Echtzeitbetriebssysteme — Übung

Oliver Jack

Ernst-Abbe-Hochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Sommersemester 2025

Ernst-Abbe-Hochschule Jena
University of Applied Sciences

Übung 4: Timer

Timer

- Zyklische Ausführung von Tasks.
- Timer sind Mechanismen, mit denen sich Aufgaben nach einem bestimmten Intervall selbst signalisieren.
- Timer bauen auf den Uhr- und Signalfunktionen auf. Die Uhr stellt eine absolute Zeitbasis zur Verfügung.
- Die Standard-Timer-Funktionen bestehen einfach aus dem Erstellen, Löschen und Setzen eines Timers.
- Wenn ein Timer abläuft, muss `sigaction()` (siehe sigLib) vorhanden sein, damit der Benutzer das Ereignis behandeln kann.
- Die Funktion „high resolution sleep“, `nanosleep()`, ermöglicht das Schlafen im Subsekundenbereich mit der Auflösung der Uhr.
- Die clockLib-Bibliothek sollte installiert und `clock_settime()` gesetzt werden, bevor irgendwelche Timer-Routinen verwendet werden.

Beschreibung

- `timer_cancel()` - bricht einen Timer ab
- `timer_connect()` - verbindet eine Benutzerroutine mit dem Timer-Signal
- `timer_create()` - weist einen Timer unter Verwendung der angegebenen Uhr als Zeitbasis zu (POSIX)
- `timer_delete()` - entfernt einen zuvor erstellten Timer (POSIX)
- `timer_gettime()` - ermittelt die verbleibende Zeit bis zum Ablauf und den Wert für das Nachladen (POSIX) ermittelt die verbleibende Zeit bis zum Ablauf und den Reload-Wert (POSIX)
- `timer_getoverrun()` - gibt die Überschreitung des Timer-Ablaufs zurück (POSIX)
- `timer_settime()` - setzt die Zeit bis zum nächsten Ablauf und schaltet den Timer scharf (POSIX)
- `nanosleep()` - hält die aktuelle Task an, bis das Zeitintervall abgelaufen ist (POSIX)

Beispiel VxWorks_timer.c

```
/* POSIX timers */
#include "vxWorks.h"
#include "time.h"
#include "timexLib.h"
#include "taskLib.h"
#include "sysLib.h"
#include "stdio.h"
#define TIMER_START 10
#define TIMER_INTERVAL 5
/* timer is connected to timerhandle() */
void timerhandle(timer_t timerID, int targ) {
        int i;
        printf("timerhandle␣invoked␣with␣targ␣=␣%d\n", targ);
        /* some CPU eating stuff */
        for (i = 0; i < 200000; i++) {
        };
}
```

Beispiel VxWorks_timer.c

```
int execTimer(void) {
        sysClkRateSet(100);
        timer_t timerID;
        struct itimerspec value, ovalue, gvalue;
        int t_arg = 12321;
        int i;
        if (timer_create(CLOCK_REALTIME, NULL, &timerID) == ERROR) {
                printf("create FAILED\n");
                return (ERROR);
        }
        if (timer_connect(timerID, (VOIDFUNCPTR) timerhandle, t_arg) == ERROR) {
                printf("connect FAILED\n");
                return (ERROR);
        }
        value.it_value.tv_nsec = 0;
        value.it_value.tv_sec = TIMER_START;
        value.it_interval.tv_nsec = 0;
        value.it_interval.tv_sec = TIMER_INTERVAL;
        printf("timer set up for start after %ld sec and interval %ld sec\n",
                        value.it_value.tv_sec, value.it_interval.tv_sec);
        if (timer_settime(timerID, TIMER_RELTIME, &value, &ovalue) == ERROR) {
                printf("timer_settime FAILED\n");
                return (errno);
        }
```

Beispiel VxWorks_timer.c

```
        /* some diagnostics during 25 sec */
        for (i = 0; i < 25; i++) {
                if (timer_gettime(timerID, &gvalue) == ERROR) {
                        printf("gettime␣FAILED\n");
                        return (errno);
                }
                printf("gvalue.it_value.tv_sec␣=␣%ld\n", gvalue.it_value.tv_sec);
                printf("gvalue.it_interval.tv_sec␣=␣%ld\n", gvalue.it_interval.
                    tv_sec);
                taskDelay((unsigned int)CLOCKS_PER_SEC);
        }
        if (timer_cancel(timerID) == ERROR) {
                printf("cancel␣FAILED\n");
                return (errno);
        }
        if (timer_delete(timerID) == ERROR) {
                printf("delete␣FAILED\n");
                return (errno);
        }
        return (OK);
}
```

Experiment

Aufgabe: Bringen Sie das Beispielprogramm VxWorks_timer.c zum laufen.

Aufgabe: Variieren Sie `TIMER_START`, `TIMER_INTERVAL` und `sysClkRateSet` und beobachten Sie die Änderung des Programmablaufs.

Aufgabe: Schreiben Sie ein Programm, in dem zyklisch alle 50 Millisekunden eine Task ausgeführt wird. Die Task bekommt eine Zeit übergeben (die vergangene Zeit seit der letzten Ausführung, Zeitintervall) und eine Geschwindigkeit. Die Task berechnet die zurückgelegte Strecke in dem Zeitintervall auf Basis der Geschwindigkeit. Das Programm soll im Sekundentakt die zurückgelegte Gesamtstrecke ausgeben.