```
/*
Name: John Robertson
BlazerId: jprob
Project #: 4
To compile: gcc -o p4 p4.c
To run: ./p4 <int between 1 and nproc inclusive>
*/

// https://stackoverflow.com/a/19724773/9295513
// https://linuxprograms.wordpress.com/2008/01/23/piping-in-threads/

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

#include "queue.h"

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

int P, counter=0, p[2];
queue* all_processes;
job_t* current_job;

void showjobs() {
    //read(p[0], &all_processes, sizeof(all_processes));
    puts("jobid\tcommand\t\tstatus");
    queue_display(all_processes);
}

void* submit(void* arg) {

    current_job = (job_t*)arg;
    queue_insert(all_processes, current_job);

    pid_t pid[BUFSIZ], apid;
    int nprocs = 0, status;
    char buf[BUFSIZ], *args[BUFSIZ];
    strcpy(buf, current_job->name);
    createarray(current_job->name, args);
    for ( ; ; ) {

        //write(p[1], &all_processes, sizeof(all_processes));
        current_job->number = counter;

        if (counter < P) {
            current_job->wait_or_run = 1; // RUNNING
            pid[nprocs++] = fork();
            if (pid[nprocs-1] == 0) {
                int fdout, fderr;
                char outFileName[BUFSIZ], errFileName[BUFSIZ];
                sprintf(outFileName, "%d.out", getpid());
                sprintf(errFileName, "%d.err", getpid());
                if ((fdout = open(outFileName, O_CREAT | O_APPEND | O_WRONLY, 0755)) ==
 -1) {
                    printf("Error opening file %s for output\n", outFileName);
                    exit(-1);
                }
                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);
                job_t temp = queue_delete(all_processes);
                while(strcmp(current_job->name, temp.name) != 0) {
                    queue_insert(all_processes, &temp);
                    temp = queue_delete(all_processes);
                }
            } else if (pid[nprocs-1] > 0) {
                counter++;
            } else {
                perror("fork");
                exit(EXIT_FAILURE);
            } break;
        }
        else {
            current_job->wait_or_run = 0; // WAITING
            do {
                apid = waitpid(-1, &status, 0);
            } while (!WIFEXITED(status) && !WIFSIGNALED(status));
            counter--;
        }
    }

    do {
        apid = waitpid(-1, &status, 0);
    } while (apid != -1);

    return (NULL);

}

int main(int argc, char **argv) {
        if (argc != 2) { printf("Usage: %s <# of processes>\n", argv[0]); exit(-1); }
        P = atoi(argv[1]);
        all_processes = queue_init(BUFSIZ);
        int running = 1;

        pthread_t ptid;

        while(running) {
                char* line = NULL;
                size_t maxlen = 0;
                printf("Enter command> ");
                getline(&line, &maxlen, stdin);
                if (' ' == line[0]) {
                    puts("Invalid command");
                }
                else if (strncmp("quit", line, 4) == 0) {
                        puts("Exiting program...bye!");
                        running = 0;
                        //pthread_join(ptid, NULL);
                }
                else if (strncmp("showjobs", line, 8) == 0) {
                    showjobs();
                        //close(p[0]);
                }
                else if (strncmp("submit", line, 6) == 0) {
            job_t new_job;
            strtok(line, " ");
            new_job.name = strtok(NULL, "\n");
            new_job.number = -1;
            new_job.wait_or_run = 1;



            //pipe(p);
            pthread_create(&ptid, NULL, submit, (void*)&new_job);
            //close(p[1]);



                }
                else {
                        puts("Invalid command");
                }
        }
        return 0;
}
```