```
/*
Assignment Four
Frank Wu & Hsuan Jou Lee
email: yihong@uab.edu   hsuanjou@uab.edu
Compile:gcc -lpthread -Wall -o hw4 hw4.c queue.h
Excute: ./hw4 10 (maximum amount of jobs)

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <fcntl.h>
#include "queue.h"
/*necessary headers implemented which include the queue header */

void *complete_job(void *arg);
void *complete_jobs(void *arg);
void handle_input();
/*Define jobs that are completed
https://github.com/eolatham/job-scheduler/blob/master/job_scheduler.c
*/

int CONCUR ;        /* Maximum amount of jobs executed at the same time */
int CURRENT;      /* amount of jobs working now */
job JOBS[1000]; /* array of submitted jobs */
queue *q_job;       /* pointer of wating jobs */

int main(int argc, char **argv)
{
    char *fnerr;    /*filename where main stderr is redirected */
    pthread_t tid; // thread ID

    if (argc != 2)
    {
        printf("Usage: %s CONCURRENCY\n", argv[0]);
        exit(EXIT_SUCCESS);
    }

    CONCUR = atoi(argv[1]);
    if (CONCUR < 1)
        CONCUR = 1;
    else if (CONCUR > 8)
        CONCUR = 8;

    printf("Concurrency: %d\n\n", CONCUR);

    /* redirect main stderr to file */
    fnerr = malloc(sizeof(char) * (strlen(argv[0]) + 5));
    sprintf(fnerr, "%s.err", argv[0]);
    dup2(open_log(fnerr), STDERR_FILENO);

    q_job = queue_init(100);

    /* use a new thread to complete jobs */
    pthread_create(&tid, NULL, complete_jobs, NULL);

    /* use the main thread to handle input */
```

```
        handle_input();

        exit(EXIT_SUCCESS);
    }
    /*
    The "Main" portion are reference from
    https://github.com/eolatham/job-scheduler/blob/master/job_scheduler.c
    To check the concurrency does not go over the maximum and is not lower than one.
    */
    void handle_input()
    {
        int i;              /* job counter */
        char line[1000]; /* input line buffer */
        char *kw;           /* input keyword */
        char *cmd;          /* input submit job command */


        i = 0;
        while (printf("> ") && get_line(line, 1000) != -1)
        {
            if ((kw = strtok(get_copy(line), " \t\n\r\x0b\x0c")) != NULL)
            {
                if (strcmp(kw, "submit") == 0)
                {
                    if (i >= 1000)
                        printf("Space full!\n");
                    else if (q_job->count >= q_job->size)
                        printf("Job queue full!\n");
                    else
                    {
                        cmd = left_strip(strstr(line, "submit") + 6);
                        JOBS[i] = create_job(cmd, i);
                        queue_insert(q_job, JOBS + i);
                        printf(" %d jobs added to the queue\n", i++);
                    }
                }
                else if (strcmp(kw, "showjobs") == 0 ||
                        strcmp(kw, "submithistory") == 0)
                    queue_display(JOBS, i, kw);
            }
        }
        kill(0, SIGINT); /* kill the current process group upon reaching EOF */
    }
    //Fix the input to the format we want.


    void *complete_jobs(void *arg)
    {
        job *jp; /* job pointer */

        CURRENT = 0;
        for (;;)
        {
            if (q_job->count > 0 && CURRENT < CONCUR)
            {
                /* pull next job from queue */
                jp = queue_delete(q_job);

                /* use a new thread to complete job */
                pthread_create(&jp->tid, NULL, complete_job, jp);

                /* mark new thread detached to free its resources when done */
                pthread_detach(jp->tid);
```

```
            }
            sleep(1); /* wait a second, then check again */
        }
        return NULL;
    }
    /*
    The above portion are referenced from
    https://github.com/eolatham/job-scheduler/blob/master/job_scheduler.c
    Keep checking for and completed jobs.
    Sleep for one second and check again

    */

    /* Complete the job pointed to by arg using fork/exec. */
    void *complete_job(void *arg)
    {
        job *jp;     /* job pointer from arg */
        char **args; /* array of args to be parsed from job command */
        pid_t pid;   /* process ID */

        jp = (job *)arg;

        ++CURRENT;
        jp->stat = "working";
        jp->start = current_datetime_str();

        pid = fork();
        if (pid == 0) /* child process */
        {
            dup2(open_log(jp->fnout), STDOUT_FILENO); /* redirect job stdout */
            dup2(open_log(jp->fnerr), STDERR_FILENO); /* redirect job stderr */
            args = get_args(jp->cmd);
            execvp(args[0], args);
            fprintf(stderr, "Error: command execution failed for \"%s\"\n", args[0]);
            perror("execvp");
            exit(EXIT_FAILURE);
        }
        else if (pid > 0) /* parent process */
        {
            waitpid(pid, &jp->estat, WUNTRACED);
            jp->stat = "complete";
            jp->stop = current_datetime_str();

            if (!WIFEXITED(jp->estat))
                fprintf(stderr, "Child process %d did not terminate normally!\n", pid);
        }
        else
        {
            fprintf(stderr, "Error: process fork failed\n");
            perror("fork");
            exit(EXIT_FAILURE);
        }

        --CURRENT;
        return NULL;
    }



    /*
    Job creation and fill in the features, status, and start-stop time.
    */
    job create_job(char *cmd, int jid)
```

```
    {
        job j;
        j.jid = jid;
        j.cmd = get_copy(cmd);
        j.stat = "waiting";
        j.estat = -1;
        j.start = j.stop = NULL;
        sprintf(j.fnout, "%d.out", j.jid);
        sprintf(j.fnerr, "%d.err", j.jid);
        return j;
    }

    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("%d ", q->buffer[(q->start + i) % q->size]);
            printf("\n");
        } else
            printf("queue empty, nothing to display\n");
    }

    /*
    This is the course material provided by professor.
    If mode is "showjobs", list all non-completed jobs from jobs,
    displaying each job's ID, command, and status.
    If mode is "submithistory", list all completed jobs from jobs,
    displaying each job's job ID, thread ID, command,
    exit status, start time, and end time.
    */


    /*
    This is the course material provided by professor
    Create the queue data structure and initialize it.
    Adapted from Professor Bangalore's queue example.
    */
    queue *queue_init(int n)
    {
        queue *q = malloc(sizeof(queue));
        q->size = n;
        q->buffer = malloc(sizeof(job *) * n);
        q->start = 0;
        q->end = 0;
        q->count = 0;

        return q;
    }

    /*
    This is the course material provided by professor
    Insert the job pointer jp into the queue q, update the pointers and count,
    and return the number of items in the queue (-1 if q is null or full).
    Adapted from Professor Bangalore's queue example.
    */
    int queue_insert(queue *q, 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;
        ++q->count;

        return q->count;
    }

    /*
    This is the course material provided by professor
    Delete a job pointer from the queue, update the pointers and count,
    and return the job pointer deleted (-1 if q is null or empty).
    Adapted from Professor Bangalore's queue example.
    */
    job *queue_delete(queue *q)
    {
        if ((q == NULL) || (q->count == 0))
            return (job *)-1;

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

        return j;
    }

    /*
    This is the course material provided by professor
    Delete the queue data structure.
    Adapted from Professor Bangalore's queue example.
    */
    void queue_destroy(queue *q)
    {
        free(q->buffer);
        free(q);
    }

    /*
    Below are portions referenced from
    https://github.com/eolatham/job-scheduler/blob/master/job_scheduler.c
    Record input character-by-charactermm.;
    */
    int get_line(char *s, int n)
    {
        int i, c;
        for (i = 0; i < n - 1 && (c = getchar()) != '\n'; ++i)
        {
            if (c == EOF)
                return -1;
            s[i] = c;
        }
        s[i] = '\0';
        return i;
    }

    //Return one if it is a whitespace. Otherwise 0
    int is_space(char c)
    {
        if (c == ' ' || c == '\t' || c == '\r' || c == '\x0b' || c == '\x0c')
        {
            return 1;
        }
        else
        {
```

```
            return 0;
        }
    }
    /*Reference from https://github.com/eolatham/job-scheduler/blob/master/job_schedule
r.c
    Return a pointer to the first non-whitespace character in s. */
    char *left_strip(char *s)
    {
        int i;

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

        return s + i;
    }

    //Return a null-terminated copy of the null-terminated string s
    char *get_copy(char *s)
    {
        int i, c;
        char *copy;

        i = -1;
        copy = malloc(sizeof(char) * strlen(s));
        while ((c = s[++i]) != '\0')
            copy[i] = c;
        copy[i] = '\0';

        return copy;
    }

    /*
    Return a null-terminated copy of the null-terminated
    string s up until the first newline or the end of s.
    */
    char *get_copy_until_newline(char *s)
    {
        int i, c;
        char *copy;

        i = -1;
        copy = malloc(sizeof(char) * strlen(s));
        while ((c = s[++i]) != '\0' && c != '\n')
            copy[i] = c;
        copy[i] = '\0';

        return copy;
    }

    /*
    Return a copy of the result of ctime() with the current
    timestamp from time(). Returning a copy is important
    because subseqent calls to ctime() mutate strings
    previously made by ctime().
    */
    char *current_datetime_str()
    {
        time_t tim = time(NULL);
        return get_copy_until_newline(ctime(&tim));
    }

    /*
```

```
    https://github.com/eolatham/job-scheduler/blob/master/job_scheduler.c
    Parse space/tab separated tokens in order from the given string
    into a null-terminated array of strings and return it.
    */
    char **get_args(char *line)
    {
        char *copy = malloc(sizeof(char) * (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(sizeof(char) * (strlen(arg) + 1));
            strcpy(args[i], arg);
            args = realloc(args, sizeof(char *) * (++i + 1));
            copy = NULL;
        }
        args[i] = NULL;
        return args;
    }
    /* https://github.com/eolatham/job-scheduler/blob/master/job_scheduler.c
    Open a log file with the given filename and return its file descriptor. */
    int open_log(char *fn)
    {
        int fd;
        if ((fd = open(fn, O_CREAT | O_APPEND | O_WRONLY, 0755)) == -1)
        {
            fprintf(stderr, "Error: failed to open \"%s\"\n", fn);
            perror("open");
            exit(EXIT_FAILURE);
        }
        return fd;
    }
```