ECE-350 L4 Report

Tony, Shen, t48shen
Moyez Farook, Mansoor, mfmansoo
Anthony Natale, De Luca, andeluca
Anson, Wan, a23wan

DATA STRUCTURES AND ALGORITHMS

For this lab, we reused structures from all previous labs to facilitate scheduling of real-time tasks based on EDF protocol, specifically using our FIFO queue to organize and sort tasks by their period

TEST SUITE DESCRIPTIONS

Note: we use helper functions initialize_msg_(MSG_TYPE) and validate_message in our test suites.

Test Suite 1 - ae_tasks1_G10.c

SUMMARY:
This test suite will create tasks1-3, elevate all to real-time w/ different periods, have them all send messages back to priv_task1 and ensure the order of messages received is correct (through a long period of time). There will also be another (non real-time) task4, which will do the same thing (send a message back to priv_task1), and should only execute when all of the real-time tasks are in the SUSPENDED state. We should therefore have an expected order of execution that can be verified through the received messages to priv_task1.

The test suite will also verify that rt_tsk_get, tsk_get, tsk_ls, mbx_ls, and mbx_get return the expected results throughout program execution.

TESTS:
- EDF scheduling policy
    - integration with previous preemptive priority scheduling (SUSPENDED)
    - includes all Lab4 system calls
    - tests all _get and _ls functions
    - non real-time task runs only when all real-time tasks SUSPENDED
        - once real-time task unsuspended, immediately preempts
- some real-time edge cases?
- uses artificial delay

STEPS:
- priv_task1

        - create_mbx() for priv_task1
        - tsk_set_prio() priority of priv_task1 to LOWEST
        - **TEST CASE 1: create_mbx(), tsk_set_prio(), mbx_ls(), tsk_ls()**
        - tsk_create() task1 (prio=LOW)
        - tsk_create() task2 (prio=MED)
        - tsk_create() task3 (prio=HIGH)
        - tsk_create() task4 (prio = MED)
        - **TEST CASE #: tsk_create()**
        - recv_msg()
            - (repeat 31 times to verify correct order printed)
        - **TEST CASE #: recv_msg() gives messages in order (use multiple?)**
        - test_exit()
- task1
    - rt_tsk_set() to elevate to real-time
        - period = 0.1s
    - inside finite while loop (6 iterations):
        - send_msg() of “1” to priv_task1
        - rt_task_susp() to yield remaining period to other real-time tasks
    - tsk_exit()
- task2
    - rt_tsk_set() to elevate to real-time
        - period = 0.2s
    - inside finite while loop (3 iterations):
        - send_msg() of “2” to priv_task1
        - rt_task_susp() to yield remaining period to other real-time tasks
    - tsk_exit()
- task3
    - rt_tsk_set() to elevate to real-time
        - period = 0.3s
    - inside finite while loop (2 iterations):
        - send_msg() of “3” to priv_task1
        - rt_task_susp() to yield remaining period to other real-time tasks
    - tsk_exit()
- task4
    - inside finite while loop (20 iterations):
        - send_msg() of “4” to priv_task1

Test Suite 2 - ae_tasks2_G10.c

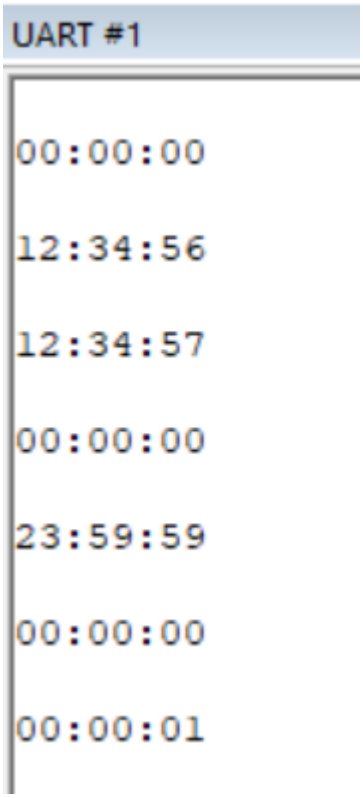
This test suite focuses on these aspects of the Wall Clock:

- **Wall Clock properly displayed**
  - **Correct format HH:MM:SS, 24 hour**
- **Task registers itself correctly.**
- **All 3 commands**
  - **%WR**
    - **Properly resets the wall clock to 00:00:00. Sends to console display task, displays on console terminal updating every second.**
  - **%WS HH:MM:SS**
    - **Properly sets the wall clock to HH:MM:SS. Sends to console display task, displays on console terminal updating every second.**
    - **If invalid input, no operation.**
  - **%WT**
    - **Properly removes the wall clock from the console terminal.**

Tests:

- Test 1
  - Verify sending a %WS HH:MM:SS command to the TID_WCLCK tid's mailbox works with no error
  - Manually verify that at the next period for the Wall Clock (After 1 second) the wall clock updates to HH:MM:SS
- Test 2
  - Verify sending a %WS HH:MM:SS where HH:MM:SS is an invalid string to the TID_WCLCK tid's mailbox works with no error.
  - Manually verify that at the next period for the wall clock, the wall clock goes up 1 second and does not set the wall clock to the invalid value in the message.
- Test 3
  - Verify sending a %WR command to the TID_WCLCK tid's mailbox works with no error
  - Manually verify that after 1 second, the wall clock updates to 00:00:00
- Test 4
  - Verify sending a %WT command to the TID_WCLCK tid's mailbox works with no error
  - Manually verify that after 1 second, the wall clock is no longer displaying on the console terminal
- Test 5
  - Verify sending %WS 23:59:59 command to the TID_WCLCK tid's mailbox works with no error.
  - Manually verify that after 1 second, the wall clock updates to 24:59:59
  - Manually verify that after another second, the wall clock updates to 00:00:00

Expected Output on Console for Test Suite 2:


UART #1
00:00:00
12:34:56
12:34:57
00:00:00
23:59:59
00:00:00
00:00:01

Test Suite 3 - ae_tasks3_G10.c

SUMMARY:
This test suite tests the functionality of the KCD and CON tasks with the newly implemented real-time task support with EDF scheduling policy. We employ two real-time tasks with the same period, one created after the other. We register (and re-register) command IDs inside each real-time task, and verify that the same command strings sent from various tasks (eg. task1->task2, task1->task1) are sent and received by the correct mailbox. We verify that the task that most recently registered a common command takes precedence.

Additionally, we verify that %LT and %LM work, and call tsk_ls() and mbx_ls() to manually check.

TESTS:
- 2 real-time tasks with the same period, executes in order created each period
- KCD task, CON task
    - registering command ids inside real-time task
        - re-registering command id - most recent
    - sending command id from one real-time task to another
        - also send to self
    - %LT - manually check matches w/ tsk_ls()
    - %LM - manually check matches w/ mbx_ls()
- rt_tsk_get()

STEPS:
- priv_task1
    - create_mbx()
    - tsk_set_prio() priority of priv_task1 to LOWEST
    - **TEST CASE 1: create_mbx(), tsk_set_prio(), mbx_ls(), tsk_ls()**
    - tsk_create() task1 (prio=MED) - will preempt, run to rt_task_susp
    - tsk_create() task2 (prio=HIGH) - will preempt, run to rt_task_susp
    - //step3
        - send_msg() KEY_IN command %A to KCD
            - should go to task2, since most recently registered
    - **TEST CASE 4: tsk_create(), send_msg(), mbx_get(TID_KCD) same**
    - artificially delay long enough to let task1 and task2 unsuspend
        - should allow to unsuspend multiple times
    - //to fill
    - test_exit()
- task1
    - create_mbx()
    - send_msg() KCD_REG command A
    - send_msg() KCD_REG command C
    - rt_tsk_set() to elevate to real-time
        - period = 0.1s
    - //step1
        - send_msg() KEY_IN command %C to KCD

        - should go to self
    - **TEST CASE 2: send_msg(), mbx_get(TID_KCD) down**
    - rt_task_susp()
- //step4
    - recv_msg() to get KEY_IN command %C from task1
    - **TEST CASE 5: recv_msg(), validate_msg()**
    - send_msg() KEY_IN command %B to KCD
        - should go to task2
    - **TEST CASE 6: send_msg(), mbx_get(TID_KCD) down**
    - rt_task_susp()
- //step6
    - send_msg() KEY_IN command %LT to KCD
    - **TEST CASE 9: send_msg(), mbx_get(TID_KCD) down**
    - tsk_ls()
    - rt_tsk_get()
    - **TEST CASE 10: tsk_ls(), rt_tsk_get()**
    - rt_task_susp()
- tsk_exit()

- task2
    - create_mbx()
    - send_msg() KCD_REG command B
    - rt_tsk_set() to elevate to real-time
        - period = 0.1s
    - //step2
        - send_msg() KCD_REG command A to KCD
        - **TEST CASE 3: send_msg(), mbx_get(TID_KCD) down**
        - rt_task_susp()
    - //step5
        - recv_msg() to get KEY_IN command %A from priv_task1
            - most recent
        - **TEST CASE 7: recv_msg(), validate_msg()**
        - recv_msg() to get KEY_IN command %B from task1
        - **TEST CASE 8: recv_msg(), validate_msg()**
        - rt_task_susp()
    - //step7
        - send_msg() KEY_IN command %LM to KCD
        - **TEST CASE 11: send_msg(), mbx_get(TID_KCD) down**
        - mbx_ls()
        - rt_tsk_get()
        - **TEST CASE 12: mbx_ls(), rt_tsk_get()**
        - rt_task_susp()
    - tsk_exit()

<u>Expected Output on Console for Test Suite 3:</u>

```
00:00:00
 % L T
TID: 1 State: READY

TID: 2 State: SUSPENDED

TID: 3 State: SUSPENDED

TID: 13 State: SUSPENDED

TID: 14 State: READY

TID: 15 State: RUNNING
  % L M
TID: 1 State: READY Free: 128

TID: 2 State: SUSPENDED Free: 128

TID: 3 State: SUSPENDED Free: 128

TID: 13 State: SUSPENDED Free: 128

TID: 14 State: READY Free: 39

TID: 15 State: RUNNING Free: 512

00:00:01

00:00:02
```