

/*

FreeRTOS V8.2.3 - Copyright (C) 2015 Real Time Engineers Ltd.
All rights reserved

VISIT <http://www.FreeRTOS.org> TO ENSURE YOU ARE USING THE LATEST VERSION.

This file is part of the FreeRTOS distribution.

FreeRTOS is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License (version 2) as published by the Free Software Foundation >>>> AND MODIFIED BY <<<< the FreeRTOS exception.

```
*****
>>! NOTE: The modification to the GPL is included to allow you to  !<<
>>! distribute a combined work that includes FreeRTOS without being !<<
>>! obliged to provide the source code for proprietary components  !<<
>>! outside of the FreeRTOS kernel.                                !<<
*****
```

FreeRTOS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. Full license text is available on the following link: <http://www.freertos.org/a00114.html>

```
*****
*
*
* FreeRTOS provides completely free yet professionally developed,  *
* robust, strictly quality controlled, supported, and cross      *
* platform software that is more than just the market leader, it  *
* is the industry's de facto standard.                            *
*
*
* Help yourself get started quickly while simultaneously helping  *
* to support the FreeRTOS project by purchasing a FreeRTOS      *
* tutorial book, reference manual, or both:                      *
* http://www.FreeRTOS.org/Documentation                            *
*
*
*****
```

<http://www.FreeRTOS.org/FAQHelp.html> - Having a problem? Start by reading the FAQ page "My application does not run, what could be wrong?". Have you defined configASSERT()?

<http://www.FreeRTOS.org/support> - In return for receiving this top quality embedded software for free we request you assist our global community by participating in the support forum.

<http://www.FreeRTOS.org/training> - Investing in training allows your team to be as productive as possible as early as possible. Now you can receive FreeRTOS training directly from Richard Barry, CEO of Real Time Engineers Ltd, and the world's leading authority on the world's leading RTOS.

<http://www.FreeRTOS.org/plus> - A selection of FreeRTOS ecosystem products,

including FreeRTOS+Trace - an indispensable productivity tool, a DOS compatible FAT file system, and our tiny thread aware UDP/IP stack.

<http://www.FreeRTOS.org/labs> - Where new FreeRTOS products go to incubate. Come and try FreeRTOS+TCP, our new open source TCP/IP stack for FreeRTOS.

<http://www.OpenRTOS.com> - Real Time Engineers Ltd. license FreeRTOS to High Integrity Systems Ltd. to sell under the OpenRTOS brand. Low cost OpenRTOS licenses offer ticketed support, indemnification and commercial middleware.

<http://www.SafeRTOS.com> - High Integrity Systems also provide a safety engineered and independently SIL3 certified version for use in safety and mission critical applications that require provable dependability.

1 tab == 4 spaces!

```
*/
```

```
/* Standard includes. */
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
/* FreeRTOS kernel includes. */
```

```
#include "FreeRTOS.h"
```

```
#include "task.h"
```

```
#include <queue.h>
```

```
#include <timers.h>
```

```
#include <math.h>
```

```
/* This demo uses heap_5.c, and these constants define the sizes of the regions that make up the total heap. This is only done to provide an example of heap_5 being used as this demo could easily create one large heap region instead of multiple smaller heap regions - in which case heap_4.c would be the more appropriate choice. */
```

```
#define mainREGION_1_SIZE    3001
```

```
#define mainREGION_2_SIZE    18105
```

```
#define mainREGION_3_SIZE    1107
```

```
/*
```

```
* This demo uses heap_5.c, so start by defining some heap regions. This is
```

```
* only done to provide an example as this demo could easily create one large
```

```
* heap region instead of multiple smaller heap regions - in which case heap_4.c
```

```
* would be the more appropriate choice. No initialisation is required when
```

```
* heap_4.c is used.
```

```
*/
```

```
static void prvInitialiseHeap(void);
```

```
/*
```

```
* Prototypes for the standard FreeRTOS callback/hook functions implemented
```

```
* within this file.
```

```
*/
```

```

void vApplicationMallocFailedHook(void);
void vApplicationIdleHook(void);
void vApplicationStackOverflowHook(TaskHandle_t pxTask, char *pcTaskName);
void vApplicationTickHook(void);

/*
 * Writes trace data to a disk file when the trace recording is stopped.
 * This function will simply overwrite any trace files that already exist.
 */
static void prvSaveTraceFile(void);

/* The user trace event posted to the trace recording on each tick interrupt.
Note tick events will not appear in the trace recording with regular period
because this project runs in a Windows simulator, and does not therefore
exhibit deterministic behaviour. */
traceLabel xTickTraceUserEvent;
static portBASE_TYPE xTraceRunning = pdTRUE;

/*-----*/

#define SIZE 10
#define ROW SIZE
#define COL SIZE

//Communication_task count value flag.
uint32_t communication_flag = 0;

TickType_t two_hundred_ms = pdMS_TO_TICKS(200);
TickType_t one_thousand_ms = pdMS_TO_TICKS(1000);

//Task handles.
TaskHandle_t matrix_handle;
TaskHandle_t communication_handle;
TaskHandle_t arranger_handle;

//Created tasks.
static void matrix_task(void)
{
    int i;
    double **a = (double **)pvPortMalloc(ROW * sizeof(double *));
    for (i = 0; i < ROW; i++) a[i] = (double *)pvPortMalloc(COL * sizeof(double));
    double **b = (double **)pvPortMalloc(ROW * sizeof(double *));
    for (i = 0; i < ROW; i++) b[i] = (double *)pvPortMalloc(COL * sizeof(double));
    double **c = (double **)pvPortMalloc(ROW * sizeof(double *));
    for (i = 0; i < ROW; i++) c[i] = (double *)pvPortMalloc(COL * sizeof(double));

    double sum = 0.0;
    int j, k, l;

    for (i = 0; i < SIZE; i++)
    {

```

```

        for (j = 0; j < SIZE; j++)
        {
            a[i][j] = 1.5;
            b[i][j] = 2.6;
        }
    }

    while (1)
    {
        long simulationdelay;
        for (simulationdelay = 0; simulationdelay < 1000000000; simulationdelay++)
        ;
        for (i = 0; i < SIZE; i++)
        {
            for (j = 0; j < SIZE; j++)
            {
                sum = 0.0;
                for (k = 0; k < SIZE; k++)
                {
                    for (l = 0; l < 10; l++)
                    {
                        sum = sum + a[i][k] + b[k][j];
                    }
                }
                c[i][j] = sum;
                printf("Sum: %f\n", sum);
            }
        }

        vTaskDelay(100);
    }
}

static void communication_task(void *p)
{
    while (1)
    {

        communication_flag = xTaskGetTickCount();

        printf("Sending data...\n");
        fflush(stdout);
        vTaskDelay(100);
        printf("Data sent!\n");
        fflush(stdout);
        vTaskDelay(100);
    }
}

static void arranger_task(void)
{

```

```

TickType_t xLastWakeTime;

xLastWakeTime = xTaskGetTickCount();

for (;;)
{
    if (communication_flag < two_hundred_ms)
    {
        vTaskPrioritySet(communication_handle, 2);
    }
    else if (communication_flag > one_thousand_ms)
    {
        vTaskPrioritySet(communication_handle, 4);
        break;
    }

    vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(10));
}

}

int main(void)
{
    /* This demo uses heap_5.c, so start by defining some heap regions. This
    is only done to provide an example as this demo could easily create one
    large heap region instead of multiple smaller heap regions */
    prvInitialiseHeap();

    /* Initialise the trace recorder and create the label used to post user
    events to the trace recording on each tick interrupt. */
    vTraceInitTraceData();
    xTickTraceUserEvent = xTraceOpenLabel("tick");

    xTaskCreate((pdTASK_CODE)matrix_task, (signed char *)"Matrix", 1000, NULL, 3,
    &matrix_handle);
    xTaskCreate((pdTASK_CODE)communication_task, (signed char *)"Communication",
    configMINIMAL_STACK_SIZE, NULL, 1, &communication_handle);
    xTaskCreate((pdTASK_CODE)arranger_task, (signed char *)"Arranger", 1000, NULL, 4,
    &arranger_handle);

    //This starts the real-time scheduler
    vTaskStartScheduler();
    for (;;)
    return 0;
}
/*-----*/

void vApplicationMallocFailedHook(void)
{
    /* vApplicationMallocFailedHook() will only be called if
    configUSE_MALLOC_FAILED_HOOK is set to 1 in FreeRTOSConfig.h. It is a hook
    function that will get called if a call to pvPortMalloc() fails.
    pvPortMalloc() is called internally by the kernel whenever a task, queue,

```

timer or semaphore is created. It is also called by various parts of the demo application. If heap_1.c or heap_2.c are used, then the size of the heap available to pvPortMalloc() is defined by configTOTAL_HEAP_SIZE in FreeRTOSConfig.h, and the xPortGetFreeHeapSize() API function can be used to query the size of free heap space that remains (although it does not provide information on how the remaining heap might be fragmented). */
vAssertCalled(__LINE__, __FILE__);

```
}  
/*-----*/
```

```
void vApplicationIdleHook(void)  
{
```

```
}  
/*-----*/
```

```
void vApplicationStackOverflowHook(TaskHandle_t pxTask, char *pcTaskName)  
{
```

```
    (void)pcTaskName;  
    (void)pxTask;
```

```
    /* Run time stack overflow checking is performed if  
    configCHECK_FOR_STACK_OVERFLOW is defined to 1 or 2. This hook  
    function is called if a stack overflow is detected. */  
    vAssertCalled(__LINE__, __FILE__);
```

```
}  
/*-----*/
```

```
void vApplicationTickHook(void)  
{
```

```
}  
/*-----*/
```

```
void vAssertCalled(unsigned long ulLine, const char * const pcFileName)  
{
```

```
    static portBASE_TYPE xPrinted = pdFALSE;  
    volatile uint32_t ulSetToNonZeroInDebuggerToContinue = 0;
```

```
    /* Parameters are not used. */  
    (void)ulLine;  
    (void)pcFileName;
```

```
    printf("ASSERT! Line %d, file %s\r\n", ulLine, pcFileName);
```

```
    taskENTER_CRITICAL();  
    {
```

```
        /* Stop the trace recording. */  
        if (xPrinted == pdFALSE)  
        {
```

```
            xPrinted = pdTRUE;  
            if (xTraceRunning == pdTRUE)
```

```

        {
            vTraceStop();
            prvSaveTraceFile();
        }
    }

    /* You can step out of this function to debug the assertion by using
    the debugger to set ulSetToNonZeroInDebuggerToContinue to a non-zero
    value. */
    while (ulSetToNonZeroInDebuggerToContinue == 0)
    {
        __asm { NOP };
        __asm { NOP };
    }
}
taskEXIT_CRITICAL();
}
/*-----*/

static void prvSaveTraceFile(void)
{
    FILE* pxOutputFile;

    pxOutputFile = fopen("Trace.dump", "wb");

    if (pxOutputFile != NULL)
    {
        fwrite(RecorderDataPtr, sizeof(RecorderDataType), 1, pxOutputFile);
        fclose(pxOutputFile);
        printf("\r\nTrace output saved to Trace.dump\r\n");
    }
    else
    {
        printf("\r\nFailed to create trace dump file\r\n");
    }
}
/*-----*/

static void prvInitialiseHeap(void)
{
    /* This demo uses heap_5.c, so start by defining some heap regions. This is
    only done to provide an example as this demo could easily create one large heap
    region instead of multiple smaller heap regions - in which case heap_4.c would
    be the more appropriate choice. No initialisation is required when heap_4.c is
    used. The xHeapRegions structure requires the regions to be defined in order,
    so this just creates one big array, then populates the structure with offsets
    into the array - with gaps in between and messy alignment just for test
    purposes. */
    static uint8_t ucHeap[configTOTAL_HEAP_SIZE];
    volatile uint32_t ulAdditionalOffset = 19; /* Just to prevent 'condition is always true'
warnings in configASSERT(). */

```

```

const HeapRegion_t xHeapRegions[] =
{
    /* Start address with dummy offsets                                     Size */
    { ucHeap + 1,
mainREGION_1_SIZE },
    { ucHeap + 15 + mainREGION_1_SIZE,
mainREGION_2_SIZE },
    { ucHeap + 19 + mainREGION_1_SIZE + mainREGION_2_SIZE,
mainREGION_3_SIZE },
    { NULL, 0 }
};

/* Sanity check that the sizes and offsets defined actually fit into the
array. */
configASSERT((ulAdditionalOffset + mainREGION_1_SIZE + mainREGION_2_SIZE +
mainREGION_3_SIZE) < configTOTAL_HEAP_SIZE);

vPortDefineHeapRegions(xHeapRegions);
}
/*-----*/

```