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

typedef struct Job
{
    int job_id;
    int exit_status;
    char *start_time;
    char *end_time;
    char *status;
    char error_msg[10];
    char out_msg[10];
    char *command;
    pthread_t thread_id;
} Job;

Job job_q[500];
int current_length;
int first_index;
int num_of_jobs;

int max_threads_allowed;
int active_jobs;

int get_output_pointer(char *fn)
{
    int fp = open(fn, O_APPEND | O_CREAT | O_WRONLY, 0755);

    if (fp != -1)
        return fp;

    fprintf(stderr, "%s \n", fn);
    exit(EXIT_FAILURE);
}

char * get_string(char *current_str){
    char *new_string = malloc(sizeof(char) * strlen(current_str));
    int i;
    for(i=0;current_str[i] != '\0'; i++){
        new_string[i] = current_str[i];
    }
    new_string[i] = '\0';
    return new_string;
}

int read_input(char *input, int max_characters)
{
    int input_letter, i=0;
    while (i < max_characters - 1)
    {
        input_letter = getchar();
        if (input_letter == '\n')
            break;
        if (input_letter == EOF)
            return -1;
        input[i++] = input_letter;
    }
    input[i] = '\0';
    return i;
}

char **get_command_args(char *command)
{
    char **command_args = malloc(sizeof(char *));
    int i=0, j=0, k=0;
    for (;;)
    {
        for(k=i;command[k] !=' ' && command[k] != '\0';k++){

        }
        command_args[j] = malloc(sizeof(char) * k + 1);
        strncpy(command_args[j], command+i, k-i);
        command_args = realloc(command_args, sizeof(char *) * (j + 2));
        j++;
        if (command[k] == '\0')
            break;
        i=k+1;
    }
    command_args[j] = NULL;
    return command_args;
}

char * remove_leading_spcl_char(char *command)
{
    int start;
    for (start=0;command[start]!='\0';start++){
        if(command[start] == ' ' || command[start] == '\x0b')
            continue;
        else
            break;
    }
    return command + start;
}

char *get_current_datetime()
{
    time_t current_time = time(NULL);
    return get_string(ctime(&current_time));
}


int create_job_and_add_to_queue(char *command){
        Job job;
    job.job_id = num_of_jobs;
    sprintf(job.out_msg, "%d_out.txt", job.job_id);
    sprintf(job.error_msg, "%d_err.txt", job.job_id);
    job.exit_status = -1;
    job.status = "Waiting";
    job.start_time = NULL;
    job.end_time = NULL;
    job.command = command;
    job_q[num_of_jobs++] = job;
    ++current_length;
    return current_length;
}

void show_jobs(){
    int i;
        if (num_of_jobs != 0)
    {
        printf("Job Id \t Command \t\t\t Status\n");
        for (i = 0; i < num_of_jobs; ++i)
        {
            if (strcmp(job_q[i].status, "Success") != 0 && strcmp(job_q[i].status, "Fai
led") != 0)
                printf("%d\t%s\t\t\t%s\n",job_q[i].job_id, job_q[i].command, job_q[i].s
tatus);
        }
    }
}

void show_history(){
    int i;
        if (num_of_jobs != 0)
    {
        printf("JobId \t Command \t\t\t StartTime \t EndTime \t Status\n");
            for (i = 0; i < num_of_jobs; ++i)
            {
                if (strcmp(job_q[i].status, "Success") == 0 || strcmp(job_q[i].status,
"Failed") == 0)
                    printf(" %d  %s %s %s %s\n",
                           job_q[i].job_id,
                           job_q[i].command,
                           job_q[i].start_time,
                           job_q[i].end_time,
                           job_q[i].status
                           );
            }
    }
}

Job *remove_from_queue()
{
    if (current_length == 0)
        return NULL;
    Job *job = (Job *)&job_q[first_index++];
    --current_length;
    return job;
}

void * process_job(void *job_to_process)
{
    int i;
    pid_t process_id;

    Job *job = (Job *)job_to_process;
    job->status = "Running";
    job->start_time = get_current_datetime();
    char **command_args = get_command_args(job->command);

    ++active_jobs;

    process_id = fork();
    if (process_id == 0)
    {
        dup2(get_output_pointer(job->out_msg), STDOUT_FILENO);
        dup2(get_output_pointer(job->error_msg), STDERR_FILENO);
        execvp(command_args[0], command_args);
    }
    else if (process_id > 0)
    {
        waitpid(process_id, &job->exit_status, WUNTRACED);
        job->status = "Success";
        job->end_time = get_current_datetime();
    }
    else
    {
        job->status = "Failed";
        job->end_time = get_current_datetime();
        fprintf(stderr, "Error: Fork failed\n");
    }
    for(i=0; command_args[i] != NULL; i++)
        free(command_args[i]);
    free(command_args);
    --active_jobs;
    return NULL;
}

void * run_jobs(void *arg)
{
    active_jobs = 0;
    Job *job;
    while(1)
    {
        if (active_jobs <= max_threads_allowed && current_length > 0)
        {
            job = remove_from_queue();
            if(job != NULL){
                    pthread_create(&job->thread_id, NULL, process_job, job);
                    pthread_detach(job->thread_id);
            }
        }
        sleep(1);
    }
    return NULL;
}


int main(int argc, char **argv)
{
    pthread_t tid;
    char *input_parsed;
    char *command;
    char input[500];

    if (argc != 2)
    {
        printf("Missing argument, Max number of threads\n");
        exit(EXIT_SUCCESS);
    }

    max_threads_allowed = atoi(argv[1]);
    printf("Max threads set to: %d\n\n", max_threads_allowed);

    current_length = 0;
    first_index = 0;
    num_of_jobs = 0;

    pthread_create(&tid, NULL, run_jobs, NULL);

    while (1)
    {
        printf("Enter command> ");
        int return_val = read_input(input, 500);
        if(return_val == -1)
```

```
            break;

        command = get_string(input);

        input_parsed = strtok(command, " \t\n\r");

        if (input_parsed != NULL)
        {
            if (strcmp(input_parsed, "submit") == 0)
            {
                if (num_of_jobs >= 500)
                    printf("Max jobs for queue reached.\n");
                else
                    create_job_and_add_to_queue(remove_leading_spcl_char(&input[7]));
            }
            else if (strcmp(input_parsed, "showjobs") == 0)
                show_jobs();
            else if(strcmp(input_parsed, "submithistory") == 0)
                show_history();
            else
                printf("Command not recognized!\n");
        }
    }

    exit(EXIT_SUCCESS);
}
```