Comunicación entre procesos

Electrónica IV

Mg.Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

https://facetvirtual.facet.unt.edu.ar/course/view.php?id=165

Comunicación entre procesos

- La forma más simple de comunicación entre procesos es la utilización de variables compartidas
- El uso de variables compartidas genera problemas de sección crítica y de espera activa

Comunicación entre procesos

- Los sistemas operativos ofrecen mecanismos que permiten la comunicación entre procesos sin estos
- Estos mecanismos evitan los problemas de sección critica, por esta razón muchas veces se habla de comunicación y sincronización como el mismo problema

Eventos

- El mensaje más simple que se puede enviar entre dos tareas es un valor lógico verdadero/falso que indica que una ocurre una condición previamente pactada entre ambas tareas

- En general todos los sistemas operativos implementan un sistema de eventos que permite este tipo de comunicación

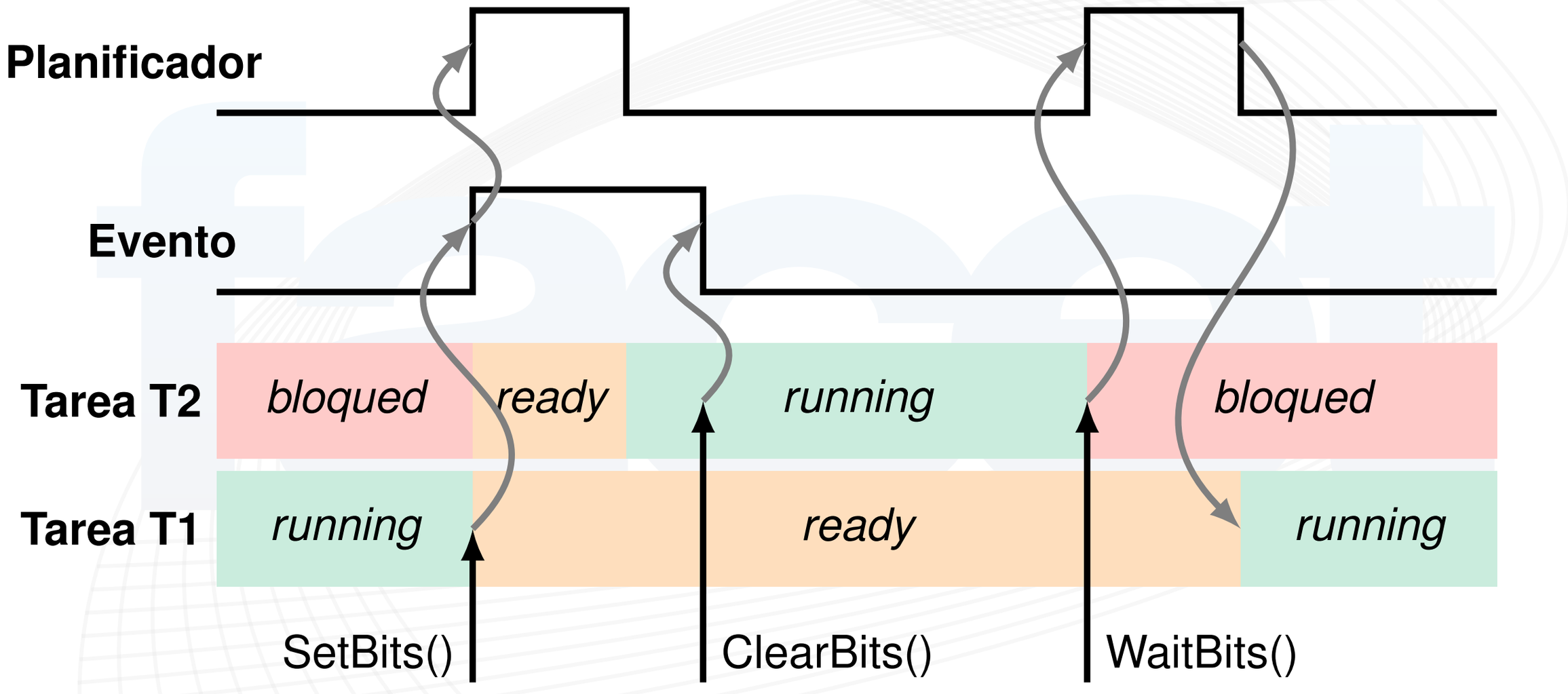
Eventos

- En general los eventos se manejan como bits y se agrupan en conjuntos afines, de forma tal que una tarea puede esperar por más de un evento

- Como el mecanismo de eventos es controlado por el sistema operativo la espera de un evento es pasiva

- Se implementa sacando a la tarea del estado de Ready hasta que se produzca el evento esperado

Eventos en FreeRTOS



Funciones para manejo de eventos

- **`xEventGroupCreate[Static]()`**
  - Crea un grupo de eventos
- **`xEventGroupDelete()`**
  - Borra un grupo de eventos
- **`xEventGroupSetBits[FromISR]()`**
  - Activa determinados bits de eventos en un grupo
- **`xEventGroupClearBits[FromISR]()`**
  - Limpia determinados bits de eventos en un grupo

Funciones para manejo de eventos

- **`xEventGroupGetBits[FromISR]()`**
  - Devuelve el estado de los eventos de un grupo
- **`xEventGroupWaitBits()`**
  - Espera que determinados eventos se activen. Puede esperar por uno cualquiera o por todos especificados en una mascara y limpiar los eventos recibidos automáticamente.
- **`xEventGroupSyncBits()`**
  - Fija uno o mas eventos y espera por otro eventos del mismo grupo en forma atómica. Se utiliza para sincronización de procesos

Semáforos

- Son un mecanismo de comunicación y sincronización universal
- Usa una variable numérica con un valor inicial que solo puede ser accedida por dos operaciones
  - *p()* o *wait()* espera que la variable tenga un valor mayor que cero y la decrementa
  - *v()* o *signal()* incrementa la variable
- Ambas operaciones son atómicas

Una cola con semáforos

- Se quiere implementar una cola con capacidad para almacenar *n* datos
- Dos procesos interactúan a través de esta cola, uno produce datos y otro los consume
- La comunicación se puede resolver utilizando tres semáforos:
  - *mutex* inicia en 1 y protege la sección critica
  - *lleno* inicia en n y espera un lugar en la cola
  - *vacío* inicia en 0 y espera un dato en la cola

Una posible implementación

```
void Productor(void) {
    while (true) {
        /* Produce
        un dato */
        wait(lleno);
        wait(mutex);
        /* Agrega el dato
        a la cola */
        signal(mutex);
        signal(vacio);
    }
}
```

```
void Consumidor(void) {
    while (true) {
        wait(vacio);
        wait(mutex);
        /* Retira el dato
        de la cola */
        signal(mutex);
        signal(lleno);
        /* Consume
        el dato */
    }
}
```

Funciones para semáforos

- **`xSemaphoreCreateCounting[Static]()`**
  - Crea un nuevo semáforo numérico
- **`xSemaphoreCreateBinary[Static]()`**
  - Crea un semáforo binario (sin herencia de prioridad)
- **`xSemaphoreTake[FromISR]()`**
  - Toma una unidad del semáforo (se bloque si es cero)
- **`xSemaphoreGive[FromISR]()`**
  - Devuelve una unidad al semáforo
- **`vSemaphoreDelete()`**
  - Elimina un semáforo

Colas nativas o buzones

- Muchos sistemas operativos implementan colas como un servicio nativo
- Estas colas resuelven el problema de sección critica y espera activa
- En general almacenan punteros, lo que permite el envío de cualquier tipo de información
- En FreeRTOS las colas pueden ser FIFO o LIFO

Funciones para colas

- **`xQueueCreate[Static]()`**
  - Crea una cola
- **`xQueuePeek[FromISR]()`**
  - Lee el primer elemento de la cola sin retirarlo
- **`xQueueReceive[FromISR]()`**
  - Recupera el primer elemento de la cola
- **`xQueueSend[FromISR]()`**
  - Agrega un elemento al final de la cola
- **`xQueueSendToFront[FromISR]()`**
  - Agrega un elemento al inicio de la cola
- **`xQueueReset()`**
  - Reinicia la cola como vacía