```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <stdbool.h>

// GROUP F
// AYRTON LEDESMA
// ayrton.ledesma_fuentes@okstate.edu

//Prototype functions
void Server(char *employeeName, char *jobTitle, char *status);
void* salarySearcher(void* received_struct);
void* satisfactionSearcher(void* received_struct);
void Assistant(char *employeeName, char *jobTitle, char *status);
void* historyFunction(void* received_struct);

struct Person
{
   int id;
   char *name[500];
   float satisfactionLevel;
   int projects;
   int avg_hours;
   int timeCompany;
   int workAccidents;
   int promotion;
   char *jobTitle[55];
   double basePay;
```

```
    double overTime;
    double benifit;
    char *status[10];
};

struct SalaryHolder{
    int id;
    char *jobtitle[55];
    double basePay;
    double overTimePay;
    double benefit;
    char *status[10];
};

struct SatisfactionHolder{
    int id;
    float satisfaction_Level;
    int number_project;
    int average_monthly_hours;
    int time_spend_company_in_yrs;
    int work_accident;
    int promotion_last_5years;
};

void* salarySearcher(void* received_struct)
{
    struct SalaryHolder *requestedEmployee = (struct SalaryHolder*) received_struct;
    int requestedID;
    requestedID = requestedEmployee->id;
```

```
   char stateInfo[1000];
   char line[1000];
   FILE *fptr;

   int id;
   char *jobTitle[500];
   float basePay;
   float overTime;
   float benefit;
   char *status[10];

   int i = 0;

   if ((fptr = fopen("Salaries.txt", "r")) == NULL) {
      printf("Error! Could not open Salaries.txt");
      exit(1);
   }

   //Skipping first line
   fgets(stateInfo, 100, fptr);
   while(fscanf(fptr, "%d\t%[^\n^\t]%*c\t%f\t%f\t%f\t%s\n", &id, jobTitle, &basePay, &overTime,
&benefit, status) != NULL)
   {
      if(id == requestedID){ //fails at 81393, 84507, 84961
         //printf("\nINFORMATION FOUND BY SALARY THREAD \n\nID: \t\t\t%d\nJOBTITLE:
\t\t%s\nBASEPAY: \t\t%f\nOVERTIME: \t\t%f\nBENEFIT: \t\t%f\nSTATUS: \t\t%s\n", id, jobTitle,
basePay, overTime, benefit, status);
         requestedEmployee->basePay = basePay;
         requestedEmployee->overTimePay = overTime;
```

```
            requestedEmployee->benefit = benefit;
            *requestedEmployee->status = *status;
            *requestedEmployee->jobtitle = *jobTitle;
        }
    }
    fclose(fptr);
    return (void*)requestedEmployee;
}

void* satisfactionSearcher(void* received_struct)
{
    struct SatisfactionHolder *requestedEmployee = (struct SatisfactionHolder*) received_struct;

    int requestedID;
    requestedID = requestedEmployee->id;

    char stateInfo[1000];
    char line[1000];
    FILE *fptr;

    int idLocal;
    float satisfaction_Level_Local;
    int number_project_Local;
    int average_monthly_hours_Local;
    int time_spend_company_in_yrs_Local;
    int work_accident_Local;
    int promotion_last_5years_Local;

    if ((fptr = fopen("SatisfactionLevel.txt", "r")) == NULL) {
```

```
    printf("Error! Could not open SatisfactionLevel.txt");
    exit(1);
  }

  //Skipping first line
  fgets(stateInfo, 1000, fptr);

  //Scanning the entire document and storing data into variables
  while(fscanf(fptr, "%d\t%f\t%d\t%d\t%d\t%d\t%d\n", &idLocal, &satisfaction_Level_Local,
&number_project_Local, &average_monthly_hours_Local, &time_spend_company_in_yrs_Local,
&work_accident_Local, &promotion_last_5years_Local) != EOF)
  {
    //when the current id equals the requested ID, put all information into structure
    if(idLocal == requestedID){
      //printf("\nINFORMATION FOUND BY SATISFACTION LEVEL THREAD \n\nID:\t\t\t%d
\nSATISFACTION LEVEL:\t%f \nNUMBER PROJECT:\t\t%d \nAVERAGE MTLY HOURS:\t%d\nTIME
SPEND:\t\t%d \nWORK ACCIDENT:\t\t%d \nPROMOTION:\t\t%d\n", idLocal,
satisfaction_Level_Local, number_project_Local, average_monthly_hours_Local,
time_spend_company_in_yrs_Local, work_accident_Local, promotion_last_5years_Local);
      requestedEmployee->id = idLocal;
      requestedEmployee->average_monthly_hours = average_monthly_hours_Local;
      requestedEmployee->number_project = number_project_Local;
      requestedEmployee->satisfaction_Level = satisfaction_Level_Local;
      requestedEmployee->work_accident = work_accident_Local;
      requestedEmployee->promotion_last_5years = promotion_last_5years_Local;
      requestedEmployee->time_spend_company_in_yrs = time_spend_company_in_yrs_Local;
    }
  }
  fclose(fptr);
  //return structure with all the information obtained from the file
  return (void*)requestedEmployee;
```

```
}


//Function: Reads ID_NAME.txt until it finds the id of the name requested. If search fails, it returns
0
int searchForEmployeeID(char *employeeName)
{
   char stateInfo[1000];
   FILE *fptr;
   char *test = employeeName;
   char *str = test;
   int i = 0;
   int ID;
   bool arr[2] = { true, false };
   bool foundEmployee = false;

   if ((fptr = fopen("ID_Name.txt", "r")) == NULL) {
      printf("Error! Could not open ID_Name.txt");
      exit(1);
   }

   while(fgets(stateInfo, 1000, fptr) != NULL){
      if((strstr(stateInfo, str)) != NULL) {
         foundEmployee = true;
                              char *line = stateInfo, *p = line;
         while (*p) {
            if (isdigit(*p)) {
               ID = strtol(p, &p, 10);
            }
            else {
```

```
            p++;
          }
        }
                }
    i+= 1;
  }
  fclose(fptr);

  if(foundEmployee == false || ID > 140000){
    printf("\nCould not find requested name. Please enter a valid name!\n");
    return 0;
  }
  else {
    return ID;
  }
}

//Server function, receives employee, jobTitle and status information of employee
void Server(char *employeeName, char *jobTitle, char *status){
  struct Person *finalInformation;

  bool arr[2] = { true, false };

  bool currentlySearching = true;
  bool employeeHasBeenFound = false;

  //printf("\n\n ---------- SERVER STARTS ---------- \n\n");
  //printf("Name:    %s\n", employeeName);
  //printf("Job Title %s\n", jobTitle);
```

```
  //printf("Status:   %s\n\n", status);

  int idNum;

  //Look for employee id in the first txt document. Returns the ID number, 0 if it cant find it
  idNum = searchForEmployeeID(employeeName);
  if(idNum > 0){
    employeeHasBeenFound = true;
    printf("\nServer found employee ID: %d\n\nLooking for employee...\n", idNum);
  }

  if(employeeHasBeenFound){
    //Threads declaration
    pthread_t ptid_Salaries;
    pthread_t ptid_Satisfaction;

    struct SatisfactionHolder currentEmployeeSatisfaction;
    struct SalaryHolder currentEmployeeSalary;

    //Passing data to the structure that will be used by the threads
    currentEmployeeSatisfaction.id = idNum;
    currentEmployeeSalary.id = idNum;
    *currentEmployeeSalary.jobtitle = &jobTitle;
    *currentEmployeeSalary.status = &status;

    //Create threads - Start at their respective function and are passed the address of their
respective structure declared in lines 213 and 214
    pthread_create(&ptid_Satisfaction, NULL, &satisfactionSearcher,
&currentEmployeeSatisfaction);
    pthread_create(&ptid_Salaries, NULL, &salarySearcher, &currentEmployeeSalary);
```

```
//Initializing place where information obtained from threads
void* salary_thread_result;
void* satisfaction_thread_result;

//Waiting for threads to terminate
pthread_join(ptid_Satisfaction, &satisfaction_thread_result);
pthread_join(ptid_Salaries, &salary_thread_result);

//printf("\nThreads completed...\n");
//printf("\nEXAMPLE INFORMATION RECEIVED FROM THREADS\n");
struct SalaryHolder* reqEmployee_Salary = salary_thread_result;
struct SatisfactionHolder* reqEmployee_Satisfaction = satisfaction_thread_result;

struct Person allInfo;
//Saving the 13 values of each requested employee as a structure
allInfo.id = reqEmployee_Salary->id;
allInfo.avg_hours = reqEmployee_Satisfaction->average_monthly_hours;
allInfo.basePay = reqEmployee_Salary->basePay;
allInfo.benifit = reqEmployee_Salary->benefit;
*allInfo.jobTitle = jobTitle;
*allInfo.jobTitle = reqEmployee_Salary->jobtitle;
*allInfo.name = employeeName;
allInfo.overTime = reqEmployee_Salary->overTimePay;
allInfo.projects = reqEmployee_Satisfaction->number_project;
allInfo.promotion = reqEmployee_Satisfaction->promotion_last_5years;
allInfo.satisfactionLevel = reqEmployee_Satisfaction->satisfaction_Level;
*allInfo.status = reqEmployee_Salary->status;
allInfo.timeCompany = reqEmployee_Satisfaction->time_spend_company_in_yrs;
```

```
        allInfo.workAccidents = reqEmployee_Satisfaction->work_accident;

        //Printing information found
        printf("\n      EMPLOYEE'S INFORMATION\n");
        printf("\nID: \t\t\t%d", allInfo.id);
        printf("\nNAME: \t\t\t%s", *allInfo.name);
        printf("\nJOB TITLE: \t\t%s", *allInfo.jobTitle);
        printf("\nBASE PAY: \t\t%f", allInfo.basePay);
        printf("\nOVERTIME PAY: \t\t%f", allInfo.overTime);
        printf("\nBENEFIT: \t\t%f", allInfo.benifit);
        printf("\nSTATUS: \t\t%s", *allInfo.status);
        printf("\nSATISFACTION: \t\t%f", allInfo.satisfactionLevel);
        printf("\nPROJECTS: \t\t%d", allInfo.projects);
        printf("\nAVG HOURS: \t\t%d", allInfo.avg_hours);
        printf("\nTIME SPENT: \t\t%d", allInfo.timeCompany);
        printf("\nWORK ACCIDENT: \t\t%d", allInfo.workAccidents);
        printf("\nPROMOTION: \t\t%d\n", allInfo.promotion);
    }
    //Completed
    currentlySearching = false;
    employeeHasBeenFound = false;
}

//Making connection with Manager - Passing user's input
void Assistant(char *employeeName, char *jobTitle, char *status){
    bool arr[2] = { true, false };
    bool isInHistory = false;

    struct Person personInHistory;
```

```
    initialize(personInHistory);

    *personInHistory.name = &employeeName;
    *personInHistory.jobTitle = &jobTitle;
    *personInHistory.status = &status;

    pthread_t ptid_History;
    printf("\nCHECKING ON HISTORY FIRST \n");
    pthread_create(&ptid_History, NULL, &historyFunction, &personInHistory);

    void* history_thread_result;
    //Waiting for thread to terminate
    pthread_join(ptid_History, &history_thread_result);
    struct Person* reqEmployee_History = history_thread_result;

    //If employee was found on history file
    if(reqEmployee_History->id != -1){
        printf("Searching was successful. The data was found on the history files....\n");
        printStruct(reqEmployee_History);
    }

    else {
        //Employee is not on the history file
        printf("\nUser was not found on history. It will look on the server\n");
        Server(employeeName, jobTitle, status);
    }
}
```