Timers

Older interfaces

У ранніх версіях Unix для реалізації таймерів використовувалися прості інтерфейси, які сьогодні вважаються застарілими. Найвідоміші з них:

- alarm() — викликає сигнал SIGALRM через задану кількість секунд. Простий у використанні, але дуже обмежений: лише один таймер на процес.

- sleep(), usleep() — блокують виконання потоку/процесу на заданий час. Не дають можливості обробляти події асинхронно.

- setitimer() — дозволяє створити інтервальний таймер, який надсилає сигнал (SIGALRM, SIGVTALRM, SIGPROF) через вказаний інтервал. Дає трохи більше контролю, ніж alarm, але все одно має багато обмежень:

    - Таймерів всього три (по одному на тип сигналу).

    - Важко інтегрувати з асинхронними системами або epoll.

    - В основному керуються через сигнали, а не дескриптори.

Interval timers

setitimer() дозволяє створити таймер, який:

- Викликає сигнал через вказаний початковий інтервал (it_value).

- Повторюється через заданий інтервал (it_interval), якщо не 0.

Типи таймерів:

- ITIMER_REAL: генерує SIGALRM — реальний час (wall-clock).

- ITIMER_VIRTUAL: генерує SIGVTALRM — час виконання процесу.
- ITIMER_PROF: генерує SIGPROF — час користувача + ядра.

Використання getitimer() дозволяє запитати поточне значення таймера.

Застосування: профілювання програм, автоматичне завершення, примітивні timeouts.

A simple CLI digital clock

На базі setitimer() або alarm() можна реалізувати простий цифровий годинник у CLI:

1. Використовуйте SIGALRM, щоб щосекунди оновлювати екран.
2. В обробнику сигналу малюйте годину:хвилини:секунди.
3. Обробку можна реалізувати через write() (вона reentrant-безпечна).

```
void handler(int sig) {
   time_t now = time(NULL);
   struct tm *tm_info = localtime(&now);
   char buf[64];
   strftime(buf, sizeof(buf), "%H:%M:%S\n", tm_info);
   write(STDOUT_FILENO, buf, strlen(buf));
   alarm(1);
}
```

The newer POSIX interval timers mechanism

Сучасна альтернатива — POSIX timers, які дають змогу:

- Створювати кілька таймерів.
- Працювати з точністю до наносекунд.
- Отримувати події у вигляді сигналів або дескрипторів.
- Інтегрувати таймери в epoll (через timerfd_create — розширення Linux).

Основні функції:

- timer_create() — створити таймер.
- timer_settime() — запустити або скинути.
- timer_gettime() — запитати статус.
- timer_delete() — знищити.

Таймери можуть генерувати сигнал (SIGEV_SIGNAL) або запускати окремий потік (SIGEV_THREAD).

Arming, disarming and querying a POSIX timer

При створенні таймера через timer_create() ми можемо обрати:

- Чи буде таймер одноразовим чи інтервальним.
- Який сигнал буде згенеровано.
- Який обробник його оброблятиме (через sigaction).

Таймер активується функцією timer_settime():

- it_value – час першого спрацювання.

- it_interval – інтервал повторного спрацювання (0 — одноразовий).

Таймер можна вимкнути, передавши it_value = {0, 0}.

Запитати статус можна через timer_gettime():

- Повертає поточне залишкове значення і період.

Example program – POSIX interval timers

```
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

void handler(int sig, siginfo_t *si, void *uc) {
    write(STDOUT_FILENO, "Tick\n", 5);
}

int main() {
    struct sigaction sa = {0};
    sa.sa_flags = SA_SIGINFO;
    sa.sa_sigaction = handler;
    sigaction(SIGRTMIN, &sa, NULL);

    timer_t timerid;
```

```
    struct sigevent sev = {0};
    sev.sigev_notify = SIGEV_SIGNAL;
    sev.sigev_signo = SIGRTMIN;
    timer_create(CLOCK_REALTIME, &sev, &timerid);
    struct itimerspec its;
    its.it_value.tv_sec = 1;
    its.it_value.tv_nsec = 0;
    its.it_interval.tv_sec = 1;
    its.it_interval.tv_nsec = 0;

    timer_settime(timerid, 0, &its, NULL);

    while (1)
        pause();
}
```

Код в прикладі вище демонструє:

- Використання SIGRTMIN для уникнення конфліктів зі стандартними сигналами.
- Таймер із повторним інтервалом.
- Асинхронний обробник sa_sigaction.

Завдання до практичних

1. Змініть програму з прикладу Example program – POSIX interval timers так, щоб вона оновлювала системний лічильник у змінній (наприклад, рахувала тики).

2. Реалізуйте два незалежних POSIX таймери, які працюють на різних інтервалах та логують в окремі файли.

3. Реалізуйте програму, яка оновлює лічильник кожні 100 мс та вимикає таймер через 5 секунд.

4. Додайте можливість динамічного змінення інтервалу таймера без перезапуску програми.

5. Дослідіть, як поводиться таймер у стані sleep/suspend (через CLOCK_MONOTONIC vs CLOCK_REALTIME).

6. Реалізуйте зворотний відлік з виведенням часу що залишився до події.

7. Напишіть тест, який перевіряє точність POSIX-таймера на великих інтервалах (від 1 до 60 секунд).

8. Вбудуйте timerfd_create() замість timer_create() для синхронної обробки через select() або epoll.

9. Реалізуйте CLI-таймер, який виводить повідомлення через рівно N секунд без використання сигналів (через clock_nanosleep()).

10. Порівняйте поведінку alarm(), setitimer() та timer_create() у програмі з двома потоками.
11. Реалізуйте планувальник подій, який дозволяє динамічно додавати/видаляти події з таймером, використовуючи POSIX-таймери або timerfd_create.
12. Створіть daemon, який запускає зовнішній процес через задані інтервали часу, логуючи запуск і завершення.

13. Дослідіть поведінку POSIX-таймерів при високому навантаженні (запуск сигналу частіше, ніж програма встигає обробити).
14. Створіть гру-реакцію (reaction time game), яка вимірює час реакції користувача на появу сигналу з таймера.
15. Розробіть систему нагадувань, яка дозволяє користувачеві налаштовувати багаторазові або одноразові події з різними інтервалами.

16. Напишіть власний аналог cron, що використовує POSIX-таймери для планування запуску команд.

17. Вивчіть, як зміна системного часу (через date або ntp) впливає на CLOCK_REALTIME та CLOCK_MONOTONIC таймери.

18. Реалізуйте механізм автоматичного вимкнення серверу після простою (немає запитів протягом N секунд) із використанням таймера.

19. Побудуйте систему, яка фіксує втрати тика таймера через переповнення сигналів (SIGEV_SIGNAL) при навантаженні.

20. Використовуйте таймери для реалізації автоматичного лог-ауту користувача у CLI-додатку після N хвилин бездіяльності.

21. Дослідіть поведінку таймерів у багатопоточному додатку — створіть таймер у кожному потоці і порівняйте результат.

22. Напишіть бенчмарк, який порівнює точність і стабільність sleep(), nanosleep(), setitimer() і timer_create().

23. Реалізуйте механізм "debounce" для подій, що надходять від користувача (наприклад, клавіатура) через POSIX-таймер.

24. Створіть систему захисту від DoS: обмежте частоту певних дій користувача через таймери блокування.

25. Побудуйте з використанням POSIX-таймера пульсацію LED або UART-сигналу на вбудованому пристрої (реальна система або симуляція).

26. Реалізуйте таймер з обліком відставання та компенсацією дрифту (з урахуванням того, що події не завжди точно вчасно спрацьовують).

27. Імітуйте таймерну чергу (timer queue), де заплановані події обробляються в точний час, з підтримкою пріоритетів і скасування.