



INSTITUTO DE  
INGENIERÍA  
ELÉCTRICA



FACULTAD DE  
INGENIERÍA  
UDELAR



UNIVERSIDAD  
DE LA REPÚBLICA  
URUGUAY

# Introducción a los sistemas operativos de tiempo real (2/2)

## Sistemas embebidos para tiempo real

Este material didáctico fue elaborado por docentes del Departamento de Electrónica de la Universidad de la República a lo largo a varios años. Se pone a disposición de la comunidad bajo la licencia “Creative Commons Attribution 4.0 International License”.

Ver detalles de la licencia aquí: <https://creativecommons.org/licenses/by/4.0/>



Co-funded by the  
Erasmus+ Programme  
of the European Union

# Índice

- Clase anterior:
  - Introducción
  - Tareas y el planificador (scheduler)
  - Tareas y datos
  - Datos compartidos
- Hoy: semáforos y datos compartidos

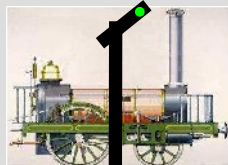
# Datos compartidos

- “Exclusión mutua” (MutEx) para evitar datos corruptos
  - Asegurar acceso exclusivo de datos compartidos
  - Métodos: deshabilitar interrupciones, semáforos
- Comunicación entre tareas o entre tarea e ISR
  - Por ejemplo para avisar de un evento (“está libre la impresora” )
  - Método: semáforos

# Semáforos y recursos compartidos



Recurso compartido: tramo de vía



# Semáforos en RTOS

- Nomenclatura:
  - Take–Release (posta, llave)
  - Raise–Lower (semáforo ferroviario)
  - Wait–Signal (esperar-avisar)
  - Pend–Post (esperar-avisar)
- Semáforos para exclusión mutua (MutEx)
- Semáforos para comunicación entre tareas.



# Semáforos: protección de datos

```
/* "Levels Task" */
void vCalculateTankLevels(void)
{
    /* low priority task */
    int i = 0;
    while (TRUE) {
        !! read float levels in task i
        !! do bunches of calculations
        TakeSemaphore( );
        tankdata[i].lTimeUpdated =
            !! current time
        tankdata[i].lTankLevel =
            !! result of long calculation
        ReleaseSemaphore( );
        !! pick next tank to handle, etc.
        i = !! next tank
    }
}
```

```
struct {
    long lTankLevel;
    long lTimeUpdated;
} tankdata[MAX_TANKS];
```

```
/* "Button Task" */
void vRespondToButton(void)
{
    /* high priority task */
    int i;
    while (TRUE) {
        !! Block until button pressed
        i = !! ID of button pressed
        TakeSemaphore( );
        !! output lTankLevel
        !! output lTimeUpdated
        ReleaseSemaphore( );
    }
}
```

# Flujo de ejecución con semáforos

## vCalculateTankLevel

```
TakeSemaphore( );
```

```
tankdata[i].lTimeUpdated
```

usuario presiona un botón,  
se desbloquea tarea botón

semáforo no disponible,  
se bloquea tarea botón

```
tankdata[i].lTankLevel
```

```
ReleaseSemaphore( );
```

liberando el semáforo,  
se desbloquea la tarea botón

tarea botón se bloquea,  
continúa con tarea niveles

## vRespondToButton

```
!! Block until button pressed
```

```
i = !! ID of button pressed
```

```
TakeSemaphore( );
```

```
!! output tank level,  
timestamp
```

```
ReleaseSemaphore( );
```

# Ejemplo de semáforos (μC/OS)

- Notación
  - El prefijo OS indica funciones del kernel
  - Creación/inicialización: `OSSemCreate(...)`
  - Tomar un semáforo: `OSSemPend(...)`
  - Tomar y liberar un semáforo: `OSSemPost(...)`
  - Estructura `OS_EVENT` representa el semáforo
- Ejemplo:
  - `OS_EVENT *p_semTemp;`
  - `p_semTemp = OSMemCreate(1);`
  - `OSSemPend(p_semTemp, WAIT_FOREVER);`
- Otro servicio del RTOS
  - Retardo: `OSTimeDly(...)`



```

#define PRIORITY_READ 11
#define PRIORITY_CONTROL 12
#define STK_SIZE 1024
static unsigned ReadStk[STK_SIZE];
static unsigned CtrlStk[STK_SIZE];
static int iTemperatures[2];

OS_EVENT *p_semTemp;

void main (void) {
    OSInit( );
    OSTaskCreate (vReadTmpTsk, NULLP, (void *) &ReadStk[STK_SIZE], PRIORITY_READ);
    OSTaskCreate (vCtrlTask, NULLP, (void *) &CtrlStk[STK_SIZE], PRIORITY_CONTROL);
    p_semTemp = OSSemCreate(1);
    OSStart ( );
}

```

```

void vRdTmpTsk (void) {
    while (TRUE) {
        OSTimeDly(5);
        OSSemPend(p_semTemp, WAIT_FOREVER);
        !! read in iTemperatures[0];
        !! read in iTemperatures[1];
        OSSemPost (p_semTemp);
    }
}

```

```

void vCtrlTsk(void) {
    while (TRUE) {
        OSSemPend (p_semTemp, WAIT_FOREVER);
        if(iTemperatures[0] != iTemperatures[1])
            !! set off howling alarm;
        OSSemPost (p_semTemp);
        !! do other useful work
    }
}

```

# Semáforos y funciones reentrantes

```
void Task1(void)
{
    vCountErrors (1);
}

void Task2(void)
{
    vCountErrors (2);
}
```

```
static int cErrors;

void vCountErrors(int cNewErrors)
{
    cErrors += cNewErrors;
}
```

# Semáforos y funciones reentrantes

```
void Task1(void)
{
    vCountErrors (1);
}

void Task2(void)
{
    vCountErrors (2);
}
```

```
static int cErrors;
static NU_SEMAPHORE semErrors;

void vCountErrors(int cNewErrors)
{
    NU_Obtain_Semaphore(&semErrors,NU_SUSPEND);
    cErrors += cNewErrors;
    NU_Release_Semaphore(&semErrors);
}
```

- Terminología:
  - Se ha "protegido" `cErrors` con semáforos
- Notación:
  - En este caso RTOS: Nucleus (prefijo NU)

# Múltiples semáforos

- En un sistema pueden coexistir varios semáforos:
  - Cada semáforo protege un sólo recurso compartido.
  - Cada tarea sólo tiene que esperar si el recurso que tiene que utilizar está bloqueado.
  - El RTOS no sabe qué recursos están protegidos por qué semáforos. Eso es tarea del programador.
- Counting semáforos
  - Control sobre un conjunto de N recursos.

# Semáforo como método de comunicación

- Entre tareas, o entre tareas e ISR.
- Ejemplo: impresión de reportes.
  - Datos:
    - Buffer donde se arma el reporte, números líneas totales e impresas.
  - Tareas:
    - PrinterTask: espera por el reporte y luego envía línea a línea desde el buffer a la impresora.
  - ISR:
    - La impresora interrumpe al final de cada línea y carga la siguiente línea a imprimir.

# Actividad en grupo

- Objetivo: estudiar el semáforo como método de comunicación entre la ISR y la tarea.
- Considerar el código de la siguiente slide:
  - ¿Dónde tengo que colocar el semáforo **semPrinter** para compartir la impresora?
  - ¿Dónde debe hacerse el post *OSSemPost(...)* y el pend *OSSemPend(...)*
- Observaciones:
  - El semáforo solamente es compartido por las funciones que se muestran en la slide
  - Considerar el caso de inicializar el semáforo ocupado
- Grupos: 2 a 4 participantes
- Tiempo: 10 minutos

# Semáforos para señalización

```
static char a_chPrint[10][21]; //output buffer
static int iLinesTotal;
static int iLinesPrinted;
static OS_EVENT *semPrinter;
```

```
void vPrinterInterrupt (void) {
    if (iLinesPrinted == iLinesTotal)
        /* report done */
    else
        /* report not done: print next line */
        vHardwarePrinterOutputLine(
            a_chPrint[iLinesPrinted++]);
}
```

```
void vPrinterTask(void) {
    BYTE byError;
    int wMsg;

    while (TRUE) {
        /* wait for a job to print */
        wMsg = (int) OSQPend (QPrinterTask,
            WAIT_FOREVER, &byError);
        !! Format the report into a_chPrint
        iLinesTotal = !! count of lines in report
        /* print first line of report */
        iLinesPrinted = 0;
        vHardwarePrinterOutputLine(
            a_chPrint[iLinesPrinted++]);
    }
}
```

- ¿Dónde tengo que “colocar” el semáforo **semPrinter** para compartir la impresora?  
O sea: ¿dónde debe hacerse el post **OSSemPost(...)** y el pend **OSSemPend(...)**?
- Observaciones:
  - El semáforo solamente es compartido por estas funciones
  - Considerar el caso de inicializar el semáforo ocupado

# Semáforos para señalización

```
static char a_chPrint[10][21]; //output buffer
static int iLinesTotal;
static int iLinesPrinted;
static OS_EVENT *semPrinter;

void vPrinterInterrupt (void) {
    if (iLinesPrinted == iLinesTotal)
        /* report done: release semaphore */
        OSSemPost(semPrinter);
    else
        /* report not done: print next line */
        vHardwarePrinterOutputLine(
            a_chPrint[iLinesPrinted++]);
}
```

```
void vPrinterTask(void) {
    BYTE byError;
    int wMsg;
    SemPrinter = OSSemInit(0); //semaphore taken

    while (TRUE) {
        /* wait for a job to print */
        wMsg = (int) OSQPend (QPrinterTask,
            WAIT_FOREVER, &byError);
        !! Format the report into a_chPrint
        iLinesTotal = !! count of lines in report
        /* print first line of report */
        iLinesPrinted = 0;
        vHardwarePrinterOutputLine(
            a_chPrint[iLinesPrinted++]);
        /* wait for job done to finish */
        OSSemPend(semPrinter, WAIT_FOREVER,
            &byError);
    }
}
```



# Semáforos para señalización

```
static char a_chPrint[10][21]; //output buffer
static int iLinesTotal;
static int iLinesPrinted;
static OS_EVENT *semPrinter;

void vPrinterInterrupt (void) {
    if (iLinesPrinted == iLinesTotal)
        /* report done: release semaphore */
        OSSemPost(semPrinter);
    else
        /* report not done: print next line */
        vHardwarePrinterOutputLine(
            a_chPrint[iLinesPrinted++]);
}
```

```
void vPrinterTask(void) {
    BYTE byError;
    int wMsg;
    SemPrinter = OSSemInit(1); //sem released

    while (TRUE) {
        /* wait for a job to print */
        wMsg = (int) OSQPend (QPrinterTask,
            WAIT_FOREVER, &byError);
        OSSemPend(semPrinter, WAIT_FOREVER,
            &byError);

        !! Format the report into a_chPrint
        iLinesTotal = !! count of lines in report
        /* print first line of report */
        iLinesPrinted = 0;
        vHardwarePrinterOutputLine(
            a_chPrint[iLinesPrinted++]);
        /* wait for job done to finish */
    }
}
```

# Discusión sobre caso anterior

- Semáforo usado como señal entre dos tareas
  - Una tarea hace el *pend* y la otra tarea hace el *post*.
  - Comparación con exclusión mutua: la misma tarea llama a *pend*, y luego a *post*.
- Especial cuidado con:
  - Valor inicial del semáforo: debe ser elegido bien.
    - En el caso anterior el semáforo fue inicializado como no disponible. Podría haberse hecho diferente.
    - Depende de la aplicación.
  - Orden cuando existen múltiples *pends*.
    - En el caso anterior, la tarea espera por una cola y el semáforo (potencial “abrazo mortal”).

# Problemas

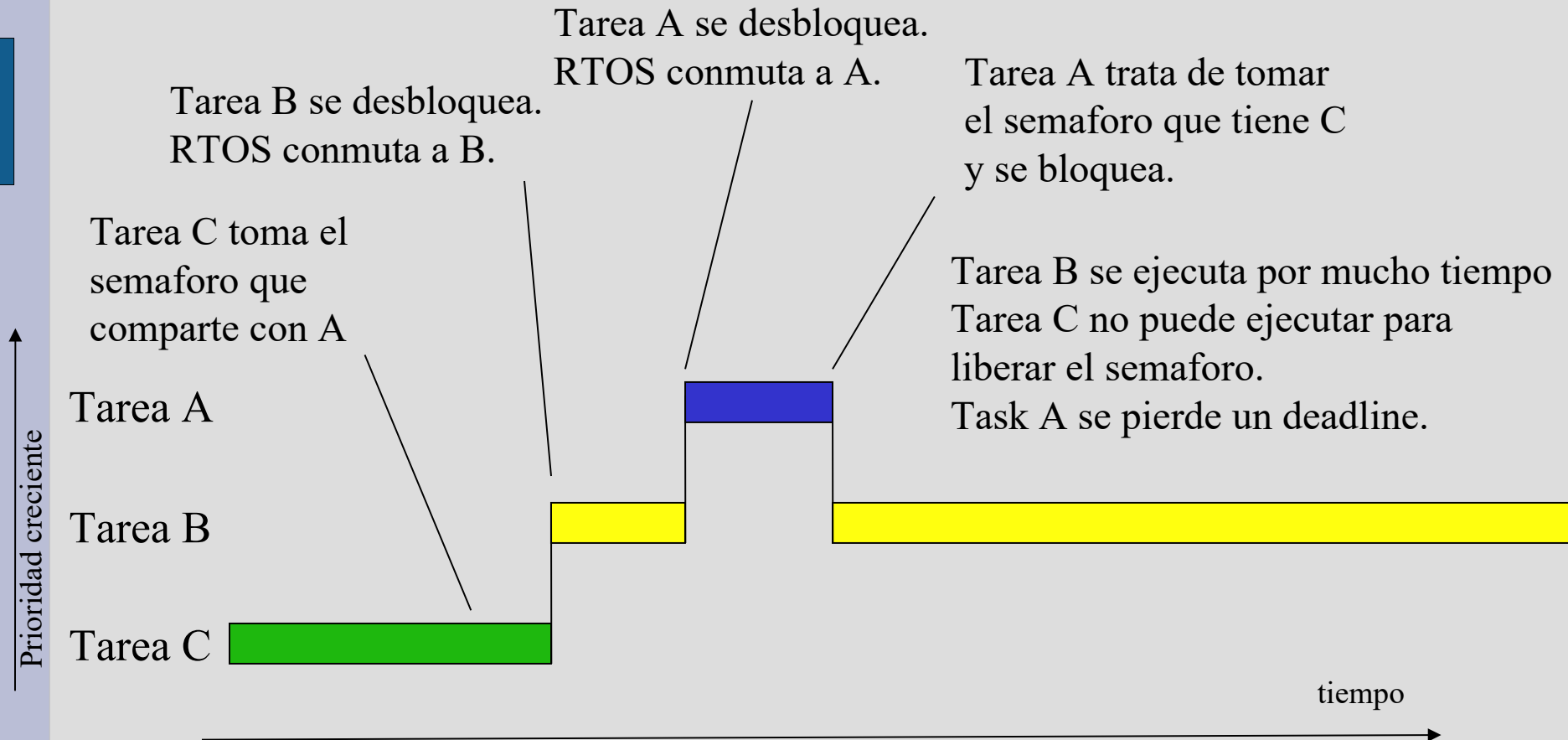
- Básicos
  - Olvidar de tomar (pend) el semáforo.
  - Olvidar de liberar (post) el semáforo.
  - Tomar o liberar el semáforo equivocado.
  - Ocupar el semáforo por demasiado tiempo.
- Inversión de prioridad
- Deadlock, o abrazo mortal

# Inversión de prioridad

- Ejemplo:
  - Tareas C, B y A de prioridad creciente
  - Tareas C y A comparten un recurso



# Inversión de prioridad



# Abrazo mortal (bloqueo mutuo)

```
OS_EVENT *p_sem_A, *p_sem_B;
int a, b; /* Variables compartidas */
void Tarea1(void) {
    OSSemPend(p_sem_A, WAIT_FOREVER);
    OSSemPend(p_sem_B, WAIT_FOREVER);
    a = b;
    OSSemPost(p_sem_B);
    OSSemPost(p_sem_A);
}
void Tarea2(void) {
    OSSemPend(p_sem_B, WAIT_FOREVER);
    OSSemPend(p_sem_A, WAIT_FOREVER);
    b = a;
    OSSemPost(p_sem_A);
    OSSemPost(p_sem_B);
}
```



# Opciones para datos compartidos

- Deshabilitar interrupciones
  - manera más drástica
  - afecta tiempo de respuestas de todas las tareas y ISR
  - único método para compartir datos con ISR
- Semáforos
  - afecta aquellas que esperan el mismo semáforo, sin impacto en ISR
  - aumenta probabilidad de errores de programación
- Deshabilitar conmutación de tareas (deshab. planificador)
  - planificador (scheduler) no conmuta tareas
  - funcionalidad presente en algunos RTOS
  - Overhead para habilitar y deshabilitar es pequeño.
  - Tiempo de respuesta: afecta todas las tareas pero no ISR

# Resumen

- RTOS
  - Características, diferencias con OS convencional
- Tareas
  - Bloque fundamental, estados, contexto, scheduler...
- Funciones reentrantes
- Recursos compartidos:
  - Datos, hardware.
- Semáforos
  - Mutex o comunicación entre tareas, problemas
- Alternativas para prevenir el bug de datos compartidos.



# Anexo: ejemplo de servicios en $\mu$ C/OS

OSInit()	OSSemCreate()
OSIntEnter()	OSSemPend()
OSIntExit()	OSSemPost()
OSMboxCreate()	OSStart()
OSMboxPend()	OSTaskChangePrio()
OSMboxPost()	OSTaskCreate()
OSQCreate()	OSTaskDel()
OSQPend()	OSTimeDly()
OSQPost()	OSTimeGet()
OSSchedLock()	OSTimeSet()
OSSchedUnlock()	OSTimeTick()
OS_ENTER_CRITICAL()	OS_EXIT_CRITICAL()

# Bibliografía

- “An Embedded Software Primer”, David E. Simon
  - Chapter 6: Introduction to Real-Time Operating Systems
- “MicroC OS II: The Real Time Kernel”, Jean J. Labrosse.