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

char *StringCopy(char *s)
{
    int i;
    char c;
    char *cpy;

    i = 0;
    cpy = malloc(sizeof(char) * strlen(s));
    c = s[i];

    while (c != '\0')
    {
        cpy[i] = c;
        i++;
        c = s[i];
    }
    cpy[i] = '\0';

    return cpy;
}

struct Job
{
    int jobid;
    pthread_t threadid;
    char *cmd;
    char *jobstatus;
    int ejobstatus;
    char *begin;
    char *finish;
    char fnout[16];
    char fnerr[16];
};

struct Queue
{
    int length;
    struct Job **buffer;
    int begin;
    int finish;
    int count;
```

```
};

char *FullCopy(char *s)
{
    int i, c;
    char *cpy;

    i = 0;
    cpy = malloc(sizeof(char) * strlen(s));
    while ((c = s[i++]) != '\0' && c != '\n')
        cpy[i] = c;
    cpy[i] = '\0';

    return cpy;
}

char *Now()
{
    time_t tim = time(NULL);
    return FullCopy(ctime(&tim));
}

int PRESENT;

void *BeginCommand(void *arg)
{
    struct Job *jp;
    char **args;
    pid_t pid;

    jp = (struct Job *)arg;

    PRESENT++;
    jp->jobstatus = "working";
    jp->begin = Now();

    pid = fork();
    if (pid == 0)
    {
        int fd, fd2;
        if ((fd = open(jp->fnout, O_CREAT | O_APPEND | O_WRONLY, 0755)) == -1)
            exit(0);
        else if ((fd2 = open(jp->fnerr, O_CREAT | O_APPEND | O_WRONLY, 0755))
         == -1)
            exit(0);

        dup2(fd, STDOUT_FILENO);
        dup2(fd2, STDERR_FILENO);

        char *cpy = malloc(sizeof(char) * (strlen(jp->cmd) + 1));
        strcpy(cpy, jp->cmd);
```

```
        char *arg;
        char **args = malloc(sizeof(char *));
        int i = 0;
        while ((arg = strtok(cpy, " \t")) != NULL)
        {
            args[i] = malloc(sizeof(char) * (strlen(arg) + 1));
            strcpy(args[i], arg);
            args = realloc(args, sizeof(char *) * (i++ + 1));
            cpy = NULL;
        }
        args[i] = NULL;

        execvp(args[0], args);
        exit(0);
    }
    else if (pid > 0)
    {
        waitpid(pid, &jp->ejobstatus, WUNTRACED);
        jp->jobstatus = "complete";
        jp->finish = Now();
    }
    else
    {
        exit(0);
    }

    PRESENT--;
    return 0;
}

struct Queue *JOBQUEUE;
int QUEUE;

void *BeginAllCommands(void *arg)
{
    struct Job *jp;

    PRESENT = 0;
    do
    {
        if (JOBQUEUE->count > 0 && PRESENT < QUEUE)
        {
            if ((JOBQUEUE == NULL) || (JOBQUEUE->count == 0))
                jp = (struct Job *)NULL;

            else
            {
                struct Job *j = JOBQUEUE->buffer[JOBQUEUE->begin];
                JOBQUEUE->begin = (JOBQUEUE->begin + 1) % JOBQUEUE->length;
                JOBQUEUE->count--;
```

```
                jp = j;
            }

            pthread_create(&jp->threadid, NULL, BeginCommand, jp);

            pthread_detach(jp->threadid);
        }
        sleep(1);
    }
    while (1);
    return NULL;
}

struct Job ALLJOBS[512];

int GetLine(char *s, int n)
{
    int i;
    int c;
    for (i = 0; i < n - 1 && (c = getchar()) != '\n'; i++)
    {
        if (c == EOF)
            return -1;
        s[i] = c;
    }
    s[i] = '\0';
    return i;
}

char *DeleteSpace(char *s)
{
    int i;

    i = 0;
    while (s[i] == ' ')
        i++;

    return s + i;
}

int main(int argc, char *argv[])
{
    char *fnerr;
    pthread_t threadid;

    QUEUE = atoi(argv[1]);

    JOBQUEUE = malloc(sizeof(struct Queue));
    JOBQUEUE->length = 128;
    JOBQUEUE->buffer = malloc(sizeof(struct Job *) * 128);
    JOBQUEUE->begin = 0;
```

```
JOBQUEUE->finish = 0;
JOBQUEUE->count = 0;

pthread_create(&threadid, NULL, BeginAllCommands, NULL);

int i;
char line[512];
char *kw;
char *cmd;

i = 0;
while (printf("Enter Command> ") && GetLine(line, 512) != -1)
{
    if ((kw = strtok(StringCopy(line), " ")) != NULL)
    {
        if (strcmp(kw, "submit") == 0)
        {
            if (i >= 512)
                printf("Full job history; restart the program to add
                 more\n");
            else if (JOBQUEUE->count >= JOBQUEUE->length)
                printf("Job struct Queue full; retry when more jobs have
                 finished\n");
            else
            {
                cmd = DeleteSpace(strstr(line, "submit") + 6);

                struct Job j;
                j.jobid = i;
                j.cmd = StringCopy(cmd);
                j.jobstatus = "waiting";
                j.ejobstatus = -1;
                j.begin = j.finish = NULL;
                sprintf(j.fnout, "%d.out", j.jobid);
                sprintf(j.fnerr, "%d.err", j.jobid);
                ALLJOBS[i] = j;

                if ((JOBQUEUE != NULL) && (JOBQUEUE->count !=
                 JOBQUEUE->length))
                {
                    JOBQUEUE->buffer[JOBQUEUE->finish % JOBQUEUE->length]
                     = ALLJOBS + i;
                    JOBQUEUE->finish = (JOBQUEUE->finish + 1) %
                     JOBQUEUE->length;
                    JOBQUEUE->count++;
                }

                printf("Struct Job %d was added to the struct Job struct
                 Queue.\n", i++);
            }
        }
```

```
            else if (strcmp(kw, "showjobs") == 0 || strcmp(kw,
             "submithistory") == 0)
            {
                int ix;
                if (ALLJOBS != NULL && i != 0)
                {
                    if (strcmp(kw, "showjobs") == 0)
                    {
                        printf("jobid\tcommand\t\tstatusus\n");
                        for (ix = 0; ix < i; ix++)
                        {
                            if (strcmp(ALLJOBS[ix].jobstatus, "complete") != 0)
                                printf("%d\t%s\t%s\n",
                                    ALLJOBS[ix].jobid,
                                    ALLJOBS[ix].cmd,
                                    ALLJOBS[ix].jobstatus);
                        }
                    }
                    else if (strcmp(kw, "submithistory") == 0)
                    {
                        printf("Job
                         ID\tCommand\t\tstarttime\tendtime\tstatusus\n");
                        for (ix = 0; ix < i; ix++)
                        {
                            if (strcmp(ALLJOBS[ix].jobstatus, "complete") == 0)
                                printf("%d\t%s\t%s\t%s\t%s\n",
                                    ALLJOBS[ix].jobid,
                                    ALLJOBS[ix].cmd,
                                    ALLJOBS[ix].begin,
                                    ALLJOBS[ix].finish,
                                    "Success");
                        }
                    }
                }
            }
            else if (strcmp(kw, "exit") == 0)
                break;
        }
    }
    kill(0, SIGINT);

    return 0;
}
```