Lab 4 Report

Group Members:

Geoffery Qin (c22qin)

Rebecca Zhou (yx5zhou)

Kehan Xing (k5xing)

Yiru Wu (y645wu)

Data Structure

suspend_tsk (QUEUE)

- This structure is used to store all the suspended tasks
- It has the same structure as our ready queues

variable name	type	purpose
HEAD	TCB*	Points to the head of the suspend list
TAIL	TCB*	Points to the tail of the suspend list
current_size	unsigned int	Stores the current size of the suspend list

Ring Buffer (MB)

- This structure is used to store the message in the mailbox

variable name	type	purpose
memory_start	U8*	Stores the start of the ring buffer which will be returned from the **k_mpool_alloc** function
memory_end	U8*	Stores the end of the ring buffer which is **memory_start** plus the size of the buffer
capacity	size_t	Size of the buffer
current_size	size_t	Current capacity of the buffer
head	U8*	Stores the current location of read (receiving message)
tail	U8*	Stores the current location of write (sending message)

Mailbox (MAILBOX)

- This is the structure of the mailbox which will contain a ring buffer and waiting list.
- The waiting list will only be used for blocking receive and send

variable name	type	purpose
active	int	Set to 1 if the mailbox is active
receiver_id	task_t	Stores the task id of the owner of the mailbox
memory_buffer	MB	Instantiation of Ring buffer
waiting list	QUEUE	Instantiation of Waiting List

ready_queue (QUEUE)

- This structure is used for waiting list and ready queues
  - We added an additional queue to our total ready queues for real time tasks

variable name	type	purpose
HEAD	TCB*	Points to the head of QUEUE
TAIL	TCB*	Points to the tail of QUEUE
current_size	unsigned int	Stores the current size of the QUEUE

cmdnode (CMDNODE)

variable name	type	purpose
next	CMDNODE*	Points to the next CMDNODE in the CMDLIST
command	char	Stores the command character
command_handler_tid	task_t	Stores the command handler tid

cmdlist (CMDLIST)

variable name	type	purpose
head	CMDNODE*	Points to the head of the list
tail	CMDNODE*	Points to the tail of the list

inputnode (INPUTNODE)

variable name	type	purpose
next	INPUTNODE*	Points to the next INPUTNODE in the INPUTQUEUE
character	char	Stores the character

inputqueue (INPUTQUEUE)

variable name	type	purpose
head	INPUTNODE*	Points to the head of the queue
tail	INPUTNODE*	Points to the tail of the queue

Helper Functions

void update_clock_string(uint8_t clock_string, int h, int m, int s)*

This function sets the clock_string with inputted hours, minutes, and seconds values at corresponding positions of clock_string.

void update_clock_display(uint8_t clock_string, int h, int m, int s)*

This function calls update_clock_string function to update clock_string with the hours, minutes, and seconds values user inputted and embed the message containing clock_string with ANSI Escape Sequence to set the clock_string display position on the upper right corner(Row 1, Column 72) of the interrupt-driven uart window.

*int verify_valid_input(uint8_t *input)*

This function verifies the user input for the wall clock to prevent malicious or invalid inputs on time. The input of the function has 9 indexes and if the input is not valid at any location, the function will return -1. If the input is valid and passes all the checks for each index, the function will return RTX_OK.

int char_to_int(char key)

This function converts a char to int by minus the value by 0 which does not change the value of the variable, but the return type is converted to int

Algorithm Functions

*int k_rt_tsk_set(TIMEVAL *p_tv);*

This function set the current running task to be a real time task with a period p_tv. It returns RTX_OK if it works and it returns –1 if current task is already real time task or the period is not available.

int k_rt_tsk_susp(void));

This function set current running real time task to be suspended until its next period start. It returns RTX_OK if it works and –1 current task is not real time task.

*int k_rt_tsk_get(task_t tid, TIMEVAL *buffer);*

This function stores the period of task with the specified tid to the buffer. It returns RTX_OK if it works and –1 if the specifies task is not real time task or the task is not exist.

void TIMER0_IRQHandler(void);

When timer0 interrupts, this function will check suspended task list and release the tasks if its new period starts. Also the function will check task priority and closer deadline of real time task to decide preemption.

void task_wall_clock(void)

This function will create a mailbox for itself, sends a KCD_REG message to KCD task to register "W" command, and call rt_tsk_set to set its period to 1 second and elevate itself to a real-time task. Then, the function will enter an infinite loop where it updates clock hours, minutes, and seconds values if the clock has started, calls recv_msg_nb to check if there is any KCD_CMD message from KCD, and call rt_tsk_susp function to suspend itself until its next period. If the function receives %WR KCD_CMD message, the function will reset the clock to 00:00:00, started the clock, and displayed the clock_string on the interrupt_driven UART window. If the function receives %WS HH:MM:SS KCD_CMD message, the function will reset the clock to HH:MM:SS that user inputted, started the clock, and displayed the clock_string on the interrupt_driven UART window. If the function receives %WT KCD_CMD message, the function will remove the clock_string from the interrupt_driven UART window.

Test Cases

ae_tasks1_G12.c

This test suite creates 1 task during initialization, then the initial task will create other three tasks. Overall, it contains 4 tasks in total, 4 high priority. It mainly tests the basic function of rt_tsk_set, rt_tsk_susp, receive_block of real time task. The test suite covers the following scenarios.

- Set a non-real-time task to real time
- Calling rt_tsk_susp, the task should be suspended and switch task
- Calling non blocking receive, the task should not be blocked
- Real time tasks with earlier deadlines will preempt other real time task

If everything passes, we should get a score of 16/16 at the end.

ae_tasks2_G12.c

This test suite creates 2 tasks during initialization, then the initial task priv_task1 will create three MEDIUM priority tasks. It mainly tests the execution sequence of real-time tasks and non-real-time tasks with the functions rt_tsk_set and rt_tsk_susp. The test suite covers the following scenarios:

- Set a non-real-time task to real time
- Call rt_tsk_susp to switch to another non-real-time task
- Set two other non-real-time tasks to real-time with different period and update their execution sequence every time
- Check the result sequence with the expected sequence

If everything passes, we should get a score of 29/29 at the end.

ae_tasks3_G12.c

This test suite creates 2 tasks during initialization, then each initial task will create another high priority task. It mainly tests the kcd, cdisp, wall_clock, and uart_handler. The test suite covers the following scenarios.

Based on this scenario, we tested the following cases:

- Task 0 register Command D, and Task 1 register Command D after, check if Task 1 receive D KCD_CMD message when user type %D
- Task 2 register Command G, Task 3 register Command F, and check if Task 2 receive G KCD_CMD message when user type %G and Task 3 receive F KCD_CMD message when user type %F.
- User type %LT in the interrupt_driven UART, check if all non-dormant tasks' tid and state information are displayed in the interrupt_driven UART
- User type %LM in the interrupt_driven UART, Check if all non-dormant tasks' tid, state, mailbox information are displayed in the interrupt_driven UART
- User type %WR in the interrupt_driven UART, check if the clock_string are reset to 00:00:00, started the clock, and displayed on the upper right corner(Row 1, Column 72) of the interrupt_driven UART
- User type %WS HH:MM:SS in the interrupt_driven UART, check if the clock_string are set to HH:MM:SS user inputted, started the clock, and displayed on the upper right corner(Row 1, Column 72) of the interrupt_driven UART
- User type %WT, check if the clock_string are removed from the upper right corner(Row 1, Column 72) of the interrupt_driven UART

There will not be a score information printed on the polling UART window in the end as all 4 tasks created are in infinite loop calling recv_msg(), and the user commands result will be displayed on interrupt-driven UART window as user will type different command in the window.