# Documentación
# Sistema operativo

Universidad de Málaga

Máster en sistemas electrónicos para entornos inteligentes


Antonio León Pérez

# Índice

**Capítulo** **Página**

**1 Procesos** **2**
1.1 Parámetros . . . 2
1.1.1 STACK_SIZE . . . 2
1.1.2 MAX_PROCESS . . . 2
1.2 Estructuras de datos . . . 2
1.2.1 processState . . . 2
1.2.2 processData . . . 2
1.3 Funciones . . . 2
1.3.1 create_task . . . 2

**2 Planificador** **3**
2.1 Funciones . . . 3
2.1.1 context_switch . . . 3
2.1.2 init_scheduler . . . 3
2.1.3 scheduler . . . 3
2.1.4 irq_timer_handler . . . 3
2.1.5 set_priority_current_task . . . 3

**3 Temporizadores** **3**
3.1 Parámetros . . . 3
3.1.1 DEFAULT_PROCESS_TIME . . . 3
3.2 Funciones . . . 4
3.2.1 timer_init . . . 4
3.2.2 clear_timer_flag . . . 4
3.2.3 timer_start . . . 4
3.2.4 timer_stop . . . 4

**4 Mutex** **4**
4.1 Estructura de datos . . . 4
4.1.1 mutex . . . 4
4.2 Funciones . . . 4
4.2.1 lock . . . 4
4.2.2 unlock . . . 4

**5 Interrupciones** **5**
5.1 Funciones . . . 5
5.1.1 initGIC . . . 5
5.1.2 GICConnectTImer . . . 5
5.1.3 GICEnable . . . 5
5.1.4 ExceptionInit . . . 5
5.1.5 ExceptionRegisterHandler . . . 5
5.1.6 ExceptionEnable . . . 5
5.1.7 clearCallback . . . 5

# 1. Procesos

## 1.1. Parámetros

### 1.1.1. STACK_SIZE

Tamaño de la pila de datos reservada para cada hebra. Su valor por defecto es 1024

### 1.1.2. MAX_PROCESS

Número máximo de hebras que se pueden llegar a crear. Su valor por defecto es 5.

## 1.2. Estructuras de datos

### 1.2.1. processState

Enumerado de los estados en los que se puede encontrar una tarea, actualmente sus valores son stopped (parado), running (en ejecución) y ready (preparado para ejecución).

### 1.2.2. processData

Es la estructura que compone a una hebra. Sus parámetros son:

- **void process_function**: La función que ejecuta la hebra.
- **uint32_t stack_pointer**: El puntero a la pila de datos de la hebra.
- **uint32_t priority**: Prioridad de la hebra.
- **processState state**: Estado actual de la hebra.

## 1.3. Funciones

### 1.3.1. create_task

Función para reservar los recursos de una hebra y dejarla preparada para la ejecución del sistema operativo. Sus argumentos son:

- **void process_function**: La función que se le va a asignar a la hebra.
- **uint32_t priority**: La prioridad que se le va a asignar a la hebra.

La función devolverá 1 en caso de que todo haya salido bien, en caso contrario devolverá 0.

# 2. Planificador

## 2.1. Funciones

### 2.1.1. context_switch

Hace el cambio de contexto entre 2 hebras, sus argumentos son:

- **uint32_t **old_sp**: Puntero a la pila de datos de la función que se va a dejar de ejecutar.
- **uint32_t **new_sp**: Puntero a la pila de datos de la función que va a pasar a ejecutarse.

### 2.1.2. init_scheduler

Inicializa los recursos recursos para empezar la ejecución del sistema operativo y pone en ejecución la primera hebra de mayor prioridad.

Devolverá -1 en caso de que no haya procesos a ejecutar o devolverá 0 si todo se ha iniciado correctamente.

### 2.1.3. scheduler

Escoge la siguiente hebra a ejecutar y realizará un cambio de contexto, el orden para escoger la siguiente hebra es, primero prioridad, y en caso de igual prioridad, orden partiendo desde la tarea que se estaba ejecutando anteriormente.

### 2.1.4. irq_timer_handler

Manejador del sistema operativo para llamar al planificador cuando salte un temporizador.

### 2.1.5. set_priority_current_task

Cambia la prioridad del hebra en ejecución. Su único argumento es:

- **uint32_t new_priority**: La nueva prioridad que se le asigna a la hebra actual.

# 3. Temporizadores

## 3.1. Parámetros

### 3.1.1. DEFAULT_PROCESS_TIME

El timpo de espera del temporizador del sistema operativo. Su valor por defecto es 0x3FFFFFFF.

## 3.2. Funciones

### 3.2.1. timer_init

Inicializa los registros del temorizador para su ejecución, también se puede usar para cambiar el tiempo de espera. Su único argumento es:

- **uint32_t load_value**: Tiempo de espera del temporizador.

### 3.2.2. clear_timer_flag

Limpia el bit de temporizador del registro para comunicar que se ha recibido correctamente la interrupción.

### 3.2.3. timer_start

Empieza la cuenta atrás del temporizador.

### 3.2.4. timer_stop

Para la cuenta atrás del temporizador.

# 4. Mutex

## 4.1. Estructura de datos

### 4.1.1. mutex

Es la estructura que define los cerrojos del sistema. Se conforma de un único valor:

- **volatile int value**: Valor numérico del cerrojo, siendo ele valor 1 bloqueado y 0 desbloqueado.

## 4.2. Funciones

### 4.2.1. lock

Bloquea de manera atómica el cerrojo. Su argumento es:

- **mutex *mut**: El cerrojo que se va a bloquear.

### 4.2.2. unlock

Desbloquea de manera atómica el cerrojo. Su argumento es:

- **mutex *mut**: El cerrojo que se va a desbloquear.

# 5. Interrupciones

## 5.1. Funciones

### 5.1.1. initGIC

Inicializa los registros del controlador general de interrupciones.

### 5.1.2. GICConnectTImer

Inicializa la interrupción del temporizador desde el controlador general y le asocia un manejador. Su argumento es:

- **u32 Int_Id**: Identificador del vector de interrupción del temporizador, por defecto es 29.
- **Xil_ExceptionHandler Handler**: Manejador de interrupción que se va a asociar al temporizador.

### 5.1.3. GICEnable

Habilita las interrupciones del controlador general de interrupciones asociadas a una interrupción. Su argumento es:

- **u32 Int_Id**: Identificador de la interrupción a habilitar.

### 5.1.4. ExceptionInit

Inicializa los registros necesarios para el manejo de excepciones desde el controlador general de interrupciones.

### 5.1.5. ExceptionRegisterHandler

Asocia la rutina de excepciones del sistema al controlador general de interrupciones.

### 5.1.6. ExceptionEnable

Permite la ejecución de interrupciones e interrupciones del procesador.

### 5.1.7. clearCallback

Limpia el registro de interrupción para comunicar que se ha atendido la interrupción.