### Homework 3

Deadline: see web page
Assignments: Non-programming parts are to be solved individually (turned in electronically, written parts in ASCII text, NO Word, postscript, etc. permitted).

Some programming assignments have to be solved in your group of up to 2 group members.

Please use the OS Lab machines (Linux) in 2236 EB2, either by going to the OS lab or by remotely logging in to eb2-2236-awXX.csc.ncsu.edu, where XX=01..09. All programs have to be written in C/C++, translated with gcc and turned in with a corresponding Makefile (unless indicated otherwise).

Programming exercises turned in electronically, individual ones by each students and group assignments just by 1 student. Finally, peer evaluations are also turned in electronically.

1. (20 points, 5 each) A system contains 3 periodic tasks: T1(4,1),T2(6,1), and T3(8,3). Determine if the task set is schedulable according to the following schedulability analysis methods:
   - (a) non-preemptive EDF using blocking time modeling (Theorem 6-18)
   - (b) non-preemptive EDF using Jeffay's theorem
   - (c) non-preemptive RM utilization with blocking
   - (d) non-preemptive RM using TDA with blocking

   Turn in file problem1.txt.

2. (80 points -- groups) Modify the existing static priority scheduler of EV3RT to support **EDF** scheduling. EDF scheduling will be supported just within tasks of equal static priority.
   - The hints below point you to an abstraction for cyclic tasks (used for cyclic executives) and prioritized periodic tasks (abstracted as cyclic tasks with priorities). You can reuse their specification of a period.
   - You need to add a counter in the TCB to track the job index (in kernel.h).
   - The periodic-test sample code shows how a timer interrupt can be triggered to issue a callback (to ask_activator()). This calls act_tsk(). You need to make a change here to increment the number of job invocations for the current task.
   - act_tsk() in manage_task.c eventually calls make_runnable() of tasks.c. At this point, the new task is appended to the tail of the queue at its priority level via queue_insert_prev(). You need to change this code. Within the current priority level, walk the queue (see Inline functions in queue.h) to place the new task in the queue in EDF order. I suggest to walk the queue until the next element is head (circular queue!) or its absolute deadline is after that of the new task. Then insert the new task after the current one (and before the next one) -- or at the head if the queue is empty. Incremented a counter if a task is being preempted (i.e., the counter is associated with the task task is being preempted).
   - To calculate absolute deadlines for comparison, multiply the number of invocations with the period. You may need additional TCB fields to be able to access the period, which you can store in this field in task_activation(). (A cleaner way would be to change the cfg tool to generate such code in ev3api_cfg.c as part of the build process. But you are not required to do so.)

   Hints:
   - Use the uITRON4.0 Specification - TRON Forum, the EV3RT paper, and this tutorial tutorial of Toppers as a reference. Notice that it describes a different Topper target platform, but the principle description is correct. Unfortunately, all figures are lost due to the Wayback Engine. But it's still useful. A Google translation with figures (Figures not translated) can be used side-by-side.
   - Check out the c/h/cfg files for the periodic-task sample code.
   - make app=periodic-task
   - Inspect the auto-generated files in sdk/OBJ/, such as ev3api_cfg.c with the initialization of task structures and module_cfg.c with the _module_ctsk_tab (and _module_cfg_tab) generated from the app.cfg file.
   - These tables are decoded after the the program is loaded (see handle_module_cfg_tab()) to create pk_ctsk structures and call acre_tsk() to create tasks, see target/ev3_gcc/dmloader/src/dmloader.c
   - All *_tsk() routines are in the kernel/, have a look at task.c, task_manage.c (which implements act_tsk() and populates the TCBs) etc. The get_tcb() macro and the TCB structure are in task.h.
   - Modify files in the directory kernel, esp. task.h and task.c, (and possibly others). The core scheduling functionality is through calls to dispatch() (assembly code in arch/arm_gcc/common/core_support.S).
   - Have a look at time_event.c/h, which implements the timer interrupt in signal_time().
   - cyclic.c starts/stops sta_cyc()/stp_cyc() periodic tasks if they are initially inactive (created w/ TA_NULL), its callback function call_cychdr() adds the next periodic event via enqueue_cyc(). It is called from signal_time() on cyclic events.
   - cyclic.c defines macros for tnum_cyc and tnum_scyc, the number total number of cyclic tasks and the number of "static system" (predefined) cyclic tasks. You can find your task ID in the exinf field of the task_initialization_block (see tutorial, part 14). The tskid passwd into Toppers syscalls actually differs by an offset that's the difference between tnum_cyc and tnum_scyc. If you subtract this difference from exinf, you can compare to tskid. (I wish they would document that.)
   - Try to keep your modifications minimal in the kernel directory, use declarations of new data structures within your test program.

   Turn in a README2 file (separately) and an edf.zip file of your entire ev3rt-hrp2, but BE SURE TO RUN "make clean" first so that only sources are submitted!!! The README2 indicates all changed files, which should have comments (in English, not Japanese, please) and explains the approach (which may differ from the one suggested above). The program will be tested by these commands:

```
unzip edf.zip
cd ev3rt-hrp2
cd base-workspace
make realclean
make app=loader
cp uImage /run/media/...
#... (see HW1 for for building ev3rt)
cd ../sdk/workspace
make clean
make app=edf
cp app /run/media/.../ev2rt/apps/edf
#sample output:
Start... calibrated loop time 101 @ max 146

@ABBACBABCABBAC@ABBACBABCABBAC
011121232143351164472585296 6:2
|....|....|....|....|....|....
task 0 was preempted 4 times
  @ 1437
  @ 1500
  @ 1602
  @ 1909
task 1 was preempted 0 times
task 2 was preempted 6 times
  @ 300
  @ 600
```

```
  @ 1200
  @ 1800
  @ 2100
  @ 2700
task 3 was preempted 6 times
  @ 500
  @ 900
  @ 1002
  @ 2000
  @ 2400
  @ 2502

   0  100  200  300  400  500  600  700  800  900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900 2000 2100 2200 2300 2400 2500 2600 2700 2800 2900
```

Use bluetooth to print out which task executes at any time. Create an EDF task set of at least 3 tasks with non-harmonic periods and preemptions. Then record the task ID (say: A or B or C) of the running task and its job index every 100ms. (Hint: You can use an even higher priority task to record this information.) Do NOT print out this information right away, but instead record it in an array! (Why?) The execution length of tasks is controlled by running some code that is calibrated up front to run for the right amount of time.
Also, when an EDF task is activated, it checks if another task was preempted; if so, it increments a count for the task that is being preempted and record the timestamp of preemption. Introduce a lower priority task as well with a task ID of "@".
Execute for two hyperperiods, then print a Gantt chart indicating task execution (see above), the preemption counts/timestamps, and the discrete times shown in the Gantt chart. Explain the output in your README.
The above example shows the **rate monotone** schedule (not EDF) for the following task set (times in ms): TA=(300, 100), TB=(500, 200), TC=(1500, 300). The background task T@=(1500, 100) has a lower static priority. Notice that there are more preemption points that one might expect, and there is a reason for that! (explain)
Please demonstrate EDF (and RM) for the following task set, which gives a different execution depending on policy: TA=(300, 100), TB=(500, 300), TC=(1500, 100).

- Peer evaluation: Each group members has to submit a peer evaluation form.

**Grading:**

Group assignments are graded by groups. All members receive the same number of base points but these base points are multiplied by a peer evaluation factor, which is determined as the average of the evaluation score by your peer (optionally minus outliers, up to the TA). Each group member is individually responsible to **know** how any of the parts are solved. While the individual author of a module is indicated, this does not affect grades.

**What to turn in for programming assignments:**

- commented program(s) as source code, comments may count as much as 10% of the points (see class policy on guidelines on comments)
- Makefiles (if required)
- test programs as source (and input files, if required)
- README (documentation to outline solution and list commands to install/execute -- if required)
- in each file, include the following information as a comment at the top of the file, where "username" is your unity login name and the single author is the person who wrote this file:

```
Single Author info:

username FirstName MiddleInitial LastName

Group info:

username FirstName MiddleInitial LastName

username FirstName MiddleInitial LastName

username FirstName MiddleInitial LastName
```

**How to turn in:**

For programming assignments, turn in exactly one set of files per group (unless the assignment is an individual one)! It does not matter which member of the group submits, as long as it is only one. Use the "Submit Homework" link on the course web page. Please upload all files individually.

Remember: If you submit a file for the second time, it will overwrite the original file.

**Group problems:**

Should there be any problems in the groups, then please read the policies for this class. Then consult the TA at the earliest possible point. Problems may include: a group member drops the class, does not show up for meetings, does not contribute any work, has personal problems related to the group etc. If the TA cannot resolve the problem, the instructor shall be consulted right away.