```c
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <fcntl.h>
#include <pthread.h>
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
typedef struct pthread_args
{
  pthread_t ptid;
    int place;
    char *command;
  char *args;
} JOBS;
/* create the queue data structure and initialize it */
queue *queue_init(int n) {
  queue *q = (queue *)malloc(sizeof(queue));
  q->size = n;
  q->buffer = malloc(sizeof(int)*n);
  q->start = 0;
  q->end = 0;
  q->count = 0;

  return q;
}

/* insert an item into the queue, update the pointers and count, and
   return the no. of items in the queue (-1 if queue is null or full) */
int queue_insert(queue *q, int item) {
  if ((q == NULL) || (q->count == q->size))
     return -1;

  q->buffer[q->end % q->size] = item;
  q->end = (q->end + 1) % q->size;
  q->count++;
  printf("job %d added to the queue\n",q->count);
  return q->count;
}

/* delete an item from the queue, update the pointers and count, and
   return the item deleted (-1 if queue is null or empty) */
int queue_delete(queue *q) {
  if ((q == NULL) || (q->count == 0))
     return -1;

  int x = q->buffer[q->start];
  q->start = (q->start + 1) % q->size;
  q->count--;

  return x;
}

/* display the contents of the queue data structure */
void queue_display(queue *q) {
  int i;
  if (q != NULL && q->count != 0) {
```

```
60      printf("queue has %d elements, start = %d, end = %d\n",
61        q->count, q->start, q->end);
62      printf("queue contents: ");
63      for (i = 0; i < q->count; i++)
64            printf("%d ", q->buffer[(q->start + i) % q->size]);
65      printf("\n");
66    } else{
67      printf("queue empty, nothing to display\n");
68    }
69  }
70
71  /* delete the queue data structure */
72  void queue_destroy(queue *q) {
73    free(q->buffer);
74    free(q);
75  }
76
77  void *run_job(void *arg){
78    char buf[BUFSIZ];
79    char bufTwo[BUFSIZ];
80    int fdout,fderr;
81    struct pthread_args *task;
82    task=(JOBS *)arg;
83    char* argList[]={task->command,task->args,NULL};
84    pid_t pid;
85    pid=fork();
86    if(pid==0){
87      snprintf(buf,sizeof(buf),"job_%d.out",task->place);
88      if((fdout=open(buf,O_CREAT|O_APPEND|O_WRONLY,0755,task->place))==-1){
89        printf("Error opening file %s for output\n",buf);
90      }
91      dup2(fdout,1);
92      dup2(fdout,2);
93      execvp(task->command,argList);
94      snprintf(bufTwo,sizeof(buf),"job_%d.err",task->place);
95      if ((fderr = open(buf, O_CREAT | O_APPEND | O_WRONLY, 0755,task->place)) == -1) {
96      printf("Error opening file %s for error\n",bufTwo);
97      exit(-1);
98      }
99      dup2(fderr,1);
100     perror("exec");
101   }
102
103   return(NULL);
104 }
105 int main(int argc ,char **argv){
106   int size=1000;
107   JOBS *job= (JOBS *)malloc(sizeof(JOBS)*size);
108   int i=0;
109   if(argc<2){
110     printf("Usage: %s <Max job count>\n",argv[0]);
111     exit(-1);
112   }else if(argc>2){
113     printf("Too many arguments! Usage: %s <Max job count>\n",argv[0]);
114   }
115     queue* q;
116   int max_jobs;
117   sscanf(argv[1],"%d",&max_jobs);
118   max_jobs=max_jobs+1;
119   q=queue_init(100);
```

```
  int loop=0;
  while(loop==0){
    printf("Enter command: ");
    char *commandline[1000];
    commandline[0]=(char *)malloc(1000*sizeof(char*));
    commandline[1]=(char *)malloc(1000*sizeof(char*));
    commandline[2]=(char *)malloc(1000*sizeof(char*));
    char *store=(char*) malloc(10000*sizeof(char*));
    fgets(store,10000,stdin);
    char* delim=" ";
    commandline[0]=strdup(strtok(store,delim));
    commandline[1]=strdup(strtok(NULL,delim));
    commandline[2]=strdup(strtok(NULL,delim));
    if(strcmp(commandline[0],"showjobs")==0){
      queue_display(q);
    }else{
      if(strcmp(commandline[0],"submit")==0){
        job[i].place=(q->count)+1;
        job[i].command=commandline[1];
        job[i].args=commandline[2];
        queue_insert(q,q->count);
        pthread_create(&job[i].ptid, NULL,run_job,(void *)&job[i]);
        i++;
      }
    }

  }
    return 0;
}
```