```
/* Driver program to test the queue implementation */
/* Use the Makefile provided or compile using: gcc queue.c HW4.c -lpthread */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include <pthread.h>
#include "queue.h"

#define MAXJOBLEN 1000
#define JOBQLEN 100

int max_job;
int curr_work_num_jobs;
job total_jobs[MAXJOBLEN];
queue *waiting_jobs;

void *curr_job_complete(void *arg);
void *all_jobs_complete(void *arg);
void queue_handler();


int main(int argc, char **argv)
{
    if ((argc != 2) || (atoi(argv[1])%2!=0))
    {
        printf("Usage: %s Number of (Even)Jobs to run at a single time\n", argv[0]);
        exit(-1);
    }

    char *err_file;
    pthread_t tid;

    max_job = atoi(argv[1]);
    if (max_job < 1)
        max_job = 1;
    else if (max_job > 8)
        max_job = 8;

    printf("Maximum Jobs to be run at any given time : %d\n\n", max_job);

    err_file = malloc(sizeof(char) * (strlen(argv[0]) + 5));
    sprintf(err_file, "%s.err", argv[0]);
    dup2(file_open(err_file), STDERR_FILENO);

    waiting_jobs = queue_init(JOBQLEN);
    pthread_create(&tid, NULL, all_jobs_complete, NULL);
    queue_handler();

    exit(-1);
}

void queue_handler()
{
    int i;
    char *line = NULL, *temp = NULL;
    char *argument, *command;
    const char s[2] = " ";
    size_t len = 0;
    temp = malloc(sizeof(char) * strlen(s));
    i = 0;

    printf("> ");
    while (getline(&line, &len, stdin) != -1)
    {
        strcpy(temp, line);
        if ((argument = strtok(temp, s)) != NULL)
```

```
        {
            if (strcmp(argument, "submit") == 0)
            {
                if (i >= MAXJOBLEN)
                    printf("Job history full; restart the program to schedule more\n");
                else if (waiting_jobs->count >= waiting_jobs->size)
                    printf("Job queue full; try again after more jobs complete\n");
                else
                {
                    command = get_command(line);
                    total_jobs[i] = create_job(command, i);
                    queue_insert(waiting_jobs, total_jobs + i);
                    printf("Job %d added to the queue\n", i++);
                }
            }

            else if (strcmp(argument, "showjobs\n") == 0)
                show_jobs(total_jobs, i);
            else if (strcmp(argument, "submithistory\n") == 0)
                submit_history(total_jobs, i);
        }
        printf("> ");
    }
    kill(0, SIGINT);
}


void *all_jobs_complete(void *arg)
{
    job *current_jobs;
    curr_work_num_jobs = 0;
    for (;;)
    {
        if (waiting_jobs->count > 0 && curr_work_num_jobs < max_job)
        {
            current_jobs = queue_delete(waiting_jobs);
            pthread_create(&current_jobs->tid, NULL, curr_job_complete, current_jobs);
            pthread_detach(current_jobs->tid);
        }
        sleep(2);
    }
    return NULL;
}

void *curr_job_complete(void *arg)
{
    job *current_jobs;
    char **args;
    pid_t pid;

    current_jobs = (job *)arg;
    ++curr_work_num_jobs;
    current_jobs->stat = "working";
    current_jobs->start = current_time();

    pid = fork();
    if (pid == 0) /* child process */
    {
        dup2(file_open(current_jobs->output_file), STDOUT_FILENO);
        dup2(file_open(current_jobs->err_file), STDERR_FILENO);
        args = get_args(current_jobs->command);
        execvp(args[0], args);
        fprintf(stderr, "Error: for Argument %s \n", args[0]);
        perror("execvp");
        exit(-1);
    }
    else if (pid > 0) /* parent process */
    {
        waitpid(pid, &current_jobs->exit_stat, WUNTRACED);
        current_jobs->stat = "complete";
```

```
        current_jobs->stop = current_time();

        if (!WIFEXITED(current_jobs->exit_stat))
            fprintf(stderr, "Child process %d did not terminate normally!\n", pid);
    }
    else
    {
        fprintf(stderr, "Error: process fork failed\n");
        perror("fork");
        exit(-1);
    }

    --curr_work_num_jobs;
    return NULL;
}
```