```
// Jessica Elkins
// Project 4 - CS332
// 4/21/20
// BlazerID: jelkins3
// This program is a job scheduler that executes non-interactive jobs, and takes the
// number of processes you want running at a time as a command line argument.

// HOW IT WORKS:
// - Submit a command by typing "submit <command>"
// - Display the jobs currently running or waiting to executed by typing "showjobs"
// - Type "quit" to exit the program
// - Once you exit the program, you will find the output and error files of the
//   submitted commands with their corresponding jobid: <jobid.out> and <jobid.err>

// TO COMPILE: gcc mysched.c -o mysched -lpthread
// TO RUN: ./mysched <# of cores>


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


typedef struct _queue {
        int size; //max size of the queue
        char **buffer; // queue buffer
        int start; // index to the start of the queue
        int end; // index to the end of the queue
        int count; // no. of elements in the queue
        char *status; // status of element
        int jobid; // job id for the element
} queue;

// global variables
int P, numOfJobs = 1, counter = 0;
queue *q, *q2, *q3;

/* create the queue data structure and initialize it */
queue *queue_init(int n){
        queue *q = (queue *)malloc(sizeof(queue));
        q->size = n;
        q->buffer = (char**)malloc(sizeof(char*)*n);
        q->start = 0;
        q->end = 0;
        q->count = 0;
        q->status = malloc(sizeof(char*)*10);
        return q;
}

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

        q->status = status;
        q->buffer[q->end % q->size] = (char *)malloc((sizeof item + 1)* sizeof(char));
```

```
        strcpy(q->buffer[q->end % q->size], item);
        q->end = (q->end + 1) % q->size;
        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) */
char* queue_delete(queue *q){
        if((q == NULL) || (q->count == 0))
                return "empty";

        char *x;
        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) {
                //printf("queue has %d elements, start = %d, end = %d\n", q->count, q->
start, q->end);
                //printf("queue contents: ");
                for(i = 0; i < q->count; i++){
                        printf("%s \t \t  %s ", q->buffer[(q->start + i) % q->size], q-
>status);
                        printf("\n");
                }
        }else{
                //printf("queue empty, nothing to display\n");
        }
}

/* delete the queue data structure */
void queue_destroy(queue *q){
        free(q->buffer);
        free(q);
}

void createarray(char *buf, char **array){
        int i, count, len;
        len = strlen(buf);

        for(i = 1, array[0] = &buf[1], count = 1; i < len; i++){
                if(buf[i] == ' '){
                        buf[i] = '\0';
                        array[count++] = &buf[i+1];
                }
        }
        array[count] = (char *)NULL;
}

void *submit(){
        pid_t pid, apid;
        int nprocs = 0, status; // counter = 0;
        char *line, buf[BUFSIZ], *args[BUFSIZ], *q3input;
        int number;
```

```
        // deleting element from the queue and storing it in line
        line = queue_delete(q);
        q3input = queue_delete(q2);
        number = q3input[0] - 48;
        queue_insert(q3, q3input, "Running");

        // create execvp input
        createarray(line, args);

        pid = fork();
        if(pid == 0) {
                int fdout, fderr;
                char outFileName[BUFSIZ], errFileName[BUFSIZ];

                // making file names
                sprintf(outFileName, "%d.out", number);
                sprintf(errFileName, "%d.err", number);

                // open file to write stdout to
                if((fdout = open(outFileName, O_CREAT | O_APPEND | O_WRONLY, 0755)) ==
-1){
                        printf("Error opening file %s for output\n", outFileName);
                        exit(-1);
                }

                // open file to write stderr to
                if((fderr = open(errFileName, O_CREAT | O_APPEND | O_WRONLY, 0755)) ==
-1){
                        printf("Error opening file %s for output\n", errFileName);
                        exit(-1);
                }

                dup2(fdout, 1);
                dup2(fderr, 2);

                execvp(args[0], args);
                perror("exec");
                exit(-1);
        } else if (pid > 0) { //this is the parent
                //printf("Child process %ld started \n", (long)pid);
                counter++;
        } else { // we have an error
                perror("fork");
                exit(EXIT_FAILURE);
        }

        do {
                apid = waitpid(pid, &status, 0);

                if (WIFEXITED(status)) {
                        //printf("child process %ld exited, status = %d\n", (long)apid,
 WEXITSTATUS(status));
                } else if (WIFSIGNALED(status)) {
                        //printf("child process %ld killed by signal %d\n", (long)apid,
 WTERMSIG(status));
                }
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        counter--;
        //printf("Child process %ld ended\n", (long)apid);

        queue_delete(q3);
}
```

```
void *compute(void *args){

        // waits for more input from user
        for( ; ;){
                if(counter < P && q->count > 0 && q != NULL){
                        pthread_t tid;
                        // create thread
                        pthread_create(&tid, NULL, submit, NULL);
                        sleep(2);

                }
        }
}

// display the jobs that are running and waiting
void showjobs(char *args){
        printf("jobid \t  command \t \t  status \n");
        queue_display(q3);
        queue_display(q2);
}

int main(int argc, char **argv){
        char line[BUFSIZ];
        char input[BUFSIZ], command[BUFSIZ];
        char *args, q2input[BUFSIZ];
        const char token = ' ';
        int length, i, jobid = 1;

        if(argc != 2){
                printf("Usage: %s [# of jobs] \n", argv[0]);
                exit(-1);
        }

        P = atoi(argv[1]);
        //printf("P value is: %d \n", P);

        //creating the queue
        q = queue_init(10);

        q2 = queue_init(10);
        q3 = queue_init(10);

        // starting the thread that will execute jobs
        pthread_t tid;
        pthread_create(&tid, NULL, compute, NULL);

        while((strcmp("quit", input)) != 0){
                printf("Enter command> ");

                // getting command entered
                fgets(input, BUFSIZ, stdin);
                length = strlen(input);

                // taking new line character off
                if(input[length-1] == '\n')
                        input[length-1] = '\0';

                // tokenizing the string
                args = strchr(input, token);

                i = 0;
                while(input[i] != ' '){
                        command[i] = input[i];
```

```
                        i++;
                }
                command[i] = '\0';

                if((strcmp("quit", input)) != 0){

                        if((strcmp("submit", command)) == 0){
                                printf("Job %d added to queue.\n", jobid);
                                // insert arg to queue
                                queue_insert(q, args, "Waiting");
                                sprintf(q2input, "%d \t %s", jobid, args);
                                queue_insert(q2, q2input, "Waiting");
                                jobid++;
                        }else if((strcmp("showjobs", command)) == 0){
                                showjobs(args);
                        }else if((strcmp("quit", command)) == 0){
                                break;
                        }else{
                                printf("Invalid command...\n");
                        }

                }

        }

        return 0;

}
```