Exp No. 4. Multi-processor scheduling using FCFS

Dr S.Rajarajan, SoC

Process	P0	P1	P2	P3	P4	P5	P6
AT	0	2	4	5	5	6	7
BT	4	2	3	3	4	5	3
CT	4	4	7	8	9	12	11

Procedure

i. Get the number processes, arrival times and burst times
ii. Get the number of CPUs
iii. Whenever a CPU becomes free and one or more processes arrived by that time and waiting in the queue, schedule the first arrived process on that CPU
iv. Repeat the above steps until all the processes are completed
v. Display the completion time and turn-around time of processes

Sample problem

Numberof CPUs: 3

CPU1

0 - 4 P0	5 – 8 P3	8 – 11 P6			

CPU2

2 - 4 P1	5 – 9 P4				

CPU3

4 - 7 P2	7 – 12 P5				

```c
#include<stdio.h>
#include<stdlib.h>

int *processors;

struct process
{
    int pid,at,bt,ft,status,sel;
}rq[10];
void dispatcher(int t,int n,int cpu)
{
    for(int i=0;i<n;i++)
    {
    //checking if the process is selected by the processor, if the process
    //has arrived and have completed the process or not
        if(rq[i].status!=1 && rq[i].sel==0 && rq[i].at<=t)
        {
        //finding the free processor
            for(int j=0;j<cpu;j++)
            {
                if(processors[j]==-1)
                {
                //assigning the selected process to the processor
                    rq[i].sel=1;
                    processors[j]=i;
                    break;
                }
            }
        }
    }
}

void fcfsIO(int n,int cpu)     //initializing the processors as free(-1)
{
    for(int i=0;i<cpu;i++)
    {
            processors[i]=-1;
    }
    int time=0;
    int i=0;
    while(i<n)
    {
        dispatcher(time,n,cpu);
        //looping through all the processors to complete their process
        for(int k=0;k<cpu;k++)  //checking if the processor has process
        {
```

```
            if(processors[k]!=-1)
            {
                    rq[processors[k]].bt--;
        //checking if the processor has completed the process
                    if(rq[processors[k]].bt==0)
                    {
                            rq[processors[k]].status=1;
                            rq[processors[k]].ft=time+1;
                            processors[k]=-1;
                            i++;
                    }
            }
        }
        time++;
    }
}

int main()
{
    //getting the input
    FILE *fptr;
    fptr=fopen("mpsInput.txt","r");
    int n,cpu;
    processors=(int*) malloc (cpu*sizeof(int));
    fscanf(fptr,"%d %d",&n,&cpu);
    for(int i=0;i<n;i++)
    {
        fscanf(fptr,"%d %d",&rq[i].at,&rq[i].bt);
        rq[i].status=0;
        rq[i].pid=i+1;
        rq[i].sel=0;
    }
    fclose(fptr);

    //calling the function
    fcfsIO(n,cpu);

    //displaying the output
    printf("PID\tAT\tBT\tFT\tSTATUS\n");
    for(int i=0;i<n;i++){
        printf("%d\t%d\t%d\t%d\t%d\n",rq[i].pid,rq[i].at,rq[i].bt,rq[i].ft,rq[i].s
    }
}
```