CPU scheduling based on FCFS with CPU and IO burst times

Dr S.Rajarajan
SOC
SASTRA

Step 1: Take the process with the lowest arrival time
Step 2: If the BT1 of process !=0 then
Step 3: Update current time to Current time + Burst time1 of chosen process
Step 4: Update BT1 to zero
Step 5: Update Arrival Time of process as AT +IO
Step 6: else
Step 7: Update current time to Current time + Burst time2 of chosen process
Step 8: Update BT2 to zero
Step 9: Update Completion time of process as CT = Current time
Step 10: Sort processes as per arrival time
Step 11: Repeat previous steps until all the processes are completed

Sample Problem

Process	A	B	C	D	E
AT	0	2	3	5	7
BT1	3	2	4	6	2
I/O	3	4	3	2	4
BT2	2	3	1	3	5

WT = TT - (BT1+IO+BT2)

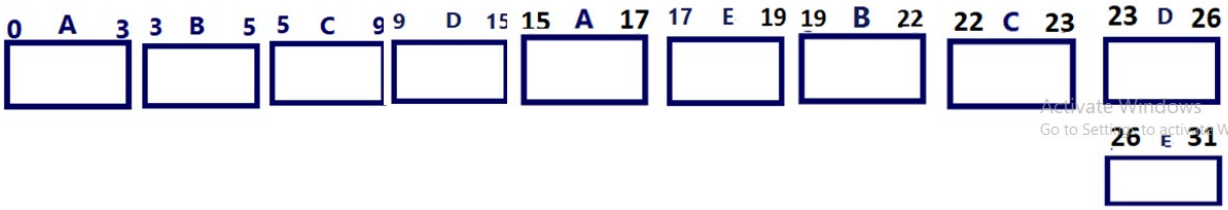
	A	B	C	D	E
CT					
TT					
WT					

Solution

Process	A	B	C	D	E
AT	6	9	12	17	23
BT1					
I/O					
BT2	2	3	1	3	5

WT = TT - (BT1+IO+BT2)

	A	B	C	D	E
CT	17	22	23	26	31
TT	17	20	20	21	24
WT	9	11	12	10	13



Activate Windows
Go to Settings to activate Wind

```c
#include <stdio.h>
struct process
{
    int at;  //arrival time to be updated
    int at_actual; // actual arrival time
    int bt1; //CPU burst time before IO
    int bt2; //CPU burst time after IO
    int io;  //IO burst time
    int status;// completed -1, not yet completed - 0
    int ft; // finish time
    int pid;
}ready_list[10], temp;
int n, cur_time=0, idle_time=0;
int dispatcher();
void sortReadyList();
int main()
{
    int i,j,pid, p=100;
    printf("Enter number of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Process %d\n",i+1);
        printf("***********\n");
        printf("Enter Arrival Time:");
        scanf("%d",&ready_list[i].at);
        ready_list[i].at_actual=ready_list[i].at;
        printf("Enter burst Time1:");
        scanf("%d",&ready_list[i].bt1);
        printf("IO burst time:");
        scanf("%d",&ready_list[i].io);
        printf("Enter burst Time2:");
        scanf("%d",&ready_list[i].bt2);
        ready_list[i].status=0; // 0 - not yet completed, 1 - already completed
        ready_list[i].pid= p++;
    }
    i=0;
    // Until all the n processes are completed
    while(i < n)
    {
        pid=dispatcher(); // To identify the next process to be scheduled
        if(ready_list[pid].status == 1)
        {
            i++;
            ready_list[pid].ft=cur_time;
        }
```

```
    }
    printf("Process Id\tFinish Time\tTT\n");
    printf("**********\t**********\t**\n");
    for(i=0;i<n;i++)
    {
    printf("%d\t\t%d\t\t%d\n",ready_list[i].pid,ready_list[i].ft,(ready_list[i].ft
    -ready_list[i].at_actual)) ;
    }
    // total time that CPU was not running any processes
    printf("Total CPU idle time: %d", idle_time);
}
// Function to pick the next process with lowest arrival time
int dispatcher()
{
    int i,index=-1,j;

    back:
    sortReadyList();
    for(i=0;i<n;i++)
    {
            // To check that ith process has arrived either newly or after IO
            if(ready_list[i].at <= cur_time)
            {
                // To check that ith process is not yet completed
                if(ready_list[i].status != 1)
                {
                    // To check that first CPU burst is completed or not
                    if(ready_list[i].bt1 > 0)
                    {
                        cur_time = cur_time + ready_list[i].bt1;
                        // Updated arrival time after IO completion
                        ready_list[i].at = cur_time + ready_list[i].io;
                        ready_list[i].bt1 = 0;
                        index=i; // index of the process that is currently chosen
                    }
                    // To check that second CPU burst is completed or not
                    else if(ready_list[i].bt2 > 0)
                    {
                        ready_list[i].status = 1; // Since second BT is completed
                        cur_time = cur_time + ready_list[i].bt2;
                        ready_list[i].bt2 = 0;
                        index=i; // index of the process that is currently chosen
                    }
                    break; // Since next process is chosen, end the loop
                }
```

```
            }
        }
        if(index == -1) // Next process not yet available at the current time
        {
            cur_time++; // To move the clock until it reach the arrival time of next p
            idle_time++; // Since CPU has been idle waiting for next process
            goto back;
        }
        return index;
}
void sortReadyList() // To sort processes as per arrival time
{
        int i,j;
        for(i=0;i<n-1;i++)
            for(j=0;j<n-1;j++)
            {
                if(ready_list[j].at > ready_list[j+1].at)
                {
                    temp=ready_list[j];
                    ready_list[j]=ready_list[j+1];
                    ready_list[j+1]=temp;
                }
            }
}
```