```c
/* Sample solution to Lab-7 assignment.
 * To Compile: gcc -Wall -o lab7 lab7_solution.c
 * To Run: ./lab7 commands.txt
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.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];
    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);

        time(&t1);
        pid = fork();
        if (pid == 0)
        {
            int redi = getpid();
            char node1[10], node2[10];
            sprintf(node1, "%d", redi);
            sprintf(node2, "%d", redi);
            strcat(node1, ".out");
            strcat(node2, ".err");

            int redirect_log = open(node1, O_CREAT | O_APPEND | O_WRONLY, 0777);
            int redirect_err = open(node2, O_CREAT | O_APPEND | O_WRONLY, 0777);
            if (redirect_log == -1)
            {
                printf("error");
                return 0;
            }

            dup2(redirect_log, 1);
            dup2(redirect_err, 2);
            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'; /* replace \n included by fgets with \t */
            strcat(buf, ctime(&t1));     /* append start time to command with arguments
 */
            buf[strlen(buf) - 1] = '\t'; /* replace \n added by ctime at the 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;
}
```