Master en Computación

Programación Concurrente

Bloque II: Programación concurrente en POSIX

Tema 4. Gestión del Tiempo

4.1 Conceptos Básicos

***Reloj*:**

- objeto que mide el paso del tiempo

***Resolución* (o "Tick") del reloj:**

- el intervalo de tiempo más pequeño que el reloj puede medir

La *Época*:

- las 0 horas, 0 minutos, 0 segundos del 1 de enero de 1970, UTC (*Coordinated Universal Time*)
- segundos desde la Época

```
sec + min*60 + hour*3600 + yday*86400
    + (year-1970)*31536000 + ((year-1969)/4)*86400
```

Relojes en un sistema POSIX

- Reloj del sistema:
  - mide los segundos transcurridos desde la Época
  - se usa para marcar las horas de creación de ficheros, etc.
- Reloj de "Tiempo Oficial" (`CLOCK_REALTIME`)
  - reloj que mide el tiempo transcurrido desde la Época
  - se usa para *timeouts* y para crear temporizadores
  - puede coincidir o no con el reloj del sistema
  - resolución máxima: 20 ms. ¡Precaución!
  - resolución mínima: 1 nanosegundo
- Reloj monótono (`CLOCK_MONOTONIC`)
  - el tiempo medido con este reloj siempre crece de forma monótona (no se puede cambiar su hora)
- Otros relojes definidos por la implementación

Tipo `timespec`

El tipo `timespec` permite especificar el tiempo con alta resolución:

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

- tiempo en nanosegundos = $tv\_sec*10^9+tv\_nsec$
- $0 \leq tv\_nsec < 10^9$

Fin de los tiempos según POSIX (2^{31}-1 seg. después de la época)

```
$ date -d @2147483647
Tue Jan 19 04:14:07 CET 2038
```

Incómodo para hacer operaciones

MaRTE OS proporciona `<misc/timespec_operations.h>`

- con operaciones de comparación, suma, resta, multiplicación y división de "timespecs"

4.2 Reloj del Sistema

Leer la hora:

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

- `time()` devuelve los segundos desde la Época, según el reloj del sistema
- si `tloc` no es `NULL`, también devuelve la hora en `*tloc`
- Funciones relacionadas (ver manual): `gettimeofday()`, `ctime()`, etc.

Alarma:

```
unsigned int alarm (unsigned int seconds);
```

- envía `SIGALRM` cuando han transcurrido los segundos especificados
- sólo se puede programar una alarma para cada proceso

4.3 Relojes de Tiempo Oficial y Monótono

El resto de relojes soportados por el sistema se definen mediante constantes del tipo `clockid_t`

- **POSIX define dos: reloj de "tiempo oficial" y reloj monótono**
- **cada SO puede definir más relojes si lo desea**

Reloj de "tiempo oficial" (`CLOCK_REALTIME`)

- **NO es el más indicado para usar en aplicaciones de tiempo real**
- **su hora puede ser cambiada**
- **la máxima resolución permisible es de 20 ms**

Reloj monótono (`CLOCK_MONOTONIC`)

- **el más *indicado para aplicaciones de tiempo real***
- **su hora NO puede ser cambiada**
- **su origen de tiempos es indeterminado**

4.4 Funciones para Manejar 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);
```


4.5 Funciones *sleep*

Dormir un thread:

```
#include <unistd.h>
unsigned int sleep (unsigned int seconds);
```

- **el thread se suspende hasta que transcurren los segundos indicados, o se ejecuta un manejador de señal**
- **valor retornado:**
  - **0 si duerme el intervalo solicitado**
  - **número de segundos que faltan para completar el intervalo si vuelve a causa de que una señal ha sido entregada al thread y su acción asociada era ejecutar un manejador**

```
int usleep(useconds_t useconds); // obsoleta
```

- **el thread se suspende hasta que transcurren los microsegundos indicados, o se ejecuta un manejador de señal**

Sleep de alta resolución

- **Relativo:**

```
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

- **Absoluto o relativo, con especificación del reloj:**

```
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 opciones; si la opción `TIMER_ABSTIME` está, es absoluto; si no (`flags=0`), es relativo

4.6 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);
    CHK( clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
                         &next_time, NULL) );
  }
}
```


Condición de carrera con las operaciones de suspensión relativas

Observar que con un *sleep* relativo existe una condición de carrera inevitable en un sistema multitarea

```
pseudocódigo thread_pseudoperiodico {
  // lee la hora de la primera activación de la tarea
  next_time = clock_gettime;

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

    // espera al próximo periodo
    next_time+=periodo;
    suspend_time=next_time-clock_gettime;
    sleep(suspend_time);
  }
}
```

El error se da si la tarea es expulsada

- **después de haber leído la hora**
- **pero antes de iniciar la suspensión**

Ejemplo: threads periódicos

```
#include <stdio.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>
#include <misc/error_checks.h>
#include <misc/timespec_operations.h>

void * periodic (void *arg);

// Programa principal, que crea dos threads periódicos
int main () {
  pthread_t t1,t2;
  struct timespec per_params1,per_params2;

  per_params1.tv_sec=0;
  per_params1.tv_nsec=500000000;
  CHK( pthread_create (&t1, NULL, periodic, &per_params1) );

  per_params2.tv_sec=1;
  per_params2.tv_nsec=500000000;
  CHK( pthread_create (&t2, NULL, periodic, &per_params2) );

  sleep(30);
  return 0;
}
```



```
// Cuerpo del thread periódico que usa clock_nanosleep
void * periodic (void *arg) {
  struct timespec next_time; // hora de activación
  struct timespec my_period = * (struct timespec*)arg;

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

  // lazo que se ejecuta periódicamente
  while (1) {
    printf("Thread con periodo sec=%ld nsec=%ld activo \n",
           my_period.tv_sec, my_period.tv_nsec);

    // espera al próximo periodo
    incr_timespec(&next_time,&my_period);
    CHK( clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME,
                         &next_time, NULL) );
  }

  return NULL;
}
```