CSE-316 Simulation Project

Made by - Priyansh Tiwari .

Reg. Number - 11711226

Roll Number - 13

Section - K17KV

Submitted to - Mr. Sumit Mittu

April 14, 2019

Author: Priyansh Tiwari

INDEX

Serial Number	Content	Page Number
1	Description	3
2	Algorithm and Complexity	3
3	Constraints	4
4	Test Cases	7
5	GitHub Revision Confirmation	8

Font Used – Times New Roman

Font Size- 12

Line Spacing – 1.5

Student Name : Priyansh Tiwari Registration Number : 11711226

Email : priyansht645@gmail.com

GitHub Link : https://github.com/priyansh14/K17KVA13-OS-PROJECT

1.Description :

The problem is to implement Preemptive Shortest Job First CPU Scheduling Algorithm on a set of processes. The burst times of these processes are to be read from the first line of a text file named “CPU_BURST.txt”. The burst times are also to be validated and rejected in case they are not greater than or equal to 0 i.e. if they are not positive. The arrival time of the processes should be taken as every 3 units of time for each process. This implies that Process 1 arrives at 0 ms , Process 2 arrives at 3 ms , Process 3 arrives at 6 ms , Process 4 arrives at 9 ms and Process 5 arrives at 12 ms. The solution should compare the CPU burst time required by the newly arrived process against the remaining time of the currently executing process and decide whether to preempt the currently executing process or let it continue. The waiting and Turn Around times are to be calculated for each process and then the average waiting time and average turn around times are to be displayed in the Output.

2. Algorithm and Complexity:

The algorithm that I have implemented in my solution for this problem is as follows :

a. Traverse all the processes and stop only when all are finished.
b. Find a process with minimum remaining time every single iteration.
c. Subtract its time by 1 every iteration.
d. If the remaining time becomes 0, increment the counter of Completed Processes by 1.
e. The completion time of current process is taken as current time (current value of iteration) + 1.

f. Calculate waiting time for each process which gets completed using the formula : waitingTime[i] = Completion Time – Arrival Time – CPU Burst Time.

g. Increase the iteration (time lap) by 1.
Here the loop of Step a ends.

h. Find the Turn Around Time for each process by adding Burst Time and Waiting Time.
Turn Around Time = Burst Time + Waiting Time.

i. Find the average waiting and average turn around time by doing their sun and dividing by 5.
End Of Algorithm.

COMPLEXITY OF CODE :

1. The function for calculation waiting time consists of nested loops for iterating over processes. As we know , the complexity for nested loops is their product that is N^2.
2. The search for shortest process requires the traversal of all the processes which take O(N) time.
3. Finding the turn-around time also requires traversing to each process and doing a calculation which takes constant time and traversal takes O(N) time.
4. Finding the Average Waiting and Turn Around Times are constant order operations O(1).
5. The overall complexity of the code is O(N^2 + N + N +1). As we ignore the smaller terms and take the biggest term, the **final overall complexity** of the algorithm becomes **O(N^2)**. Since the value of N is small (5), the code runs extremely fast.

3. CONSTRAINT SATISFACTION :

The constraints of the problem with their respective code snippets are as follows :

1. **READ CPU BURST FROM A FILE** : This constraint required that the Burst times for the processes to be retrieved from the text file called CPU_BURST.txt.

2. **VALIDATE THE INTEGERS** : This constraint required that the burst times should be accepted only if they are positive. The snippet for constraints 1 and 2 is :

   SNIPPET :

```
 //Code for this part starts here
FILE *file;
file=fopen("CPU_BURST.txt","r");
while ( fscanf(file, "%d", & number ) == 1 )   //CONSTRAINT 1
{
  if(number<=0)    //CONSTRAINT  2
        {
             printf("THERE ARE NEGATIVE NUMBERS IN THE FILE AND
             THEY ARE INVALID.EXITING PROGRAM NOW.");
              exit(1);
        }
        temp[i]=number;     //If it is positive, then store it in an temporary array .
        i++;

  }

    //SNIPPET FINISHED.
```

3. **USE PREEMPTIVE SJF ALGORITHM** : This constraint required the solution to schedule the processes using preemptive Shortest Job First algorithm a.k.a Shortest Remaining Time First. The snippet for this constraint is :

   SNIPPET :

```
//Code starts here
void calculateWaitTime(struct Process p[],int n,int waitTime[])
{
     int i,j;
     int rem[n];
     for( i=0;i<n;i++)
           rem[i]=p[i].burstTime;
```

```
    int completedProcesses=0,time=0,minimum=INT_MAX;
    int smallest=0,finishTime;
    boolean c=false;

while(completedProcesses!=n)  //until all processes are finished
{
    for(j=0;j<n;j++)
    {
            if((p[j].arrivalTime<=time)&&(rem[j]<minimum)&&rem[j]>0){
            minimum=rem[j];
            c=true;
            smallest=j;
            }
    }
    if(c==false){
            time+=1;
            continue;
    }
    rem[smallest]--;
    minimum=rem[smallest];
    if(minimum==0)
    minimum=INT_MAX;

    if(rem[smallest]==0)
    {
            completedProcesses+=1;
            c=false;
            finishTime=time+1;

            //Calculation of waiting time
```

```
        waitTime[smallest]=finishTime-p[smallest].burstTime-p[smallest].arrivalTime;

    if(waitTime[smallest]<0)
        waitTime[smallest]=0;
    }
    time++;
    }
    }
```

//SNIPPET CODE ENDED.

4. **ARRIVAL TIME** : This constraint required that processes should arrive every 3 seconds.

   SNIPPET :

```
   // Code starts here
   for(i=0;i<5;i++)
   {
   p[i].arrivalTime=i*3;  //According to question,the processes arrive every 3 second.
   p[i].p_no=i+1; //Process no of the processes is given here.
   }
   //SNIPPET CLOSED.
```

   These were the constraints of the problem and all of them were satisfied accordingly.

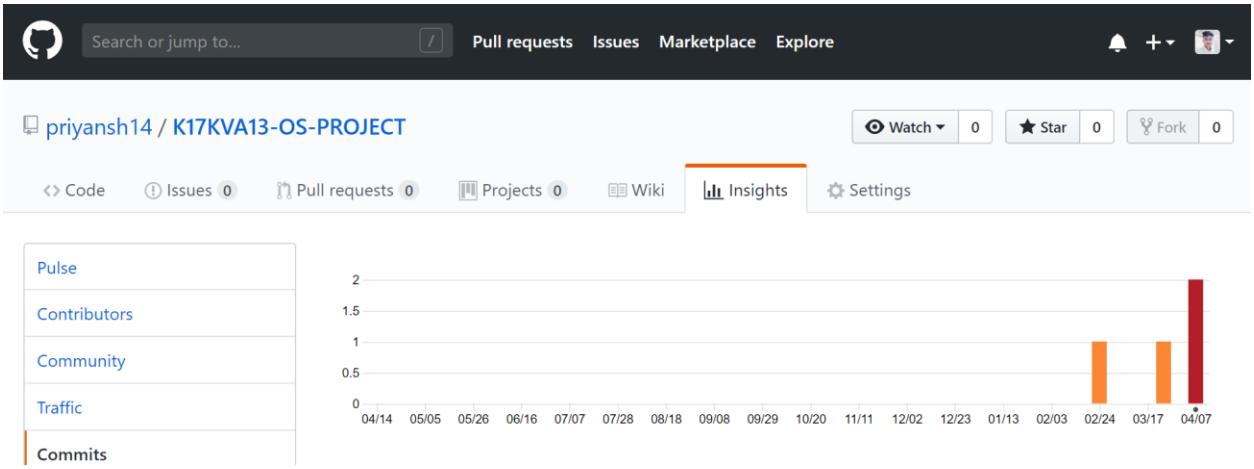
4. Test Cases Applied :

I applied the following test cases :

1. Putting a 0 in the CPU_BURST.txt file. Test case was passed and INVALID message was displayed.
2. Putting a negative integer in the text file. Test case was passed and INVALID message got printed.
3. Arrival Times were checked by separately printing them.
4. Conducted Unit Testing and System Testing on 3 different PCs.
5. Used Equivalence Class Partitioning Testing Technique.

5. Revisions on GitHub :

Yes, I have made more than 5 revisions on GitHub. I have done 8 commits in the master branch. The first commit was on 24th February 2019 when the problem statements were assigned whereas the last commit was on 13th April 2019. A graph of the activity on the GitHub repo is as follows :



The first orange bar indicates the first commit on 02/24.

The total files on the repo are 8.

GitHub Link : https://github.com/priyansh14/K17KVA13-OS-PROJECT