scast305@ocelot:~/cop4610 103% cat schedule.c

```
/*
 * Author: Samuel Casto
 * PantherID: 6330314
 * Description: This program accepts a file input consisting of a number of proc
esses and sequentially the processes burst time and arrival time. The programwil
l take these values, and depending on whether FCFS or SJF was input, calculate t
he average waiting time and average turnaround time of the concurrent processes.
 * */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//struct to hold our arrival and burst times
struct process {
	int procNum;
	int arrival;//time variables for the process
	int burst;
	int start;
	int end;
	int wait;
	int Tat;
};

//compare function to sort structs based on arrival time then procNum
int compareFCFS(const void *proc1, const void *proc2){
	//creating our structs to compare
	const struct process *p1 = (const struct process *)proc1;
	const struct process *p2 = (const struct process *)proc2;
```

```
    //if proc1 arrived before proc2 then we return <0 so that proc1 is sorte
d before proc2

    if(p1->arrival < p2->arrival) return -1;

    //if proc2 arrived before proc1 then we return >0 so that proc2 is sorte
d before proc1

    if(p1->arrival > p2->arrival) return 1;


    //if we are here then the processes have the same arrival time and we ne
ed to sort by procNum as the lower num will be first

    if((p1->arrival == p2->arrival) && (p1->procNum < p2->procNum)) return -
1;

    else return 1;

}


//compare function to sort structs based on burst assuming they've arrived

int compareSJF(const void *proc1, const void *proc2){

    //creating our structs to compare

    const struct process *p1 = (const struct process *)proc1;

    const struct process *p2 = (const struct process *)proc2;

    //sorting by burst and in the event of tie, procNum which is the order t
hey were scanned

    if(p1->burst < p2->burst) return -1;

    if(p1->burst > p2->burst) return 1;

    return (p1->procNum - p2->procNum);

}


int main(int argc, char **argv){

    //we are going to open the file, read the lines and store the values in
arrays(probably), and then loop through the arrays

    //performing our needed math before outputting the results most likely
```

```
	//checking for correct num of args
	if(argc != 3){
		fprintf(stderr,"Usage: %s fileName algorithm\n",argv[0]);
		exit(1);
	}

	//declaring variables
	FILE* file;
	int numProc;
	char* algo;

	//assigning variables from args
	algo = argv[2];
	file = fopen(argv[1],"r");
	//checking if algo has an acceptable string
	if(strcmp(algo,"FCFS") != 0 && strcmp(algo,"SJF") != 0){
		//if both are true then we do not have a proper algorithm select
ed
		fprintf(stderr,"Improper algorithm selection. Select FCFS or SJF
\n");
		exit(1);
	}
	//checking if the file was opened properly
	if(file == NULL){
		fprintf(stderr, "File was not opened.\n");
		exit(1);
	}
	//if we are here then the file has opened properly and we need to now re
ad the data and store it into our variables
	if(fscanf(file, "%d", &numProc) != 1){
```

```
        fprintf(stderr, "Error reading number of processes.\n");
        exit(1);
    }

    //now we need to create our arrays holding our burst time and arrival ti
me based on the size of numProc
    struct process array[numProc];
    //looping through the file until we have all our times
    for(int i = 0; i < numProc; i++){
        int burst, arrival;
        fscanf(file, "%d %d", &burst,&arrival);
        array[i].procNum = i;
        array[i].burst = burst;
        array[i].arrival = arrival;
        array[i].start = 0;
        array[i].end = 0;
        array[i].wait = 0;
        array[i].Tat = 0;
    }

    //we have our times stored now we need to compute average waiting times
and turnaround times for the selected algorithm
    //
    //Turnaround time(Tat) = exit time - arrival time
    //wait time = Tat - burst time
    //
    //we will already know the arrival time, we will need to compute our sta
rt times and exit times
```

```
    //if we sort the array by arrival time, then we can just add the burst t
imes together based on that order
    //have a temp variable keep track of our total time spent, and then calc
ulate the start and exit times based
    //on the sorted array


    //FCFS scheduling algorithm selected
    if(strcmp(algo,"FCFS") == 0){
        //declaring variables to help keep track of start/end times amon
g others
        int curTime = 0;
        double avgWait = 0;
        double avgTat = 0;
        qsort(array, numProc, sizeof(struct process), compareFCFS);
        //here we have a properly sorted array by arrival time, then pro
cNum
        curTime = array[0].arrival;
        for(int i = 0; i < numProc; i++){
            //updating our start and end times to calculate Tat/wait
time
            array[i].start = curTime;
            curTime +=  array[i].burst;
            array[i].end = curTime;
            array[i].Tat = array[i].end - array[i].arrival;
            array[i].wait = array[i].Tat - array[i].burst;
            avgWait += array[i].wait;
            avgTat += array[i].Tat;
        }

        //outputting our average wait time and Tat
```

```
        avgWait = avgWait / numProc;
        avgTat = avgTat / numProc;
        printf("Average wait time using SJF: %f\n",avgWait);
        printf("Average turnaround time using SJF: %f\n",avgTat);
        //for extra clarity
        printf("\n");
    }//end of FCFS scheduling algorithm


    //SJF scheduling algorithm selected
    if(strcmp(algo,"SJF") == 0){
        int curTime = 0;
        double avgWait = 0;
        double avgTat = 0;
        qsort(array, numProc, sizeof(struct process), compareFCFS);
        //here we have an array sorted currently on arrival time and aft
er processing the first burst we will use SJF
        //
        //since this loop needs to update after each run, we will use a
while loop that calls the compareSJF function to resort
        int i = 0;
        while(i < numProc){
            if(curTime >= array[i].arrival){
                array[i].start = curTime;
                curTime +=  array[i].burst;
                array[i].end = curTime;
                array[i].Tat = array[i].end - array[i].arrival;
                array[i].wait = array[i].Tat - array[i].burst;
                avgWait += array[i].wait;
```

```
                avgTat += array[i].Tat;
                i++;
                //we need to make sure we are not including proc
esses that have already happened
                qsort(array + i, numProc - i, sizeof(struct proc
ess), compareSJF);
            }
            else {
                // Move to the next process if the current proce
ss hasn't arrived yet
                curTime = array[i].arrival;
            }
        }//end of while loop

        avgWait = avgWait / numProc;
        avgTat = avgTat / numProc;
        printf("Average wait time using SJF: %f\n",avgWait);
        printf("Average turnaround time using SJF: %f\n",avgTat);
        //for extra clarity
        printf("\n");
    }//end of SJF scheduling algorithm

    printf("Process completion order:\n");
    //outputting processes in order of completion
    for (int i = 0; i < numProc; i++) {
        printf("Process %d\n", array[i].procNum);
        //printf("Arrival Time: %d\n", array[i].arrival);
        //printf("Burst Time: %d\n", array[i].burst);
        //printf("Start Time: %d\n",array[i].start);
        //printf("End Time: %d\n",array[i].end);
```

```
            //printf("Wait Time: %d\n",array[i].wait);
            //printf("Tat Time: %d\n",array[i].Tat);
            //printf("\n");
        }

        return 0;
}
```