```c
/*
 * Group Number: Group D
 * Group member name: Haidar Musaqlab
 * Email: haidar.musaqlab@okstate.edu
 */

// header inclusion
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <strings.h>
#include <string.h>
#include <pthread.h>
#include "Query.h"
#include "Record.h"

#define ID_NAME_FILE "IdName.txt"
#define SALARIES_FILE "Salaries.txt"
#define SATISFACTION_LEVEL_FILE "SatisfactionLevel.txt"
#define PORT 8000
#define LENGTH 1024

int getId(Query query) {
    // function to get id from employee name
    char buffer[LENGTH];
    int id = -1;
    FILE *fp;
    int matches[32];
    int count;
    char temp[256];
    char *token;
    char *rest = buffer;
    const char delim[2] = "\t";

    fp = fopen(ID_NAME_FILE, "r"); // open file for read
    if(fp == NULL) { // File couldn't be opened
        return id;
    }

    while(fgets(buffer, LENGTH, fp)) {
        if(strstr(buffer, query.employee_name)) { // found match
            sscanf(buffer, "%d\t", &id);
            matches[count] = id;
            count++;
        }
    }
    fclose(fp);

    if (count <= 1)
        return id;
    else{
        fp = fopen(SALARIES_FILE, "r");

        // Look for possible matches in salaries file
        while(fgets(buffer, LENGTH, fp)) {
            token = strtok_r(buffer, delim, &rest);
            id = (int)atoi(token);
            token = strtok_r(NULL, delim, &rest);
            strcpy(temp, token);

            // Checking possible matches
            for (int i = 0 ; i < count; i++){
                if (id == matches[i] && strcmp(query.job_title, temp) == 0){
                    fclose(fp);
                    return id;
                }
            }
        }
    }

    printf("no match found\n");
    fclose(fp);
    return -1; // get the id
}

void *SatisfactionLevelSearch(void *arg) {
    // thread function to get satisaaction level details
    Record *record = (Record *)arg;
    char buffer[LENGTH];
    const char delim[2] = "\t";
    char *token;
    char *rest = buffer;
    int i = 0;
    FILE *fp;

    fp = fopen(SATISFACTION_LEVEL_FILE, "r");
    if(fp == NULL) { // File couldn't be opened
        record->id = -1;
    }
    else { // file opened
        while(fgets(buffer, LENGTH, fp)) { // get the data parsed with strtok
            token = strtok_r(buffer, delim, &rest);
            if(atoi(token) == record->id) {
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->satisfaction_level = (float)atof(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->number_project = atoi(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->average_monthly_hours = atoi(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->time_spend_company_in_yrs = atoi(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->work_accident = atoi(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->promotion_last_5years = atoi(token);
                i = 1;
            }

        }
        if(i == 0)
            record->id = -1; // not found

    }
    fclose(fp);
    pthread_exit(NULL); // exit thread
}

void *SalariesSearch(void *arg) {
    // thread function to get salaries details
    Record *record = (Record *)arg;
    char buffer[LENGTH];
    const char delim[2] = "\t";
    char *token;
    char *rest = buffer;
    int i = 0;
    FILE *fp;

    fp = fopen(SALARIES_FILE, "r");
    if(fp == NULL) { // File couldn't be opened
```

```c
        record->id = -1;
    }
    else { // file opened
        while(fgets(buffer, LENGTH, fp)) {
            token = strtok_r(buffer, delim, &rest);
            if(atoi(token) == record->id) { // match found
                token = strtok_r(NULL, delim, &rest); // split with delim
                strcpy(record->job_title, token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->base_pay = (float)atof(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->overtime_pay = (float)atof(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                record->benefit = (float)atof(token);
                token = strtok_r(NULL, delim, &rest); // split with delim
                strcpy(record->status, token);
                record->status[2] = '\0';
                i = 1;
            }
        }
        if(i == 0)
            record->id = -1; // no match
    }
    fclose(fp);
    pthread_exit(NULL); // exit thread
}

int main(int argc, char **argv) {
    // main function server
    Record record;
    Query query;
    pthread_t tid1, tid2;
    char buffer[LENGTH];
    int connFd, fd, len;
    int id;

    struct sockaddr_in servaddr, cli;

    fd = socket(AF_INET, SOCK_STREAM, 0); // create socket
    if(fd == -1) { // server creation failed
        printf(">> Error creating socket. Try again!!\n");
        exit(-1);
    }

    // socket detials endpoint
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET; // protocol
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY); // any ip
    servaddr.sin_port = htons(PORT); //port num

    // bind socket name
    if((bind(fd, (struct sockaddr *)&servaddr, sizeof(servaddr))) != 0) {
        printf(">> Bind call failed\n"); // bind call failed
        exit(-1);
    }

    // listen for connections
    if((listen(fd, 3)) != 0) {
        printf(">> Listen call failed\n"); // listen call failed
        exit(-1);
    }

    int b1, b2, b3;
    len = sizeof(cli);
    connFd = accept(fd, (struct sockaddr *)&cli, &len); // accept conn
    if(connFd < 0) { // connect call failed
```

```c
        printf(">> Accept call failed\n");
        exit(-1);
    }
    while(1) {

        // communicate with Assistant here
        if(read(connFd, &query, sizeof(Query)) > 1){

            // End if needed
            if (strcmp(query.employee_name, "exit") == 0 && strcmp(query.job_title, "exit") == 0 && strcmp(query.status, "ex") ==0)
                break;


            // Clear existing values
            bzero(&record, sizeof(Record));

            id = getId(query); // get the id from employee name
            record.id = id;
            strcpy(record.employee_name, query.employee_name);

            // create 2 threads
            pthread_create(&tid1, NULL, SatisfactionLevelSearch, (void *)&record);
            pthread_create(&tid2, NULL, SalariesSearch, (void *)&record);
            pthread_join(tid1, NULL); // join thread 1
            pthread_join(tid2, NULL); // join thread 2

            printf("done.\n");
            printf("%d %s %s %.2f %.2f %.2f %s %.2f %d %d %d %d %d\n",
                    record.id, record.employee_name,

                    record.job_title,
                    record.base_pay, // not working
                    record.overtime_pay, // not working
                    record.benefit, // not working
                    record.status, // not working

                    record.satisfaction_level, record.number_project,
                    record.average_monthly_hours, record.time_spend_company_in_yrs, record.work_accident,
                    record.promotion_last_5years);

            // write back to client assistant
            send(connFd, &record, sizeof(Record), 0); // changed from write to send.. mroyster
            //close(connFd); // close client conn
        }
    }
    close(fd); // close socket fd
    return 0;
}
```