```
/*
I, SHREYAS SRINIVASA, declare that I have completed this assignment completely and entirely
 on my own, without any unauthorized consultation from others or unauthorized access to onl
ine websites. I have read the UAB Academic Honor Code and understand that any breach of the
 UAB Academic Honor Code may result in severe penalties.

Student Signature/Initials: SS

Date: 04/14/2023

To compile: gcc -Wall -lpthread -o job_scheduler job_scheduler.c utils.c queue.c
To run: ./job_scheduler P
*/

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

#define INPUT_LIMIT 1000      /* maximum input line length */
#define MAX_JOBS 1000         /* maximum number of jobs that are allowed to be executed */
#define MAX_JOB_QUEUE_SIZE 50 /* maximum job queue size */

int P;              /* global max thread count */
int W;              /* global number of currently WORKING jobs */
job JOBS[MAX_JOBS]; /* global array of all submitted jobs */
queue *JOB_QUEUE;   /* global job queue */

int main(int argc, char **argv)
{
    char *fnerr; /* error log file */

    pthread_t tid;

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

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

    printf("The value of P chosen is %d\n", P);

    fnerr = malloc(sizeof(char) * (strlen(argv[0]) + 5));
    sprintf(fnerr, "%s.err", argv[0]);
    dup2(open_file(fnerr), STDERR_FILENO);

    JOB_QUEUE = queue_init(MAX_JOB_QUEUE_SIZE);

    /*manager thread */
    pthread_create(&tid, NULL, job_manager, NULL);

    /* main thread */
    run();

    exit(EXIT_SUCCESS);
}
```

```
/* Handle submit command*/
void submit(int *counter, char *input)
{
    int i = *counter;
    char *command;
    /*Restricts job creation beyond maj job limit*/
    if (i >= MAX_JOBS)
        printf("Job history full; restart the program to schedule more\n");
    /*Checks for queue overflow*/
    else if (JOB_QUEUE->count >= JOB_QUEUE->size)
        printf("Job queue full; try again after more jobs complete\n");
    /*Removes the scheduler keyword from the input line, creates a new job struct and inser
ts it into the queue*/
    else
    {
        command = left_strip(strstr(input, "submit") + 6);
        JOBS[i] = create_job(command, i);
        queue_insert(JOB_QUEUE, JOBS + i);
        printf("Added job %d to the job queue\n", ++i);
    }
    *counter = i;
}

/*Handles showjobs scheduler command*/
void show_jobs(job *jobs, int n)
{
    int i;
    if (jobs != NULL && n != 0)
    {
        printf("%-5s %-40s %-10s\n", "jobid", "command", "status");
        for (i = 0; i < n; ++i)
        {
            /*Prints all jobs with status not equal to COMPLETE*/
            if (strcmp(jobs[i].status, "COMPLETE") != 0)
                printf("%-5d %-40s %-10s", jobs[i].jid + 1, jobs[i].command, jobs[i].status
);
        }
        printf("\n");
    }
}

/*Handles submithistory scheduler command*/
void submit_history(job *jobs, int n)
{
    int i;
    if (jobs != NULL && n != 0)
    {
        printf("%-5s %-40s %-30s %-30s %-10s\n", "jobid", "command", "starttime", "endtime"
, "status");
        for (i = 0; i < n; ++i)
        {
            /*Prints all jobs with status equal to COMPLETE*/
            if (strcmp(jobs[i].status, "COMPLETE") == 0)
                printf("%-5d %-40s %-30s %-30s %-10s", jobs[i].jid + 1, jobs[i].command, jo
bs[i].start_time, jobs[i].stop_time, jobs[i].status);
        }
        printf("\n");
    }
}

/*
User input processing
*/
void run()
{
    int i;                    /* job counter */
    char input[INPUT_LIMIT]; /* input buffer */
    char *keyword;            /* scheduler command keyword */
```

```
    printf("submit COMMAND [ARGS] : Schedule a job\n"
           "showjobs : List all jobs are are in WAITING or WORKING status\n"
           "submithistory : List all jobs are are in COMPLETED status\n\n");

    i = 0;
    while (printf("> ") && read_line(input, INPUT_LIMIT) != -1)
    {
        if ((keyword = strtok(duplicate(input), " \t\n\r\x0b\x0c")) != NULL)
        {
            if (strcmp(keyword, "submit") == 0)
            {
                submit(&i, input);
            }
            else if (strcmp(keyword, "showjobs") == 0)
                show_jobs(JOBS, i);
            else if (
                strcmp(keyword, "submithistory") == 0)
                submit_history(JOBS, i);
            else
            {
                printf("Unsupported command, please try again\n");
            }
        }
    }
    kill(0, SIGINT); /* kill the current process group upon reaching EOF */
}

/*Thread routine for managing all the jobs*/
void *job_manager(void *args)
{
    job *jp;

    W = 0;
    /*Non-terminating loop which checks for queued jobs and the thread limit and creates a
worker thread for each new job each second*/
    while (1)
    {
        if (JOB_QUEUE->count > 0 && W < P)
        {

            jp = queue_delete(JOB_QUEUE);
            pthread_create(&jp->tid, NULL, job_runner, jp);
            pthread_detach(jp->tid);
        }
        sleep(1);
    }
    return NULL;
}
/*Worker thread routine which uses fork-exec pattern to complete the assigned job*/
void *job_runner(void *args)
{
    job *jp;          /* job pointer from arg */
    char **exec_args; /* array of args to be parsed from job command */
    pid_t pid;        /* process ID */

    jp = (job *)args;

    W++;
    jp->status = "WORKING";
    jp->start_time = current_datetime_str();

    pid = fork();
    if (pid == 0) /* child process */
    {
        dup2(open_file(jp->fout), STDOUT_FILENO); /* redirect job stdout */
        dup2(open_file(jp->ferr), STDERR_FILENO); /* redirect job stderr */
        exec_args = create_exec_args(jp->command);
```

```
        execvp(exec_args[0], exec_args);
        fprintf(stderr, "Error: command execution failed for \"%s\"\n", exec_args[0]);
        perror("execvp");
        exit(EXIT_FAILURE);
    }
    else if (pid > 0) /* parent process */
    {
        waitpid(pid, &jp->exit_status, WUNTRACED);
        jp->status = "COMPLETE";
        jp->stop_time = current_datetime_str();

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

    W--;
    return NULL;
}

/*Creates a new job struct by setting the appropriate initial values*/
job create_job(char *command, int jid)
{
    job j;
    j.jid = jid;
    j.command = duplicate(command);
    j.status = "WAITING";
    j.exit_status = -1;
    j.start_time = j.stop_time = NULL;
    sprintf(j.fout, "%d.out", j.jid + 1);
    sprintf(j.ferr, "%d.err", j.jid + 1);
    return j;
}
```