

feasibility_tests.c

```

1  /
  *****
2  * Copyright (C) 2023 by Sam Siewert, Parth Thakkar
3  *
4  * Redistribution, modification or use of this software in source or binary
5  * forms is permitted as long as the files maintain this copyright. Users are
6  * permitted to modify this and use it to learn about the field of embedded
7  * software. Parth Thakkar and the University of Colorado are not liable for
8  * any misuse of this material.
9  *
  *****/
10
11 /**
12  * @file    feasibility_tests.c
13  * @brief   This example code provides feasibility decision tests for single
  core fixed
14  * priority rate monotonic systems only (not dynamic priority such as deadline
15  * driven EDF and LLF). These are standard algorithms which either estimate
16  * feasibility (as the RM LUB does) or automate exact analysis (scheduling
  point,
17  * completion test) for a set of services sharing one CPU core. This can be
  emulated on Linux SMP
18  * multi-core systems by use of POSIX thread affinity, to "pin" a thread to a
19  * specific core. Coded based upon standard definition of:
20  * 1) RM LUB based upon model by Liu and Layland
21  * 2) Scheduling Point - an exact feasibility algorithm based upon Lehoczky,
  Sha, and Ding exact analysis
22  * 3) Completion Test - an exact feasibility algorithm
23  *
24  *
25  * @author   Parth Thakkar, Sam Siewert
26  * @date     20th Sept 2023
27  *
28  */
29
30 #include <math.h>
31 #include <stdio.h>
32 #include <stdbool.h>
33 #include <stdlib.h>
34
35 #define TRUE 1
36 #define FALSE 0
37
38 #define U32_T unsigned int
39 #define EXAMPLES 10
40
41 #define MAX_TASKS 4
42
43 typedef struct
44 {
45     U32_T wcet[MAX_TASKS];
46     U32_T period[MAX_TASKS];
47     U32_T deadline[MAX_TASKS];
48     U32_T num_tasks;
49 } task_set_t;
50
51 // Example tasks

```

```
52
53 task_set_t tasks[EXAMPLES] = {
54     {
55         // 0
56         .period = {2, 10, 15},    // Example Periods
57         .wcet = {1, 1, 2},        // Example Worst Case Execution Times
58         .deadline = {2, 10, 15},  // Example Deadlines
59         .num_tasks = 3            // Number of tasks in this set
60     },
61     {
62         // 1
63         .period = {2, 5, 7},      // Example Periods
64         .wcet = {1, 1, 2},        // Example Worst Case Execution Times
65         .deadline = {2, 5, 7},    // Example Deadlines
66         .num_tasks = 3            // Number of tasks in this set
67     },
68     {
69         // 2
70         .period = {2, 5, 7, 13},  // Example Periods
71         .wcet = {1, 1, 1, 2},     // Example Worst Case Execution Times
72         .deadline = {2, 5, 7, 13}, // Example Deadlines
73         .num_tasks = 4            // Number of tasks in this set
74     },
75     {
76         // 3
77         .period = {3, 5, 15},     // Example Periods
78         .wcet = {1, 2, 3},        // Example Worst Case Execution Times
79         .deadline = {3, 5, 15},   // Example Deadlines
80         .num_tasks = 3            // Number of tasks in this set
81     },
82     {
83         // 4
84         .period = {2, 4, 16},     // Example Periods
85         .wcet = {1, 1, 4},        // Example Worst Case Execution Times
86         .deadline = {2, 4, 16},   // Example Deadlines
87         .num_tasks = 3            // Number of tasks in this set
88     },
89     {
90         // 5
91         .period = {2, 4, 16},     // Example Periods
92         .wcet = {1, 1, 4},        // Example Worst Case Execution Times
93         .deadline = {2, 4, 16},   // Example Deadlines
94         .num_tasks = 3            // Number of tasks in this set
95     },
96     {
97         // 6
98         .period = {2, 5, 7, 13},  // Example Periods
99         .wcet = {1, 1, 1, 2},     // Example Worst Case Execution Times
100        .deadline = {2, 3, 7, 15}, // Example Deadlines
101        .num_tasks = 4            // Number of tasks in this set
102    },
103    {
104        // 7
```

```
111     .period = {3, 5, 15},    // Example Periods
112     .wcet = {1, 2, 4},      // Example Worst Case Execution Times
113     .deadline = {3, 5, 15}, // Example Deadlines
114     .num_tasks = 3          // Number of tasks in this set
115 },
116 {
117     // 8
118     .period = {2, 5, 7, 13}, // Example Periods
119     .wcet = {1, 1, 1, 2},    // Example Worst Case Execution Times
120     .deadline = {2, 5, 7, 13}, // Example Deadlines
121     .num_tasks = 4           // Number of tasks in this set
122 },
123 {
124     // 9
125     .period = {6, 8, 12, 24}, // Example Periods
126     .wcet = {1, 2, 4, 6},     // Example Worst Case Execution Times
127     .deadline = {6, 8, 12, 24}, // Example Deadlines
128     .num_tasks = 4           // Number of tasks in this set
129 };
130
131 // Feasibility test functions
132 bool completion_time_feasibility(task_set_t *task_set);
133 bool scheduling_point_feasibility(task_set_t *task_set);
134 bool rate_monotonic_least_upper_bound(task_set_t *task_set);
135 int edf_feasibility(task_set_t *task_set, bool deadline);
136 int llf_feasibility(task_set_t *task_set, bool deadline);
137 bool deadline_monotonic_feasibility(task_set_t *task_set);
138
139 int main(void)
140 {
141     for (int i = 0; i < EXAMPLES; i++)
142     {
143         printf("\n*****\n");
144         printf("Example %d\n", i);
145
146         // Calculate and print total utilization
147         double U = 0.0;
148         for (int j = 0; j < tasks[i].num_tasks; j++)
149         {
150             U += (double)tasks[i].wcet[j] / tasks[i].period[j];
151         }
152         printf("C: ");
153         for (int j = 0; j < tasks[i].num_tasks; j++)
154         {
155             printf("%d ", tasks[i].wcet[j]);
156         }
157         printf("\nT: ");
158         for (int j = 0; j < tasks[i].num_tasks; j++)
159         {
160             printf("%d ", tasks[i].period[j]);
161             if(tasks[i].period[j] == 0){
162                 printf("Pperiod is zero\n");
163                 exit(0);
164             }
165         }
166         printf("\nD: ");
167         int dm = 0;
```

```
170     for (int j = 0; j < tasks[i].num_tasks; j++)
171     {
172         printf("%d ", tasks[i].deadline[j]);
173         if(tasks[i].deadline[j] ==0){
174             printf("Deadline is zero\n");
175             exit(0);
176         }
177         if (tasks[i].deadline[j] != tasks[i].period[j])
178         {
179             dm++;
180         }
181     }
182     // printf("\nUtility : %4.2f%%\n", U * 100);
183
184     // Perform and print feasibility tests
185     printf("RM LUB: %s\n", rate_monotonic_least_upper_bound(&tasks[i]) ? "
Feasible" : "Infeasible");
186     printf("Completion time feasibility: %s\n",
completion_time_feasibility(&tasks[i]) ? "Feasible" : "Infeasible");
187     printf("Scheduling point feasibility: %s\n",
scheduling_point_feasibility(&tasks[i]) ? "Feasible" : "Infeasible");
188
189     if (dm != 0)
190     {
191         printf("Deadline monotonic: %s\n", deadline_monotonic_feasibility(&
tasks[i]) ? "Feasible" : "Infeasible");
192
193         printf("\n(Period)");
194         printf("EDF on Period: %s\n", edf_feasibility(&tasks[i], false) ? "
Feasible" : "Infeasible");
195         printf("LLF on Period: %s\n", llf_feasibility(&tasks[i], false) ? "
Feasible" : "Infeasible");
196
197         printf("\n(Deadline)");
198         printf("EDF on Deadline: %s\n", edf_feasibility(&tasks[i], true) ?
"Feasible" : "Infeasible");
199         printf("LLF on Deadline: %s\n", llf_feasibility(&tasks[i], true) ?
"Feasible" : "Infeasible");
200     }
201     else if (i > 4)
202     {
203         printf("\n(Period)");
204         printf("EDF: %s\n", edf_feasibility(&tasks[i], false) ? "Feasible"
: "Infeasible");
205         printf("LLF: %s\n", llf_feasibility(&tasks[i], false) ? "Feasible"
: "Infeasible");
206     }
207
208     // Add other feasibility tests here
209
210     printf("\n");
211 }
212 }
213
214 bool rate_monotonic_least_upper_bound(task_set_t *task_set)
215 {
216     double utility_sum = 0.0, lub = 0.0;
217     int idx;
218
219     // Sum the C(i) over the T(i) for utility calculation
220     printf("\n\n");
```

```
221     for (idx = 0; idx < task_set->num_tasks; idx++)
222     {
223         utility_sum += ((double)task_set->wcet[idx] / (double)task_set->
period[idx]);
224         printf("Task %d, WCET=%u, Period=%u, Utility Sum = %lf\n", idx,
task_set->wcet[idx], task_set->period[idx], utility_sum);
225     }
226     printf("\nTotal Utility Sum = %lf\n", utility_sum);
227
228     // Compute LUB for the number of services
229     lub = (double)task_set->num_tasks * ((pow(2.0, (1.0 / (double)task_set->
num_tasks))) - 1.0);
230     printf("LUB = %lf\n", lub);
231
232     // Compare the utility sum to the bound and return feasibility
233     if (utility_sum <= lub)
234         return TRUE;
235     else
236         return FALSE;
237 }
238
239 bool completion_time_feasibility(task_set_t *task_set)
240 {
241     int i, j;
242     U32_T an, anext;
243     int set_feasible = TRUE;
244
245     // For all tasks in the analysis
246     for (i = 0; i < task_set->num_tasks; i++)
247     {
248         an = 0;
249         anext = 0;
250
251         for (j = 0; j <= i; j++)
252         {
253             an += task_set->wcet[j];
254         }
255
256         while (1)
257         {
258             anext = task_set->wcet[i];
259
260             for (j = 0; j < i; j++)
261                 anext += ceil((double)an / (double)task_set->period[j]) *
task_set->wcet[j];
262
263             if (anext == an)
264                 break;
265             else
266                 an = anext;
267         }
268
269         if (an > task_set->period[i])
270         {
271             set_feasible = FALSE;
272         }
273     }
274
275     return set_feasible;
276 }
```

```
277
278 bool scheduling_point_feasibility(task_set_t *task_set)
279 {
280     int rc = TRUE, i, j, k, l, status, temp;
281
282     // For all tasks in the analysis
283     for (i = 0; i < task_set->num_tasks; i++)
284     { // iterate from highest to lowest priority
285         status = 0;
286
287         // Look for all available CPU minus what has been used by higher
priority tasks
288         for (k = 0; k <= i; k++)
289         {
290             // find available CPU windows and take them
291             for (l = 1; l <= (floor((double)task_set->period[i] / (double)
task_set->period[k])); l++)
292             {
293                 temp = 0;
294
295                 for (j = 0; j <= i; j++)
296                 task_set->period[k] / (double)task_set->period[j]);
297
298                 // Can we get the CPU we need or not?
299                 if (temp <= (l * task_set->period[k]))
300                 {
301                     // insufficient CPU during our period, therefore infeasible
302                     status = 1;
303                     break;
304                 }
305             }
306             if (status)
307                 break;
308         }
309
310         if (!status)
311             rc = FALSE;
312     }
313     return rc;
314 }
315
316 int llf_feasibility(task_set_t *task_set, bool deadline)
317 {
318     double totalU = 0.0;
319     if (!deadline)
320     {
321         for (int i = 0; i < task_set->num_tasks; i++)
322         {
323             totalU += (double)task_set->wcet[i] / task_set->period[i];
324         }
325     }
326     else
327     {
328         for (int i = 0; i < task_set->num_tasks; i++)
329         {
330             totalU += (double)task_set->wcet[i] / task_set->deadline[i];
331         }
332     }
```

```
333     printf("Total utility in LLF: %f ", totalU);
334     if (totalU <= 1.0)
335     {
336         printf("Which is less than 1.0 \n");
337         return TRUE;
338     }
339     else
340     {
341         printf("Which is less than 1.0 \n");
342         return FALSE;
343     }
344 }
345
346 bool deadline_monotonic_feasibility(task_set_t *task_set)
347 {
348     //Ensure tasks are sorted by their deadlines before running this
feasibility test.
349     int status = 0;
350     for (int i = 0; i < task_set->num_tasks; i++)
351     {
352         float interference = 0;
353         float utilization = 0;
354         for (int j = 0; j < i; j++)
355         {
356             interference += (ceil((float)task_set->deadline[i] / (float)
task_set->period[j])) * (float)task_set->wcet[j];
357         }
358         utilization = ((float)(task_set->wcet[i]) / (float)task_set->
deadline[i]) + (interference / (float)task_set->deadline[i]));
359         if (utilization > 1)
360         {
361             status = 1;
362             break;
363         }
364     }
365     if (status == 1)
366         return FALSE;
367     else
368         return TRUE;
369 }
370
371 int edf_feasibility(task_set_t *task_set, bool deadline)
372 {
373     double totalU = 0.0;
374     if (!deadline)
375     {
376         for (int i = 0; i < task_set->num_tasks; i++)
377         {
378             totalU += (double)task_set->wcet[i] / task_set->period[i];
379         }
380     }
381     else
382     {
383         for (int i = 0; i < task_set->num_tasks; i++)
384         {
385             totalU += (double)task_set->wcet[i] / task_set->deadline[i];
386         }
387     }
388     printf("\nTotal utility in EDF: %f ", totalU);
```

```
389     if (totalU <= 1.0)
390     {
391         printf("Which is less than 1.0 \n");
392         return TRUE;
393     }
394     else
395     {
396         printf("Which is less than 1.0 \n");
397         return FALSE;
398     }
399 }
400
```