#include <stdio.h>

#include "schedule.h"

struct PCB{

    int pid;

    int begin\_tick;

    int burst\_tick;

    int remaining;

    int finishtick;

    int firstAllocated;

}pcb[50];

void switchPCB(struct PCB \*arr1, struct PCB \*arr2);

void FCFSorder();   //declaration

int kindOfSchedule;

int numOfPCB;

// fn: read\_proc\_list

// desc: read process file list

// param

//  file\_name: process list name

void read\_proc\_list(const char\* file\_name){

    FILE\* file1;

    file1 = fopen(file\_name, "r");

    if(file1==NULL){    //if there's no such file

        printf("no file exists\n");

        return;

    }

    fscanf(file1, "%d", &numOfPCB);     //check the number of the PCBs

    for (int i = 0; i<numOfPCB; i++){

        fscanf(file1, "%d %d %d", &(pcb[i].pid), &(pcb[i].begin\_tick), &(pcb[i].burst\_tick));   //read integers from file1

        pcb[i].remaining = pcb[i].burst\_tick;   //pcb[i].burst\_tick remains on pcb[i].remaining

    }

    if (kindOfSchedule==1)

        FCFSorder();

}

/\*\*

 \* void FCFSorder()

 \*

 \* Summary of the FCFSorder function :

 \*      The FCFSorder function switches pcb in order of begining time of the pcb

 \*      in ascending order.

 \*

 \* Parameters : nothing

 \*

 \* Return value : nothing

 \*

 \*

 \*/

void FCFSorder(){

    for (int i = 0; i<numOfPCB-1; i++){

        for (int j = i; j<numOfPCB; j++){

            if(pcb[i].begin\_tick > pcb[j].begin\_tick){

                switchPCB(&pcb[i], &pcb[j]);    //calls switchPCB function

            }

        }

    }

}

/\*\*

 \* void switchPCB(struct PCB \*arr1, struct PCB \*arr2)

 \*

 \* Summary of the switchPCB function :

 \*      The switchPCB functions switches the order of given 2 arrays

 \*

 \* Parameters : struct PCB pointer

 \*

 \* Return value : nothing

 \*

 \* Description :

 \*

 \*      The arrays can be modified anywhere

 \*

 \*

 \*/

void switchPCB(struct PCB \*arr1, struct PCB \*arr2){

    int tempPid = arr1->pid;

    int tempBeginTick = arr1->begin\_tick;

    int tempBurstTick = arr1->burst\_tick;

    int tempRemaining = arr1->remaining;

    int tempFinishTick = arr1->finishtick;

    arr1->pid = arr2->pid;

    arr1->begin\_tick = arr2->begin\_tick;

    arr1->burst\_tick = arr2->burst\_tick;

    arr1->remaining = arr2->remaining;

    arr1->finishtick = arr2->finishtick;

    arr2->pid = tempPid;

    arr2->begin\_tick = tempBeginTick;

    arr2->burst\_tick = tempBurstTick;

    arr2->remaining = tempRemaining;

    arr2->finishtick = tempFinishTick;  //switches arr2 and arr1

}

// fn: set\_schedule

// desc: set scheduling method

//

// param: method

//

// return none

void set\_schedule(int method){

    if (method == 1){

        kindOfSchedule = 1;

        //FCFS (Nonpreemptive)

    }

    else if (method == 2){

        kindOfSchedule = 2;

        //Shortest Job First (Nonpreemptive)

    }

    else if (method == 3){

        kindOfSchedule = 3;

        //Shortest Remaining Time First (Prremptive)

    }

}

// fn: do\_schedule

// desc: scheduling function called every tick from main

// param

//  tick: time tick beginning from 0

// return

//     -1: when all process are terminated

//      0: CPU is idle

// others: PID od running state

int do\_schedule(int tick){

    if(tick==0){

        for (int i = 0; i<50; i++){

            pcb[i].firstAllocated = 0;

        }

    }

    if (kindOfSchedule == 1){   //FCFS (Nonpreemptive)

        for (int i = 0; i < numOfPCB; i++){

            if (pcb[i].remaining>0){

                if(pcb[i].begin\_tick<=tick){    //after it came in

                    if(pcb[i].firstAllocated==0){

                        if(pcb[i].begin\_tick==0){   //if it begins at 0

                            pcb[i].firstAllocated = 0;

                        }

                        else

                            pcb[i].firstAllocated = tick;   //pcb[i].firstAlloacted has tick now

                    }

                    pcb[i].remaining--;

                    if(pcb[i].remaining==0){

                        pcb[i].finishtick = tick;   //when it's done

                    }

                    return i+1;     //the index of pcb + 1

                }

            }

            int check4 = 0;

            for (int i = 0; i<numOfPCB; i++){

                if (pcb[i].remaining>0){

                    check4 = 1;

                }

            }

            if (check4==0)

                return -1;  //when all process are terminated

        }

        return 0;

    }

    if(kindOfSchedule == 2){//Shortest Job First (Nonpreemptive)

        for (int i = 0; i < numOfPCB; i++){

            if (pcb[i].remaining<pcb[i].burst\_tick && pcb[i].remaining>0){    //running

                if(pcb[i].begin\_tick<=tick){    //after it came in

                    pcb[i].remaining--;         //CPU is allocated on pcb[i]

                    if(pcb[i].remaining==0){

                        pcb[i].finishtick = tick;   //when it's done

                    }

                    return i+1;

                }

                else

                    return 0;

            }

            int check = 0;

            for (int j = 0; j<numOfPCB; j++){

                if (pcb[j].remaining>0){

                    check = 1;      //see if there's any remaining work

                }

            }

            if(check==0){

                return -1;  //when all process are terminated

            }

        }

        int smallest;

        for (int i = 0; i<numOfPCB; i++){

            if(pcb[i].remaining!=0 && pcb[i].begin\_tick<=tick){

                smallest = i;   //initializes smallest

                break;

            }

        }

        for (int i = 0; i<numOfPCB; i++){

            if(pcb[i].remaining!=0 && pcb[i].begin\_tick<=tick && pcb[smallest].burst\_tick>pcb[i].burst\_tick){

                smallest = i;   //find the pcb that has smallest burst time

            }

        }

        if(pcb[smallest].firstAllocated==0){

            if(pcb[smallest].begin\_tick==0){            //pcb[smallest] begins at 0

                pcb[smallest].firstAllocated = 0;

            }

            else

                pcb[smallest].firstAllocated = tick;    //pcb[smallest] is allocated for the first time

        }

        pcb[smallest].remaining--;                      //pcb[smallest] is allocated

        if(pcb[smallest].remaining==0){                 //pcb[smallest] is finished

            pcb[smallest].finishtick = tick;

        }

        return smallest+1;          //returns the index of pcb[smallest] + 1

    }

    //3. Shortest Remaining Time First (Prremptive)

    if (kindOfSchedule == 3){

        int check2 = 0;

        for (int i = 0; i<numOfPCB; i++){

            if (pcb[i].remaining>0){

                check2 = 1;

            }

        }       //checks if there's any remaining work to do

        if (check2 == 0){

            return -1; //terminated

        }

        int ShortestIndex = -1;

        for (int i = 0; i < numOfPCB; i++){

            if(tick>=pcb[i].begin\_tick){

                if(ShortestIndex == -1 && pcb[i].remaining>0){

                    ShortestIndex = i;  //first one

                }

                else if(pcb[i].remaining<pcb[ShortestIndex].remaining && pcb[i].remaining>0){

                    ShortestIndex = i;  //found shortest remaining one

                }

            }

        }

        if (ShortestIndex>=0){  //if shortest remaining one is found

            if(pcb[ShortestIndex].firstAllocated==0){

                if(pcb[ShortestIndex].begin\_tick==0){   //pcb[ShortestIndex] works at 0

                    pcb[ShortestIndex].firstAllocated = 0;

                }

                else

                    pcb[ShortestIndex].firstAllocated = tick;   //pcb[ShortestIndex] is first allocated

            }

            pcb[ShortestIndex].remaining--;

            if(pcb[ShortestIndex].remaining == 0){

                pcb[ShortestIndex].finishtick = tick;       //pcb[ShortestIndex] is finished

            }

            return ShortestIndex+1;             //returns the index of pcb[ShortestIndex] + 1

        }

        else

            return 0;

    }

}

// fn: print\_performance();

// desc: print system performance

void print\_performance(){

    double avgTurnAroundTime = 0, avgWaitingTime = 0, avgResponseTime = 0;

    for (int i = 0; i < numOfPCB; i++){

        avgTurnAroundTime += pcb[i].finishtick - pcb[i].begin\_tick + 1;

        avgResponseTime += pcb[i].firstAllocated - pcb[i].begin\_tick;

        avgWaitingTime += pcb[i].finishtick - pcb[i].begin\_tick - pcb[i].burst\_tick + 1;    //calculates sum of time of them

    }

    avgResponseTime /= numOfPCB;

    avgTurnAroundTime /= numOfPCB;

    avgWaitingTime /= numOfPCB; //calculates average time of them

    printf("===============================================================================================\n");

    printf(" PID       begin        finish      Turn around time       Waiting time      Responsetime\n");

    printf("===============================================================================================\n");

    for (int i = 0; i<numOfPCB; i++){

        printf("%-11d %-13d %-17d %-20d %-20d %-20d\n", pcb[i].pid, pcb[i].begin\_tick,

        pcb[i].finishtick + 1, pcb[i].finishtick - pcb[i].begin\_tick + 1, pcb[i].finishtick - pcb[i].begin\_tick - pcb[i].burst\_tick + 1,

        pcb[i].firstAllocated - pcb[i].begin\_tick);     //displays

    }

    printf("===============================================================================================\n");

    printf("average :                            %10.2lf %20.2lf %20.2lf\n", avgTurnAroundTime, avgWaitingTime, avgResponseTime);

    //display result on screen

}

After saving integers from the file and decided which scheduling method to use, it reads data from the file and if it’s fcfs, it rearranges the order of pcbs in read\_proc\_list function). After these, do\_schedule function works until (res < 0 || tick > 50). In do\_schedule function, it works in certain way with the scheduling method.

For FCFS scheduling, which is 1, it find which pcb still has work to do. It saves tick on the pcb’s firstAllocated if its beginning tick is lower than tick or same as the tick and it’s allocated for the first time. Pcb[i].remaining-- since it’s allocated. And see if pcb[i] is finished which means pcb[i].remaining == 0. It saves the tick on pcb[i].finishtick if it(the pcb)’s all finished. Then, return the index of the pcb + 1. After all of these, it checks if all the pcbs are finished.

For SJF scheduling, it checks if there’s any pcb which was running previously by checking pcb[i].remaining<pcb[i].burst\_tick && pcb[i].remaining>0. pcb[i].remaining-- since pcb[i] is allocated. Then it checks if pcb[i] is finished by pcb[i].remaining==0. If it’s finished, saves tick(finish time) on pcb[i].finishtick. then returns the index of the pcb +1. If There’s no pcb which was running previously, find the shortest remaining time which exists on tick time and saves the index on smallest. Then, if it’s found, pcb[smallest].remaining--; since it’s allocated. Then checks if the pcb is finished, if so, pcb[smallest]. finishtick = tick to save the finish time. Then returns the index of pcb[smallest] + 1.

For SRTF scheduling, it checks if there’s any remaining work to do. If there’s no such thing, returns -1 since they’re all terminated. It finds the index of the pcb with shortest remaining time which exists on tick time. If it’s found, checks if it’s allocated for the first time and saves the tick time on cb[ShortestIndex].firstAllocated. pcb[ShortestIndex].remaining--; since it’s allocated. Then checks if pcb[ShortestIndex] is finished which is pcb[ShortestIndex].remaining == 0. If so, saves the tick time on pcb[ShortestIndex].finishtick. then it returns the index of the pcb[ShortestIndex] + 1

It prints the performance after all the pcbs are allocated.