



SISTEMAS CONCURRENTE Y DISTRIBUIDOS

4. Prácticas de STR



Indice

- Objetivos
- Gestión de Tiempo con Relojes POSIX
- Retardos
- Práctica



Objetivos

- En esta práctica se van a realizar una implementación de sistemas de tiempo real basados en el método de planificación cíclica
- Para lo cual se debe:
 - Estudiar la planificabilidad del sistema de tiempo real utilizando un método de planificación
 - Implementar, compilar y ejecutar dicho sistema de tiempo real en una plataforma POSIX como Linux utilizando lenguaje C/C++.



Introducción

- POSIX (Portable Operating System Interface for Computer Environments) son un conjunto de normas o especificaciones IEEE/ISO que definen la interfaz entre las aplicaciones y el Sistema Operativo.
 - Establecen los servicios fundamentales que debe tener el sistema operativo: manejo de archivos, control de dispositivo, multiprogramación, comunicación entre procesos (IPC), etc.
 - Proporcionan una API común a las aplicaciones que facilita la portabilidad, mantenimiento y reutilización de las aplicaciones.
- La denominación oficial es IEEE Std. 1003.1-2008 y ha evolucionado desde las primeras versiones en los años 1980 hasta la última revisión de la especificación en 2013.
 - <http://pubs.opengroup.org/onlinepubs/9699919799/>



Introducción

- Para la realización del programa vamos a estudiar un aspecto clave de POSIX para utilizar los planificadores de los sistemas operativos: **La Gestión del tiempo**



GESTIÓN DEL TIEMPO



Gestión del Tiempo

- En un sistema de tiempo real se necesita disponer de mecanismos adecuados para:
 - ☐ medir el tiempo
 - ☐ controlar la duración de las acciones del sistema.

- Para ello se requieren:
 - ☐ **Relojes** que midan con precisión el paso del tiempo.
 - ☐ Temporizadores/**Retardos** que permitan activar hilos (o procesos) en instantes de tiempo prefijados.
 - ☐ Timeouts que permitan definir límites temporales a las acciones del sistema.



Relojes en POSIX

- El reloj sirve para medir el paso continuo del tiempo
 - Los sistemas operativos deben tener al menos un sistema de medición del tiempo a través del reloj del sistema.
- Los sistemas POSIX disponen de relojes de alta precisión, denominados relojes POSIX, para la medida del tiempo.
 - El sistema operativo POSIX implementa el reloj utilizando el reloj hardware o un chip de temporización más preciso que contenga.
 - En Linux el reloj se implementa utilizando el contador TSC de los procesadores
- Los relojes POSIX utilizan siempre como referencia la Época: CUT (Coordinated Universal Time)
 - 0 h 0 m 0s del 1 de enero de 1970

Relojes en POSIX

- El tiempo se representa con la estructura ***timespec***
 - Utiliza 62 bits para el almacenamiento del tiempo.
 - Resolución: 1 ns. (1 nanosegundo = 1 mil millonésima de segundo)
 - Epoca: 0h UT 1/1/1970
 - Se pueden representar 2^{32} s, equivale a 136 años
 - No hay problema de saturación del contador
 - El tiempo en segundos se calcula **`tv_sec + tv_nsec * 10-9`**

```
#include <time.h>

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



Gestión de Relojes POSIX

- Funciones para manejo de relojes POSIX (se debe incluir la cabecera `<time.h>`):

- ☐ 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);*



Gestión de Relojes POSIX

- Hay definidos varios relojes mediante el tipo `clockid_t` (entero):
 - `CLOCK_REALTIME`: Reloj por defecto.
 - Granularidad máxima de 20 ms (50Hz).
 - Puede haber ajustes para corregir fechas
 - En Linux la resolución es de 10 ms o de 1 ms.
 - `CLOCK_MONOTONIC`: Igual que `CLOCK_REALTIME` pero no se realizan ajustes, por tanto su cuenta es creciente sin saltos bruscos.
- Linux tiene definido los siguientes relojes:
 - `CLOCK_REALTIME`, `CLOCK_MONOTONIC`

Medida del tiempo

- Para medir el tiempo hay que establecer dos marcas de tiempo, antes y después de la secuencia de instrucciones a medir, y después se calcula la diferencia de tiempo

```
#include <time.h>

int main (int argc, char *argv[])
{
    struct timespec times0, times1, total;

    clock_gettime(CLOCK_MONOTONIC, &times0);    //marca uno
    //secuencia de instrucciones a medir
    clock_gettime(CLOCK_MONOTONIC, &times1);    //marca dos

    //medir el tiempo
    total.tv_sec = times1.tv_sec - times0.tv_sec;
    total.tv_nsec = times1.tv_nsec - times0.tv_nsec;
    if (total.tv_nsec < 0) {
        total.tv_nsec = 1E9 + times1.tv_nsec - times0.tv_nsec;
        total.tv_sec - -;
    }
    // imprimir los resultados separando tv_sec de tv_nsec
}
```



Ejemplo

- Compilar y ejecutar el programa `medidatiempo.c` que permite medir:
 - La resolución del reloj POSIX a través de `clock_getres`.
 - La medida del tiempo de ejecución de un comando o un programa que se pasa como parámetro
- Para compilar con el compilador de c:

```
gcc medidatiempo.c utilRT.c -o medidatiempo
```
- Se enlaza `utilRT.c` que incluye la implementación de funciones de conversión de tiempo.
- En algunos sistemas Linux es necesario incluir la librería `rt` para compilar, es decir,

```
gcc medidatiempo.c utilRT.c -o medidatiempo -lrt
```
- Ejecutar varias veces el programa para medir el tiempo que tarda en ejecutarse algunos comandos de linux como `ls`



Retardos

- Los retardos son funciones que permiten suspender la ejecución de un proceso o un hilo hasta que transcurra el intervalo de tiempo especificado.
 - El sistema operativo garantiza que después de transcurrido el intervalo de tiempo devolverá el control al proceso/hilo que invocó el retardo.
 - No es un mecanismo tan preciso como los temporizadores.
- Tradicionalmente los retardos se invocan con la función sleep

unsigned int sleep (unsigned int seconds);



Retardos

- En POSIX hay dos funciones para establecer retardos de mayor resolución y precisión.
- La función **nanosleep** puede suspender el proceso/hilo con un intervalo de tiempo relativo de mayor resolución:
*int nanosleep(const struct timespec *rqtp,
 struct timespec *rmtp);*
 - Con **nanosleep()** se suspende la ejecución del proceso/hilo hasta que transcurra el intervalo especificado o hasta que se recibe una señal que lo interrumpe
 - **rqtp** es el intervalo de tiempo que se va a suspender
 - **rmtp** si es distinto de NULL, y la espera se interrumpe por recibirse una señal, entonces se escribe en esta variable el intervalo de tiempo restante que no se ha dormido (es un parámetro opcional de salida).



Retardos

- La función **clock_nanosleep** permite establecer el retardo con una mayor resolución (como nanosleep) y además:
 - Posibilita elegir el tipo de reloj POSIX utilizado clockid_t
 - Opcionalmente, permite establecer retardos hasta que llegue un instante de tiempo determinado (**no duerme durante un intervalo sino que duerme hasta un instante en tiempo absoluto**). Para eso se pone en flags TIMER_ABSTIME

```
int clock_nanosleep (clockid_t clock_id, int flags,  
                    const struct timespec *rtpq, struct timespec *rmpt);
```

- En el primer parámetro podemos asociar el reloj p.e. CLOCK_MONOTONIC.
- Si se utilizan retardos de tiempo absolutos se puede reducir la deriva que se produce normalmente en un retardo.



Planificación cíclica

- La implementación del método de planificación cíclica no requiere que el entorno de ejecución disponga de ningún módulo de planificación ni de despacho.
- Es necesario establecer el plan de ejecución antes de su implementación. Para ello, se puede utilizar el método sistemático visto en teoría que se basa en:
 - Determinar el ciclo principal T_M
 - Determinar el ciclo secundario T_s o marco.
 - Establecer la ordenación de ejecución de cada tarea según los atributos temporales de cada tarea.
- Se puede establecer una prioridad al ejecutivo cíclico, aunque no es necesario.



Planificación cíclica

- La implementación del ejecutivo cíclico se realiza directamente siguiendo el esquema siguiente:

```
//inicializar el planificador
int nciclos, marco;
struct timespec origen, activacion, cicloMarco;
clock_gettime (CLOCK_MONOTONIC, &origen);
//activacion debe tener la suma de origen + cicloMarco

//planificador ciclico
while (true) {
    switch (marco) {
        case 1: TareaA(); TareaB ();... //marco secundario 1
        case 2: TareaC(); TareaD ();....
        ....
        case nciclos-1: TareaB(); ...
    }
    marco = (marco+1)%nciclos; //cambia de marco secundario
    //bloquea hasta el siguiente tick
    clock_nanosleep (CLOCK_MONOTONIC, TIMER_ABSTIME, activacionTiempo,NULL);
    //calcular la siguiente activacion sumando activacionTiempo más cicloMarco
}
```



EJERCICIO



Planteamiento

- Dado un sistema de tiempo real formado por 4 tareas periódicas cuyos parámetros temporales vienen definido en la siguiente tabla:

| Tarea | T Periodo (ms) | C Tiempo de Cómputo (ms) |
|-------|-------------------|-----------------------------|
| A | 5000 | 1000 |
| B | 5000 | 1500 |
| C | 10000 | 2000 |
| D | 20000 | 2400 |



Ejercicio

- Implementación de un sistema de tiempo real utilizando un método de planificación basado en planificación cíclica.



Planificación cíclica

- Determinar si el conjunto de tareas es planificable con el modelo de ejecutivo cíclico para lo cual se debe calcular el ciclo secundario y realizar la planificación offline del sistema.
- Implementar, compilar y ejecutar el sistema de tiempo real con las características de tiempo señalados anteriormente utilizando el modelo de ejecutivo cíclico. Para ello utilizar la plantilla `ejecutivociclico.c`.
- Comprobar que se cumplen los instantes de activación del ejecutivo cíclico de acuerdo al plan de ejecución.
- ¿Sería planificable el sistema si la tarea D tiene un tiempo de computo de 2500 ms? ¿Qué cambios habría que hacer en el ejecutivo cíclico anterior?