```c
/*******************************************************************************
*                                                                              *
*  To Compile: gcc -Wall -O -o lab7 lab7.c                                     *
*  To run: ./lab7 <instruction file>                                           *
*  Output/Error stream is redirected to <child_pid>.out and <child_pid.err>    *
*                                                                              *
*  Author: Noah Sellers                                                        *
*  Email: sellersn@uab.edu                                                     *
*  Date: November 4, 2025                                                      *
*******************************************************************************/

#include <stdio.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main (int argc, char** argv) {
    char *file_path = argv[1];
    FILE *instruction_file;
    FILE *log_file;

    instruction_file = fopen(file_path, "r");
    if (instruction_file == NULL) {
        fprintf(stderr, "Problem opening file");
        exit(-1);
    }

    log_file = fopen("log", "w");
     if (log_file == NULL) {
        fprintf(stderr, "Problem opening file");
        exit(-1);
    }
    int LINESIZE_CHARS = 1024;
    char line[LINESIZE_CHARS];

    while (fgets(line, LINESIZE_CHARS, instruction_file) != NULL) {
        line[strcspn(line, "\n")] = 0;

        char *args[64];
        char *token;
        int i = 0;

        char line_copy[LINESIZE_CHARS];
        strncpy(line_copy, line, LINESIZE_CHARS);

        token = strtok(line_copy, " ");
        while (token != NULL) {
            args[i] = token;
            i++;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;

        time_t start_time = time(NULL);
        time_t end_time;

        pid_t pid;

        pid = fork();

        if (pid == -1) {
            perror("fork");
            exit(-1);
        }
        else if (pid == 0) {

            //Get the child’s actual PID
            pid_t child_pid = getpid();

            char stdout_file[32];
            char stderr_file[32];

            //Write the filenames to the corresponding buffers
            sprintf(stdout_file, "%d.out", child_pid);
            sprintf(stderr_file, "%d.err", child_pid);

            //Create the files and open them for writing
            FILE *fp_out = fopen(stdout_file, "w");
            FILE *fp_err = fopen(stderr_file, "w");

            //Check for errors
            if (fp_out == NULL || fp_err == NULL) {
                perror("fopen redirect");
                exit(-1);
            }

            //Convert the FILE pointers to int file descriptors
            int fd_out = fileno(fp_out);
            int fd_err = fileno(fp_err);

            //Redirect stdout and stderr to child_pid.out file
            if (dup2(fd_out, 1) == -1) { //1 is stdout
                perror("dup2 stdout");
                exit(-1);
            }

            if(dup2(fd_err, 2) == -1) { //2 is stderr
                perror("dup2 stderr");
                exit(-1);
            }

            //Close the file pointers
            fclose(fp_out);
            fclose(fp_err);

            execvp(args[0], args);

            perror("execvp");
            exit(-1);
        }

        else {
            wait(NULL);
            end_time = time(NULL);
        }

        char start_time_string[30];
        char end_time_string[30];

        strcpy(start_time_string, ctime(&start_time));
        strcpy(end_time_string, ctime(&end_time));

        start_time_string[strcspn(start_time_string, "\n")] = 0;
        end_time_string[strcspn(end_time_string, "\n")] = 0;

        fprintf(log_file, "%s\t%s\t%s\n", line, start_time_string, end_time_string);
    }

    fclose(instruction_file);
    fclose(log_file);
}
```