Student Name: Pasula Prasad
Student ID email address: prasadpasula1752@gmail.com
Github Link: https://github.com/Prasadpasula1752/RoundRobin
Code:

```
#include<stdio.h>


int count,i,j,time,remain,flag=0,time_quantum;

int wait_time=0,turnaround_time=0,fa[10],fb[10],rt[10],sa[10],sb[10];

int f,s,p;

int ra[15],rb[15];

int fname[15],sname[15],rname[15];


int input(int *a, int *b,int *c)

{

   int n;

   printf("Enter Total Process:\t ");

   scanf("%d",&n);

   remain=n;

   for(count=0;count<n;count++)

   {

      printf("Enter the name: ");

      scanf("%d",&c[i]);

      printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

      scanf("%d",&a[count]);

      scanf("%d",&b[count]);

      printf("\n");

   }


   return n;

}


int round_robin()
```

```
{
    for(int i=0;i<p;i++)
    {
       rt[i]=rb[i];
    }
    remain=p;
    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
    for(time=0,count=0;remain!=0;)
    {
     if(rt[count]<=time_quantum && rt[count]>0)
     {
       time+=rt[count];
       rt[count]=0;
       flag=1;
     }
     else if(rt[count]>0)
     {
       rt[count]-=time_quantum;
       time+=time_quantum;
     }
     if(rt[count]==0 && flag==1)
     {
       remain--;
       printf("P[%d]\t|\t%d\t|\t%d\n",rname[count],time-ra[count],time-ra[count]-rb[count]);
       wait_time+=time-ra[count]-rb[count];
       turnaround_time+=time-ra[count];
       flag=0;
     }
     if(count==p-1)
       count=0;
     else if(ra[count+1]<=time)
```

```
        count++;
       else
        count=0;
     }
     printf("\nAverage Waiting Time= %f\n",wait_time*1.0/p);
     printf("Avg Turnaround Time = %f",turnaround_time*1.0/p);

     return 0;
}

int sort()
{
   int swap;
   char sw[15];
   for (i=0;i<p-1;i++)
   {
      for (j=0;j< p-i-1;j++)
      {
         if (ra[j] > ra[j+1])
         {
            swap = ra[j];
            ra[j]= ra[j+1];
            ra[j+1]= swap;

            swap = rb[j];
            rb[j]= rb[j+1];
            rb[j+1]= swap;

            swap = rname[j];
            rname[j]= rname[j+1];
            rname[j+1]= swap;
```

```
        }
      }
   }
}

int sortedMerge()
{
   int i = 0, j = 0, k = 0;
   while (i < f) {
      ra[k] = fa[i];
      rb[k] = fb[i];
      rname[k]=fname[i];
      i += 1;
      k += 1;
   }
   while (j < s) {
      ra[k] = sa[j];
      rb[k] = sb[j];
      rname[k] = sname[j];
      j += 1;
      k += 1;
   }
   p=k;
   sort();
}


int main()
{
   printf("Enter data for FACULTY\n");
   f=input(fa,fb,fname);
```

```
    printf("Enter data for STUDENT\n");
    s=input(sa,sb,sname);

    printf("Enter Time Quantum:\t");
    scanf("%d",&time_quantum);

    sortedMerge();

    round_robin();
}
```

1.**Description:**

This repository contains program used to implement following problem: Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty, where faculty queue is given a higher priority. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.

Solution Proposed for the problem : The given problem is based upon solving queries of persons of different classes i.e. Faculty and Students. Thus, these queries can be compared to different processes in terms of operating system where each process has its demands and needs resources and time for its execution. And this demands of processes are handled by the CPU. In the given scenario, Mr. Sudesh Sharma, Linux expert, can be considered as a CPU, who solves the queries of either Faculty or Student by allocating proper resources to their individual demands and processing them by allocating them time accordingly. Now, Mr. Sharma, wants to provide priority for each query based upon its class, as well as, he wants to dedicate a fixed amount of time to every request. Thus in Operating System, if we divide the requests into two separate queues i.e. Faculty and Student such that the first queue contains faculty queries has higher priority and the second contains student queries which has lower priority, then we can resolve the problem, by allocating them required resources based upon their priorities as done in the scheduling algorithm in operating systems.

This program contains the implementation of modified Round Robin algorithm and a custom Job Merger algorithm.

2.Algorithm:

ROUND ROBIN SCHEDULING ALGORITHM

• • We first have a queue where the processes are arranged in first come first serve order.

• • A quantum value is allocated to execute each process.

• • The first process is executed until the end of the quantum value. After this, an interrupt is generated and the state is saved.

- • The CPU then moves to the next process and the same method is followed.
- • Same steps are repeated till all the processes are over.

3.complexity: dynamic round-robin (DYRR) packet scheduling algorithm with high efficiency and good fairness. DYRR algorithm introduces dynamic round-robin concept, that is, the allowance given to each of the flows in a given round is not fixed, but is related with the number of bytes sent of this and other flows of the last round scheduling. The time complexity of the DYRR algorithm is O(1). Results from performance simulation analysis shows that DYRR algorithm can effectively smooth output burst, realize fair scheduling, and have a good time delay characteristic.

4.constraints: Here we use teacher queue and student queue which we merge in another queue.

5.code output: