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

struct job;
struct queue;

struct job
{
    int jid;
    pthread_t tid;
    char *cmd;
    char *stat;
    char *start;
    char *stop;
    char fnout[15];
    char fnerr[15];
};

struct queue
{
    int size;
    struct job **buffer;
    int start;
    int end;
    int count;
};

char out[] = "out";
char err[] = "err";

int Insert(struct queue *, struct job *);
void JobsList(struct job *, int, char *);
struct job *FreeQueue(struct queue *);
char *CopyLine(char *);
void FreeEntireQueue(struct queue *);
struct job InitJob(char *, int);
struct queue *InitQueue(int);
char *DateTime();
int CountCharacters(char *, int);
char *RemoveSpace(char *);
char **ParseArguments(char *);
char *CreateCopy(char *);
void *ProcessAllJobs();
void *ProcessJob();
int CreateFile(char *);
void ReadInput();
int CheckSpace(char);

struct job InitJob(char *cmd, int jid)
{
    struct job j;
    j.jid = jid;
    j.cmd = CreateCopy(cmd);
    j.stat = "waiting";
    j.start = j.stop = NULL;
    sprintf(j.fnout, "%d.%s", j.jid, out);
    sprintf(j.fnerr, "%d.%s", j.jid, err);
    return j;
}

void list_status(struct job *jobs, int n)
{
    printf("jobid\tcommand\t\tstatus\n");
    for (int i = 0; i < n;)
    {
        if (strcmp(jobs[i].stat, "Success") != 0)
        {
            printf("%d\t %s\t %s\n", jobs[i].jid, jobs[i].cmd, jobs[i].stat);
        }
        i++;
    }
}

void list_all_status(struct job *jobs, int n)
{
    printf("Job ID\tCommand\t\tstarttime\tendtime\tstatus\n");
    for (int i = 0; i < n;)
    {
        if (!strcmp(jobs[i].stat, "Success"))
        {
            printf("%d\t %s\t %s\t %s\t %s\n", jobs[i].jid, jobs[i].cmd,
jobs[i].start, jobs[i].stop, "Success");
        }
        i++;
    }
}

void JobsList(struct job *jobs, int n, char *mode)
{
    int i;
    if (jobs != NULL && n != 0)
    {
        if (!strcmp(mode, "submithistory"))
        {
            list_all_status(jobs, n);
            return;
        }
        if (!strcmp(mode, "showjobs"))
        {
            list_status(jobs, n);
            return;
        }
    }
}
int Insert(struct queue *q, struct job *jp)
{
    if ((q == NULL) || (q->count == q->size))
        return -1;

    q->buffer[q->end % q->size] = jp;
    q->end = (q->end + 1) % q->size;
    return ++q->count;
}

struct queue *InitQueue(int n)
{
    struct queue *q = malloc(sizeof(struct queue));
    q->size = n;
    q->buffer = malloc(sizeof(struct job *) * n);
    q->start = 0;
    q->end = 0;
    q->count = 0;

    return q;
}

struct job *FreeQueue(struct queue *q)
{
    if ((q == NULL) || (q->count == 0))
        return NULL;

    struct job *j = q->buffer[q->start];
    q->start = (q->start + 1) % q->size;
    q->count--;

    return j;
}

void FreeEntireQueue(struct queue *q)
{
    for (int i = 0; i < q->count; i++)
    {
        free(q->buffer[i]);
    }
    free(q);
}

int CheckSpace(char c)
{
    return !(c != ' ' && c != '\t');
}

char *RemoveSpace(char *s)
{
    while (CheckSpace(*s))
    {
        s++;
    }
    return s;
}

int CountCharacters(char *s, int n)
{
    int c, i = 0;
    while (--n > 0 && (c = getchar()) != EOF && c != '\n')
    {
        s[i++] = c;
    }
    if (c == '\n')
```

```c
    {
        s[i++] = c;
    }
    s[i] = '\0';
    return i;
}

char *CreateCopy(char *s)
{
    char *copy = malloc(strlen(s) + 1);
    strcpy(copy, s);
    copy[strlen(s)] = '\0';
    return copy;
}

char *DateTime()
{
    time_t tim = time(NULL);
    return CopyLine(ctime(&tim));
}

char *CopyLine(char *s)
{
    char *copy = malloc(strlen(s) + 1);
    int i = 0;
    while (s[i] != '\n')
    {
        copy[i] = s[i];
        i++;
    }
    copy[i] = '\0';
    return copy;
}

int CreateFile(char *fn)
{
    int fd = open(fn, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd == -1)
    {
        fprintf(stderr, "Error: failed to open \"%s\"\n", fn);
        perror("open");
        return -1;
    }
    return fd;
}

char **ParseArguments(char *line)
{
    char *copy = malloc((strlen(line) + 1));
    strcpy(copy, line);

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

int ARGUMENT;
int CURRENT;
struct job RUNNING_JOBS[1000];
struct queue *CURRENT_JOBS;

int main(int argc, char **argv)
{
    char *fnerr;
    pthread_t tid;

    if (argc != 2)
    {
        printf("Usage: %s Queue size\n", argv[0]);
        return 0;
    }

    ARGUMENT = atoi(argv[1]);

    CURRENT_JOBS = InitQueue(1000);

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

    ReadInput();

    return 0;
}

void ReadInput()
{
    int i;
    char line[1000];
    char *kw;
    char *cmd;

    i = 0;
    while (printf("Enter Command> ") && CountCharacters(line, 1000) != -1)
    {
        if ((kw = strtok(CreateCopy(line), " \t\n")) != NULL)
        {
            if (strcmp(kw, "submit") == 0)
            {
                cmd = RemoveSpace(strstr(line, "submit") + 6);
                cmd[strlen(cmd) - 1] = '\0';
                RUNNING_JOBS[i] = InitJob(cmd, i);
                Insert(CURRENT_JOBS, RUNNING_JOBS + i);
                printf("Added struct job %d to the struct job struct queue\n", i+
+);
            }
            else if (strcmp(kw, "showjobs") == 0 || strcmp(kw, "submithistory") ==
0)
                JobsList(RUNNING_JOBS, i, kw);
            else if (strcmp(kw, "exit") == 0)
                exit(0);
        }
    }
    kill(0, SIGINT);
}

void *ProcessAllJobs(void *arg)
{
    struct job *jp;

    CURRENT = 0;
    for (int i = 0; i <= CURRENT;)
    {
        if (CURRENT_JOBS->count > 0 && CURRENT < ARGUMENT)
        {
            jp = FreeQueue(CURRENT_JOBS);

            pthread_create(&jp->tid, NULL, ProcessJob, jp);

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

void *ProcessJob(void *arg)
{
    struct job *jp;
    char **args;
    pid_t pid;

    jp = (struct job *)arg;

    jp->stat = "Working";
    jp->start = DateTime();

    ++CURRENT;
    pid = fork();
    if (pid == 0)
    {
        args = ParseArguments(jp->cmd);
        dup2(CreateFile(jp->fnout), STDOUT_FILENO);
        dup2(CreateFile(jp->fnerr), STDERR_FILENO);
        execvp(args[0], args);
        return 0;
    }
    else if (pid > 0)
    {
        int status;
        waitpid(pid, &status, WUNTRACED);
        jp->stop = DateTime();
        jp->stat = "Success";
    }

    --CURRENT;
    return 0;
}

//contribution
```

```
// skatkum
// edam
```