```
/*
                                                        CS532 SYSTEMS PROGRAMING

                                                    Hari Srujan Goppu; Sai Vathsal Jammula
                                                          hgoppu     ;      sjammula

                                                              HOME WORK#4

complie: gcc HW4.c -lpthread

execute: ./a.out noofjobs
         ./a.out 2

Enter commands :> submit ./hw3 1000
job1 added to the queue
Enter commands :>  submit ./hw2 1000
job2 added to the queue
Enter commands :>  submit ./hw1 1000
job3 added to the queue
Enter commands :> showjobs
jobid    cmds                    status
2       ./hw1 1000                      running
Enter commands :> showjobs
jobid    cmds                    status
Enter commands :> submithistory
jobid    cmds                    starttime       endtime         status
 0  ./hw3 1000 Sat Nov 19 15:49:07 2022 Sat Nov 19 15:49:07 2022 success
 1  ./hw2 1000 Sat Nov 19 15:49:14 2022 Sat Nov 19 15:49:22 2022 success
 2  ./hw1 1000 Sat Nov 19 15:49:18 2022 Sat Nov 19 15:49:26 2022 success
Enter commands :> exit

*/

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



typedef struct struct_jobvalues
{
    int jid;
    pthread_t tid;
    char *begintime;
    char *endtime;
    char jobout[10];
    char joberr[10];
    char *cmds;
    char *jobstat;
    int flagexit;
} struct_jobvalues;
typedef struct StQueue
{
    int quesize;
    struct_jobvalues **buffer;
    int Qnumber;
    int Qstart;
```

```
    int Qend;

} StQueue;
StQueue *object_jobqueue;
int P;
struct_jobvalues data1[1500];
int workstat;

char *getdata(char *str);
struct_jobvalues newjob(char *cmds, int jid);
void jobdatadetails(struct_jobvalues *jobsList, int val, char *new_com);
int acesscoomond(char *lineInfo, int datanumberb);
int spacechecking(char val);
char *pointerreter(char *str);
StQueue *createQueue(int num);
int addQueue(StQueue *dataqueue, struct_jobvalues *jval);
struct_jobvalues *removeQueue(StQueue *dataqueue);
char *acesslinesinstring(char *data2);
char *currentdatetime();
char **acessthevalues(char *data);
int logcreater(char *fn);
void jobProcess();
void *sucessjob(void *arg);
void *historyjobs(void *arg);



int main(int argc, char **argv)
{
    pthread_t tid;
    char *errorjob;
    if (argc != 2)
    {
        exit(EXIT_SUCCESS);
    }
    P = atoi(argv[1]);
    errorjob = malloc(sizeof(char) * (strlen(argv[0]) + 5));
    sprintf(errorjob, "%s.err", argv[0]);
    dup2(logcreater(errorjob), STDERR_FILENO);
    object_jobqueue = createQueue(P);
    pthread_create(&tid, NULL, historyjobs, NULL);
    jobProcess();
    exit(EXIT_SUCCESS);
}

char *getdata(char *str)
{
    int i=-1;
    int chars;
    char *temp;
    temp = malloc(sizeof(char) * strlen(str));
    while (chars = str[++i]){
        if(chars !='\0'){
            temp[i] = chars;
        }
    }
    temp[i] = '\0';
    return temp;
}
struct_jobvalues newjob(char *cmds, int jid)
{
    struct_jobvalues job;
    job.jid = jid;
```

```
    job.cmds = getdata(cmds);
    job.flagexit = -1;
    job.begintime = NULL;
    job.endtime = NULL;
    job.jobstat = "waiting";
    sprintf(job.jobout, "%d.out", job.jid);
    sprintf(job.joberr, "%d.err", job.jid);
    return job;
}

void jobdatadetails(struct_jobvalues *jobsList, int val, char *new_com)
{
    int i;
    if (jobsList != NULL && val != 0)
    {
        if (strcmp(new_com, "showjobs") == 0)
        {
            printf("jobid \t cmds \t\t\t status\n");
            for (i = 0; i < val; ++i)
            {
                if (strcmp(jobsList[i].jobstat, "success") != 0)
                    printf("%d\t%s\t\t\t%s\n",jobsList[i].jid,
                           jobsList[i].cmds,
                           jobsList[i].jobstat);
            }
        }
        else if (strcmp(new_com, "submithistory") == 0)
        {
            printf("jobid \t cmds \t\t\t starttime \t endtime \t status\n");
            for (i = 0; i < val; ++i)
            {
                if (strcmp(jobsList[i].jobstat, "success") == 0)
                    printf(" %d  %s %s %s %s\n",
                           jobsList[i].jid,
                           jobsList[i].cmds,
                           jobsList[i].begintime,
                           jobsList[i].endtime,
                           jobsList[i].jobstat
                           );
            }
        }
    }
}

int acesscoomond(char *lineInfo, int datanumberb)
{
    int i, str;
    for (i = 0; i < datanumberb - 1 && (str = getchar()) != '\n'; ++i)
    {
        if (str == EOF){
            return -1;
        }
        lineInfo[i] = str;
    }
    lineInfo[i] = '\0';
    return i;
}

int spacechecking(char val)
{
    return (val == ' ' ||val == '\r' ||val == '\t' ||val == '\n' ||val == '\x0b' );
}
```

```
char *pointerreter(char *str)
{
    int i=0;

    while (spacechecking(str[i])){
        ++i;
    }
    return str + i;
}
StQueue *createQueue(int num)
{
    StQueue *dataqueue = malloc(sizeof(StQueue));
    dataqueue->quesize = num;
    dataqueue->Qnumber = 0;
    dataqueue->buffer = malloc(sizeof(struct_jobvalues *) * num);
    dataqueue->Qstart = 0;
    dataqueue->Qend = 0;
    return dataqueue;
}


int addQueue(StQueue *dataqueue, struct_jobvalues *jval)
{
    if ((dataqueue == NULL) || (dataqueue->Qnumber == dataqueue->quesize)){
        return -1;
    }
    dataqueue->buffer[dataqueue->Qend % dataqueue->quesize] = jval;
    dataqueue->Qend = (dataqueue->Qend + 1) % dataqueue->quesize;
    ++dataqueue->Qnumber;
    return dataqueue->Qnumber;
}

struct_jobvalues *removeQueue(StQueue *dataqueue)
{
    if ((dataqueue == NULL) || (dataqueue->Qnumber == 0)){
        return (struct_jobvalues *)-1;
    }
    struct_jobvalues *jval = dataqueue->buffer[dataqueue->Qstart];
    dataqueue->Qstart = (dataqueue->Qstart + 1) % dataqueue->quesize;
    --dataqueue->Qnumber;
    return jval;
}
char *acesslinesinstring(char *data2)
{
    int i=-1, val;
    char *temp;
    temp = malloc(sizeof(char) * strlen(data2));
    while ((val = data2[++i]) != '\0' && val != '\n'){
        temp[i] = val;
    }
    temp[i] = '\0';
    return temp;
}
char *currentdatetime()
{
    time_t curtime = time(NULL);
    return acesslinesinstring(ctime(&curtime));
}

char **acessthevalues(char *data)
{
    char *newdata = malloc(sizeof(char) * (strlen(data) + 1));
    char **dataArg = malloc(sizeof(char *));
```

```
    char *value;
    int i = 0;
    strcpy(newdata, data);
    while ((value = strtok(newdata, " \t")) != NULL)
    {
        dataArg[i] = malloc(sizeof(char) * (strlen(value) + 1));
        strcpy(dataArg[i], value);
        dataArg = realloc(dataArg, sizeof(char *) * (++i + 1));
        newdata = NULL;
    }
    dataArg[i] = NULL;
    return dataArg;
}

int logcreater(char *fn)
{
    int flagf;
    if ((flagf = open(fn, O_CREAT | O_APPEND | O_WRONLY, 0755)) == -1)
    {
        fprintf(stderr, "Error: unable to open \"%s\"\n", fn);
        perror("open");
        exit(EXIT_FAILURE);
    }
    return flagf;
}

void jobProcess()
{
    char *inputText;
    char *cmds;
    int i=0;
    char data[1500];

    while (printf("Enter commands :> ") && acesscoomond(data, 1500) != -1)
    {
        if ((inputText = strtok(getdata(data), " \t\n\r\x0b")) != NULL)
        {
            if (strcmp(inputText, "submit") == 0)
            {
                if (i >= 1500){
                    printf("Queue is full\n");
                }
                else if (object_jobqueue->Qnumber >= object_jobqueue->quesize){
                    printf("Add jobs after some time\n");
                }
                else
                {
                    cmds = pointerreter(strstr(data, "submit") + 6);
                    data1[i] = newjob(cmds, i);
                    addQueue(object_jobqueue, data1 + i);
                    printf("job%d added to the queue\n", ++i);
                }
            }
            else if (strcmp(inputText, "showjobs") == 0 ||strcmp(inputText, "submithistory") =
= 0){
                jobdatadetails(data1, i, inputText);
            }
            else if (strcmp(inputText, "exit") == 0) {

            exit(0);
            }
        }
    }
```

```
    kill(0, SIGINT);
}

void *sucessjob(void *arg)
{
    pid_t pid;
    char **val;
    ++workstat;
    struct_jobvalues *jobVal;
    jobVal = (struct_jobvalues *)arg;
    jobVal->jobstat = "running";
    jobVal->begintime = currentdatetime();

    pid = fork();
    if (pid == 0)
    {
        dup2(logcreater(jobVal->jobout), STDOUT_FILENO);
        dup2(logcreater(jobVal->joberr), STDERR_FILENO);
        execvp(acessthevalues(jobVal->cmds)[0], acessthevalues(jobVal->cmds));
        fprintf(stderr, "error:  failed for \"%s\"\n", val[0]);
        perror("execvp");
        exit(EXIT_FAILURE);
    }
    else if (pid > 0)
    {
        waitpid(pid, &jobVal->flagexit, WUNTRACED);
        jobVal->jobstat = "success";
        jobVal->endtime = currentdatetime();
        if (!WIFEXITED(jobVal->flagexit)){
            fprintf(stderr, "Child process %d did not terminate normally!\n", pid);
        }
    }
    else
    {
        fprintf(stderr, "error: fork failed\n");
        perror("fork");
        exit(EXIT_FAILURE);
    }
    --workstat;
    return NULL;
}

void *historyjobs(void *arg)
{
    workstat = 0;
    struct_jobvalues *jobVal;
    for (;;)
    {
        if (object_jobqueue->Qnumber > 0 && workstat < P)
        {
            jobVal = removeQueue(object_jobqueue);
            pthread_create(&jobVal->tid, NULL, sucessjob, jobVal);
            pthread_detach(jobVal->tid);
        }
        sleep(1);
    }
    return NULL;
}
```