ensure that there is no collision between threads while printing out the outputs. If the thread is successfully obtained the lock it will go through the pipestream variables. Basically, by using fdopen I opened the pipe's read end after that I checked through it whether it had information or not. If it has an information thread will start to print out the console's number. After that, I add an extra layer of control to check that the pipe_line buffer has information inside to program it not to crash. Moreover, after controlling that it can reach print statements. After completing its execution it releases the lock by setting the islistening variable to 0, closes the pipestream variable and lastly, with pthread_exit we are sure that the thread is finished.

Main flow:
First of all, I initialized the arrays to keep track of the processes and arrays. To start the parsing operation, I first opened commands.txt and after that, I created a file to parse into it. To read the lines, I used the fgets function which will read through the end of the file. After that, I set pre-defined variables for the Command, Inputs, Options, Redirection, and Background jobs. To handle the words I used the strtok function (“new” is the word variable). After that, I created a command array, to handle execvp operations, a redirected input name variable to get the input name, and a redirected output name to get the output name. After that with a while loop, I partialized the the lines and set these variables. After setting these variables it is time to send them to the command array which will be executed by execvp because of that I used the strdup function to set these variables and a NULL at the end. Moreover, by using the dup function I saved the current stdout to an integer variable after that using dup2 I redirected stdout to parse text and write through the file. After that with the set saved_stdout, I got the stdout back to the console and closed the saved_stdout. Now it is time to execute the commands. It can basically divided into two parts one is the wait command which has only a command name in it. With using two for loops it waits through all of the operations executed by processes and threads. The other part is divided by the redirection types. There are three types of redirection: “-”, “>”, “<”. With if else statements our commands go through these lines. - and < are very similar to each other because each writes to the console and the collision between them is not wanted therefore we need a thread in them.

Let’s start with “-”, this operation initializes a thread and a pipe first and then uses the fork() system call to create the process. After using the fork() system call, it divides the relation into two: Parent and child. The child goes through the else-if statement and closes the read end of the pipe. After these operations, it will direct the stdout to the read end of the pipe and then execute. While these are operating parent will be going through the else part of the code, closing the write end of the pipe, and creating a thread. After creating the thread I checked whether it was a background job or not. As it is stated in the assignment file if it is not a background job the process and the thread will be looped through and immediately end, if it is not I push back through the arrays to keep track of them.

Secondly, “<”, this operation initializes a thread and a pipe first and then uses the fork() system call to create the process. After using the fork() system call, it divides the relation into two: Parent and child. The child goes through the else-if statement opens the input file directs the

“stdin” to that file, and lastly closes the file. After these operations, it closes the read end of the pipe directs the stdout to the write end of the pipe and then closes the write end, and then executes the execvp. While these are operating parent will be going through the else part of the code, closing the write end of the pipe, and creating a thread. As it is stated in the assignment file if it is not a background job the process and the thread will be looped through and immediately end, if it is not I push back through the arrays to keep track of them.

Lastly, “>”, this operation uses the fork() system call. After using the fork() system call, it divides the relation into two: Parent and child. The child goes through the else-if statement opens the output file redirects the stdout to the file and then closes the file lastly executes the execvp. While these are operating parent will be going through the else part of the code, creating a thread. As it is stated in the assignment file if it is not a background job the process will be looped through and immediately end, if it is not I push back through the arrays to keep track of them.

These were the operations, after these operations, I will close the files and with two loops program will iterate through the processes and threads to finish them.