Computer Systems Engineering Technology
CST 347 – Real-Time Operating Systems

Lab 06 – Command Line Interface (CLI) Name_____

Possible Points: 10

Instructions

Complete the following procedures. Answer any questions given on this lab. Zip up your final project and answers to the questions, then upload to blackboard.

Procedure

Integrating FreeRTOS+CLI into your project

Copy the FreeRTOS-Plus-CLI directory to be at the same level as your FreeRTOS folder. This directory is provided as a .zip file with this lab. You will also have to modify your project to include this additional folder in the include paths. The folder should contain the FreeRTOS_CLI.c, FreeRTOS_CLI.h, History.txt and LICENSE_INFORMATION.txt.

There is a PDF file on the Blackboard site that will give you a quick tutorial on the USE and Integration of FreeRTOS+CLI. Also you will find a file called examplecmd.c that contains the code to implement the task-stats command. This file only contains the command implementation not the calls to FreeRTOS+CLI to interpret the calls.

CLI Commands

You will need to implement the following Command Line Interface commands. You are provided with example code for the "***task-stats***" command in the **examplecmd.c** file. The additional commands will have a similar structure with a different command operation implementation. Some commands will take command line parameters and some will not. Parameters for a command will be extracted using the `FreeRTOS_CLIGetParameter()` call. The parameters will be returned in ASCII and must be converted to the appropriate type by your command processing function. In this lab, the parameters will be integer values so `atoi()` is all that is needed. Each command entered must have the exact number of parameters required, including 0, or it will not be recognized by the CLI command processor.

task-stats – Provide a listing tasks and task related information

start_led **LED_int DELAY_ms** – Start <**LED_int**> blinking using a toggle delay time of <**DELAY_ms**>. If the specified LED is already blinking, the LED will not be affected but a user message "LED already started" will be sent to the terminal.

***stop_led* LED_int** – Stop <**LED_int**> from blinking. If the specified LED is not currently blinking, the LED will not be affected but a user message "LED already stopped" will be sent to the terminal.

***change_led* LED_int DELAY_ms** – Change <**LED_int**>'s blink delay time to <**DELAY_ms**>. If the specified LED is not currently blinking, the LED will not be affected but a user message "LED is stopped" will be sent to the terminal.

Tasks

1. RX Task – The RX task will be created with task priority 3. The UART RX task will "take" the InputByteBuffer mutex. It will block if the mutex is not available. [The UART ISR will "give" the InputByteBuffer mutex when a character is received through the UART.] When RX Task acquires the mutex, it will call the GetChar UART driver function. It will use the character data acquired through the RX operation of the ISR. In response, it will send the character data to the TX queue for echoing purposes.

In addition, characters will be built into a command string that will be null-terminated when a CR character is received. Once the command string is complete, it will call FreeRTOS_CLIProcessCommand() function to process. From this call, the returned output buffer (statically created with 300 bytes) will be portioned into the 50 byte maximum TX queue messages. (49 + null)

2. TX Task – The UART TX task will block awaiting a TXQueue message. The message will be a string of maximum 50 characters for display to the terminal. (49 + null) The TX task will call the PutStr UART driver function which will start and interrupt-driven string transfer to the TX. The TX task will be created with task priority 2.

3. LEDn Task – The LEDn task will toggle the appropriate LED. These will be created by the start_led CLI command and deleted by the stop_led CLI command. In addition, the change_led CLI command will message the task to change the blink rate. The TX task will be created with task priority 1.

UART Interrupt Driven Driver

Both the RX and the TX use the same ISR. Operations will be within the single ISR functions.

1. UARTPutStr – It will "take" the OutputStringBuffer mutex. It will write the string and make it available to the TX function of the ISR. The data will be a null-terminated string that the ISR can read.

2. UARTGetChar – This function will return the character buffer that the RX ISR populated.

UART Interrupt Service Routine

1. RX function – This operation will populated the character buffer to be made available to the UART driver. It will use UARTGetData() from the plib to extract the RX data value. After getting the value, clear the RX interrupt flag and then "give" the InputByteBuffer mutex.

2. TX function – This operation will use UARTSendDataByte() to send characters to the terminal. It will clear the TX interrupt flag AND disable the TX interrupt when character is NULL. (End of String) Lastly, it will "give" the OutputStringBuffer mutex.

Queue

TX Queue – Same as previous lab.

LEDn Queue – Same as past labs.

Semaphores

InputByteBuffer – This will be created as a mutex and initialized to 0. It will be "given" by the RX operation of the UART ISR and "taken" by the RX Task.

OutputStringBuffer – This will be created as a mutex and initialized to 1. It will be "taken" by the TX Task and "given" by the TX operation of the UART ISR.

FreeRTOS API Calls

`xSemaphoreCreateMutex()`

`xSemaphoreGive()`

`xSemaphoreTake()`

`xSemaphoreGiveFromISR()`

`xSemaphoreTakeFromISR()`

Additional Considerations

Additions

`#include semphr.h`, where necessary to reference semaphore APIs.

Add the following `#define` FreeRTOSConfig.h

```
#define configCOMMAND_INT_MAX_OUTPUT_SIZE 300
```

Change the following `#define` FreeRTOSConfig.h so that the sample code can run `prvTaskStatsCommand()`

```
#define configUSE_TRACE_FACILITY   1
```