

Carrera de Especialización en Sistemas Embebidos

Sistemas Operativos en Tiempo Real

Clase 5: Colas



Asociación Civil para la Investigación,
Promoción y Desarrollo de los
Sistemas Electrónicos Embebidos



Comunicación es datos entre contextos

- En un modelo de funcionamiento **productor/consumidor** de datos:

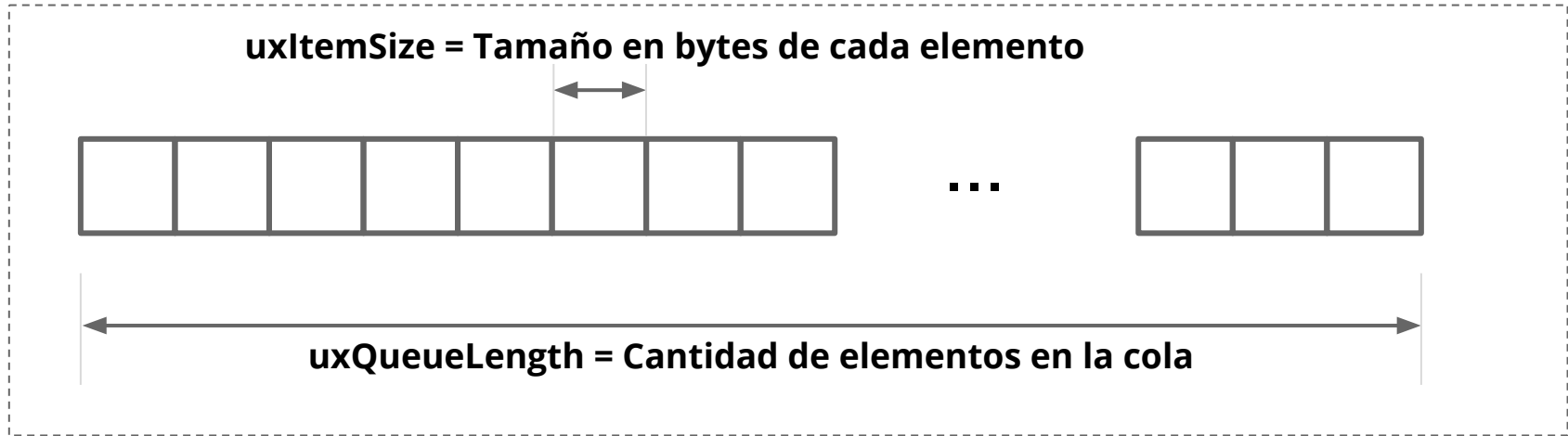
¿Como se puede enviar esos datos desde una tarea hacia otra ?

- Cuando los datos se producen más rápido que lo que se los consume (momentáneamente), se necesita un **almacenamiento temporario** de los mismos.
- Se podría utilizar semáforos, con algún algoritmo que utilice memoria global, pero eso significa más complejidad a la aplicación y carga al programador.

- FreeRTOS permite instanciar colas de **distintas cantidades** de elementos y **de distinto tamaño** de elementos.
- Incluye **mecanismo de sincronización**:
 - Cuando una tarea intenta obtener un dato de una cola, pero la cola está vacía se bloqueará durante un tiempo o para siempre.
 - Cuando un contexto intenta agregar un dato de una cola, pero la cola está llena se bloqueará durante un tiempo o para siempre.
- Agregar o Quitar un elemento de la cola, son operaciones **POR COPIA**.
- La cola funciona según un algoritmo de cola circular, **FIFO**.

- Creación de colas:

```
QueueHandle_t mi_cola = xQueueCreate( UBaseType_t uxQueueLength,  
                                       UBaseType_t uxItemSize );
```



API: Creación de Cola (Ejemplos)



```
QueueHandle_t cola_1 = xQueueCreate( 4 , sizeof(uint8_t) );
```

Crearé el objeto **cola_1** que permita almacenar de manera circular hasta 4 datos de tipo entero no signado de 8 bits.

```
QueueHandle_t cola_2 = xQueueCreate( 10, sizeof(tEstructura) );
```

Crearé el objeto **cola_2** que permita almacenar de manera circular hasta 10 elementos con la estructura **tEstructura**

Si **tEstructura** se define como sigue cada elemento ocupará 16(*) bytes.

```
typedef struct
{
    uint16_t  campo1;
    uint32_t  campo2;
    char      campo3[10];
} tEstructura;
```

***OJO!** Depende de la alineación del bus del CPU y del compilador.

API: Creación de Cola (Ejemplos)

```
QueueHandle_t cola_1 = xQueueCreate( 4 , sizeof(uint8_t) );
```

Crearé el objeto **cola_1** que permita almacenar de manera circular hasta 4 datos de tipo entero no signado de 8 bits.

```
QueueHandle_t cola_2 = xQueueCreate( 10, sizeof(tEstructura
```

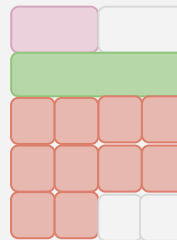
Crearé el objeto **cola_2** que permita con la estructura **tEstructura**

Si **tEstructura** se define como sig

```
typedef struct
{
    uint16_t  campo1;
    uint32_t  campo2;
    char      campo3[10];
} tEstructura;
```

```
{
  Uint16_t  -> 2 bytes
  Uint32_t  -> 4 bytes
  char*10   -> 10 bytes
}
```

16 bytes



18 bytes

Instancia así por defecto, por el bus y el set de instrucciones (32 bits)

compilador.

API: Uso de Cola: Agregar Elemento



```
BaseType_t xQueueSend( QueueHandle_t xQueue,  
                        const void*   pvItemToQueue,  
                        TickType_t    xTicksToWait );
```

xQueue = Nombre de la cola con el valor devuelto por **xQueueCreate**

pvItemToQueue = Dirección de memoria del elemento a agregar.

xTicksToWait = Tiempo en ticks que como máximo deberá bloquearse la tarea en caso de que la cola esté llena.

La funcion retorna:

pdTRUE: Si el elemento se envió correctamente

pdFALSE: Si el elemento no se envió, y el llamado dio timeout.

- Regla de funcionamiento: Cuando un contexto intenta agregar un dato de una cola, pero la cola está llena se bloqueará durante un tiempo o para siempre.

API: Uso de Cola: Agregar Elemento



```
BaseType_t xQueueSend( QueueHandle_t xQueue,  
                        const void* pvItemToQueue,  
                        TickType_t xTicksToWait );
```

xQueue = Nombre de la cola con el valor devuelto por xQueueCreate

pvItemToQueue = Dirección de memoria

xTicksToWait = Tiempo en ticks que co
tarea en caso de que la cola esté llena

La función retorna:

pdTRUE: Si el elemento se envió cor

pdFALSE: Si el elemento no se envió

Dato = 1

xQueue

2	3	4			
---	---	---	--	--	--

xQueueSend(Cola, &Dato, Infinito);

xQueue

2	3	4	1		
---	---	---	---	--	--

- Regla de funcionamiento: Cuando se intenta agregar un dato de una cola, pero la cola está llena se bloqueará durante un tiempo o para siempre.

API: Uso de Cola: Quitar Elemento



```
BaseType_t xQueueReceive( QueueHandle_t xQueue,  
                           void* pvBuffer,  
                           TickType_t xTicksToWait );
```

xQueue = Nombre del objeto con el valor devuelto por **xQueueCreate**

pvBuffer = Dirección de memoria del lugar en donde se almacenará el elemento removido.

xTicksToWait = Tiempo en ticks que como máximo deberá bloquearse la tarea en caso de que la cola esté vacía.

La funcion retorna:

pdTRUE: Si el elemento se recibió correctamente

pdFALSE: Si el elemento no se recibió , y el llamado dio timeout.

- Regla de funcionamiento: Cuando una tarea intenta obtener un dato de una cola, pero la cola está vacía se bloqueará durante un tiempo o para siempre.

API: Uso de Cola: Quitar Elemento



```
 BaseType_t xQueueReceive( QueueHandle_t xQueue,  
                          void* pvBuffer,  
                          TickType_t xTicksToWait );
```

xQueue = Nombre del objeto con el valor devuelto por xQueueCreate

pvBuffer = Dirección de memoria del elemento removido.

xTicksToWait = Tiempo en ticks que col que la cola esté vacía.

La función retorna:

pdTRUE: Si el elemento se recibió co

pdFALSE: Si el elemento no se recib

Dato = 1;

xQueue

2	3	4				
---	---	---	--	--	--	--

xQueueReceive(Cola, &Dato, Infinito);

xQueue

3	4					
---	---	--	--	--	--	--

Dato

2

- Regla de funcionamiento: Cuando una tarea intenta obtener un dato de una cola, pero la cola está vacía se bloqueará durante un tiempo o para siempre.

- xQueuePeek: Consulta un elemento de la cola sin removerlo. Opera igual que xQueueReceive.
- uxQueueMessagesWaiting: Devuelve la cantidad de mensajes esperando ser removidos de la cola.
- uxQueueSpacesAvailable: Devuelve la cantidad de espacios para mensajes, disponibles en la cola.
- vQueueDelete: Destruye la cola, liberando toda memoria dinámica que haya necesitado cuando se llamó a xQueueCreate
- xQueueReset: Restablece la cola vaciandola, volviendo a su estado inicial (cuando se llamó a xQueueCreate)

API: Uso de Cola: Otras funciones



- xQueuePeek: Consulta un elemento de la cola sin removerlo. Opera igual que xQueueReceive.

- uxQueueMessagesWaiting: Devuelve la cantidad de mensajes esperando ser removidos de la cola.
- uxQueueSpacesAvailable: Devuelve la cantidad de espacios disponibles en la cola.
- vQueueDelete: Destruye la cola cuando se llamó a vQueueCreate.
- xQueueReset: Restablece la cola a su estado inicial (cuando se llamó a xQueueCreate).

Dato = 1;

xQueue 2 3 4    

xQueuePeek(Cola, &Dato, Infinito);

xQueue 2 3 4    

Dato 2

API: Uso de Cola: Otras funciones



- xQueuePeek: Consulta un elemento de la cola sin removerlo. Opera igual que xQueueReceive.
- uxQueueMessagesWaiting: Devuelve la cantidad de mensajes esperando ser removidos de la cola.
- uxQueueSpacesAvailable: Devuelve la cantidad de espacios para mensajes, disponibles en la cola.
- vQueueDelete: Destruye la cola. No se puede utilizar si la cola se necesita cuando se llamó a xQueueCreate o xQueueCreateStatic.
- xQueueReset: Restablece la cola a su estado inicial (cuando se llamó a xQueueCreate)

xQueue 2 3 4

xQueueMessagesWaiting(Cola); -> 3

API: Uso de Cola: Otras funciones



- xQueuePeek: Consulta un elemento de la cola sin removerlo. Opera igual que xQueueReceive.
- uxQueueMessagesWaiting: Devuelve la cantidad de mensajes esperando ser removidos de la cola.
- uxQueueSpacesAvailable: Devuelve la cantidad de espacios para mensajes, disponibles en la cola.

- vQueueDelete: Destruye la cola, liberando toda memoria dinámica que haya necesitado cuando se llamó a xQueueCreate.
- xQueueReset: Restablece la cola (cuando se llamó a xQueueCreate).

xQueue 2 3 4

uxQueueSpacesAvailable(Cola); -> 4

API: Uso de Cola: Otras funciones



- xQueuePeek: Consulta un elemento de la cola sin que xQueueReceive.
- uxQueueMessagesWaiting: Devuelve el número de mensajes removidos de la cola.
- uxQueueSpacesAvailable: Devuelve el número de espacios disponibles en la cola.
- vQueueDelete: Destruye la cola, liberando toda memoria dinámica que haya necesitado cuando se llamó a xQueueCreate
- xQueueReset: Restablece la cola vaciándola, volviendo a su estado inicial (cuando se llamó a xQueueCreate)

Dato = 1;

xQueue

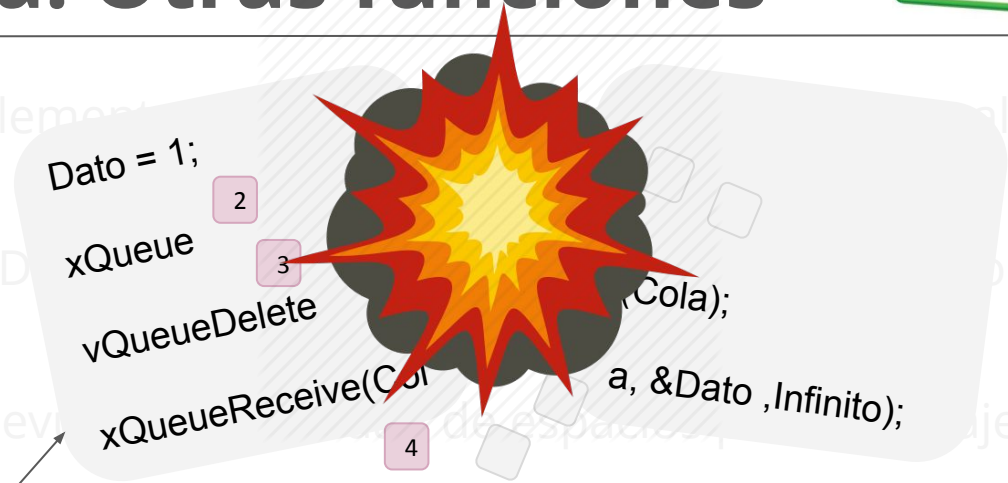
2	3	4				
---	---	---	--	--	--	--

vQueueDelete(Cola);

xQueueSend(Cola, &Dato, Infinito);

API: Uso de Cola: Otras funciones

- xQueuePeek: Consulta un elemento de la cola sin que xQueueReceive.
- uxQueueMessagesWaiting: Devuelve el número de mensajes removidos de la cola.
- uxQueueSpacesAvailable: Devuelve el número de espacios disponibles en la cola.



- **vQueueDelete**: Destruye la cola, liberando toda memoria dinámica que haya necesitado cuando se llamó a xQueueCreate
- xQueueReset: Restablece la cola vaciandola, volviendo a su estado inicial (cuando se llamó a xQueueCreate)

API: Uso de Cola: Otras funciones



- xQueuePeek: Consulta un elemento de la cola sin que xQueueReceive.
- uxQueueMessagesWaiting: Devuelve el número de mensajes pendientes de ser removidos de la cola.
- uxQueueSpacesAvailable: Devuelve el número de espacios disponibles en la cola.
- vQueueDelete: Destruye la cola, liberando toda memoria dinámica que haya necesitado cuando se llamó a xQueueCreate

Dato = 1;

xQueue

2	3	4				
---	---	---	--	--	--	--

xQueueReset(Cola);

xQueue

--	--	--	--	--	--	--

uxQueueSpacesAvailable(Cola); -> 7

--

- xQueueReset: Restablece la cola vaciandola, volviendo a su estado inicial (cuando se llamó a xQueueCreate)

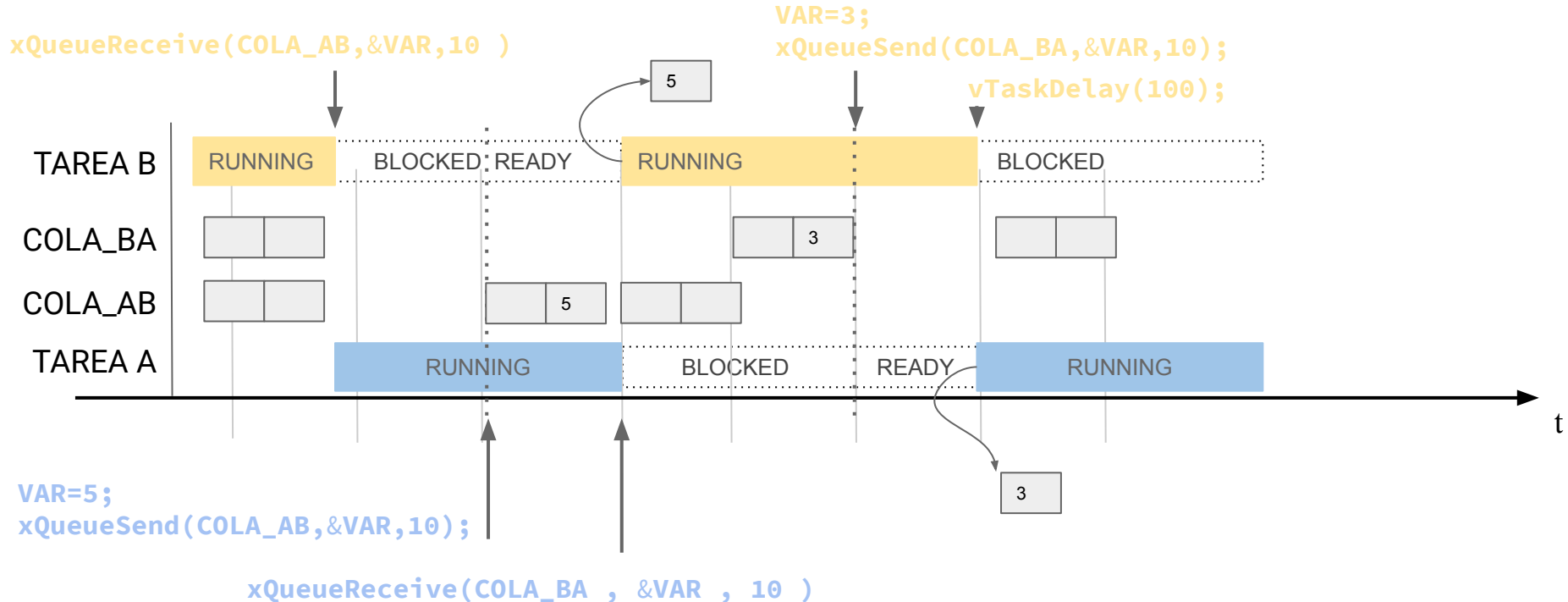
Uso de elementos por referencia



- Si se desea crear un cola con **elementos MUY grandes**, pasarlos por copia sería un **desperdicio de tiempo**. Podría utilizarse una cola cuyos elementos sean punteros simplemente apilar referencias a elementos globales (o locales a la tarea)
- Cuidado: los elementos deben mantener su validez hasta que sean consumidos.
- Hay que **gestionar** los elementos en **memoria** "manualmente" lo que conlleva un trabajo extra al programador.

Ejemplo

- Dos tareas desean comunicar datos de manera bidireccional.



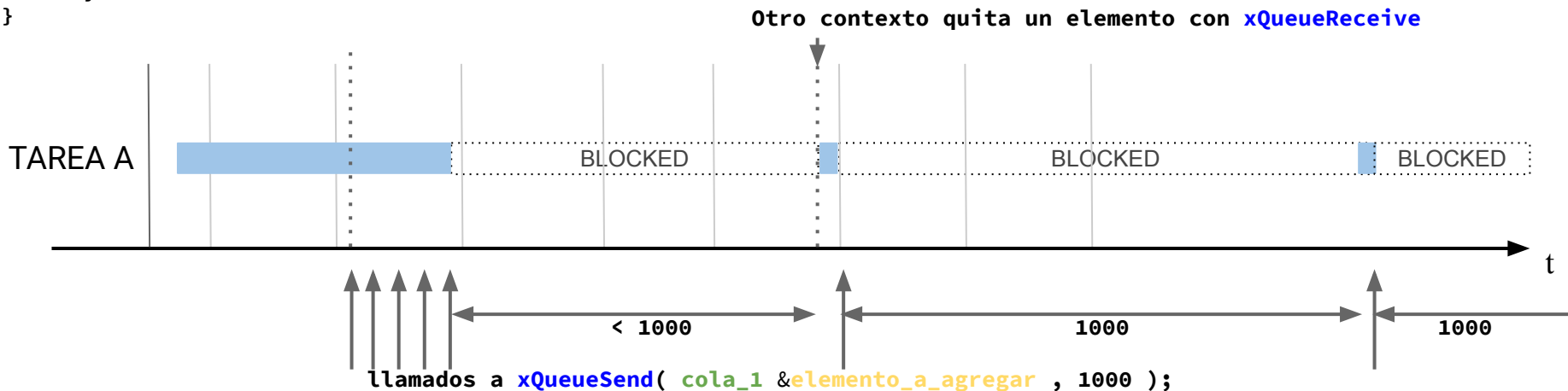
Ejemplo



`QueueHandle_t cola_1 = xQueueCreate(4 , sizeof(uint8_t));` //LOS ELEMENTOS DE COMUNICACIÓN ENTRE TAREAS DEBEN SER GLOBALES

```
void TareaA( void* params )
{
    BaseType_t rv;
    uint8_t elemento_a_agregar = 20 ;

    while(1)
    {
        /* RESTO DE CÓDIGO */
        rv = xQueueSend( cola_1 , &elemento_a_agregar , 1000 );
        elemento_a_agregar++;
        /* RESTO DE CÓDIGO */
    }
}
```



Ejemplo



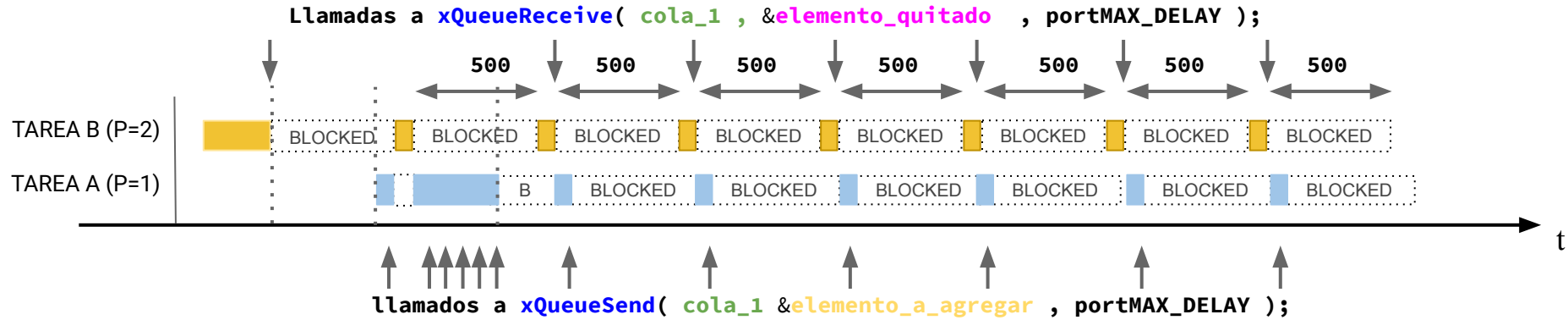
```
QueueHandle_t cola_1 = xQueueCreate( 4 , sizeof(uint8_t) );    //LOS ELEMENTOS DE COMUNICACIÓN ENTRE TAREAS DEBEN SER GLOBALES

void TareaA( void* params )
{
    BaseType_t rv;
    uint8_t elemento_a_agregar = 20 ;
    while(1)
    {
        /* RESTO DE CÓDIGO */
        rv = xQueueSend( cola_1 , &elemento_a_agregar , portMAX_DELAY );
        elemento_a_agregar++;
        /* RESTO DE CÓDIGO */
    }
}

void TareaB( void* params )
{
    BaseType_t rv;
    uint8_t elemento_quitado;

    while(1)
    {
        rv = xQueueReceive( cola_1 , &elemento_quitado , portMAX_DELAY );
        vTaskDelay( 500 / portTICK_RATE_MS );
    }
}
```

Ejemplo



Similitudes de Colas y Semáforos



- FreeRTOS implementa semáforos utilizando la API de colas.
- Un semáforo binario, es una cola de 1 elemento.
- Un semáforo contador, es una cola de N elemento.
- Es por eso que la API de semáforos está implementada con macros.
 - Llama a la API de colas.

Licencia



"Colas en FreeRTOS"

Por Esp. Ing. Franco Bucafusco, se distribuye bajo una [licencia de Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/)

Bibliografia

- <https://www.freertos.org>
- Introducción a los Sistemas operativos de Tiempo Real, Alejandro Celery - 2014
- FreeRTOS - Colas , Cusos INET, Franco Bucafusco, 2017