# Sistemas en Tiempo Real – Práctica Posix 3

## Programación con tiempo real

Crear un programa que incluya los siguientes hilos:

- Hilo que espera la ocurrencia de la señal SIGINT, momento en que incrementará el número de señales recibidas (basarse en el ejercicio 2 de la Práctica Posix 2).
- Hilo periódico de periodo 5 segundos usando retardos absolutos que realice las siguientes acciones:
  - Obtiene el tiempo de los relojes CLOCK_MONOTONIC, CLOCK_REALTIME y CLOCK_THREAD_CPUTIME_ID y los muestra en pantalla siguiendo el formato [día/mes/año horas:minutos:segundos.milisegundos]
    - Ejemplo: 15/10/2021 12:27:34.785.
  - Ejecuta el código *for(i = 0; i < 100000000; i++);* // no hace nada, solo las vueltas del bucle para // incrementar el tiempo de proceso. Poner ';' al final.
- Hilo periódico de periodo 2 segundos utilizando un temporizador (que generará la señal SIGUSR1) que realice las siguientes acciones:
  - Mostrará por pantalla el número de señales SIGINT que ha recibido la aplicación.

Todos los hilos terminarán cuando las señales SIGINT recibidas sean iguales a las señales esperadas. Los hilos reciben estos valores por parámetro mediante una estructura compartida. Para la gestión de los hilos se debe usar la clase *hilo_t* definida en la práctica 0.

No es necesario garantizar la exclusión mutua para el incremento del número de señales recibidas ya que únicamente un hilo escribe en la variable, con lo que no se producen escrituras simultáneas.

En el campus virtual se dispone de la librería funciones_time que suministra funciones para realizar las operaciones necesarias con la estructura *timespec*.

La práctica se comprimirá en un único archivo que será entregado a través del campus virtual usando la tarea creada para tal efecto.

# Anexo 1 – Definición de funciones para crear hilos

- Definición de la función que se usará para lanzar un hilo: *void *NombreFuncion(void *parámetro)*.
- Paso de parámetros a un hilo:
  - Parámetro de tipo puntero:

```
void *funcion(void *aux) { //Por ejemplo, el parámetro es un puntero a entero
    int *n=(int *)aux;
    //Resto del código usando n de forma normal
}
```

  - Parámetro de tipo no puntero:

```
void *funcion(void *aux) { //El parámetro es un dato de tipo TDato
     Tdato d=*((TDato *)aux);
    //Resto del código usando d de forma normal
                                                    }
```

# Anexo 2 – Posix

- Señales:
  - Todos los hilos (incluido el main) deben bloquear las señales.
  - Poner un conjunto de señales a vacío: *int sigemptyset(sigset_t *set)*.
  - Añadir una señal a un conjunto de señales: *int sigaddset (sigset_t *set, int sig)*.
  - Quitar una señal de un conjunto de señales: *int sigdelset (sigset_t *set, int sig)*.
  - Establecer el conjunto de señales bloqueadas de un hilo: *int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset)*, donde *how* puede ser *SIG_BLOCK* (añade las señales en *set* a la máscara), *SIG_UNBLOCK* (retira las señales en *set* de la máscara) y *SIG_SETMASK* (establece *set* como la nueva máscara).
  - Esperar por la recepción de una señal: *int sigwait(const sigset_t *set, int *sig)*.

# Anexo 3 – Tiempo real

- Relojes:
  - Estructura *timespec*: Contiene dos campos: *tv_sec* (segundos) y *tv_nsec* (nanosegundos). Ambos campos deben tener valores temporales válidos: *tv_sec* no puede ser menor que 0 y *tv_nsec* no puede ser menor que 0 ni mayor que 999.999.999 (número máximo de nanosegundos en un segundo). IMPORTANTE: nunca suponer que estos campos tienen valores válidos, por lo que siempre hay que inicializarlos, leerlos mediante *clock_gettime*, …
  - Leer la hora de un reloj concreto: *int clock_gettime(clockid_t clockid, struct timespec *tp)*.
  - Transformar segundos a estructura *tm* (contiene campos para indicar las horas, minutos, segundos, día, mes, …): *struct tm* localtime (const time_t *timer)*. *localtime* retorna la dirección donde se guarda la estructura (siempre la misma), no una estructura nueva cada vez que se llame.
  - Dormir un hilo especificando el reloj y el tipo de retardo: *int clock_nanosleep(clockid_t clock_id, int flags, const struct timespec *rtpq, struct timespec *rmpt)*. Para retardos absolutos, poner *TIMER_ABSTIME* en *flags*. *rmpt* puede ser *NULL*.
- Temporizadores:
  - Estructura *sigevent*: Almacena los datos de la acción que realizará el temporizador. Tiene varios campos:
    - *sigev_notify* → Hay que establecerlo siempre. Indica el tipo de acción que se va a realizar, que puede ser *SIGEV_NONE* (ninguna), *SIGEV_SIGNAL* (generará una señal) y *SIGEV_THREAD* lanzará un hilo)
    - *sigev_signo* → Necesario cuando se usa *SIGEV_SIGNAL*. Establece la señal que lanzará el temporizador.
    - *sigev_notify_function* → Necesario cuando se usa *SIGEV_THREAD*. Establece la función con la que se creará el hilo.
    - *sigev_notify_attributes* → Necesario cuando se usa *SIGEV_THREAD*. Atributos de la función *sigev_notify_function*.
  - Creación de un temporizador: *int timer_create(clockid_t clock_id, struct sigevent *evp, timer_t *timerid)*. Esta función crea el temporizador, pero no lo lanza.
  - Estructura *itimerspec*: Esta estructura tiene dos campos de tipo *timespec* que sirven para almacenar el instante inicial en que se disparará el temporizador (campo *it_value*) y el periodo de repetición del mismo (campo *it_interval*).
  - Armar un temporizador: *int timer_settime(timer_t timerid, int flags, const struct itimerspec *value, struct itimerspec *ovalue)*. *ovalue* puede ser *NULL*.