ECE 4680: Laboratory 8

LAB8: REAL TIME SCHEDULING USING RMA

April 23, 2019

Chintan Patel
Clemson University
Department of Electrical and Computer Engineering
cdpatel@clemson.edu

Abstract

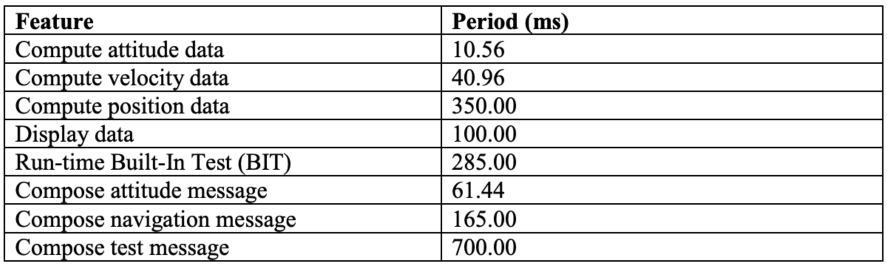
Lab 8 involved implemented a real-time scheduler using the RMA algorithm for an Inertial Navigation System. An INS is a real-time shipboard avionic system. It has strict time constraints for providing information to other shipboard devices. Typically an INS system tracks attitude, geographic position, velocity, distance and displacement. It also responds to periodic test messages sent by an external computer for checking of communication links. The student had to write a program to perform RMA with overhead blocking.

1 Implementation

The table in Figure 1 below shows the timing constraints that were required to be complied. There were eight tasks in total. All tasks shared the result table (write mode for computational tasks, read mode for all others). The attitude, navigation and test message composition tasks share the I/O channel and these messages are stored on a disk.

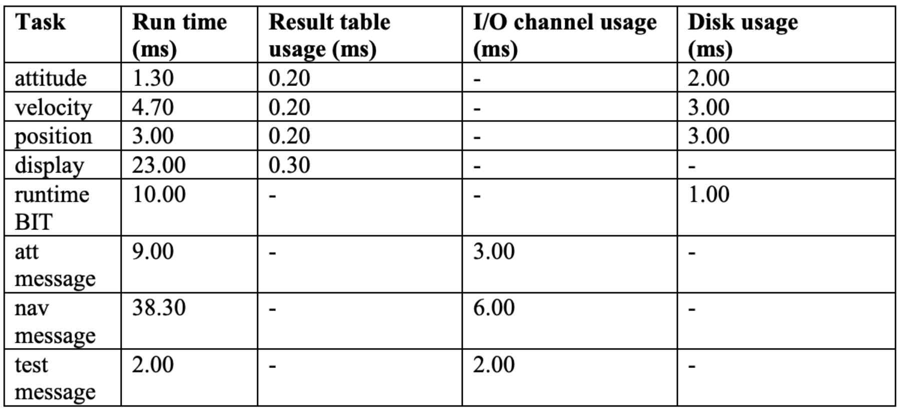
The system ran on a platform using a Motorola MC68302 microcontroller and a linux real-time kernel, which offered a priority ceiling protocol. The overhead for this system was 153 microseconds per task. The estimated execution times and resource usage times for each of the tasks can be seen in the table in Figure 2 below.

The data from Figure 1 and 2 was consolidated into the table in Figure 3 below. This data was then used to calculate the blocking values that were used in the code. The tables in Figures 4-6 show the calculated blocking values that were used to calculate the total blocking data for each task. The table in Figure 7 below shows the total blocking data used in the code for RMA algorithm. Figures 8-9 shows the RMA algorithm code that was written to create the real-time scheduler.



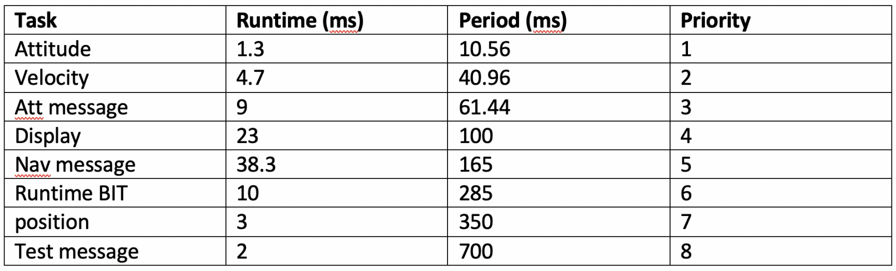
Feature	Period (ms)
Compute attitude data	10.56
Compute velocity data	40.96
Compute position data	350.00
Display data	100.00
Run-time Built-In Test (BIT)	285.00
Compose attitude message	61.44
Compose navigation message	165.00
Compose test message	700.00

Figure 1: System Timing Constraints



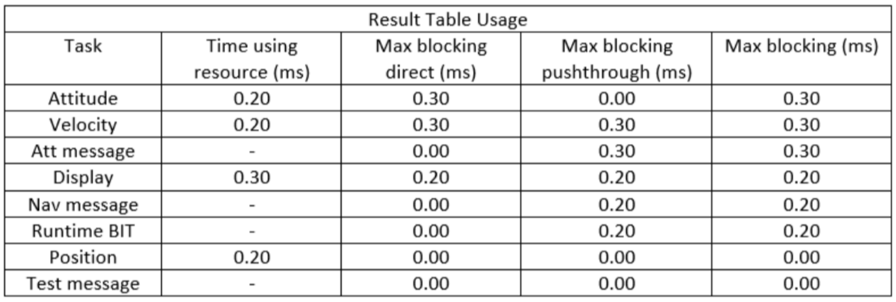
Task	Run time (ms)	Result table usage (ms)	I/O channel usage (ms)	Disk usage (ms)
attitude	1.30	0.20	-	2.00
velocity	4.70	0.20	-	3.00
position	3.00	0.20	-	3.00
display	23.00	0.30	-	-
runtime BIT	10.00	-	-	1.00
att message	9.00	-	3.00	-
nav message	38.30	-	6.00	-
test message	2.00	-	2.00	-

Figure 2: Estimated Execution and Resource Usage Times



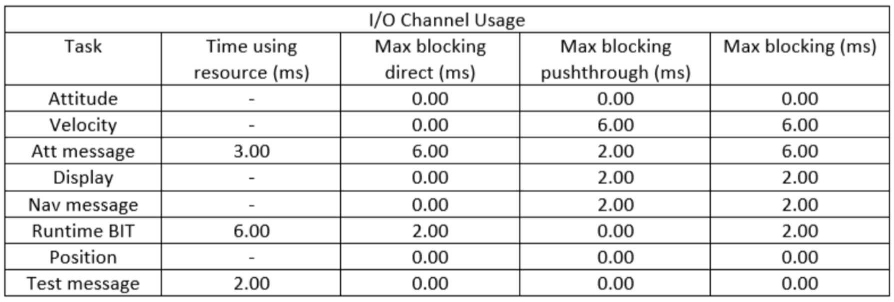
Task	Runtime (ms)	Period (ms)	Priority
Attitude	1.3	10.56	1
Velocity	4.7	40.96	2
Att message	9	61.44	3
Display	23	100	4
Nav message	38.3	165	5
Runtime BIT	10	285	6
position	3	350	7
Test message	2	700	8

Figure 3: Cosolidated task information



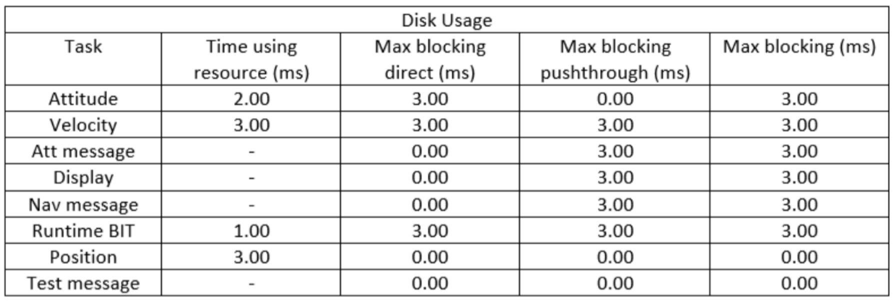
Result Table Usage				
Task	Time using resource (ms)	Max blocking direct (ms)	Max blocking pushthrough (ms)	Max blocking (ms)
Attitude	0.20	0.30	0.00	0.30
Velocity	0.20	0.30	0.30	0.30
Att message	-	0.00	0.30	0.30
Display	0.30	0.20	0.20	0.20
Nav message	-	0.00	0.20	0.20
Runtime BIT	-	0.00	0.20	0.20
Position	0.20	0.00	0.00	0.00
Test message	-	0.00	0.00	0.00

Figure 4: Result Table Usage



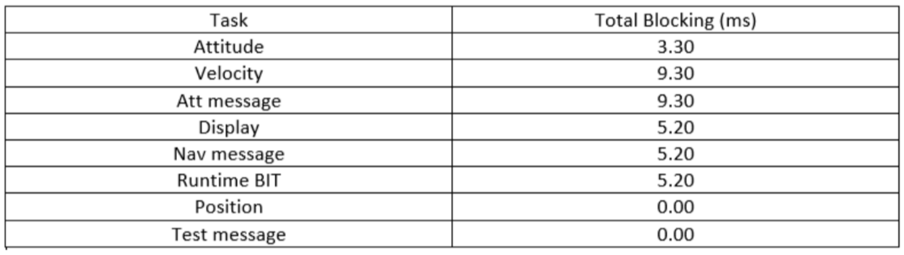
I/O Channel Usage				
Task	Time using resource (ms)	Max blocking direct (ms)	Max blocking pushthrough (ms)	Max blocking (ms)
Attitude	-	0.00	0.00	0.00
Velocity	-	0.00	6.00	6.00
Att message	3.00	6.00	2.00	6.00
Display	-	0.00	2.00	2.00
Nav message	-	0.00	2.00	2.00
Runtime BIT	6.00	2.00	0.00	2.00
Position	-	0.00	0.00	0.00
Test message	2.00	0.00	0.00	0.00

Figure 5: I/O Channel Usage



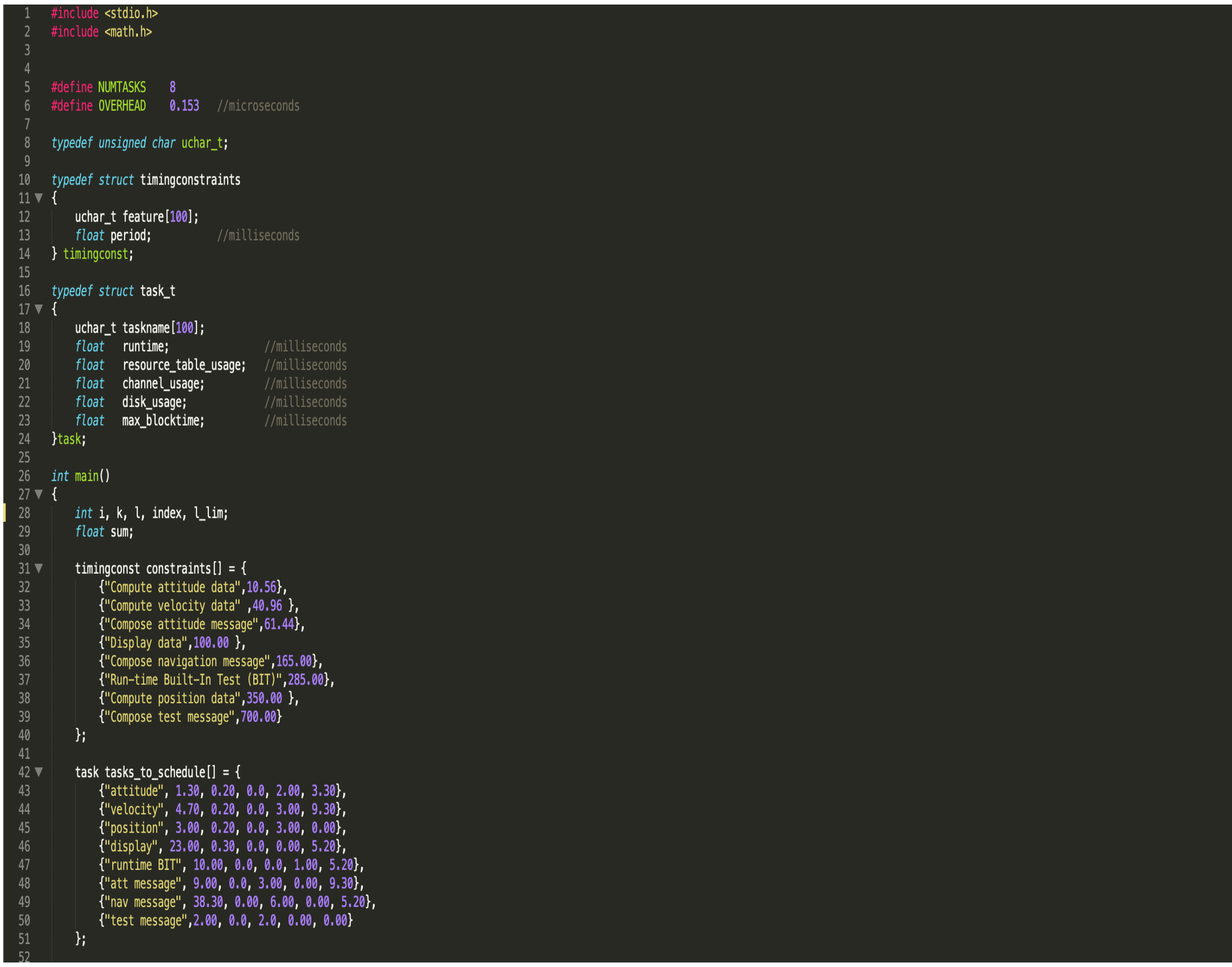
Disk Usage				
Task	Time using resource (ms)	Max blocking direct (ms)	Max blocking pushthrough (ms)	Max blocking (ms)
Attitude	2.00	3.00	0.00	3.00
Velocity	3.00	3.00	3.00	3.00
Att message	-	0.00	3.00	3.00
Display	-	0.00	3.00	3.00
Nav message	-	0.00	3.00	3.00
Runtime BIT	1.00	3.00	3.00	3.00
Position	3.00	0.00	0.00	0.00
Test message	-	0.00	0.00	0.00

Figure 6: Disk Usage



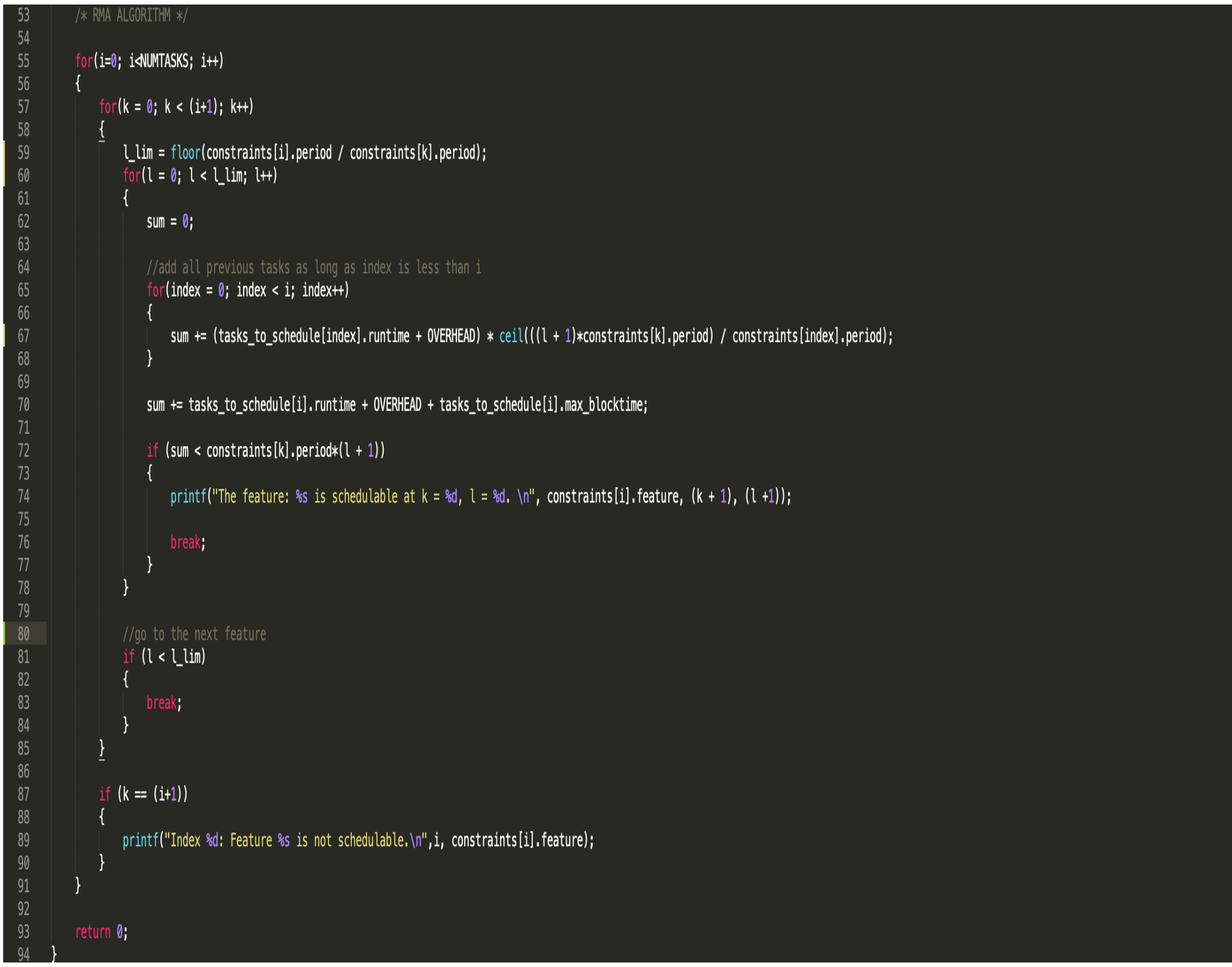
Task	Total Blocking (ms)
Attitude	3.30
Velocity	9.30
Att message	9.30
Display	5.20
Nav message	5.20
Runtime BIT	5.20
Position	0.00
Test message	0.00

Figure 7: Total Blocking Data



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


#define NUMTASKS    8
#define OVERHEAD    0.153   //microseconds

typedef unsigned char uchar_t;

typedef struct timingconstraints
{
    uchar_t feature[100];
    float period;           //milliseconds
} timingconst;

typedef struct task_t
{
    uchar_t taskname[100];
    float   runtime;                //milliseconds
    float   resource_table_usage;   //milliseconds
    float   channel_usage;          //milliseconds
    float   disk_usage;             //milliseconds
    float   max_blocktime;          //milliseconds
}task;

int main()
{
    int i, k, l, index, l_lim;
    float sum;

    timingconst constraints[] = {
        {"Compute attitude data",10.56},
        {"Compute velocity data" ,40.96 },
        {"Compose attitude message",61.44},
        {"Display data",100.00 },
        {"Compose navigation message",165.00},
        {"Run-time Built-In Test (BIT)",285.00},
        {"Compute position data",350.00 },
        {"Compose test message",700.00}
    };

    task tasks_to_schedule[] = {
        {"attitude", 1.30, 0.20, 0.0, 2.00, 3.30},
        {"velocity", 4.70, 0.20, 0.0, 3.00, 9.30},
        {"position", 3.00, 0.20, 0.0, 3.00, 0.00},
        {"display", 23.00, 0.30, 0.0, 0.00, 5.20},
        {"runtime BIT", 10.00, 0.0, 0.0, 1.00, 5.20},
        {"att message", 9.00, 0.0, 3.00, 0.00, 9.30},
        {"nav message", 38.30, 0.00, 6.00, 0.00, 5.20},
        {"test message",2.00, 0.0, 2.0, 0.00, 0.00}
    };

```

Figure 8: Code snippet

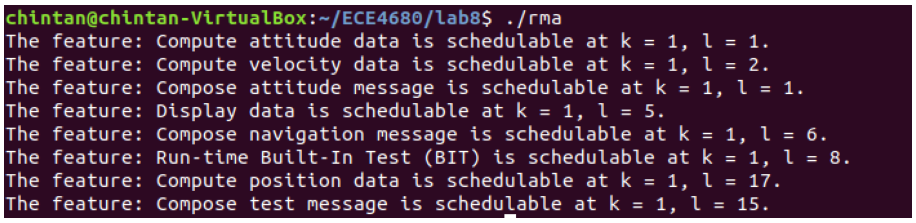


```c
    /* RMA ALGORITHM */

    for(i=0; i<NUMTASKS; i++)
    {
        for(k = 0; k < (i+1); k++)
        {
            l_lim = floor(constraints[i].period / constraints[k].period);
            for(l = 0; l < l_lim; l++)
            {
                sum = 0;

                //add all previous tasks as long as index is less than i
                for(index = 0; index < i; index++)
                {
                    sum += (tasks_to_schedule[index].runtime + OVERHEAD) * ceil(((l + 1)*constraints[k].period) / constraints[index].period);
                }

                sum += tasks_to_schedule[i].runtime + OVERHEAD + tasks_to_schedule[i].max_blocktime;

                if (sum < constraints[k].period*(l + 1))
                {
                    printf("The feature: %s is schedulable at k = %d, l = %d. \n", constraints[i].feature, (k + 1), (l +1));

                    break;
                }
            }

            //go to the next feature
            if (l < l_lim)
            {
                break;
            }
        }

        if (k == (i+1))
        {
            printf("Index %d: Feature %s is not schedulable.\n",i, constraints[i].feature);
        }
    }

    return 0;
}
```

Figure 9: Code snipped

2 Results

The RMA algorithm code from Figures 8-9 above was ran to find the values of ***l*** and ***k*** for which the theorem passed. Figure 10 below shows the output from running the code.



```
chintan@chintan-VirtualBox:~/ECE4680/lab8$ ./rma
The feature: Compute attitude data is schedulable at k = 1, l = 1.
The feature: Compute velocity data is schedulable at k = 1, l = 2.
The feature: Compose attitude message is schedulable at k = 1, l = 1.
The feature: Display data is schedulable at k = 1, l = 5.
The feature: Compose navigation message is schedulable at k = 1, l = 6.
The feature: Run-time Built-In Test (BIT) is schedulable at k = 1, l = 8.
The feature: Compute position data is schedulable at k = 1, l = 17.
The feature: Compose test message is schedulable at k = 1, l = 15.
```

Figure 10: RMA Algorithm output