RateMonotonic-Report

Shivaraj Thyagarajan Mudaliar

October 2020

Supervisor:
Marilyn Chetto

1 Goal

The aim of this project is to develop a software that stimulate the behaviour of a preemptive fixed priority Rate monotonic.In RM scheduler all task's deadline and period are equal.In this report we will study Data acquisition,schedulabilty analysis,simulation and display the task processed.

2 Rate Monotonic

- Rate monotonic scheduler
  Rate Monotonic (RM) sets priorities to tasks according to the rule: the smallerthe period the higher the priority.

- Advantages
  1.Work consuming
  2.Fixed task
  3.ONline

3 Data acquisition

In the beginning of scheduler part of the the user will give the inputs for the number of task **(Ti)** to be performed,computation time **(Ci)**, period **(Pi)** of each task. As we already know in RM scheduling algorithm there the deadline **(Di)** is equal to the period.
Function to Input number of task

```
int task_number(){
     int a;
        cout<<"Enter the total number of tasks"<<endl;
        cin>>a;
```

```
    return a;
}
```

The above given is a simple function taking the number of task from the user and returning it.

Function for computation time and period

```
computation_time(int a,int comp_time[], int period_time[],int table[][2],int rank_1[]){
//create my table
    for(int i = 0 ; i<a ;i++ ){
      cout <<"Computation time for task "<< i+1 <<endl;
      cin>>comp_time[i];
      cout <<"Period time for task "<< i+1 <<endl;
      cin>>period_time[i];
     }
```

The function above given takes the Period**(Pi)** and the computational value **(Ci)** from the user.A two simple array has been created to store both the values.

4 Schedulability Analysis

In this section we will concentrate on schedualabilty analysis of ratemontonic algorithm

- The priority task of this scheduler depends upon the task period **(Pi)**, smaller the period higher the priority of the task.
- A necessary condition test has been done only when the utilization factor is less than 1. A sufficient condition test is also been done to check if the if it is greater than utilization factor.Only when these two test pass we say that the task is schedualable.

$$\sum_{k=1}^{n} \frac{Ci}{Ti} \leq U = n\left(2^{1/n} - 1\right)$$

- The fraction of processor time spent in the execution of the task set it is called total necessary condition test(utilisation factor).Necessary condition test is to check whether the real time-systems can meet their deadlines or not.

5 Stimulation

In this section we will stimulate a periodic task to check our RM scheduler.We will take the given **WCET** and **Period** for the given following task given below in the table.

Task	WCET	Period
1	2	6
2	2	8
3	2	12

```
shiva@shiva-metal:~/Desktop/Shivaraj_ratemonotonic$ ./a.out
Enter the total number of tasks
3
Computation time for task 1
2
Period time for task 1
8
Computation time for task 2
2
Period time for task 2
6
Computation time for task 3
2
Period time for task 3
12
```

- Necessary condition test:

$$\sum_{k=1}^{n} \frac{Ci}{Ti}$$

$$\frac{2}{6} + \frac{2}{8} + \frac{2}{12} = 0.75 \tag{1}$$

The utilization factor of all the task is 0.75 which is less than 1.There is 25 percentage of ideal time which also means that there is 6 units of unused time instant remaining.

- Sufficient condition test:

$$U = n(2^{1/n} - 1)$$

(n is no of periodic task)

The **sufficient** test for the given periodic tasks is 0.77 which is **greater** than the **necessary**test, hence we can conclude that the task is feasible and scheduable with **ratemonotonic** scheduler.

```
  float utilization_factor(int task,int comp_time[], int period_time[]){
float cond=  0.0f;
float sum = 0.0f;
for (int i = 0; i < task; i++){
```

```
sum = sum + ((float)comp_time[i]/(float)period_time[i]);
//cout<<setprecision(3)<<sum;
}
div =(double)(1/task);
cond =(double)task*(pow(2,div)-1);
cout<<"The Sufficient condition test is = "<< cond<<endl;
if(sum<=cond){
     std::cout << "It is schedulable" << '\n';
}else{
   cout<<"Task did not pass sufficient condition test";
}
return sum;

}
```

The snippet given above is function which checks and displays both the **Sufficient condition** and **necessary condition test**.Computation time and period of the task have passed to the function.If the test condition are not satisfied the program is terminated.

```
utilization factor=0.75
The Sufficient condition test is = 0.779763
It is schedulable
```

- Hyperperiod:
  The hyperperiod of the task is calculated by the least common multiple(LCM) of all the periods of the task. The integer output that we get is the hyperperiod of the task.The hyperperiod for the above task set is **24**.

```
 int hyperperiod(int period_time[], int task){
//find max of the array
int lcm= 1;
while(1){
int i = 0;
   while(i<task){
     if (lcm%period_time[i] != 0) {
         lcm++;
         break;
       }
       i++;
   }
   if(i==task){
      break; }
   }
return lcm;
}
```

The above function is to calculate the hyperperiod, it takes the period and number of task in the argument.It loops till it calculates the LCM of all the task.

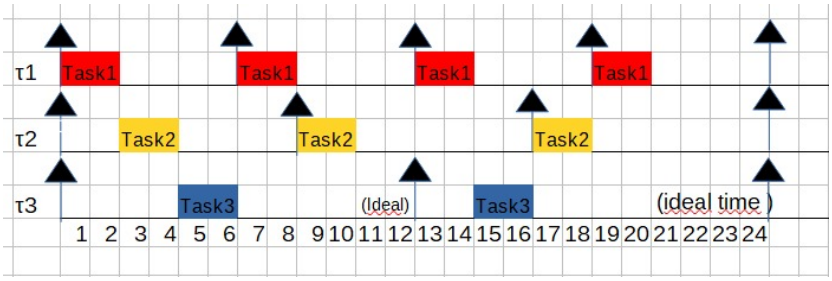
- Scheduler:
  A function **Scheduling** has been created to schedule the rate monotonic algorithm.The scheduler schedules the task to the first hyperperiod.
  The first loop in the given function resets the task completed status means every task jobs i set to 0 of each task period.A condition is also implemented to check if the highest priority task should start at current time, by checking if current time is a multiple of that task's period.Conditions also checks if the previous job that preempted by higher priority job, it will alter finish its execution if it does not misses its deadline.The computation time and execution time of each task is checked at every time instant to calcualte the ideal time of all the tasks.The following snippet given below if Scheduler function.The compelete Function definitons along with comments are apecified in the sorce code.

```
 void scheduling (int task, int priority[] int Remainingtime[]){
int currenttime = 0;
int endtime;


while(1){
  int count=0;
  int idletimectr=0;
  if(currenttime>=lcm){
  break;}//do scheduling only for first hyperperiod
  for (int i=0;i<task;i++){
      if((currenttime%priority[i][2])==0){
      Completionstatus[i]=0;
      Remainingtime[i]=0;
      }
    }

  if(currenttime%priority[0][2]==0){
    endtime=currenttime+priority[0][3];
    Completionstatus[0]=1;
    cout<<currenttime<<"-"<<endtime<<"Task"<<priority[0][1]<<endl;
    currenttime=endtime;
    Remainingtime[0] = priority[0][2]-priority[0][3];
    }

  for(int i=1;i<task;i++){
    if(Completionstatus[i]==0){
      if(Remainingtime[i] > 0){
```

```
      if(Remainingtime[i]<Remainingtime[0]){
        endtime = currenttime + Remainingtime[i];
        Completionstatus[i] = 1;
        }
      else{
        endtime = currenttime + Remainingtime[0];  //only time gap before highest p
      }
   }
     else{
        if(priority[i][3]<=Remainingtime[0]){
        endtime = (currenttime+priority[i][3]);
        Completionstatus[i] = 1;
        }
        else{//if not enough time to complete priority task i
          endtime=currenttime+Remainingtime[0];
          Remainingtime[i] = priority[i][3] - Remainingtime[0];
        }
      }
    cout<<currenttime<<"-"<<endtime<<"Task"<<priority[i][1]<<endl;
    currenttime=endtime;
    }
  }
for (int i=0;i<task;i++){
  idletimectr=idletimectr+Completionstatus[i];
}
  if (idletimectr==task){
    int start_time=currenttime;
    for(int i=0;i<task;i++){
        if(currenttime%priority[i][2]!=0){
            count++;
        }
      //cout<<count;
    if(count == task){


      while(1){
      int counter=0;
       for(int i=0;i<task;i++){
           if(currenttime%priority[i][2]!=0){
              counter++;
           }
       }
        if (counter==task){
            currenttime++;
        }
        else{
```

```
                    break;
                }
            }

            cout<<start_time<<"-"<<currenttime<<"Idle time"<<endl;
        }

    }
        }
    }
}
```

```
Computation time for task 1
2
Period time for task 1
6
Computation time for task 2
2
Period time for task 2
8
Computation time for task 3
2
Period time for task 3
12
utilization factor=0.75
The Sufficient condition test is = 0.779763
It is schedulable
Priority    Tasknumber        P[i]            C[i]
3             1             6             2
2             2             8             2
1             3             12             2
Hyperperiod is24
0-2Task1
2-4Task2
4-6Task3
6-8Task1
8-10Task2
10-12Idle time
12-14Task1
14-16Task3
16-18Task2
18-20Task1
20-24Idle time
```



6 Display main metrics

In this section we will discuss about the release time of each task, no of releases and average response of all the task.The function **displays results** and **averageresp time** computes all the metrics of all task.

- Release time:
  The release time of each task is calulated by ratio of hyperperiod of all task and period of each task.(i number of individual task)

$$N(i) = \frac{Hyperperiod}{Period(i} \tag{2}$$

- Finish time:
  The finish time is defined as the time instant at which each **job** of a particular **task** is calculated.

- Average response time:
  Average response time of task is ratio of difference between the **finish time** and **release time** of each task divided by total number of task.

$$Averageresponsetime = \frac{Hyperperiod - Period(i}{Ni} \tag{3}$$

```
The release times of task 1 are
0
6
12
18
the finish times of Task1 are
2
8
14
20
The number of releases,(N[i]), of task 2 = 3
The release times of task 2 are
0
8
16
the finish times of Task2 are
4
10
18
The number of releases,(N[i]), of task 3 = 2
The release times of task 3 are
0
12
the finish times of Task3 are
6
16
the average response times of Tasks 1 to 3 are
2
2.66667
5
respectively
shiva@shiva-metal:~/Desktop/Shivaraj_ratemonotonic$
```

7 Example 1

Task	WCET	Period
1	4	2
2	8	6
3	5	7

```
shiva@shiva-metal:~/Desktop/Shivaraj_ratemonotonic$ ./a.out
Enter the total number of tasks
3
Computation time for task 1
4
Period time for task 1
2
Computation time for task 2
5
Period time for task 2
7
Computation time for task 3
8
Period time for task 3
6
utilization factor=4.04762
The Sufficient condition test is = 0.779763
Task did not pass sufficient condition test
Priority   Tasknumber       P[i]           C[i]
3             1             2             4
2             3             6             8
1             2             7             5
shiva@shiva-metal:~/Desktop/Shivaraj_ratemonotonic$
```

8 Example 2

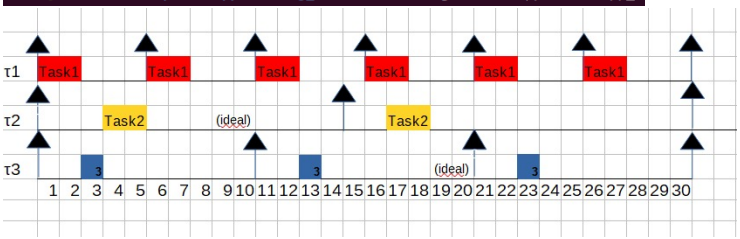
Task	WCET	Period
1	2	5
2	2	15
3	1	10

```
shiva@shiva-metal:~/Desktop/Shivaraj_ratemonotonic$ ./a.out
Enter the total number of tasks
3
Computation time for task 1
2
Period time for task 1
5
Computation time for task 2
2
Period time for task 2
15
Computation time for task 3
1
Period time for task 3
10
utilization factor=0.633333
The Sufficient condition test is = 0.779763
It is schedulable
```

```
Priority    Tasknumber        P[i]            C[i]
3              1              5              2
2              3              10              1
1              2              15              2
Hyperperiod is30
0-2Task1
2-3Task3
3-5Task2
5-7Task1
7-10Idle time
10-12Task1
12-13Task3
13-15Idle time
15-17Task1
17-19Task2
19-20Idle time
20-22Task1
22-23Task3
23-25Idle time
25-27Task1
27-30Idle time
```

```
The number of releases,(N[i]), of task 1 = 6
The release times of task 1 are
0
5
10
15
20
25
the finish times of Task1 are
2
7
12
17
22
27
The number of releases,(N[i]), of task 2 = 2
The release times of task 2 are
0
15
the finish times of Task2 are
3
13
The number of releases,(N[i]), of task 3 = 3
The release times of task 3 are
0
10
20
the finish times of Task3 are
5
```

```
the average response times of Tasks 1 to 3 are
2
0.5
4.66667
respectively
shiva@shiva-metal:~/Desktop/Shivaraj_ratemonotonic$ g++ main.cpp rmsch.cpp
```



9 Conclusion

Thus the ratemonotonic scheduler has been programmed successfully on **cpp** language

- Data Acquisition:The scheduler takes the input from the user in the function **computation time**. The priority of each is calculated from function **prioritization**
- Schedualabilty test: The necessary condition and sufficient condition test is checked in the function **utilization factor**,if the task pass the test the task set is schedulable.
- Simulation: Once the task set pass the test the **scheduling** function computes the task with its respective WCET and period.
- Display metrics: The ideal time, number of relases and average response time of each task is computed in the function **display metrics**

10 Command

cd shivaraj_ratemonotonic
g++ main.cpp rmsch.cpp
./a.out