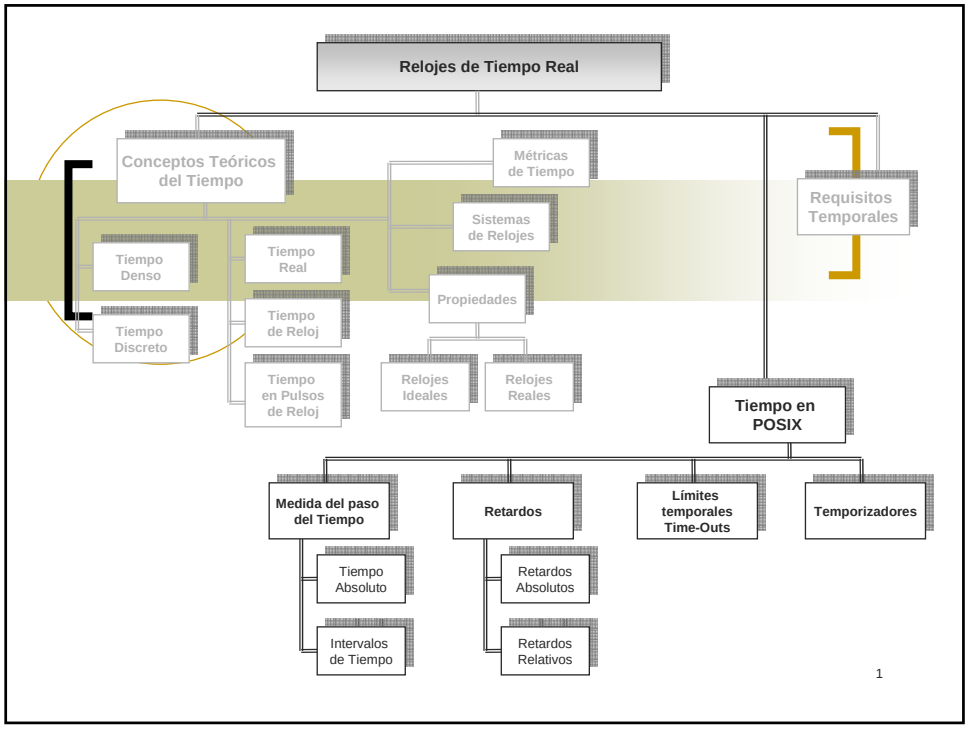

Relojes POSIX

- Vamos a estudiar mecanismos básicos para:
  - Medir el tiempo de ejecución y el uso de CPU de una tarea
  - Retrasar la ejecución de una tarea durante un tiempo
  - Establecer límites temporales para que ocurra un suceso
  - Establecer relojes temporizados

Medida del Paso del Tiempo

- El tiempo es una magnitud física fundamental (unidad en el SI es el segundo)
- Suele ser necesario realizar dos tipos de medidas:
  - Intervalos de tiempo
  - Tiempo absoluto
- Para medir valores absolutos de tiempo hace falta un sistema de referencia
- Un sistema de referencia se basa en una escala de tiempo cuyo origen se denomina *'época'*

Relojes POSIX

- El sistema de referencia utilizado para las medidas absolutas es el tiempo universal 1/1/1970 a las 0 horas.
- El intervalo máximo de medida es de 2^{32} segundos es decir más de 136 años
  - *Entero con signo (int)* 2^{31} *=> 68 años*
  - *time_t*
  - *nueva especificación de 64 bits*
- La granularidad depende de la implementación.
- No existe en POSIX una definición de intervalos de tiempo. Los intervalos de tiempo se miden por tanto por diferencia entre valores de reloj.

Tipos de Relojes en POSIX

- Reloj Calendario
  - Proporciona valores de tiempo con resolución de 1s
- Reloj de Tiempo Real Sytem V
  - La resolución de la representación es de 1 microsegundos
- Reloj de Tiempo Real:
  - Se pueden definir distintos relojes
  - Por lo menos debe haber uno denominado CLOCK_REALTIME
  - La resolución de la representación es de 1ns
  - La granularidad depende de la implementación
- Medida de recursos/tiempo de procesamiento.

Reloj de Tiempo Real POSIX

- El tiempo se representa mediante el la estructura *timespec* **(***<time.h>***)**:

```
typedef struct timespec {
        time_t   tv_sec;        /* segundos */
        long     tv_nsec;       /* nanosegundos */
} timespec_t;
```

- El tipo ***clockid_t*** (entero) sirve para identificar los diferentes relojes.
  - Por lo menos debe estar definido un reloj que abarca a todo el sistema que se identifica como *CLOCK_REALTIME*
  - La resolución máxima del reloj suele ser de 20 ms
  - Puede haber otros relojes

Reloj de Tiempo Real POSIX

- Leer el tiempo actual para el reloj clockid:

  *int **clock_gettime** (clockid_t clockid, struct timespec *tp);*

- Poner en hora el reloj clockid :

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

- Obtiene la resolución del reloj:

  *int **clock_getres** (clockid_t clockid, struct timespec *res);*

EJEMPLO: relojes POSIX

Lectura de tiempo de respuesta

Compilar con: gcc ejt1.c -o ejt1 -lposix4 –lm
Linux: gcc ejt1.c -o ejt1 -lrt –lm

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <time.h>

#define RESET_TMP()             clock_gettime (CLOCK_REALTIME, &time1)
#define SET_TMP()               clock_gettime (CLOCK_REALTIME, &time2)

#define SEC_TRANS()             (time2.tv_sec - time1.tv_sec)
#define NSEC_TRANS()            (time2.tv_nsec - time1.tv_nsec)

struct timespec     time1, time2;


void VER_TMP(char *txt)
{
        if (txt!=NULL)
                fprintf(stderr, "%s ", txt);

                fprintf(stderr, "[Tiempo consumido real: %2.9fs]\n",
                                SEC_TRANS() + (float) NSEC_TRANS() *1E-9);
}

// Continua ->
```


Ejemplo: lectura tiempo de respuesta

```
int main(void)
{
                int i;
                float sen;

                // tiempo de respuesta
                RESET_TMP();            // fija tiempo inicial
                for(i=0;  i< 1E6;  i++)
                        sen=sin((float)i);
                SET_TMP();              // fija tiempo final

                VER_TMP("Ciclo for:");          // visualiza el tiempo consumido

                exit(0);
}
```

```
Ciclo for: [Tiempo consumido real: 0.086368814s]
```

Recursos/Tiempo del Sistema

- Proporciona información del uso de recursos del sistema:

```
#include <sys/resource.h>

int getrusage (int who, struct rusage * r_usage);
```

- *who*:
  - **RUSAGE_SELF** : datos del propio proceso
    RUSAGE_CHILDREN: datos de los procesos hijos
- **r_usage:* contiene la información contable del proceso.

Ejemplo getrusage()

```
struct rusage {
    struct timeval ru_utime;/* user time used */
    struct timeval ru_stime;/* system time used */
    long       ru_nvcsw;/* voluntary context switches */
    long       ru_nivcsw;/* involuntary context switches */
    long       ru_nswap;/* swaps */
    long       ru_inblock;/* block input operations */
    long       ru_oublock;/* block output operations */
    long       ru_msgsnd;/* messages sent */
    long       ru_msgrcv;/* messages received */
    long       ru_nsignals;/* signals received */
  //.....
}
```

```
Ciclo for: [Tiempo consumido real: 0.905228928s]
Recursos:
------------------------------------------------------------
Tiempo consumido (usuario): 0.880000   segundos.
Tiempo consumido (sistema): 0.000000   segundos.
------------------------------------------------------------
ru_nvcsw:       0  -> Cambios de Contexto voluntarios.
ru_nivcsw:     26  -> Cambios de Contexto involuntarios.
ru_nswap:       0  -> Swaps (caché de memoria del proceso en HD).
------------------------------------------------------------
ru_inblock:     0  -> Operaciones de entrada bloqueadas.
ru_oublock:     0  -> Operaciones de salida bloqueadas.
ru_msgsnd:      0  -> Mensajes enviados.
ru_msgrcv:      0  -> Mensajes recibidos.
ru_nsignals:    0  -> Señales recibidas.
------------------------------------------------------------
```

Retardos

- Un retardo suspende la ejecución de un proceso durante un cierto tiempo.
- Existen dos tipos de retardos:
  - ***Retardo Relativo***: la ejecución se suspende durante un intervalo de tiempo relativo al instante actual.
  - ***Retardo Absoluto***: la ejecución se suspende hasta que se llegue a un instante determinado de tiempo absoluto.
- POSIX solo incorpora funciones para manejo de retardos relativos

Retardo Relativos

- Las funciones para el manejo de retardos relativos en POSIX son dos:

```
#include <time.h>

unsigned int sleep (unsigned int seconds);
int nanosleep (const  timespec_t *rqtp, timespec_t *rmtp);
```

- La función ***sleep()*** suspende el proceso durante los segundos indicados (*muy poca precisión*)
- La función ***nanosleep()*** (thread) permite un especificar retardos con una resolución de nanosegundos. La duración del retardo es ***rqtp***
- Devuelve 0 si es la espera es completa y -1 si el proceso se despierta antes por una **señal** enviada al *proceso/thread*.
  - Si ***rmtp*** es distinto de *NULL* almacena la diferencia entre el tiempo especificado y el tiempo que el proceso ha estado dormido realmente

Ejemplo Retardo Relativo

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

int main(void)
{
        struct timespec retardo, retardo_pend;
        int res;

        // retardo relativo 2.150 s
        retardo.tv_sec=2;
        retardo.tv_nsec=150E6;          // 150 ms

        printf("Voy a esperar %d.%09ld s ......\n", retardo.tv_sec, retardo.tv_nsec);

        res = nanosleep (&retardo, &retardo_pend);

        if (res==-1)
           printf("He despertado por una señal, Retardo pendiente: %d s %ld nsec\n",
                                retardo_pend.tv_sec, retardo_pend.tv_nsec);
        else
           printf("He despertado y termino.\n");

        exit(0);
}
```


Retardos Absolutos

- No existe en POSIX soporte directo para el manejo de retardos absolutos
- Puede aproximarse utilizando retardos relativos de la siguiente forma:

  tiempo_actual ← ***clock_gettime (…);***

  sleep(ret_absoluto - tiempo_actual);

- Desde la lectura del tiempo actual hasta que se llama a la función *sleep()* el proceso puede haber sido expulsado de la CPU (*condición de carrera*)
- Para que fuera correcto, la lectura de tiempo y el cálculo de *(ret_absoluto-tiempo_actual)* debería ser atómica.
- Su implementación precisa se hace mediante *temporizadores*

Retardo Efectivo

- El valor especificado en un retardo no es exactamente el tiempo realmente esperado

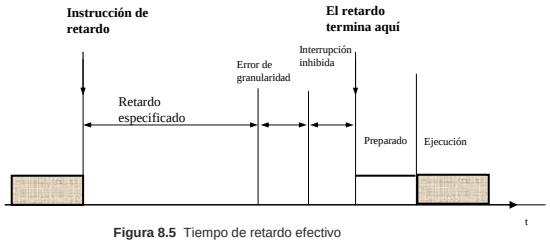

Figura 8.5 Tiempo de retardo efectivo

Límites Temporales (Time-outs)

- A menudo conviene limitar el tiempo durante el cual se espera que ocurra un suceso.
- Ejemplos:
  - Espera por un semáforo o mutex
  - Espera por una variable condicional
  - Espera de una lectura desde un dispositivo físico (un puerto de comunicaciones por ejemplo) pueden producirse también situaciones de bloqueo.
- POSIX incluye funciones para el manejo de temporizaciones asociadas a diferentes tipos de recursos.

Temporizadores

- Un temporizador es un registro contador asociado a un reloj.
  - Una vez creado se inicializa con el tiempo a contar
  - Cada pulso de reloj decrementa el contador del temporizador.
  - Al llegar a 0 se notifica un evento (**Señal**) al *thread* que lo creó y inicializa de nuevo su valor
  - Están asociados al manejo de ***señales*** (lo estudiaremos en el siguiente tema)
  - Permiten ejecutar tareas con retardos relativos y absolutos precisos
  - Permiten ejecutar tareas periódicas

Resumen Funciones POSIX (1)

TIPO DE DATOS	CABECERA	DESCRIPCIÓN
time_t	*<time.h>*	Entero 32 bits
struct timespec { *time_t tv_sec; /* segundos */* *long tv_nsec; /* nanosegundos */* *};*	*<time.h>*	Tiempo en nanosegundos
clockid_t	*<time.h>*	ID reloj (entero)
struct rusage { *struct timeval ru_utime;* *struct timeval ru_stime;* */* ... */* *};*	*<sys/time.h>*	Tiempo de uso de CPU
timer_t	*<time.h>*	ID Temporizador. (entero)
struct itimerspec { *struct timespec it_interval; /* periodo */* *struct timespec it_value; /* expiración */* *};*	*<sys/time.h>*	Estructura de un temporizador

Resumen Funciones POSIX (2)

FUNCIÓN	CABECERA	DESCRIPCIÓN
*int **clock_gettime**(clockid_t clockid, struct timespec *tp);*	*<time.h>*	Lectura de tiempo en nanosegundos (ns)
*int **clock_settime**(clockid_t clockid, const struct timespec *tp);*		Poner en hora el reloj (ns)
*int **clock_getres**(clockid_t clockid, struct timespec *res);*		Obtiene la resolución del reloj
*int **getrusage**(int who, struct rusage * r_usage);*	*<sys/resource.h>*	Tiempo de utilización de CPU (us)
*unsigned int **sleep**(unsigned int seconds);*	*<time.h>*	Retardo relativo (segundos)
*int **nanosleep**(const struct timespec *rqtp, struct timespec *rmtp);*		Reatrdo relativo (ns)
*int **timer_create** (clockid_t clock_id, struct sigevent *evp, timer_t *timerid)*	*<signal.h>* *<time.h>*	Crea un temporizador
*int **timer_settime** (timer_t timerid, int flag, const struct itimerspec *value, struct itimerspec *ovalue);*		Activa un temporizador
*int **timer_delete** (timer_t timerid);*		Elimina un temporizador

Librería manejo de relojes

- Ficheros:
  - Tiempo.h
  - Tiempo.c
- Manejo de tiempos de respuesta, retardos y visualización de resultados
- Enlace:

http://isa.umh.es/asignaturas/sitr/tiempo.zip