RATE MONOTONIC ANALYSIS

LAB 6 REPORT FOR ECE468

EMBEDDED COMPUTING

SUBMITTED BY

JULIAN COY

UNDERGRADUATE OF ELECTRICAL & COMPUTER ENGINEERING

CLEMSON UNIVERSITY

APRIL 22, 2014

Abstract

An Inertial Navigation System (INS) is a real-time shipboard avionic system. It has strict time constraints for providing information to other shipboard devices. For example, an INS tracks attitude, geographic position, velocity, distance and displacement. The goal of this experiment is to use Rate Monotonic Analysis do determine if the system is schedulable on a Motorola MC68302 microcontroller that is implementing a priority ceiling protocol.

INTRODUCTION

There are specific timing constraints that are in place for each subtask in the system. Figure 1.1 shows the time constraints of specific features. The period is the interval between task completions. In essence, this number lets us know when a specific task needs to complete execution.

Feature	**Period (ms)**
Compute attitude data	2.56
Compute velocity data	40.96
Compute position data	1,280.00
Display data	1,000.00
Compose attitude message	61.44
Compose navigation message	1,024.00

Figure 1.1: Timing Constraints

When switching tasks, the system will incur and overhead delay of 153μs. Below in Figure 1.2, you can see the specific resource usage required by each task.

Task	**Run time (ms)**	**Result table usage (ms)**	**I/O channel usage (ms)**
attitude	1.30	0.20	-
velocity	4.70	0.20	-
position	3.00	0.20	-
display	23.00	0.30	-
att message	9.00	0.15	3.00
nav message	38.30	0.30	6.00

Figure 1.2: Resource Usage

Each task will share the same result table, and two will share an I/O channel. These resources will need to be modeled as semaphorically protected for a proper RMA analysis. Due to the semaphores that will be present, it becomes necessary to analyze each task with regard to its maximum blocking time. This time is calculated from the large of two statistics: direct blocking and pass-through blocking. Direct blocking occurs when a lower priority task holds a needed resource. Pass-through blocking occurs when a medium priority task is blocked by a lower priority task that has inherited a higher priority, due to a share resource constraint.

EXPERIMENT

To test the schedulability of this system, a C program was written to perform RMA analysis upon the data sets provided. The C program followed the rate monotonic analysis formulas provided by Dr. Hoover in class.

RESULTS

The system is schedulable with k = 1 and l = 117. The following output is the verbose results of the program when supplied with the data provided earlier. The blocking values, and other information were hard coded into the program. See Code section for the program source.

Output

```
Testing for possible scheduling...


Pass for i: 1
  Fail with i: 2 k: 1 l: 1
  Fail with i: 2 k: 1 l: 2
  Fail with i: 2 k: 1 l: 3
Pass for i: 2
  Fail with i: 3 k: 1 l: 1
  Fail with i: 3 k: 1 l: 2
  ...
  Fail with i: 3 k: 1 l: 14
  Fail with i: 3 k: 1 l: 15
Pass for i: 3
  Fail with i: 4 k: 1 l: 1
  Fail with i: 4 k: 1 l: 2
  ...
  Fail with i: 4 k: 1 l: 58
  Fail with i: 4 k: 1 l: 59
Pass for i: 4
  Fail with i: 5 k: 1 l: 1
  Fail with i: 5 k: 1 l: 2
  ...
```

```
  Fail with i: 5 k: 1 l: 114
  Fail with i: 5 k: 1 l: 115
Pass for i: 5
  Fail with i: 6 k: 1 l: 1
  Fail with i: 6 k: 1 l: 2
  ...
  Fail with i: 6 k: 1 l: 115
  Fail with i: 6 k: 1 l: 116
Pass for i: 6

System is schedulable with k: 1 and l: 117.
```

Lab 6: Code

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>

#define OVERHEAD 0.153

typedef struct _task {
  double runtime, period;
  double semaphore[2];
  double max_blocking;
} TASK;

void
printErr ( char *, int );

void
RMA ( TASK [], int, bool );
```

```
int
main ( int argc, char *argv[] )
{
  ////////////////////////
  /// SET DEFAULT VALUES ///
  ////////////////////////

  bool VERBOSE = false;

  ////////////////////////
  /// VERIFY/PARSE INPUT ///
  ////////////////////////

  int i;

  for (i = 1; i < argc; i++)
  {
    if (strcmp(argv[i], "-v") == 0)
      VERBOSE = true;
    else
      printErr(argv[i], EINVAL);
  }

  /* ////////////|\\\\\\\\\\\\\ */
  /* |||     DO STUFFS     ||| */
  /* \\\\\\\\\\\\|///////////// */


  // Build a task set
  TASK task[6];

  task[0].runtime    = 1.3;
  task[0].period     = 2.56;
  task[0].semaphore[0] = 0.2;
  task[0].semaphore[0] = 0.0;
  task[0].max_blocking = 0.3;
```

```
  task[1].runtime    = 4.7;
  task[1].period     = 40.96;
  task[1].semaphore[0] = 0.2;
  task[1].semaphore[0] = 0.0;
  task[1].max_blocking = 0.3;

  task[2].runtime    = 9.0;
  task[2].period     = 61.44;
  task[2].semaphore[0] = 0.15;
  task[2].semaphore[0] = 3.00;
  task[2].max_blocking = 6.00;

  task[3].runtime    = 23.0;
  task[3].period     = 1000.0;
  task[3].semaphore[0] = 0.30;
  task[3].semaphore[0] = 0.0;
  task[3].max_blocking = 6.00;

  task[4].runtime    = 38.3;
  task[4].period     = 1024.0;
  task[4].semaphore[0] = 0.30;
  task[4].semaphore[0] = 6.00;
  task[4].max_blocking = 0.2;

  task[5].runtime    = 3.0;
  task[5].period     = 1280.0;
  task[5].semaphore[0] = 0.2;
  task[5].semaphore[0] = 0.0;
  task[5].max_blocking = 0.0;

  RMA( task, 6, VERBOSE );

  exit(0);
}
```

```
//function definitions

void
RMA( TASK task[], int count, bool VERBOSE )
{
  int i, j, k, l;
  int storedk, storedl;

  double LHS, RHS, inner;

  bool SCHEDULABLE;

  if ( VERBOSE ) printf( "Testing for possible scheduling...\n\n" );

  for ( i = 1; i <= count; i++ )
  // i iterates from 1 to n
  {
    SCHEDULABLE = false;
    for ( k = 1; k <= i; k++ )
    // k iterates from 1 to i
    {
      for ( l = 1; l <= (int) floor( task[i-1].period / task[k-1].period ); l++ )
      // k iterates from 1 to floor of Ti / Tk
      {
        // setup RHS and LHS for maths
        LHS = 0.0; // clear for loops
        RHS = (l) * task[k-1].period; // l * Tk

        // if ( VERBOSE ) printf( "\t\tLHS = " );

        // peform summation loop
        for ( j = 1; j < i; j++ )
        // sum from j=1 to i-1
        {
          inner = ( task[j-1].runtime + OVERHEAD );
          inner *= (int) ceil( l * task[k-1].period / task[j-1].period );
```

```
        LHS += inner;
      }

      // add in runtimes and blocking
      LHS = LHS + ( task[i-1].runtime + OVERHEAD + task[i-1].max_blocking );

      if ( LHS <= RHS ) {
        // set conditions to break two inner loops and continue i
        storedl = l;
        storedk = k;
        l = (int) floor( task[i-1].period / task[k-1].period ) + 1;
        k = i + 1;
        if ( VERBOSE ) printf( "Pass for i: %d\n", i );
        SCHEDULABLE = true;
      } else {
        if ( VERBOSE ) printf( "\tFail with i: %d\tk: %d\tl: %d\n", i, k, l );
      }

    } // end l
  } // end k
  if ( !SCHEDULABLE )
  {
    if ( VERBOSE ) printf( "\n" );
    printf ( "System is unschedulable at i: %d.\n", i );
    if ( VERBOSE ) printf( "\n" );
    exit(1);
  }
 } // end i

 if ( SCHEDULABLE )
 {
  if ( VERBOSE ) printf( "\n" );
  printf( "System is schedulable with k: %d and l: %d.\n", storedk, storedl );
  if ( VERBOSE ) printf( "\n" );
 }
}
```

```
void
printErr(char *str, int err)
{
  errno     = err;
  char *msg = (char *)"Error:\n  ";
  char *pstr = NULL;
  pstr      = (char *)malloc(strlen(str) + strlen(msg));

  strcat(pstr, msg);
  strcat(pstr, str);
  perror(pstr);

  exit(err);
}
```
