**Lab no 6**

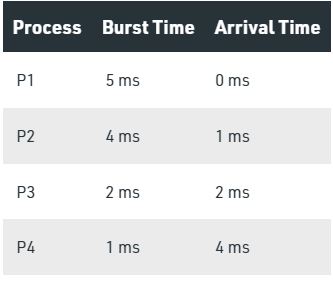
**Round Robin Scheduling**

***Objectives:***

* What is round robin scheduling?
* Implementation of round robin scheduling.

**Round Robin Algorithm:**

**Round-robin scheduling** algorithm is one of the simplest [scheduling algorithms](http://en.wikipedia.org/wiki/Scheduling_algorithm). It is designed especially for [time-sharing systems](http://en.wikipedia.org/wiki/Time-sharing_system). The [ready queue](http://en.wikipedia.org/wiki/Ready_queue) is treated as a [circular queue](http://en.wikipedia.org/wiki/Circular_queue). The algorithm assigns a [time slice](http://en.wikipedia.org/wiki/Time_slice) (also called **time quantum**) to each process in the ready queue in order, handling all processes without priority. A maximum of one time slice is allocated at once. If the remaining request is less than a time slice, only the remaining request time is allocated. Round-robin scheduling is both simple and easy to implement. It is also [starvation](http://en.wikipedia.org/wiki/Resource_starvation)-free.



CODE :

# Python3 program for implementation of

# RR scheduling

# Function to find the waiting time

# for all processes

def findWaitingTime(processes, n, bt,

                        wt, quantum):

    rem\_bt = [0] \* n

    # Copy the burst time into rt[]

    for i in range(n):

        rem\_bt[i] = bt[i]

    t = 0 # Current time

    # Keep traversing processes in round

    # robin manner until all of them are

    # not done.

    while(1):

        done = True

        # Traverse all processes one by

        # one repeatedly

        for i in range(n):

            # If burst time of a process is greater

            # than 0 then only need to process further

            if (rem\_bt[i] > 0) :

                done = False # There is a pending process

                if (rem\_bt[i] > quantum) :

                    # Increase the value of t i.e. shows

                    # how much time a process has been processed

                    t += quantum

                    # Decrease the burst\_time of current

                    # process by quantum

                    rem\_bt[i] -= quantum

                # If burst time is smaller than or equal

                # to quantum. Last cycle for this process

                else:

                    # Increase the value of t i.e. shows

                    # how much time a process has been processed

                    t = t + rem\_bt[i]

                    # Waiting time is current time minus

                    # time used by this process

                    wt[i] = t - bt[i]

                    # As the process gets fully executed

                    # make its remaining burst time = 0

                    rem\_bt[i] = 0

        # If all processes are done

        if (done == True):

            break

# Function to calculate turn around time

def findTurnAroundTime(processes, n, bt, wt, tat):

    # Calculating turnaround time

    for i in range(n):

        tat[i] = bt[i] + wt[i]

# Function to calculate average waiting

# and turn-around times.

def findavgTime(processes, n, bt, quantum):

    wt = [0] \* n

    tat = [0] \* n

    # Function to find waiting time

    # of all processes

    findWaitingTime(processes, n, bt,

                        wt, quantum)

    # Function to find turn around time

    # for all processes

    findTurnAroundTime(processes, n, bt,

                                wt, tat)

    # Display processes along with all details

    print("Processes Burst Time  Waiting",

                    "Time Turn-Around Time")

    total\_wt = 0

    total\_tat = 0

    for i in range(n):

        total\_wt = total\_wt + wt[i]

        total\_tat = total\_tat + tat[i]

        print(" ", i + 1, "\t\t", bt[i],

            "\t\t", wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f "%(total\_wt /n) )

    print("Average turn around time = %.5f "% (total\_tat / n))

# Driver code

if \_\_name\_\_ =="\_\_main\_\_":

    # Process id's

    proc = [1, 2, 3]

    n = 3

    # Burst time of all processes

    burst\_time = [10, 5, 8]

    # Time quantum

    quantum = 2;

    findavgTime(proc, n, burst\_time, quantum)

print("Time quantum given to each process : ",quantum)

**Task:**

* Run code for following process using round robin algorithm.
  + All jobs have a CPU burst of less then or at least 3
  + All jobs time quantum/slice should be 4
  + Total jobs would be 4

