**OS PROJECT REPORT FILE**

**Name**: Suman Sagar

**Reg No**: 11803595

**Roll**: 54

**Section**: K18PA

**Email**: [suman2000sagar@gmail.com](mailto:suman2000sagar@gmail.com)

**GitHub Link**: <https://github.com/num83r5iv3/OS/tree/master>

**Question:** Three students (**a, b, c**) are arriving in the mess at the same time. The id numbers of these students are **2132**, **2102**, **2453** and the food taken time from the mess table is **2**, **4** and **8** minutes. If the two students have same remaining time so it is broken by giving priority to the students with the lowest id number. Consider the longest remaining time first (LRTF) scheduling algorithm and calculate the average turnaround time and waiting time.

**Code:** <https://github.com/num83r5iv3/OS/blob/master/Simulation%20Project>

#include <stdio.h>

struct student

{int StudentId;

int FoodTakenTime;

int WaitingTime;

int TurnAroundTime;

};

void accept(struct student list[], int s);

void display(struct student list[], int s);

void scheduling(struct student list[], int s);

void waitingTime(struct student list[], int n);

void turnAroundTime(struct student list[], int n);

int main()

{struct student data[20];

int n,i;

char c='n';

do

{ printf("Please enter the No. of Students wants to eat in mess? : ");

scanf("%d", &n);

accept(data, n);

scheduling(data, n);

waitingTime(data,n);

turnAroundTime(data,n);

display(data, n);

printf("Want to continue? press 'y' : ");

scanf("%s",&c);

}while(c=='y');

return 0;

}

void accept(struct student list[80], int s)

{

int i;

for (i = 0; i < s; i++)

{

printf("\n\nEnter data for Student #%d", i + 1);

printf("\nEnter Student id : ");

scanf("%d", &list[i].StudentId);

printf("Enter time taken for food (minuts): ");

scanf("%d", &list[i].FoodTakenTime);

}

}

void display(struct student list[80], int s)

{

int i,AvgWaitingTime=0,AvgTurnAroundTime=0;

int TotalWatingTime=0,TotalTurnAroundTime=0;

printf("\n\n\t\t\tOutput according to LRTF\n");

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

printf("\n\t\t\t|Student id\tFoodTakenTime\tWaitingTime\tTurnAroundTime |");

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

for (i = 0; i < s; i++)

{

printf("\n\t\t\t|%d\t\t%d\t\t%d\t\t%d\t\t|", list[i].StudentId, list[i].FoodTakenTime,list[i].WaitingTime,list[i].TurnAroundTime);

printf("\a\n\t\t\t|---------------------------------------------------------------|");

TotalWatingTime= TotalWatingTime+list[i].WaitingTime;

TotalTurnAroundTime= TotalTurnAroundTime+list[i].TurnAroundTime;

}

printf("\n\n\t\t\tTotal Waiting Time is: = %d",TotalWatingTime);

printf("\n\t\t\tTotal Turn around Time is: = %d\n\n",TotalTurnAroundTime);

printf("\n\n\t\t\tAverage Waiting Time is: = %d",TotalWatingTime/s);

printf("\n\t\t\tAverage Turn around Time is: = %d\n\n",TotalTurnAroundTime/s);

}

void scheduling(struct student list[80], int s)

{

int i, j;

struct student temp;

for (i = 0; i < s - 1; i++)

{

for (j = 0; j < (s - 1-i); j++)

{

if (list[j].FoodTakenTime < list[j + 1].FoodTakenTime)

{

temp = list[j];

list[j] = list[j + 1];

list[j + 1] = temp;

}

else if(list[j].FoodTakenTime == list[j + 1].FoodTakenTime)

{

if(list[j].StudentId > list[j + 1].StudentId)

{

temp = list[j];

list[j] = list[j + 1];

list[j + 1] = temp;

}

}

}

}

}

void waitingTime(struct student list[80], int n)

{

int j,total;

list[0].WaitingTime=0;

for(j=1;j<n;j++)

{

list[j].WaitingTime=list[j-1].WaitingTime+list[j-1].FoodTakenTime;

}

}

void turnAroundTime(struct student list[80], int n)

{

int j,total;

for(j=0;j<n;j++)

{

list[j].TurnAroundTime=list[j].WaitingTime+list[j].FoodTakenTime;

}

}

**Description:**

This project was based on Longest Job First method, which I’ve done using LRTF. This is a pre-emptive version of Longest Job First (LJF) scheduling algorithm. Here, In this scheduling algorithm, we find the process with the maximum remaining time and then process it. We will check for the maximum remaining time after some interval of time (say 1 unit each) to check if another process having more Burst Time arrived up to that time.

There are 3 students to be served. Since they arrive at the same, we need an algorithm as to how the students are to be served food.

Given that;

LRTF (Longest Remaining Time Algorithm) is to be used.

Also, the questions states that, if two students have the same remaining time, the student with the lowest id number is served first.

Theoretical Explanation:

LRTF is to be used. Student C has the longest remaining time. So the serving starts with C.

After C is served for 4 minutes, the remaining time of B and C are equal. B has the lowest ID number. So the serve moves onto B.

After B is served for 2 minutes, the reaming time of A and B are equal. But still, B has the lowest ID number. So B is served.

Since his “food taken time” is 4 minutes, he is done with the job.

Comparing the remaining students A and C, C has a longer waiting time i.e., 4 min (he was already served 4 minutes, total time being 8 min). So, C is continued with the service.

After serving him for 2 minutes, the remaining time of C and A are equal. A has the lowest ID number. So the serve moves onto A.

After serving A for 2 minutes, he is done with the job since his “food taken time” is 2 minutes.

The serve moves onto C again to process the remaining 2 min of his “food taken time”.

**Algorithm:**

* **Step-1**: Create a structure of process containing all necessary fields like AT (Arrival Time), BT (Burst Time), CT (Completion Time), TAT (Turn Around Time), WT (Waiting Time).
* **Step-2**: Sort according to the AT;
* **Step-3**: Find the process having Largest Burst Time and execute for each single unit. Increase the total time by 1 and reduce the Burst Time of that process with 1.
* **Step-4**: When any process has 0 BT left, then update the CT (Completion Time of that process CT will be Total Time at that time).
* **Step-5**: After calculating the CT for each process, find TAT and WT.

**Formulas:**

**Turn Around Time (TAT) = Completion Time (CT) - Arrival Time (AT).**

**Wait Time (WT) = Turn Around Time (TAT) - Burst Time (BT).**

**Code Snippets:**

**Output:**

A screenshot of a cell phone

Description automatically generated

**Turn Around Time(TAT) = Completion Time(CT) - Arrival Time(AT)**

**TAT(A) = 12 - 0 = 12 min**

**TAT(B) =  8 - 0  =  8 min**

**TAT(C) = 14 - 0 = 14 min**

**Wait Time (WT) = Turn Around Time(TAT) - Burst Time(BT)**

**WT(A) = 12 - 2 = 10**

**WT(B) =   8 - 4 =   4**

**WT(C) = 14 - 8 =   6**

**Therefore,**

**The required Average Turn Around Time = (12+8+14)/3 = 11.33 minutes**

**The required Average Wait Time             = (10+4+6)/3    =  6.67 minutes**

**Main Loop:**

**A screenshot of a computer

Description automatically generated**

for (i = 0; i < s - 1; i++)

{for (j = 0; j < (s - 1-i); j++)

{if (list[j].FoodTakenTime < list[j + 1].FoodTakenTime)

{temp = list[j];

list[j] = list[j + 1];

list[j + 1] = temp}

else if(list[j].FoodTakenTime == list[j + 1].FoodTakenTime)

{if(list[j].StudentId > list[j + 1].StudentId)

{temp = list[j];

list[j] = list[j + 1];

list[j + 1] = temp; }}}}

**Waiting Structure:**

**A screenshot of a social media post

Description automatically generated**

**Turn Around Time Structure:**

**A screenshot of a social media post

Description automatically generated**

**Term Descriptions:**

**First Come First Serve (FCFS):** Simplest scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is a non-preemptive scheduling algorithm.

**Note:** First come first serve suffers from convoy effect.

**Shortest Job First(SJF):** Process which have the shortest burst time are scheduled first.If two processes have the same bust time then FCFS is used to break the tie. It is a non-preemptive scheduling algorithm.

**Longest Job First(LJF):** It is similar to SJF scheduling algorithm. But, in this scheduling algorithm, we give priority to the process having the longest burst time. This is non-preemptive in nature i.e., when any process starts executing, can’t be interrupted before complete execution.

**Shortest Remaining Time First(SRTF):** It is preemptive mode of SJF algorithm in which jobs are schedule according to shortest remaining time.

**Longest Remaining Time First(LRTF):** It is preemptive mode of LJF algorithm in which we give priority to the process having largest burst time remaining.

**Round Robin Scheduling:** Each process is assigned a fixed time(Time Quantum/Time Slice) in cyclic way. It is designed especially for the time-sharing system. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1-time quantum. To implement Round Robin scheduling, we keep the ready queue as a FIFO queue o£ processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1-time quantum, and dispatches the process. One of two things will then happen. The process may have a CPU burst of less than 1-time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1-time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail o£ the ready queue. The CPU scheduler will then select the next process in the ready queue.

**Priority Based scheduling (Non-Preemptive):** In this scheduling, processes are scheduled according to their priorities, i.e., highest priority process is scheduled first. If priorities of two processes match, then schedule according to arrival time. Here starvation of process is possible.

**Highest Response Ratio Next (HRRN)** In this scheduling, processes with highest response ratio is scheduled. This algorithm avoids starvation.

**Multilevel Queue Scheduling:** According to the priority of process, processes are placed in the different queues. Generally high priority process are placed in the top level queue. Only after completion of processes from top level queue, lower level queued processes are scheduled. It can suffer from starvation.

**Multi level Feedback Queue Scheduling:** It allows the process to move in between queues. The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time, it is moved to a lower-priority queue.

**Test Cases:**

1.

|  |  |  |  |
| --- | --- | --- | --- |
| Student Id | Food Taken Time | Waiting Time | Turn Around Time |
| 3465 | 3 | 0 | 3 |
| 4532 | 3 | 3 | 6 |
| 3595 | 2 | 6 | 8 |

**A screenshot of a cell phone

Description automatically generated**

Total waiting time is: 9

Total Turn Around time is : 17

Average waiting time is: 3

Average Turn Around time is : 5.

2.

|  |  |  |  |
| --- | --- | --- | --- |
| Student Id | Food Taken Time | Waiting Time | Turn Around Time |
| 4444 | 4 | 0 | 4 |
| 3333 | 3 | 4 | 7 |
| 2222 | 2 | 7 | 9 |
| 1111 | 1 | 9 | 10 |

A screenshot of a cell phone

Description automatically generated

Total waiting time is: 20

Total Turn Around time is : 30

Average waiting time is: 5

Average Turn Around time is : 7.

GitHub Link: <https://github.com/num83r5iv3/OS/tree/master>