```c
// Name: Jessica Elkins
// Assignment: Lab 10 - CS332
// Date: 4/7/20
// BlazerID: Jelkins3
// Description: This program opens the given file, reads the contents, uses fork-exec
// to create a new process that executes the command along with the provided arguments.
// The child process redirects the stdout and stderr to the files pid.out and pid.err.

// To Compile: gcc -Wall -o lab10 lab10.c
// To Run: ./lab10 commands.txt

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

void createarray(char *buf, char **array) {
        int i, count, len;
        len = strlen(buf);
        buf[len-1] = '\0'; /* replace last character (\n) with \0 */

        for (i = 0, array[0] = &buf[0], count = 1; i < len; i++) {
                if (buf[i] == ' ') {
                        buf[i] = '\0';
                        array[count++] = &buf[i+1];
                }
        }
        array[count] = (char *)NULL;
}

int main(int argc, char **argv) {
        pid_t pid;
        int status;
        char line[BUFSIZ], buf[BUFSIZ], *args[BUFSIZ], outFileName[BUFSIZ], errFileName
[BUFSIZ];
        // string to add to the end of the child pid for stdout
        char out[] = ".out";
        // string to add to the end of the child pif for stderr
        char err[] = ".err";
        time_t t1, t2;
        int fdout, fderr, cpid;

        if (argc < 2) {
                printf("Usage: %s <commands file>\n", argv[0]);
                exit(-1);
        }

        FILE *fp1 = fopen(argv[1],"r");
        if (fp1 == NULL) {
                printf("Error opening file %s for reading\n", argv[1]);
                exit(-1);
        }

        FILE *fp2 = fopen("output.log","w");
        if (fp2 == NULL) {
                printf("Error opening file output.log for writing\n");
                exit(-1);
        }
        while (fgets(line, BUFSIZ, fp1) != NULL) {
                strcpy(buf, line); /* save line read */
                createarray(line, args);
        #ifdef DEBUG
                int i;
                printf("%s", buf);
                for (i = 0; args[i] != NULL; i++)
                        printf("[%s] ", args[i]);
                printf("\n");
        #endif
                time(&t1);
                pid = fork();
                if (pid == 0) { /* this is child process */
                        // getting the pid of the child process
                        cpid = getpid();
                        // formatting the file name to be "pid.out"
                        sprintf(outFileName, "%d%s", cpid, out);

                        // open file to write standard output stream in append mode
                        if((fdout = open(outFileName, O_CREAT | O_APPEND | O_WRONLY, 75
5)) == -1) {
                                perror("open");
                                exit(-1);
                        }

                        // formatting the file name tp be "pid.err"
                        sprintf(errFileName, "%d%s", cpid, err);

                        // open file to write standard error steam in append mode
                        if((fderr = open(errFileName, O_CREAT | O_APPEND | O_WRONLY, 75
5)) == -1) {
                                perror("open");
                                exit(-1);
                        }

                        // replacing standard output stream with the file "pid.out"
                        dup2(fdout, 1);

                        // replacing the standing error stream with the file "pid.err"
                        dup2(fderr, 2);

                        execvp(args[0], args);
                        perror("exec");
                        exit(-1);
                } else if (pid > 0) { /* this is the parent process */
                        printf("Child started at %s", ctime(&t1));
                        printf("Wait for the child process to terminate\n");
                        wait(&status); /* wait for the child process to terminate */

                        time(&t2);
                        printf("Child ended at %s", ctime(&t2));
                        if (WIFEXITED(status)) { /* child process terminated normally *
/
                                printf("Child process exited with status = %d\n", WEXIT
STATUS(status));
                        } else { /* child process did not terminate normally */
                                printf("Child process did not terminate normally!\n");

                                /* look at the man page for wait (man 2 wait) to determ
ine
                                how the child process was terminated */
                        }
                        buf[strlen(buf) - 1] = '\t'; /* replace \n included by fgets wi
th \t */
                        strcat(buf, ctime(&t1)); /* append start time to command with a
rguments */
                        buf[strlen(buf) - 1] = '\t'; /* replace \n added by ctime at th
e end with \t */
                        strcat(buf, ctime(&t2)); /* append end time */
                        fprintf(fp2, "%s", buf);fflush(fp2);
                } else { /* we have an error */
                        perror("fork"); /* use perror to print the system error message
 */
                        exit(EXIT_FAILURE);
                }
        }
        fclose(fp1);
        fclose(fp2);

        printf("[%ld]: Exiting main program .....\n", (long)getpid());
        return 0;
}
```