Memoria de P7

POSIX: relojes y temporizadores

Shunya Zhan

08 de mayo del 2025

ÍNDICE

Introducción

En el ámbito de los sistemas operativos POSIX, la gestión precisa del tiempo es esencial para una variedad de aplicaciones, desde sistemas de monitoreo hasta la ejecución de tareas periódicas en tiempo real. Para abordar esta necesidad, las funciones de temporización y las señales en tiempo real proporcionan herramientas poderosas y flexibles.

En esta sesión, se desarrollaron varios programas en C con el objetivo de explorar y comprender los conceptos fundamentales relacionados con la medición del tiempo en sistemas POSIX. Se utilizaron funciones como clock_gettime, nanosleep, timer_create y señales en tiempo real (SIGRTMIN) para implementar temporizadores, retardos y mediciones de tiempo. Cada uno de estos programas se diseñó para ilustrar diferentes aspectos de la interacción con los relojes del sistema y la gestión de eventos periódicos.

- **El primer programa** se centró en la medición de los ticks del sistema, utilizando sysconf(_SC_CLK_TCK) para obtener el número de ticks por segundo, lo que define la resolución mínima de tiempo del sistema. Este conocimiento es fundamental para entender la granularidad con la que se pueden medir y gestionar los intervalos de tiempo.
- **El segundo programa** exploró la medición de tiempos utilizando diferentes relojes disponibles en sistemas POSIX, como CLOCK_REALTIME, CLOCK_MONOTONIC, CLOCK_PROCESS_CPUTIME_ID y CLOCK_THREAD_CPUTIME_ID. Se calcularon las resoluciones de estos relojes y se compararon los tiempos obtenidos, proporcionando una visión clara de las características y aplicaciones adecuadas para cada tipo de reloj.
- **El tercer programa** implementó un retardo relativo utilizando nanosleep, con un enfoque en la gestión de interrupciones durante el retardo. Se configuró un retardo de 2.550 segundos y se imprimió el tiempo restante en caso de interrupción, demostrando cómo manejar situaciones en las que el retardo puede ser interrumpido por señales.
- **El cuarto programa** implementó un bucle periódico utilizando clock_nanosleep con la opción TIMER_ABSTIME, que permite especificar tiempos absolutos para la ejecución de tareas periódicas. Este enfoque es útil para garantizar que las tareas se ejecuten a intervalos precisos, independientemente de las variaciones en la duración de las tareas individuales.
- **El quinto programa** creó un temporizador que utiliza señales en tiempo real (SIGRTMIN) para notificar la expiración cada 3 segundos. Se manejaron las señales con un manejador que imprime el número de expiraciones, proporcionando un ejemplo práctico de cómo utilizar temporizadores y señales para gestionar eventos periódicos en sistemas POSIX.

En conjunto, estos programas ofrecen una base sólida para comprender y aplicar técnicas de temporización y gestión de eventos en sistemas POSIX, esenciales para el desarrollo de aplicaciones en tiempo real, monitoreo y otras tareas periódicas.

Desarrollo

Medición de ticks del sistema (codigo1.c)

Objetivo: Obtener el número de ticks por segundo del sistema, lo que define la resolución mínima de tiempo.

Implementación:

- Se utiliza la función sysconf(_SC_CLK_TCK) para obtener el valor de los ticks por segundo.
- Este valor se imprime para que el usuario pueda conocer la resolución mínima de tiempo del sistema.

Medición de tiempos con diferentes relojes (codigo2.c)

Objetivo: Medir tiempos utilizando diferentes relojes del sistema y comparar sus resoluciones.

Implementación:

- Se utilizan las funciones clock_gettime y clock_getres para medir el tiempo y la resolución de los relojes CLOCK_REALTIME, CLOCK_MONOTONIC, CLOCK_PROCESS_CPUTIME_ID y CLOCK_THREAD_CPUTIME_ID.
- Se imprimen los tiempos y resoluciones obtenidos para cada reloj.

Retardo relativo con nanosleep (codigo3.c)

Objetivo: Implementar un retardo de 2.550 segundos y manejar interrupciones durante el retardo.

Implementación:

- Se utiliza la función nanosleep para implementar el retardo.
- Se maneja la interrupción del retardo y se imprime el tiempo restante en caso de interrupción.

Bucle periódico con clock_nanosleep (codigo4.c)

Objetivo: Implementar un bucle que ejecuta una tarea cada 10 ms utilizando clock_nanosleep con TIMER_ABSTIME.

Implementación:

- Se utiliza clock_nanosleep para implementar el bucle periódico.
- Se ajusta el tiempo absoluto para cada iteración del bucle.

Temporizador con señales en tiempo real (codigo5.c)

Objetivo: Crear un temporizador que utiliza SIGRTMIN para notificar la expiración cada 3 segundos.

Implementación:

- Se utiliza timer_create para crear el temporizador y timer_settime para configurarlo.
- Se maneja la señal SIGRTMIN con un manejador que imprime el número de expiraciones.

Memoria

Durante el desarrollo de los programas, se identificaron y resolvieron varios problemas relacionados con la compilación y la lógica del código. A continuación, se detallan los problemas encontrados y las soluciones implementadas:

1. **Interrupciones en nanosleep en codigo3.c:**
    - Problema: Durante el retardo implementado con nanosleep, el retardo podía ser interrumpido por señales, lo que resultaba en una interrupción del retardo sin completar el tiempo especificado.
    - Solución: Se manejan las interrupciones durante el retardo y se imprime el tiempo restante en caso de interrupción, permitiendo al usuario conocer cuánto tiempo faltaba para completar el retardo.

2. **Desbordamiento de nanosegundos en codigo4.c:**
    - Problema: En el programa codigo4.c, la suma de struct timespec no manejaba correctamente el desbordamiento de nanosegundos, lo que podía causar errores en la temporización.
    - Solución: Se ajustaron los nanosegundos para que, si excedían 1 segundo (1,000,000,000 nanosegundos), se incrementara el valor de los segundos y se restara el exceso de nanosegundos.

Compilación y ejecución

Compilación: Abre una terminal y levantar el docker, e introducir en terminal siguiente comando para compilar los códigos:

- gcc -o codigo codigo.c

Nota: ya que hay 5 ejercicios, pues debería compilar los 5, y se ejecuta separado.

Y los códigos principales están proporcionados junto con el fichero zip.

Ejecución: En la terminal, ejecuta el servidor con el siguiente comando:

- ./codigo

Prueba y resultado

Codigo1:

```
rpcuserclient@937a38ebb92c:~/p8$ gcc -o codigo1 codigo1.c
rpcuserclient@937a38ebb92c:~/p8$ ./codigo1
Número de ticks por segundo: 100
```

Preguntas respondidas en comentario de código.

Codigo2:

```
rpcuserclient@937a38ebb92c:~/p8$ gcc -o codigo2 codigo2.c
rpcuserclient@937a38ebb92c:~/p8$ ./codigo2
Resolución de CLOCK_REALTIME: 0 segundos, 1 nanosegundos
Resolución de CLOCK_MONOTONIC: 0 segundos, 1 nanosegundos
Resolución de CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 1 nanosegundos
Resolución de CLOCK_THREAD_CPUTIME_ID: 0 segundos, 1 nanosegundos
Tiempo ejecución CLOCK_REALTIME: 1 segundos, 222468 nanosegundos
Tiempo ejecución CLOCK_MONOTONIC: 1 segundos, 222490 nanosegundos
Tiempo ejecución CLOCK_PROCESS_CPUTIME_ID: 0 segundos, 50600 nanosegundos
Tiempo ejecución CLOCK_THREAD_CPUTIME_ID: 0 segundos, 51800 nanosegundos
```

Preguntas respondidas en comentario de código.

Codigo3:

```
rpcuserclient@937a38ebb92c:~/p8$ gcc -o codigo3 codigo3.c
rpcuserclient@937a38ebb92c:~/p8$ ./codigo3
Voy a esperar 2.150000000 s ......
He despertado y ya termino.
```

1. **Funcionamiento del programa:**
   - El programa utiliza nanosleep para realizar un retardo relativo de 2.550 segundos (2 segundos y 550 milisegundos).
   - Si el retardo es interrumpido por una señal, nanosleep devuelve -1 y almacena el tiempo restante en retardo_pend.
2. **Comportamiento esperado:**
   - Si no hay interrupciones, el programa imprimirá: "He despertado y ya termino."

- Si el retardo es interrumpido, imprimirá el tiempo restante en segundos y nanosegundos.

3. **Posibles interrupciones:**
   - Una señal enviada al proceso durante el retardo puede interrumpir nanosleep. Esto es útil para manejar eventos externos mientras se espera.

Codigo4:

```
rpcuserclient@937a38ebb92c:~/p8$ gcc -o codigo4 codigo4.c
rpcuserclient@937a38ebb92c:~/p8$ ./codigo4
next: 90506 segundos, 263734981 nanosegundos
next: 90506 segundos, 273734981 nanosegundos
next: 90506 segundos, 283734981 nanosegundos
next: 90506 segundos, 293734981 nanosegundos
next: 90506 segundos, 303734981 nanosegundos
next: 90506 segundos, 313734981 nanosegundos
next: 90506 segundos, 323734981 nanosegundos
next: 90506 segundos, 333734981 nanosegundos
next: 90506 segundos, 343734981 nanosegundos
next: 90506 segundos, 353734981 nanosegundos
next: 90506 segundos, 363734981 nanosegundos
next: 90506 segundos, 373734981 nanosegundos
next: 90506 segundos, 383734981 nanosegundos
next: 90506 segundos, 393734981 nanosegundos
next: 90506 segundos, 403734981 nanosegundos
next: 90506 segundos, 413734981 nanosegundos
next: 90506 segundos, 423734981 nanosegundos
next: 90506 segundos, 433734981 nanosegundos
next: 90506 segundos, 443734981 nanosegundos
```

1. **Comportamiento del programa:**
   - El programa utiliza clock_nanosleep con TIMER_ABSTIME para realizar retardos absolutos basados en el reloj CLOCK_MONOTONIC.
   - En cada iteración, el valor de next se incrementa en 10 ms (period).
2. **Impresión de next:**
   - Los valores de next se imprimen en cada iteración, mostrando el tiempo absoluto en segundos y nanosegundos en el que el programa espera.
3. **Problema en el código original:**
   - La línea next = next + period; no es válida para sumar estructuras timespec. Esto se corrige sumando los campos tv_sec y tv_nsec manualmente y ajustando los nanosegundos si exceden 1 segundo.
4. **Qué está pasando:**

- El programa genera un bucle infinito que espera en intervalos de 10 ms. Los valores de next aumentan progresivamente, representando los tiempos absolutos en los que el programa debería despertar.

Codigo5:

```
rpcuserclient@937a38ebb92c:~/p8$ gcc -o codigo5 codigo5.c
rpcuserclient@937a38ebb92c:~/p8$ ./codigo5
Timer expired 1 times
Timer expired 2 times
Timer expired 3 times
Timer expired 4 times
Timer expired 5 times
Timer expired 6 times
Timer expired 7 times
Timer expired 8 times
Timer expired 9 times
Timer expired 10 times
Timer expired 11 times
Timer expired 12 times
Timer expired 13 times
Timer expired 14 times
```

¿Qué hace la línea signal(SIGRTMIN, timer_handler)?

Configura un manejador de señales (timer_handler) para la señal SIGRTMIN. Cuando el temporizador expira, se envía la señal SIGRTMIN al proceso, y el manejador imprime un mensaje indicando cuántas veces ha expirado el temporizador.

¿Para qué se usa SIGRTMIN?

SIGRTMIN es una señal en tiempo real disponible en sistemas POSIX. Se utiliza para notificaciones específicas del usuario o del sistema, como en este caso, para manejar la expiración de un temporizador.

Puedes consultar las señales disponibles con el comando kill -l. En la mayoría de los sistemas, las señales en tiempo real comienzan desde SIGRTMIN y terminan en SIGRTMAX.

¿Qué está haciendo el programa?

El programa crea un temporizador que utiliza el reloj CLOCK_REALTIME.

Configura el temporizador para que expire después de 3 segundos y luego se repita cada 3 segundos.

Cuando el temporizador expira, se envía la señal SIGRTMIN, y el manejador de señales imprime un mensaje indicando cuántas veces ha expirado el temporizador.

El programa se mantiene en un bucle infinito, esperando señales con la función pause().

Conclusión

Los programas desarrollados en esta sesión han demostrado cómo interactuar eficazmente con los relojes del sistema y manejar eventos periódicos en sistemas POSIX. A través del uso de funciones como clock_gettime, nanosleep, timer_create y señales en tiempo real (SIGRTMIN), se han explorado y aplicado conceptos fundamentales para la medición del tiempo, la implementación de retardos y la gestión de temporizadores.

Cada programa abordó un aspecto específico de la temporización:

- **Medición de ticks del sistema:** Proporcionó una comprensión básica de la resolución mínima de tiempo del sistema.
- **Medición de tiempos con diferentes relojes:** Comparó las resoluciones y tiempos obtenidos de varios relojes, destacando sus aplicaciones adecuadas.
- **Retardo relativo con nanosleep**: Implementó un retardo preciso y manejó interrupciones, mostrando cómo gestionar pausas en la ejecución.
- **Bucle periódico con clock_nanosleep:** Demostró la ejecución de tareas periódicas con precisión utilizando tiempos absolutos.
- **Temporizador con señales en tiempo real:** Ilustró el uso de temporizadores y señales para notificar eventos periódicos, proporcionando un ejemplo práctico de su aplicación.

Estos programas no solo han servido para entender mejor las capacidades de temporización en sistemas POSIX, sino que también han proporcionado una base sólida para el desarrollo de aplicaciones en tiempo real, monitoreo y otras tareas periódicas. Los problemas encontrados durante el desarrollo fueron resueltos mediante ajustes en el código y la inclusión de los encabezados necesarios, asegurando así la robustez y funcionalidad de las implementaciones.

En resumen, las herramientas y técnicas exploradas son fundamentales para cualquier desarrollador que trabaje en entornos POSIX, permitiendo la creación de sistemas eficientes y precisos en la gestión del tiempo.

Anexo

- Código fuente
- Campus Virtual
- Copilot