Brian Guzman

ECE 480: Embedded Computing

Lab 8: Rate Monotonic Analysis (RMA)

The purpose of this program was to perform rate monotonic analysis (RMA) with overhead and blocking. The given value of overhead was 153us (.153ms) which was added to each task being scheduled. The algorithm was to be run on the tasks having the periods shown in the table below:

Feature	Period (ms)
Compute attitude data	10.56
Compute velocity data	40.96
Comput position data	350.00
Display data	100.00
Run-time Built-in Test (BIT)	285.00
Compose attitude message	61.44
Compose navigation message	165.00
Compose test message	700.00

Table 1: Tasks and Periods to be scheduled

These eight tasks are to run on a platform using a Motorola MC68302 microcontroller and a linux real time kernel. As shown in the tables below, each task uses different amounts of resources. To figure out the total blocking of a task, the first thing to do is give each task a priority. This is done by giving the task with the lowest period highest priority of one (1).

Feature	Period (ms)	Run time (ms)	Priority
Compute attitude data	10.56	1.30	1
Compute velocity data	40.96	4.70	2
Compose attitude message	61.44	9.00	3
Display data	100.00	23.00	4
Compose navigation message	165.00	38.30	5
Run-time Built-in Test (BIT)	285.00	10.00	6
Comput position data	350.00	3.00	7
Compose test message	700.00	2.00	8

Table 2: Prioritized tasks list and run times

There are three different resources the eight tasks will be trying to use, the following tables show the usage of each resource and the total blocking that will be used in the RMA algorithm.

	Result Table Usage			
Feature	**Time using resource (ms)**	**Direct (ms)**	**Push through (ms)**	**Max blocking (ms)**
Compute attitude data	0.20	0.30	0.00	0.30
Compute velocity data	0.20	0.30	0.30	0.30
Compose attitude message	-	0.00	0.30	0.30
Display data	0.30	0.20	0.20	0.20
Compose navigation message	-	0.00	0.20	0.20
Run-time Built-in Test (BIT)	-	0.00	0.20	0.20
Comput position data	0.2	0.00	0.00	0.00
Compose test message	-	0.00	0.00	0.00

Table 3: Result Table Usage

	I/O Channel Usage				Disk Usage				
Feature	**Time using resource (ms)**	**Direct (ms)**	**Push through (ms)**	**Max blocking (ms)**	**Time using resource (ms)**	**Direct (ms)**	**Push through (ms)**	**Max blocking (ms)**	**Total Blocking (ms)**
Compute attitude data	-	0.00	0.00	0.00	2.00	3.00	0.00	3.00	3.30
Compute velocity data	-	0.00	6.00	0.00	3.00	3.00	3.00	3.00	3.30
Compose attitude message	3.00	6.00	2.00	6.00	-	0.00	3.00	3.00	9.30
Display data	-	0.00	6.00	6.00	-	0.00	3.00	3.00	9.20
Compose navigation message	-	0.00	2.00	2.00	-	0.00	3.00	3.00	5.20
Run-time Built-in Test (BIT)	6.00	2.00	0.00	2.00	1.00	3.00	3.00	3.00	5.20
Comput position data	-	0.00	2.00	2.00	3.00	0.00	0.00	0.00	2.00
Compose test message	2.00	0.00	0.00	0.00	-	0.00	0.00	0.00	0.00

Table 4: I/O Channel, Disk Usage and Total blocking

By adding up the max blocking for each of the resources, the total blocking can be calculated which can then be used by the RMA algorithm to determine the values of *l* and *k* that would pass the theorem. The following figure shows the output of the RMA algorithm:

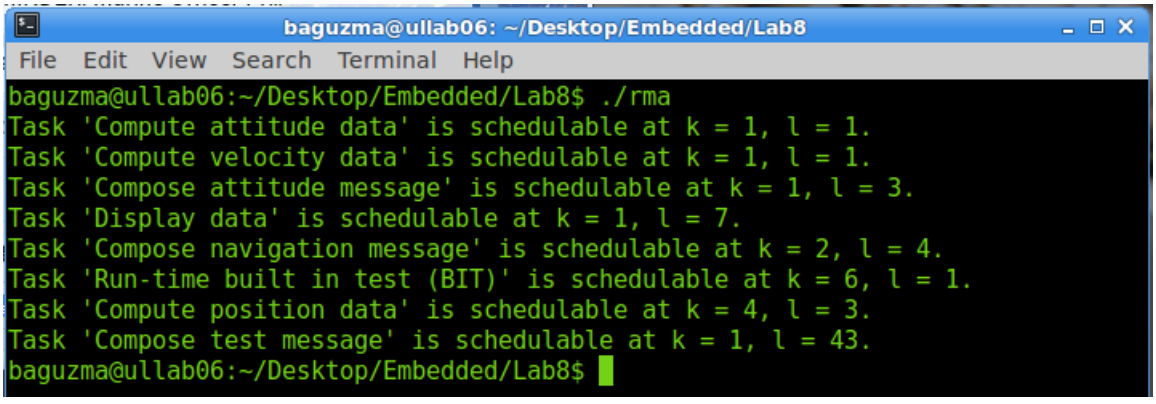

Figure 1: RMA algorithm results

```c
// Brian Guzman
// ECE 4680
// Spring 2019
// Lab 8: RMA

#include <stdio.h>
#include <math.h>


int main()
{
	char *task_names[8];
	double period[8];
	double runtime[8];
	double totb[8];
	int i,l,k;
	double sum;

	//tasks
	task_names[0] = "Compute attitude data";
	task_names[1] = "Compute velocity data";
	task_names[2] = "Compose attitude message";
	task_names[3] = "Display data";
	task_names[4] = "Compose navigation message";
	task_names[5] = "Run-time built in test (BIT)";
	task_names[6] = "Compute position data";
	task_names[7] = "Compose test message";

	//period [ms]
	period[0] = 10.56;
	period[1] = 40.96;
	period[2] = 61.44;
	period[3] = 100.00;
	period[4] = 165.00;
	period[5] = 285.00;
	period[6] = 350.00;
	period[7] = 700.00;

	//runtimes [ms]
	runtime[0] = 1.30;
	runtime[1] = 4.70;
	runtime[2] = 9.00;
	runtime[3] = 23.00;
	runtime[4] = 38.3;
	runtime[5] = 10.00;
	runtime[6] = 3.00;
	runtime[7] = 2.00;

	//total blocking [ms]
	totb[0] = 3.30;
	totb[1] = 3.30;
	totb[2] = 9.30;
	totb[3] = 9.20;
	totb[4] = 5.20;
	totb[5] = 5.20;
	totb[6] = 2.00;
	totb[7] = 0.00;
```

Source Code:

```
#include <stdio.h>
#include <math.h>


int main()
{
	char *task_names[8];
	double period[8];
	double runtime[8];
	double totb[8];
	int i,l,k;
	double sum;

	//tasks
	task_names[0] = "Compute attitude data";
	task_names[1] = "Compute velocity data";
	task_names[2] = "Compose attitude message";
	task_names[3] = "Display data";
	task_names[4] = "Compose navigation message";
	task_names[5] = "Run-time built in test (BIT)";
	task_names[6] = "Compute position data";
	task_names[7] = "Compose test message";

	//period [ms]
	period[0] = 10.56;
	period[1] = 40.96;
	period[2] = 61.44;
	period[3] = 100.00;
	period[4] = 165.00;
	period[5] = 285.00;
	period[6] = 350.00;
	period[7] = 700.00;

	//runtimes [ms]
	runtime[0] = 1.30;
	runtime[1] = 4.70;
	runtime[2] = 9.00;
	runtime[3] = 23.00;
	runtime[4] = 38.3;
	runtime[5] = 10.00;
	runtime[6] = 3.00;
	runtime[7] = 2.00;

	//total blocking [ms]
	totb[0] = 3.30;
	totb[1] = 3.30;
	totb[2] = 9.30;
	totb[3] = 9.20;
	totb[4] = 5.20;
	totb[5] = 5.20;
	totb[6] = 2.00;
	totb[7] = 0.00;

	int count = 0;
	for(i = 0; i < 8; i++)
	{
```

```
		for(k = 0; k < i+1; k++)
		{
			for(l = 0; l < floor(period[i]/period[k]); l++)
			{
				count = 0;
				sum = 0; //reset sum and count


				while(count < i)
				{
					sum += (runtime[count] + 0.153) * ceil(((l +
1)*period[k]) / period[count]);
					count++;
				}

				sum += runtime[i] + 0.153 + totb[i];

				if (sum < period[k]*(l+1))
				{
					printf("Task '%s' is schedulable at k = %d, l =
%d.\n",task_names[i],(k+1),(l+1));
					break;
				}
			}
			if(l < floor(period[i]/period[k]))
			{
				break;
			}
		}
		if(k == (i+1))
		{
			printf("Task '%s' is not schedulable\n",task_names[i]);
		}
	}
	return 0;
}
```