# DESARROLLO DE SOFTWARE PARA SISTEMAS EMPOTRADOS

## Gestión del tiempo en POSIX

Mario Aldea
Michael González
Héctor Pérez







## Planificación de threads en POSIX

- Conceptos básicos
- Tipos de relojes
- Interfaz para el manejo de relojes
- Interfaz para la suspensión de threads
- Implementación de threads periódicos

## Conceptos básicos

- El **reloj** representa un objeto que mide el paso del tiempo
  - la resolución (o "tick") del reloj es el intervalo de tiempo más pequeño que el reloj puede medir
- La **época** toma como referencia las 0 horas, 0 minutos, 0 segundos del 1 de enero de 1970, UTC (Coordinated Universal Time)
  - para calcular los segundos desde la época

    sec + min*60 + hour*3600 + yday*86400

    + (year-1970)*31536000 + ((year-1969)/4)*86400





## Conceptos básicos

- POSIX define un tipo de dato denominado **timespec** para especificar el tiempo con alta resolución:

```
struct timespec {
    time_t tv_sec; // segundos (32 bits con signo)
    long   tv_nsec;// nanosegundos
}
```

- Por tanto, el tiempo se calcularía mediante la siguiente fórmula

$$tiempo\ (nanosegundos) = tv\_sec * 10^9 + tv\_nsec$$

siendo $0 \leq tv\_nsec < 10^9$

- Este tipo de dato resulta incómodo para hacer operaciones de suma, resta, multiplicación o división

## Tipos de relojes

- Reloj del sistema
  - mide los segundos transcurridos desde la *época*

```
#include <time.h>
time_t time (time_t *tloc);
```

  - si tloc no es NULL, también devuelve la hora en *tloc





## Tipos de relojes

- Reloj de tiempo oficial
  - mide el tiempo transcurrido desde la *época*
    - su hora puede ser cambiada
    - puede coincidir o no con el reloj del sistema
  - se usa para *timeouts* y para crear temporizadores
  - definido en el estándar POSIX como **CLOCK_REALTIME**
  - la resolución máxima permisible es de 20 ms

**NO es el más indicado para usar en aplicaciones de tiempo real**

## Tipos de relojes

- Reloj monótono no decreciente
  - permite medir intervalos de tiempo
    - su origen de tiempos es indeterminado
    - su hora ***no*** puede ser cambiada
  - definido en el estándar POSIX como **CLOCK_MONOTONIC**

**CLOCK_MONOTONIC es el más indicado para usar en aplicaciones de tiempo real**





## Interfaz para el manejo de relojes

```
#include <time.h>
```

- Cambiar la hora:

```
int clock_settime (clockid_t clock_id,
                   const struct timespec *tp);
```

- Leer la hora:

```
int clock_gettime (clockid_t clock_id,
                   struct timespec *tp);
```

- Leer la resolución del reloj:

```
int clock_getres (clockid_t clock_id,
                  struct timespec *res);
```

# Interfaz para la suspensión de threads (1/2)

```
#include <unistd.h>
```

- Suspensiones relativas:

```
unsigned int sleep (unsigned int seconds);

int usleep (useconds_t useconds);

int nanosleep     (const struct timespec *rqtp,
                   struct timespec *rmtp);
```

  - **rqtp* es el tiempo a suspenderse
  - si es interrumpida por una señal, en **rmtp* retorna el tiempo que falta para finalizar la suspensión



DSSE: M. Aldea, M. González y H. Pérez

# Interfaz para la suspensión de threads (2/2)

- Suspensiones absolutas o relativas:

```
int clock_nanosleep (clockid_t clock_id,
                     int flags,
                     const struct timespec *rqtp,
                     struct timespec *rmtp);
```

  - *clock_id* es el identificador del reloj a usar
  - *flags* especifica las opciones de la suspensión
    - si la opción TIMER_ABSTIME está presente, es absoluto

- Posibles valores de *clock_id* son:
  - CLOCK_MONOTONIC
  - CLOCK_REALTIME

# Implementación de threads periódicos

- La necesidad de realizar una actividad de forma periódica es un requerimiento muy común en los sistemas de tiempo real
- En POSIX utilizaremos un thread con la siguiente estructura:

```
void * thread_periodico (void *arg) {
  struct timespec next_time; // hora de activación

  // lee la hora de la primera activación de la tarea
  clock_gettime(CLOCK_MONOTONIC, &next_time);

  // lazo que se ejecuta periódicamente
  while (1) {
    realiza la actividad periódica;

    // espera al próximo periodo
    incr_timespec(&next_time,&periodo); // sumo el periodo a la activación previa
    CHK( clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
                         &next_time, NULL) );
  }
}
```