

Universidad Tecnológica Nacional

Facultad Regional Avellaneda

Departamento de Ingeniería Electrónica

Cátedra: Técnicas Digitales III

|  |  |
| --- | --- |
| **SENSOR ULTRASÓNICO** | Trabajo Práctico INTEGRADOR:  Contenidos analizados:   * Cap1: Manejo de Tareas * Cap2: Manejo de Colas * Cap3: Gestión de Interrupciones / Semáforos * Cap4: Administración de Recursos / Exclusión * Cap5: Uso de Memoria   (Contenidos Integrados) |
| **Autores:** | Ricardo Alonso |
| Marcos Linardi |
| Angel Maggi |
|  |
|  | |
| **IMPORTANTE**:  Condición de Aprobación (Nota: 6): Sujeta a funcionamiento general del equipo con base en su funcionamiento mínimo y exposición oral.  Condición de Promoción (Nota: 7-10): Tener el 100% de la consigna desarrollada con informe y presentación oral. Todos los integrantes del grupo DEBEN estar presentes el día de la entrega del Parcial, de no ser así el parcial se debe recuperar en la instancia de recuperación. | |

**Sistemas Operativos en Tiempo Real: FREERTOS (Trabajo Práctico INT)**

**Parte Teórica:**

1. Desarrollar un programa ejemplo que permita comprender el concepto y uso de “Semáforos Mutex”.
2. Desarrollar un programa ejemplo que permita comprender el concepto y uso de “Inversión de Prioridad”. Como mínimo se deben utilizar dos tareas y una cola gestionando un recurso del microcontrolador.

* Explicar el concepto de “Herencia de Prioridad” utilizando diagramas gráficos que permitan comprenderlo.

1. Desarrollar un programa ejemplo que permita comprender el concepto de “Punto Muerto”. Como mínimo se deben utilizar dos tareas y una cola gestionando un recurso del microcontrolador.

**Parte Práctica:**

A). Realizar el siguiente trabajo práctico en la sección correspondiente al repositorio en GITHub “6.trabajos\_practicos” realizando un commit denominado “tp\_INT\_27-06-22”.

B). Se solicita realizar un sistema que permita medir distancia mediante un sensor ultrasónico. Se deben utilizar displays de 7 segmentos para indicar la distancia medida. El sistema debe tener dos alarmas; Distancia Mínima y Distancia Máxima (ambos parámetros deben ser programables). Se debe prever la necesidad de almacenamiento de los datos de Distancia Mínima y Distancia Máxima de las últimas 3 secuencias previas en la E2PROM o Módulo SD.

* El sistema debe realizarse con el uC LPC 1769 utilizando como Sistema Operativo, FreeRTOS. Como mínimo deben utilizarse dos tareas sincronizadas.
* Se debe llevar a cabo el montaje del sistema en placa experimental donde puedan ensayarse las características del mismo bajo los requerimientos solicitados.
* Se debe entregar el workspace de trabajo el día de entrega del parcial, con el cual se ensayará el funcionamiento del mismo.
* Se debe realizar un informe en formato IEEE con, como mínimo, las siguientes características:

1. Índice
2. Abstract
3. Fundamentación teórica de Hardware y Software
4. Alcance logrado
5. Diagrama en bloques (Completo y con detalles de interconexión de bloques)
6. Sobre el Hardware: se debe incluir las especificaciones técnicas, valores esperados, rangos de medición y precisión obtenida de cada módulo, sensor o dispositivo utilizado.
7. Sobre el Software: Se debe incluir una guía de código donde se especifique la relación entre las funciones, tareas y recursos del sistema operativo.

|  |  |
| --- | --- |
| **NOMBRE Y APELLIDO** | **CALIFICACIÓN EXPOSICIÓN ORAL** |
| **1.- Ricardo A. Alonso** |  |
| **2.- Marcos Linardi** |  |
| **3.- Angel Maggi** |  |

**Ejercicio N° 1:**

Desarrollar un programa ejemplo que permita comprender el concepto y uso de “Semáforos Mutex”.

Un mutex es un tipo especial de semáforo binario que se utiliza para controlar el acceso a un recurso que se comparte entre dos o más tareas. La palabra MUTEX proviene de "exclusión mutua".En este ejercicio se utiliza la UART del microcontrolador para enviar datos a una PC.

**#include** "chip.h"

**#include** "FreeRTOS.h"

**#include** "task.h"

**#include** "semphr.h"

**#include** "uart.h"

**static** **void** **Task1**(**void** \*pvParameters){

uint8\_t tx[] = "tecnicas";

**while**(1){

/\* Aseguro el recurso UART tomando el semaforo \*/

SemaphoreTake(mutex, portMAX\_DELAY);

**for**(uint32\_t i=0; i<**sizeof**(tx); i++){

**if**(tx[i] != '\0'){

/\* Envio un byte a la FIFO de UART \*/

Chip\_UART\_SendByte(LPC\_UART3, tx[i]);

/\* Esta es una demora forzada para hacer mas lenta la transmision por UART \*/

**for**(uint32\_t j=1000000; j; j--);

}

}

/\* Libero el semaforo una vez que termine de utilizar la UART \*/

xSemaphoreGive(mutex);

}

}

**static** **void** **Task2**(**void** \*pvParameters){

uint8\_t tx[] = "digitales3\n";

**while**(1){

/\* Aseguro el recurso UART tomando el semaforo \*/

xSemaphoreTake(mutex, portMAX\_DELAY);

**for**(uint32\_t i=0; i<**sizeof**(tx); i++){

**if**(tx[i] != '\0'){

/\* Envio un byte a la FIFO de UART \*/

Chip\_UART\_SendByte(LPC\_UART3, tx[i]);

/\* Esta es una demora forzada para hacer mas lenta la transmision por UART \*/

**for**(uint32\_t j=1000000; j; j--);

}

}

/\* Libero el semaforo una vez que termine de utilizar la UART \*/

xSemaphoreGive(mutex);

}

}

**int** **main**(**void**){

/\* Levanta la frecuencia del micro \*/

SystemCoreClockUpdate();

/\* Configuración inicial del micro \*/

Configuracion\_uC();

/\* Creacion del semaforo mutex \*/

mutex = xSemaphoreCreateMutex();

/\* Creacion de tareas \*/

xTaskCreate(Task1, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 1UL),

(xTaskHandle \*) NULL);

xTaskCreate(Task2, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 1UL),

(xTaskHandle \*) NULL);

/\* Inicia el scheduler \*/

vTaskStartScheduler();

/\* Nunca debería arribar aquí \*/

**return** 0;

}

**Ejercicio N° 2:**

a) Desarrollar un programa ejemplo que permita comprender el concepto y uso de “Inversión de Prioridad”. Como mínimo se deben utilizar dos tareas y una cola gestionando un recurso del microcontrolador.

b) Explicar el concepto de “Herencia de Prioridad” utilizando diagramas gráficos que permitan comprenderlo.

a) Este ejemplo es igual al anterior de transmisión de la Uart a una Pc, pero utiliza el concepto de inversión de prioridad. La Tarea 2, de mayor prioridad que la Tarea 1 tiene que esperar a que la Tarea 1, de menor prioridad, le ceda el control del mutex. Una tarea de mayor prioridad que se retrasó por una tarea de menor prioridad de esta forma, se llama "inversión de prioridad”.

**#include** "chip.h"

**#include** "FreeRTOS.h"

**#include** "task.h"

**#include** "queue.h"

**#include** "uart.h"

**static** **void** **Task1**(**void** \*pvParameters){

uint8\_t tx[] = "tecnicas";

uint32\_t dato=tx;

**while**(1){

/\* Envio a la cola la dirección de inicio de la cadena \*/

xQueueSendToBack(cola, &dato, portMAX\_DELAY);

}

}

**static** **void** **Task2**(**void** \*pvParameters){

uint8\_t tx[] = "digitales3\n";

uint32\_t dato=tx;

**while**(1){

/\* Envio a la cola la dirección de inicio de la cadena \*/

xQueueSendToBack(cola, &dato, portMAX\_DELAY);

}

}

**static** **void** **Task3**(**void** \*pvParameters){

uint8\_t \*tx;

uint32\_t dato;

uint32\_t i;

**while**(1){

/\* Recibo como dato la dirección de inicio de la cadena a transmitir \*/

xQueueReceive(cola, &dato, portMAX\_DELAY);

/\* Apunto el puntero a dicha dirección \*/

tx = dato;

i=0;

/\* Se que ambas cadenas termina con \0 asi que transmito los datos hasta encontrar dicho caracter \*/

**while**( tx[i]!='\0'){

Chip\_UART\_SendByte(LPC\_UART3, tx[i]);

i++;

**for**(uint32\_t j=1000000; j; j--);

}

}

}

**static** **void** **Task4**(**void** \*pvParameters){

vTaskDelay(500/portTICK\_RATE\_MS);

**while**(1);

}

**int** **main**(**void**){

/\* Levanta la frecuencia del micro \*/

SystemCoreClockUpdate();

/\* Configuración inicial del micro \*/

Configuracion\_uC();

cola = xQueueCreate(1,**sizeof**(uint32\_t));

/\* Creacion de tareas \*/

xTaskCreate(Task1, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 3UL),

(xTaskHandle \*) NULL);

xTaskCreate(Task2, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 1UL),

(xTaskHandle \*) NULL);

xTaskCreate(Task3, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 1UL),

(xTaskHandle \*) NULL);

xTaskCreate(Task4, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 2UL),

(xTaskHandle \*) NULL);

/\* Inicia el scheduler \*/

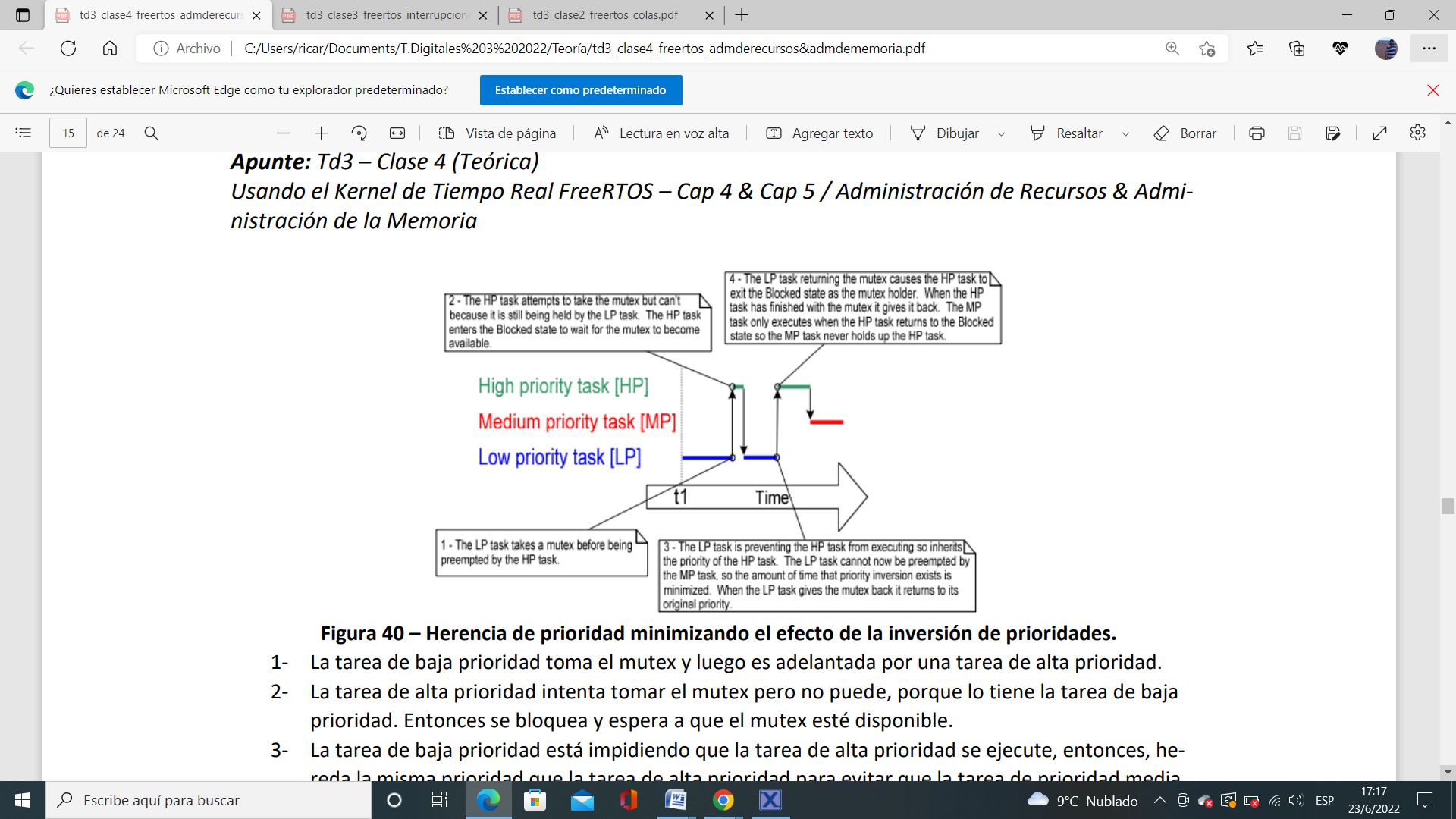
vTaskStartScheduler();

/\* Nunca debería arribar aquí \*/

**return** 0;

}

b) La herencia de prioridad consiste en elevar temporalmente la prioridad de la tarea que posee el mutex, al mismo nivel de prioridad de la tarea de más alta prioridad, que está tratando de obtener el mismo mutex. La tarea de baja prioridad que tiene el mutex "hereda" la prioridad de la tarea que está esperando poder tomar el mutex. La prioridad de la tarea que tenía el mutex se restablece automáticamente a su valor original cuando ésta ya lo devolvió.



1-La tarea de baja prioridad toma el mutex y luego es adelantada por una tarea de alta prioridad.

2-La tarea de alta prioridad intenta tomar el mutex pero no puede, porque lo tiene la tarea de baja prioridad. Entonces se bloquea y espera a que el mutex esté disponible.

3- La tarea de baja prioridad está impidiendo que la tarea de alta prioridad se ejecute, entonces, hereda la misma prioridad que la tarea de alta prioridad para evitar que la tarea de prioridad media pueda adelantársele. Con esto se logra minimizar el tiempo en el que existe la inversión de prioridad. Cuando la tarea de baja prioridad devuelve el mutex, esta recupera su prioridad original.

4- Cuando la tarea de baja prioridad devuelve el mutex, este es tomado por la tarea de alta prioridad y abandona el estado bloqueado. Cuando la tarea de alta prioridad termina de usar el mutex lo devuelve. La tarea de prioridad media solo se ejecutará cuando la tarea de alta prioridad pase a estar bloqueada nuevamente.

**Ejercicio N° 3:**

Desarrollar un programa ejemplo que permita comprender el concepto de “Punto Muerto”. Como mínimo se deben utilizar dos tareas y una cola gestionando un recurso del microcontrolador.

Un punto muerto es otro problema potencial del uso de semáforos mutex para la exclusión mutua. Se produce cuando dos tareas no pueden continuar porque ambos están a la espera de un recurso que está en manos de la otra. Considere el siguiente escenario donde la Tarea A y B ambas necesitan adquirir el mutex X y el mutex Y, con el fin de realizar una acción:

1-La tarea A se ejecuta y toma el mutex X.

2-La tarea B adelanta a la tarea A.

3-La tarea B toma el mutex Y, y luego intenta tomar el mutex X, pero no lo logra dado que lo tiene la tarea A. La tarea B entonces se bloquea, esperando a que el mutex X esté disponible.

4-La tarea A vuelve a ejecutarse e intenta tomar el mutex Y, pero no lo logra dado que la tarea B tiene tomado este mutex. Entonces la tarea A también se bloquea, esperando a que el mutex Y esté disponible. Al final de este escenario, la Tarea A está esperando al mutex tomado por la Tarea B, y Tarea B está esperando a un mutex retenido por la Tarea A. Se produce entonces un interbloqueo, porque ninguna tarea puede proceder.

Se puede evitar esto con el concepto de utilizar una “ tarea guardiana “

Las tareas guardianas proporcionan un método limpio de implementación de la exclusión mutua sin la preocupación de la inversión de prioridad o punto muerto. Una tarea guardiana es una tarea que tiene la propiedad exclusiva de un recurso. Sólo la tarea guardiana tiene permitido el acceso al recurso, cualquier otra tarea que tenga la necesidad de acceder a este recurso, sólo pueden hacerlo indirectamente mediante el uso de los servicios del guardián.

**#include** "chip.h"

**#include** "FreeRTOS.h"

**#include** "task.h"

**#include** "queue.h"

**#include** "uart.h"

**static** **void** **Task1**(**void** \*pvParameters){

uint8\_t tx[] = "tecnicas";

uint32\_t dato=tx;

**while**(1){

/\* Envio a la cola la dirección de inicio de la cadena \*/

xQueueSendToBack(cola, &dato, portMAX\_DELAY);

}

}

**static** **void** **Task2**(**void** \*pvParameters){

uint8\_t tx[] = "digitales3\n";

uint32\_t dato=tx;

**while**(1){ /\* Envio a la cola la dirección de inicio de la cadena \*/

xQueueSendToBack(cola, &dato, portMAX\_DELAY);

}

}

/\* Tarea Guardiana \*/

**static** **void** **Guardian**(**void** \*pvParameters){

uint8\_t \*tx;

uint32\_t dato;

uint32\_t i;

**while**(1){ /\* Recibo como dato la dirección de inicio de la cadena a transmitir \*/

xQueueReceive(cola, &dato, portMAX\_DELAY);

/\* Apunto el puntero a dicha dirección \*/

tx = dato;

i=0;

/\* Se que ambas cadenas termina con \0 asi que transmito los datos hasta encontrar dicho caracter \*/

**while**( tx[i]!='\0'){

Chip\_UART\_SendByte(LPC\_UART3, tx[i]);

i++;

**for**(uint32\_t j=1000000; j; j--);

}

}

}

**int** **main**(**void**){

/\* Levanta la frecuencia del micro \*/

SystemCoreClockUpdate();

/\* Configuración inicial del micro \*/

Configuracion\_uC();

cola = xQueueCreate(1,**sizeof**(uint32\_t));

/\* Creacion de tareas \*/

xTaskCreate(Task1, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 1UL),

(xTaskHandle \*) NULL);

xTaskCreate(Task2, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 1UL),

(xTaskHandle \*) NULL);

xTaskCreate(Guardian, (**char** \*) "",

configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 1UL),

(xTaskHandle \*) NULL);

/\* Inicia el scheduler \*/

vTaskStartScheduler();

/\* Nunca debería arribar aquí \*/

**return** 0;

}