Blocos de construção do projeto

6 minutos

Introdução

Vamos examinar os blocos de construção do código que são usados no projeto.

Bloco de construção 1

Declarações, definições e protótipos

Usamos este código para criar declarações, definições e protótipos:

C

```c
/* ProjectPriorityInversion.c   Priority Inversion Problem

   Create three threads, one byte pool, one mutex, two timers.
   This project contains a Priority Inversion problem;
   your task is to run it as-is and record observations.
   Then eliminate Priority Inversion by first incorporating
   Priority Inheritance, and then Preemption-Threshold,
   each time recording your results. */

   /**************************************************/
   /*    Declarations, Definitions, and Prototypes     */
   /**************************************************/

#include   "tx_api.h"
#include   <stdio.h>

#define     STACK_SIZE         1024
#define     BYTE_POOL_SIZE     9120
#define     DISPLAY_INTERVAL   5001
#define     UPDATE_INTERVAL     200

/* Define the ThreadX object control blocks */

TX_THREAD       Urgent_thread;
TX_THREAD       Important_thread;
TX_THREAD       Routine_thread;

TX_MUTEX        my_mutex;
```

```
TX_BYTE_POOL   my_byte_pool;
TX_TIMER       stats_timer, update_timer;

/* Define variables for Routine thread performance info */
ULONG   resumptions_Routine;
ULONG   suspensions_Routine;
ULONG   solicited_preemptions_Routine;

/* Define variables for Urgent thread performance info */
ULONG   resumptions_Urgent;
ULONG   suspensions_Urgent;

/* Define variables for use with mutex performance information */
ULONG mutex_puts;
ULONG mutex_gets;

/* Define variable for time analysis */
ULONG current_time;

/* Define prototypes */
void    Urgent_thread_entry(ULONG thread_input);
void    Important_thread_entry(ULONG thread_input);
void    Routine_thread_entry(ULONG thread_input);
void    print_stats(ULONG), print_update(ULONG);
```

> **Observação**
>
> Você não fará nenhuma alteração nesse código ao trabalhar com o arquivo de projeto no Visual Studio.

Bloco de construção 2

Ponto de entrada principal

Usamos este código para o ponto de entrada principal.

C

```
/*****************************************************/
/*               Main Entry Point                    */
/*****************************************************/
/* Define main entry point.  */
int main()
{
    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
}
```

> **Observação**
>
> Esse bloco de construção é idêntico aos projetos em módulos anteriores. Não há necessidade de alteração nesse código ao trabalhar com o arquivo de projeto no Visual Studio.

Bloco de construção 3

Definições de aplicativo

Usamos este código para criar definições de aplicativo:

C

```
/**************************************************/
/*             Application Definitions            */
/**************************************************/
/* Define what the initial system looks like.  */
void    tx_application_define(void* first_unused_memory)
{
    CHAR* Urgent_stack_ptr, * Important_stack_ptr,
        * Routine_stack_ptr;

    /* Create a memory byte pool from which to allocate
       the thread stacks */
    tx_byte_pool_create(&my_byte_pool, "my_byte_pool",
        first_unused_memory, BYTE_POOL_SIZE);

    /* Put system definition stuff in here, e.g., thread
       creates and other assorted create information */

    /* Allocate the stack for the Urgent_thread */
    tx_byte_allocate(&my_byte_pool, (VOID**)&Urgent_stack_ptr,
        STACK_SIZE, TX_NO_WAIT);

    /* Create the Urgent_thread.  */
    tx_thread_create(&Urgent_thread, "Urgent_thread",
        Urgent_thread_entry, 0, Urgent_stack_ptr,
        STACK_SIZE, 10, 10, TX_NO_TIME_SLICE,
        TX_AUTO_START);

    /* Allocate the stack for the Important_thread.  */
    tx_byte_allocate(&my_byte_pool, (VOID**)&Important_stack_ptr,
        STACK_SIZE, TX_NO_WAIT);

    /* Create the Important_thread.  */
    tx_thread_create(&Important_thread, "Important_thread",
        Important_thread_entry, 0, Important_stack_ptr,
```

```
        STACK_SIZE, 15, 15, TX_NO_TIME_SLICE,
        TX_AUTO_START);

    /* Allocate the stack for the Routine_thread.  */
    tx_byte_allocate(&my_byte_pool, (VOID**)&Routine_stack_ptr,
        STACK_SIZE, TX_NO_WAIT);

    /* Create the Routine_thread.
       **** for PREEMPTION-THRESHOLD, change 20,20 to 20,10  */
    tx_thread_create(&Routine_thread, "Routine_thread",
        Routine_thread_entry, 1, Routine_stack_ptr,
        STACK_SIZE, 20, 20,
        TX_NO_TIME_SLICE, TX_AUTO_START);

    /* Create the mutex used by both threads
       **** for PRIORITY INHERITANCE change to TX_INHERIT  */
    tx_mutex_create(&my_mutex, "my_mutex", TX_NO_INHERIT);

    /* Create and activate the display timer */
    tx_timer_create(&stats_timer, "stats_timer", print_stats,
        0x1234, DISPLAY_INTERVAL, 0, TX_AUTO_ACTIVATE);

    /* Create and activate the update timer */
    tx_timer_create(&update_timer, "update_timer", print_update,
        0x1234, UPDATE_INTERVAL, UPDATE_INTERVAL, TX_AUTO_ACTIVATE);
}
```

Observação

Na próxima unidade, faça essas alterações nas linhas de código realçadas:

- Empregue a herança de prioridades para evitar o problema de inversão de prioridades. Quando o mutex foi criado, a herança de prioridades não estava habilitada. Para habilitar a herança de prioridades, você altera a opção mutex de TX_NO_INHERIT para TX_INHERIT. Depois de experimentar essa alteração e registrar seus resultados, você pode alterar a opção de volta para TX_NO_INHERIT.
- Quando o thread Rotina foi criado, o limite de preempção foi definido como 20. Na próxima unidade, você vai experimentar a alteração do limite de preempção para 10. Em seguida, você realizará experimentos e registrará seus resultados

Bloco de construção 4

Não há necessidade de alteração no código a seguir ao trabalhar com o arquivo de projeto no Visual Studio.

Função de entrada do thread Urgente

Usamos este código para criar a função de entrada do thread urgente:

C

```
/****************************************************/
/*              Function Definitions                */
/****************************************************/
/* Entry function definition of the Urgent thread
   it has the highest priority */
void  Urgent_thread_entry(ULONG thread_input)
{
    while (1)
    {
        /* Processing, then Urgent Thread needs mutex */
        tx_thread_sleep(4);

        /* Get the mutex with suspension */
        tx_mutex_get(&my_mutex, TX_WAIT_FOREVER);
        tx_thread_sleep(3);

        /* Release the mutex */
        tx_mutex_put(&my_mutex);
    }
}
```

Função de entrada do thread Importante

Usaremos este código para criar a função de entrada do thread Importante:

C

```
/****************************************************/
/* Entry function definition of the Important_thread
   it has the medium-level priority */
void  Important_thread_entry(ULONG thread_input)
{
    ULONG kount;
    char  letter;
    while (1)
    {
        /* Important thread potentially causes priority inversion */

        /* Processing time */
        tx_thread_sleep(3);

        /* simulate work by Important thread  */
        for (kount = 1; kount < 100000000; kount++) letter = 'A';
```

```
    }
}
```

Função de entrada do thread de rotina

Usamos este código para criar a função de entrada do thread de rotina:

C

```
/****************************************************/
/* Entry function definition of the Routine_thread
   it has the lowest priority */
void  Routine_thread_entry(ULONG thread_input)
{
    CHAR  letter;
    ULONG kount;
    while (1)
    {
        /* Routine thread and Urgent thread compete for the mutex. */

        /* Get the mutex with suspension */
        tx_mutex_get(&my_mutex, TX_WAIT_FOREVER);

        /* Simulate work by Routine thread */
        for (kount = 1; kount < 100000000; kount++) letter = 'A';

        /* Release the mutex  */
        tx_mutex_put(&my_mutex);

        /* Processing time */
        tx_thread_sleep(1);
    }
}
```

Função de temporizador de aplicativo print_stats

Usamos este código para criar a função de temporizador de aplicativo print_stats:

C

```
/****************************************************/
/* print statistics at specified times */
void print_stats(ULONG invalue)
{
    /* Retrieve performance information on Routine thread */
    tx_thread_performance_info_get(&Routine_thread, &resumptions_Routine,
&suspensions_Routine,
        &solicited_preemptions_Routine, TX_NULL, TX_NULL,
        TX_NULL, TX_NULL, TX_NULL, TX_NULL, TX_NULL);
```

```
    /* Retrieve performance information on Urgent thread */
    tx_thread_performance_info_get(&Urgent_thread, &resumptions_Urgent,
&suspensions_Urgent,
        TX_NULL, TX_NULL, TX_NULL, TX_NULL, TX_NULL, TX_NULL, TX_NULL, TX_NULL);

    /* Retrieve performance information on my_mutex */
    tx_mutex_performance_info_get(&my_mutex, &mutex_puts, &mutex_gets, TX_NULL,
        TX_NULL, TX_NULL, TX_NULL);

    printf("\nProjectPriorityInversion: 3 threads, 1 byte pool, 1 mutex, and 2
timers\n\n");

    printf("   Routine thread resumptions: %lu\n", resumptions_Routine);
    printf("   Routine thread suspensions: %lu\n", suspensions_Routine);
    printf("Routine solicited_preemptions: %lu\n\n",
solicited_preemptions_Routine);

    printf("    Urgent thread resumptions: %lu\n", resumptions_Urgent);
    printf("    Urgent thread suspensions: %lu\n\n", suspensions_Urgent);

    printf("                  mutex puts: %lu\n", mutex_puts);
    printf("                  mutex gets: %lu\n\n", mutex_gets);

    tx_timer_deactivate(&update_timer);
}
```

Unidade seguinte: Exercício – compilar e depurar

Continuar >