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

void createarray (char *buf, char **array) {
    int i, count, len;
    len = strlen(buf);
    buf[len - 1] = '\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;
    int fdout, fderr;
    char line[BUFSIZ], buf[BUFSIZ], *args[BUFSIZ];
    time_t t1, t2;
    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);
        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) { /*Everything relevant is happening here.*/
        /*
        The child process will redirect the standard output stream (stdout)
        to a file <pid>.out and the standard error stream (stderr)
        to the file <pid>.err.
        Note that <pid> here corresponds to the process id of the child process.
        As a result of this change you will not see any output
        from the child process on the terminal,
        you have to look at the corresponding <pid>.out or <pid>.err file
```

```
        for the corresponding output and error messages.
        */
            pid_t pidc = getpid();
            char pid_out[100];
            sprintf(pid_out, "%d.out", pidc);
            char pid_err[100];
            sprintf(pid_err, "%d.err", pidc);
            if ((fdout = open(pid_out, O_CREAT | O_APPEND | O_WRONLY, 0755)) == -1) {
               printf("Error opening file %d.out for input\n", pidc);
               exit(-1);
            }
            if ((fderr = open(pid_err, O_CREAT | O_APPEND | O_WRONLY, 0755)) == -1) {
               printf("Error opening file %d.err for output\n", pidc);
               exit(-1);
            }
            dup2(fdout, STDOUT_FILENO);
            dup2(fderr, STDERR_FILENO);
            execvp(args[0], args);
            perror("exec");
            exit(-1);
        }
        else if (pid > 0) {
            printf ("Child started at %s", ctime (&t1));
            printf ("Wait for the child process to terminate\n");
            wait (&status);
            time (&t2);
            printf ("Child ended at %s", ctime (&t2));
            if (WIFEXITED (status)) {
                printf ("Child process exited with status = %d\n",
                WEXITSTATUS (status));
            }
            else {
                printf ("Child process did not terminate normally!\n");
            }
            buf[strlen (buf) - 1] = '\t';
            strcat (buf, ctime (&t1));
            buf[strlen (buf) - 1] = '\t';
            strcat (buf, ctime (&t2));
            fprintf (fp2, "%s", buf);
            fflush (fp2);

            close(fdout);
            close(fderr); /*John to self: You also did this one*/
        }
        else {
            perror ("fork");
            exit (EXIT_FAILURE);
        }
    }
    fclose (fp1);
    fclose (fp2);
    printf ("[%ld]: Exiting main program .....\n", (long) getpid ());
    return 0;
}
```