# ROUND ROBIN SCHEDULING

## Aim:

To implement the Round Robin (RR) scheduling technique

## Algorithm:

1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array rem_bt[] to keep track of remaining burst time of processes which is initially
copy of bt[] (burst times array)
5. Create another array wt[] to store waiting times of processes. Initialize this array as 0.
6. Initialize time : t = 0
7. Keep traversing the all processes while all processes are not done. Do following for i'th
process if it is not done yet.
a- If rem_bt[i] > quantum
(i) t = t + quantum
(ii) bt_rem[i] -= quantum;
b- Else // Last cycle for this process
(i) t = t + bt_rem[i];
(ii) wt[i] = t - bt[i]
(iii) bt_rem[i] = 0; // This process is over

8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

## Program Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
        float avg_wt, avg_tat;
        printf(" Total number of process in the system: ");
        scanf("%d", &NOP);
        y = NOP;
```

```
        for(i=0; i<NOP; i++)
        {
                printf("\n Enter the Arrival and Burst time of the Process[%d]\n",
i+1);
                printf(" Arrival time is: \t");  // Accept arrival time
                scanf("%d", &at[i]);
                printf(" \nBurst time is: \t"); // Accept the Burst time
                scanf("%d", &bt[i]);
                temp[i] = bt[i]; // store the burst time in temp array
        }
        printf("Enter the Time Quantum for the process: \t");
        scanf("%d", &quant);
        printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
        for(sum=0, i = 0; y!=0; )
        {
                if(temp[i] <= quant && temp[i] > 0) // define the conditions
                {
                   sum = sum + temp[i];
                   temp[i] = 0;
                   count=1;
                }
                else if(temp[i] > 0)
                {
                    temp[i] = temp[i] - quant;
                    sum = sum + quant;
                }
                if(temp[i]==0 && count==1)
                {
                    y--; //decrement the process no.
                    printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i],
sum-at[i], sum-at[i]-bt[i]);
                    wt = wt+sum-at[i]-bt[i];
                    tat = tat+sum-at[i];
                    count =0;
                }
                if(i==NOP-1)
                {
                    i=0;
                }
                else if(at[i+1]<=sum)
                {
                    i++;
                }
```

```
            else
            {
                  i=0;
            }
      }
      avg_wt = wt * 1.0/NOP;
      avg_tat = tat * 1.0/NOP;
      printf("\n Average Turn Around Time: \t%f", avg_wt);
      printf("\n Average Waiting Time: \t%f", avg_tat);
      getch();
}
```