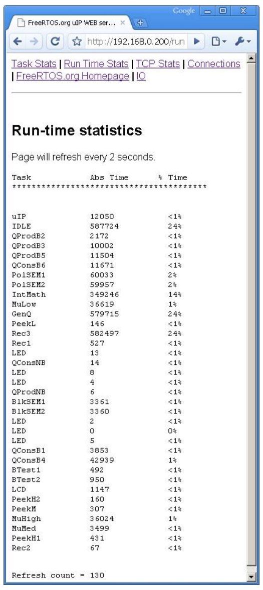
Run Time Statistics

Description

Click to enlarge


FreeRTOS.org uIP WEB ser...
http://192.168.0.200/run
Task Stats | Run Time Stats | TCP Stats | Connections | FreeRTOS.org Homepage | IO

Run-time statistics

Page will refresh every 2 seconds.

```
Task                Abs Time        % Time
****************************************


uIP                 12050            <1%
IDLE                587724           24%
QProdB2             2172             <1%
QProdB3             10002            <1%
QProdB5             11504            <1%
QConsB6             11671            <1%
PolSEM1             60033            2%
PolSEM2             59957            2%
IntMath             349246           14%
MuLow               36619            1%
GenQ                579715           24%
PeekL               146              <1%
Rec3                582497           24%
Rec1                527              <1%
LED                 13               <1%
QConsNB             14               <1%
LED                 8                <1%
LED                 4                <1%
QProdNB             6                <1%
BlkSEM1             3361             <1%
BlkSEM2             3360             <1%
LED                 2                <1%
LED                 0                0%
LED                 5                <1%
QConsB1             3853             <1%
QConsB4             42939            1%
BTest1              492              <1%
BTest2              950              <1%
LCD                 1147             <1%
PeekH2              160              <1%
PeekM               307              <1%
MuHigh              36024            1%
MuMed               3499             <1%
PeekH1              431              <1%
Rec2                67               <1%


Refresh count = 130
```

FreeRTOS can optionally collect information on the amount of processing time that has been used by each task. The vTaskGetRunTimeStats() API function can then be used to present this information in a tabular format, as shown on the right.

Two values are given for each task:

1. Abs Time (absolute time)

   This is the total 'time' that the task has actually been executing (the total time that the task has been in the Running state). It is up to the user to select a suitable time base for their application.

2. % Time (percentage time)

   This shows essentially the same information but as a percentage of the total processing time rather than as an absolute time.

Configuration and Usage

Three macros are required. These can be defined in FreeRTOSConfig.h.

1. configGENERATE_RUN_TIME_STATS

   Collection of run time statistics is enabled by #defining configGENERATE_RUN_TIME_STATS as 1. Once this has been set the other two macros must also be defined to achieve a successful compilation.

2. portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()

   The run time statistics time base needs to have a higher resolution than the tick interrupt - otherwise the statistics may be too inaccurate to be truly useful. It is recommended to make the time base between 10 and 100 times faster than the tick interrupt. The faster the time base the more accurate the statistics will be - but also the sooner the timer value will overflow.

   If configGENERATE_RUN_TIME_STATS is defined as 1 then the RTOS kernel will automatically call portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() as it is started (it is called from within the vTaskStartScheduler() API function). It is intended that the application designer uses the macro to configure a suitable time base. Some examples are provided below.

3. portGET_RUN_TIME_COUNTER_VALUE()

   This macro should just return the current 'time', as configured by portCONFIGURE_TIMER_FOR_RUN_TIME_STATS(). Again some examples are provided below.

The vTaskGetRunTimeStats() API function is used to retrieve the gathered statistics.

Examples

[Amongst many others] The LPC17xx LPCXpresso and the LM3Sxxxx Eclipse web server demo applications are configured to generate run time stats.

LM3Sxxxx example

The LM3Sxxxx Eclipse demo application already includes a 20KHz timer test. The interrupt handler was updated to simply increment a variable called ulHighFrequencyTimerTicks each time it executed. portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() simply sets this variable to 0 and portGET_RUN_TIME_COUNTER_VALUE() returns its value. To implement this the following few lines were added to FreeRTOSConfig.h:

```
extern volatile unsigned long ulHighFrequencyTimerTicks;
/* ulHighFrequencyTimerTicks is already being incremented at 20KHz.  Just set
its value back to 0. */
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() ( ulHighFrequencyTimerTicks = 0UL )
#define portGET_RUN_TIME_COUNTER_VALUE()        ulHighFrequencyTimerTicks
```

LPC17xx example

The LPC17xx demo application does not include the high frequency interrupt test, so portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() was used to configure the timer 0 peripheral to generate the time base. portGET_RUN_TIME_COUNTER_VALUE() simply returns the current timer 0 counter value. This was implemented using the following functions and macros.

```
/* Defined in main.c. */
void vConfigureTimerForRunTimeStats( void )
{
const unsigned long TCR_COUNT_RESET = 2,
                    CTCR_CTM_TIMER = 0x00,
                    TCR_COUNT_ENABLE = 0x01;

    /* Power up and feed the timer with a clock. */
    PCONP |= 0x02UL;
    PCLKSEL0 = (PCLKSEL0 & (~(0x3<<2))) | (0x01 << 2);

    /* Reset Timer 0 */
    T0TCR = TCR_COUNT_RESET;
    /* Just count up. */
    T0CTCR = CTCR_CTM_TIMER;

    /* Prescale to a frequency that is good enough to get a decent resolution,
    but not too fast so as to overflow all the time. */
    T0PR =  ( configCPU_CLOCK_HZ / 10000UL ) - 1UL;

    /* Start the counter. */
    T0TCR = TCR_COUNT_ENABLE;
}

/* Defined in FreeRTOSConfig.h. */
extern void vConfigureTimerForRunTimeStats( void );
#define portCONFIGURE_TIMER_FOR_RUN_TIME_STATS() vConfigureTimerForRunTimeStats()
#define portGET_RUN_TIME_COUNTER_VALUE() T0TC
```