#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>

#include <time.h>

typedef struct information

{

char user[50];

char process;

int arrival;

int duration;

int deadLine;

int total;

int lengthOnCPu;

int inUse;

} information;

information \*init(char \* userInput, char processInput, int arrivalInput, int durationInput, int deadLine);

void motherboard(information \*list, int ep, int cpuNum);

information \* edf(information \*list, int time);

int freeList(information \* list);

void printOutput(information \* list);

int changeLoc = -1;

int main(){

char \*trashCollector = NULL;

// this a pointer to the headers which would be taking into account but not used.

information \*processList;

int cpuNum = 0;

if((processList = malloc(4\*sizeof(information)))==NULL){

return 1;

}

printf("Please specify num of CPU:");

scanf("%d", &cpuNum);

printf("\n");

int d;

for(d=0; d<5; d++){

scanf("%s\t", trashCollector);

}

int ep = 0;

int i;

for (i = 0; i < 4; i++){

char expectedName[50] = "/0";

char expectedProcess = 'a';

int expectedArrival = 0;

int expectedDuration = 0;

int expectedDeadline;

if(scanf("%s\t%c\t%d\t%d\t%d",expectedName, &expectedProcess, &expectedArrival, &expectedDuration, &expectedDeadline)<5 && ep<0){

printf("Error in scanning the data");

return 1;

}

// Create an initialize function to help store the iincoming data in the structs

information \*newStorage;

newStorage = init(expectedName, expectedProcess, expectedArrival, expectedDuration, expectedDeadline);

processList[i] = \*newStorage;

ep++;

}

motherboard(processList, ep, cpuNum);;

freeList(processList);

return 0;

}

information \*init(char \* userInput, char processInput, int arrivalInput, int durationInput, int deadLineInput){

information \* current;

if((current = malloc(sizeof(information))) == NULL)

{

return NULL;

}

strcpy(current->user, userInput);

current->process = processInput;

current->arrival = arrivalInput;

current->duration = durationInput;

current->deadLine = deadLineInput;

return current;

}

void motherboard(information \*list, int ep, int cpuNum){

int lightsOff[ep];

int checklist[ep];

int changed[ep];

int trialTime = 0;

int count = 0;

int safteycount = 0;

int f;

int cpuCheck = 0;

int conAge = 3 \* cpuNum;

for(f=0; f<ep; f++){

checklist[f] = 1;

lightsOff[f] = 0;

}

printf("This Would Result in:\nTime");

int y;

for(y=0;y<cpuNum;y++){

printf("\tCPU%d",y+1);

}

printf("\n");

while(count !=ep && safteycount != 100){

count = 0;

cpuCheck = 0;

printf("%d\t",trialTime);

int mn;

for(mn = 0; mn<ep; mn++){

changed[mn] = 0;

if(list[mn].arrival == trialTime){

list[mn].lengthOnCPu = 1;

}if(list[mn].lengthOnCPu == conAge){

list[mn].lengthOnCPu = 1;

}

}

while(cpuCheck != cpuNum){

changeLoc = -1;

information \* temp;

temp = edf(list, trialTime);

if(changeLoc > -1){

lightsOff[changeLoc] = 1;

}

int q;

for(q=0;q<ep;q++){

if(changeLoc > -1 && changeLoc == q){

changed[q] = 1;

}

}

list = temp;

cpuCheck++;

}

int d;

for(d = 0; d<ep; d++){

if(changed[d] != 1){

list[d].lengthOnCPu = list[d].lengthOnCPu +1;

}else{

list[d].lengthOnCPu = 0;

}

}

printf("\n");

int e;

for(e=0; e<ep; e++){

if(list[e].duration == 0 && checklist[e]!=0){

checklist[e] = 0;

}

if(checklist[e] == 0){

count++;

}

if(lightsOff[e] > 0 || list[e].inUse > 0){

list[e].inUse = 0;

lightsOff[e] = 0;

if(e == 0){

}

}

}

trialTime++;

safteycount++;

}

printf("%d", trialTime);

int rr;

for(rr=0;rr<cpuNum;rr++){

printf("\tIDLE");

}

printf("\n");

printf("\n\tSummary\n");

int spot[ep];

int sumCount=0;

int u;

for(u=0; u<ep; u++){

spot[u]=0;

}

int h;

int l;

for(h=0; h<ep; h++){

for(l=0;l<ep;l++){

if(spot[h] < 1 && spot[l] < 1){

if(h != l && strcmp(list[h].user, list[l].user)==0){

spot[l] = 1;

spot[h] = 2;

sumCount++;

if(list[l].total > list[h].total){

list[h].total = list[l].total;

}

}

}

}

}

int value = ep-sumCount;

information \* sumList;

if((sumList = malloc(value \* sizeof(information))) ==NULL){

return;

}

int increment = 0;

int o;

for(o=0; o<ep; o++){

if(spot[o]!=1){

sumList[increment] = list[o];

increment++;

}

}

int p;

for(p=0;p<value;p++){

printf("\t%s\t%d\n", sumList[p].user, sumList[p].total+1);

}

}

information \* edf(information \*list, int time){

int minDuration = 999;

int processLoc;

int minDeadLine;

int check = 0;

int j;

for(j=0;j<4;j++){

minDeadLine = list[j].deadLine;

if(list[j].arrival <= time && list[j].deadLine < minDeadLine){

if(list[j].duration < minDuration && list[j].duration != 0){

minDuration = list[j].duration;

processLoc = j;

check++;

}

}

}

if(check != 0){

list[processLoc].duration = list[processLoc].duration - 1;

list[processLoc].total = time;

printf("\t%c\t%c\n", list[processLoc].process, list[processLoc + 1].process);

}

return list;

}

int freeList(information \*list){

free(list);

return 1;

}

void printOutput(information \* in){

int j;

for(j=0;j<4;j++){

printf("%s\t%c\t%d\t%d\t%dn", in[j].user, in[j].process, in[j].arrival, in[j].duration, in[j].deadLine);

}

}