```c
/*
Name: Hung-Yi Lu
BlazerId: lu0106
Project #:Homework 4
*/

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

// Mutex Lock
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

// Condition Variable
pthread_cond_t cv = PTHREAD_COND_INITIALIZER;

// job queue
queue* jobq;

// for loop
int k;

typedef struct {
    int jobid;  // job ID
    int status; // 0 is not use
    // success or failed
    int s_f; // successful is 0
    char* com;
    char* combp;
    // time start and end
    time_t s;
    time_t e;
}thread_struct;

thread_struct arr[4096];

int start(int id, char args[]){
    int status = 0; // 0 is not use
    int option = 0;
    pid_t pid;

    // child process
    if((pid = fork()) == 0){
        char* argument_list[] = {"sh", "-c", args, NULL};
        execvp(argument_list[0], argument_list);
        exit(0); // exit
    }

    // error
    else if (pid == -1){
        perror("Error: Fork");
        return -1;
    }

    // Parent process
    else {
        int wait = waitpid(pid, &status, option);
        if(wait != pid)
            perror("Error: Wait");
    }
   return status;
}

static void * cput(void * arg){
    int t;
    char args[2048];

    // Infinite
    for(;;){

        // thread, grab the lock, and release lock
        pthread_mutex_lock(&lock);     // Thread Mutex lock

        while((t = queue_delete(jobq)) == -1){
            pthread_cond_wait(&cv, &lock);
        }

        pthread_mutex_unlock(&lock);    // Mutex unlock

        sprintf(args, "%s >%d.out 2>%d.err", arr[t].com + 7, t, t);
        // running
        arr[t].status = 1;
        time(&arr[t].s); // time start
        arr[t].s_f = start(t, args);
        // time end
        time(&arr[t].e);
        // finished
        arr[t].status = 3;
    }
    return NULL;
}

void n_thread(int num){
    pthread_t thread_id[num];
    int t[num];
    int r = 0;

    r = pthread_mutex_init(&lock, NULL);
    if (r != 0)
        perror("Thread Error: Mutex Init");
    r = pthread_cond_init(&cv, NULL);
    if (r != 0)
        perror("Thread Error: Cond Init");
    jobq = queue_init(4096);

    for(k = 0; k < num; k++){
        t[k] = k;
        r = pthread_create(&thread_id[k], NULL, cput, &t[k]);
        if(r != 0)
            perror("Thread Create Error");
    }
}

int main(int argc, char **argv){
    int num;
    int c = 1;
    size_t bufflen;
    char* fgets_buff = (char *)malloc(1024);
    if(argc != 2){
        printf("%s \n", argv[0]);
        return -1;
    }

    num = atoi(argv[1]);
    if(num <= 1){
        perror("Error: Number should > 0");
        return -1;
    }
    n_thread(num);

    // Infinite
    for(;;){
        printf("Enter Command (submithistory, showjobs, submit, clear) -->");
        fgets(fgets_buff, 1024, stdin);
        bufflen = strlen(fgets_buff);
        if(bufflen <= 1)
            continue;

        bufflen = bufflen - 1;
        fgets_buff[bufflen] = '\0';

        // Show jobs case
        if (strncmp(fgets_buff, "showjobs", 8) == 0 || strncmp(fgets_buff, "Showjobs", 
8) == 0 || strncmp(fgets_buff, "SHOWJOBS", 8) == 0){
            printf("JobID   %-50s   Status\n", "Command");

            for(k=1; k < c; k++){
                // waiting
                if(arr[k].status == 2)
                    printf("%d   %s   WAITING\n", k, arr[k].combp);
                // running
                else if(arr[k].status == 1)
                    printf("%d   %s   RUNNING\n", k, arr[k].combp);
            }
            printf("===============================================================
=============\n");
        }

        // Submit History case
        else if (strncmp(fgets_buff, "submithistory", 13) == 0 || strncmp(fgets_buff, "
Submithistory", 13) == 0 || strncmp(fgets_buff, "SUBMITHISTORY", 13) == 0){
            printf("%s    %s            %s     %s     %s\n", "Id", "Command", "Start ti
me", "End time", "Status");

            for(k = 1;k < c; k++){
                thread_struct* ts = &arr[k];

                // finished
                if((ts->status) == 3){
                    char* s = ctime(&ts->s); // start time
                    char* e = ctime(&ts->e); // end time

                    if(s[strlen(s)-1] == '\n')
                        s[strlen(s)-1] = '\0';
                    if(e[strlen(e)-1] == '\n')
                        e[strlen(e)-1] = '\0';

                    if(ts->s_f)  // print FAIL case
                        printf("%d    %s            %s     %s     %s\n", k, (ts->combp)
, s, e, "FAIL");
                    else  // print SUCCESS case
```

```
                        printf("%d    %s            %s     %s     %s\n", k, (ts->combp)
, s, e, "SUCCESS");

                }
            }
            printf("=========================================================================
=============\n");
        }

        // Submit
        else if(strncmp(fgets_buff, "submit", 6) == 0 || strncmp(fgets_buff, "Submit",
6) == 0 || strncmp(fgets_buff, "SUBMIT", 6) == 0){
            arr[c].jobid = c;
            //waiting
            arr[c].status = 2;
            arr[c].combp = strdup(fgets_buff + 7);
            arr[c].com = strdup(fgets_buff);

            // thread, grab the lock, and release lock
            pthread_mutex_lock(&lock); // Mutex lock
            queue_insert(jobq, c);
            pthread_mutex_unlock(&lock); // Mutex unlock
            pthread_cond_signal(&cv);
            printf("job %d added to the queue\n", c); // success
            c++; // count
        }

        // clear screen
        else if(strncmp(fgets_buff, "clear", 5) == 0 || strncmp(fgets_buff, "Clear", 5)
 == 0 || strncmp(fgets_buff, "CLEAR", 5) == 0){
            system("clear");
        }
        // command error
        else
            printf("Error: Use 'submithistory', 'showjobs', 'submit', 'clear' Command\n
");
    }
    free(fgets_buff);
}
```