Beispiele zu Kapitel 5: Programmieren mit Zeit

Aus: Alan Burns, Andy Wellings: *Real-Time Systems and Programming Languages. Ada, Real-Time Java and C/Real-Time POSIX.*Addison Wesley, 2009.(Kapitel 7, 9 und 10)

Beispiel 5-1:Das **`Calendar`**-Package von Ada [Burns &Wellings 2009, Kap. 9.2.1]

```
package Ada.Calendar is

   type Time is private;

   subtype Year_Number is Integer range 1901..2099;
   subtype Month_Number is Integer range 1..12;
   subtype Day_Number is Integer range 1..31;
   subtype Day_Duration is Duration range 0.0..86400.0;

   function Clock return Time;

   function Year(Date:Time) return Year_Number;
   function Month(Date:Time) return Month_Number;
   function Day(Date:Time) return Day_Number;
   function Seconds(Date:Time) return Day_Duration;

   procedure Split(Date:in Time; Year:out Year_Number;
                   Month:out Month_Number; Day:out Day_Number;
                   Seconds:out Day_Duration);

   function Time_Of(Year:Year_Number; Month:Month_Number;
                   Day:Day_Number; Seconds:Day_Duration := 0.0)
                   return Time;

   function "+"(Left:Time;Right:Duration) return Time;
   function "+"(Left:Duration;Right:Time) return Time;
   function "-"(Left:Time;Right:Duration) return Time;
   function "-"(Left:Time;Right:Time) return Duration;

   function "<"(Left,Right:Time) return Boolean;
   function "<="(Left,Right:Time) return Boolean;
   function ">"(Left,Right:Time) return Boolean;
   function ">="(Left,Right:Time) return Boolean;

   Time_Error:exception;
   -- Time_Error is raised by Time_Of, Split,"+",and "-"

private
   -- implementation dependent
end Ada.Calendar;
```

Beispiel 5-2:Das **`Real_Time`**-Package von Ada[Burns &Wellings 2009, Kap. 9.2.2]

```
package Ada.Real_Time is
   type Time is private;
   Time_First: constant Time;
   Time_Last: constant Time;
   Time_Unit: constant :=  -- implementation-defined-real-number;

   type Time_Span is private;
   Time_Span_First: constant Time_Span;
   Time_Span_Last: constant Time_Span;
   Time_Span_Zero: constant Time_Span;
   Time_Span_Unit: constant Time_Span;

   Tick: constant Time_Span;
   function Clock return Time;

   function "+" (Left: Time; Right: Time_Span) return Time;
   ...

   function "<" (Left, Right: Time) return Boolean;
   ...

   function "+" (Left, Right: Time_Span) return Time_Span;
   ...

   function "<" (Left, Right: Time_Span) return Boolean;
   ...

   function "abs"(Right : Time_Span) return Time_Span;

   function To_Duration (Ts : Time_Span) return Duration;
   function To_Time_Span (D : Duration) return Time_Span;

   function Nanoseconds  (Ns: Integer) return Time_Span;
   function Microseconds (Us: Integer) return Time_Span;
   function Milliseconds (Ms: Integer) return Time_Span;

   type Seconds_Count is range -- implementation-defined

   procedure Split(T : in Time; Sc: out Seconds_Count;
                   Ts : out Time_Span);
   function Time_Of(Sc: Seconds_Count; Ts: Time_Span) return Time;

private
   -- not specified by the language
end Ada.Real_Time;
```

Beispiel 5-3:Die ANSI-C-Schnittstelle für Datum und Zeit[Burns &Wellings 2009, Kap. 9.2.3]

```
typedef ... time_t;

struct tm {
  int tm_sec;     /* seconds after the minute - [0, 61] */
                  /* 61 allows for 2 leap seconds */
  int tm_min;     /* minutes after the hour - [0, 59] */
  int tm_hour;    /* hour since midnight - [0, 23] */
  int tm_mday;    /* day of the month - [1, 31] */
  int tm_mon;     /* months since January - [0, 11] */
  int tm_year;    /* years since 1900 */
  int tm_wday;    /* days since Sunday - [0, 6] */
  int tm_yday;    /* days since January 1 - [0, 365] */
  int tm_isdst;   /* flag for alternate daylight savings time */
}; double difftime(time_t time1, time_t time2);
  /* subtract two time values */
time_t mktime(struct tm *timeptr); /* compose a time value */
time_t time(time_t *timer);
  /* returns the current time and if timer is not null */
  /*it also places the time at that location */
```

Beispiel 5-4:Die C/Real-Time POSIX-Schnittstelle für Uhren[Burns &Wellings 2009, Kap. 9.2.3]

```
#define CLOCK_REALTIME ...;
#define CLOCK_PROCESS_CPUTIME_ID ...;
#define CLOCK_THREAD_CPUTIME_ID ...;

struct timespec {
  time_t tv_sec;   /* number of seconds */
  long   tv_nsec;  /* number of nanoseconds */
};
typedef ... clockid_t;

int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
int clock_getres(clockid_t clock_id, struct timespec *res);

int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
int clock_getcpuclockid(pthread_t_t thread_id, clockid_t *clock_id);

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
/* Note, that a nanosleep return -1 if the sleep is interrupted */
/* by a signal. In this case, rmtp has the remaining sleep time */

intclock_nanosleep(clockid_t clock_id, int flags,
    const struct timespec *rqtp, struct timespec *rmtp);
/* if flag = TIMER_ABSTIME, then the sleep is absolute */
/* using the identified clock */
```

Beispiel 5-5: C/Real-Time POSIX– Datenstrukturen für die Generierung von Signalen [Burns &Wellings 2009, Kap. 7.5.1]

```
/* used with message queue notification, timers etc */

struct sigevent {
  int sigev_notify;
    /* SIGEV_SIGNAL, */
    /* SIGEV_THREAD or SIGEV_NONE */

  int sigev_signo;  /* signal to be generated */

  union sigval sigev_value; /* value to be queued */

  void (*)sigev_notify_function(union sigval s);
  /* function to be treated as thread */
  pthread_attr_t *sigev_notify_attributes;
  /* thread attributes */
};

union sigval {
  int sival_int;
  void *sival_ptr;
};
```

Beispiel 5-6: C/Real-Time POSIX – Schnittstelle für Signale [Burns &Wellings 2009, Kap. 7.5.1]

```
typedef ... sigset_t;

/* the following manipulates the signal mask */
int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
  /* how = SIG_BLOCK -> the set is added to the current set */
  /* how = SIG_UNBLOCK -> the set is subtracted from the */
  /*       current set */
  /* how = SIG_SETMASK -> the given set becomes the mask */

/* the following routines allow a signal */
/* set to be created and manipulated */
int sigemptyset(sigset_t *s); /* initialize a set to empty */
int sigfillset(sigset_t *s); /* initialize a set to full */
int sigaddset(sigset_t *s, int signum); /* add a signal */
int sigdelset(sigset_t *s, int signum); /* remove a signal */
int sigismember(const sigset_t *s, int signum);
   /* returns 1 if a member */

/* the following support signal handling */

typedef struct { /* signal parameters */
  int si_signo;
  int si_code;
  union sigval si_value;
} siginfo_t;

struct sigaction {
  void (*sa_handler) (int signum); /* non real-time handler */
  void (*sa_sigaction) (int signum, siginfo_t *data,
                        void *extra); /*real-time handler */
  sigset_t sa_mask; /* signals to mask during handler */
  int sa_flags; /*indicates if signal is to be queued */
};
```

```
int sigaction(int sig, const struct sigaction *reaction,
              struct sigaction *old_reaction);
/* sets up a signal handler, reaction, for sig */

/* the following functions allow a  */
/* process to wait for a signal */

int sigsuspend(const sigset_t *sigmask);
int sigwaitinfo(const sigset_t *set, siginfo_t *info);
int sigtimedwait(const sigset_t *set, siginfo_t *info,
                 const struct timespec *timeout);

/* the following functions allow a  */
/* signal to be sent */
int kill (pid_t pid, int sig);
   /* send the signal sig to the process pid */
int sigqueue(pid_t pid, int sig, const union sigval value);
   /* send signal and data */

/* All the above functions return -1 when errors have occurred. */
/* A shared variable errno contains the reason for the error */
```

Beispiel 5-7: C/Real-Time POSIX-Schnittstelle für Timer[Burns &Wellings 2009, Kap. 10.4.2]

```
#define TIMER_ABSTIME ..

struct itimerspec {
  struct timespec it_value; /* first timer signal */
  struct timespec it_interval; /* subsequent intervals */
};
typedef ... timer_t;


int timer_create(clockid_t clock_id, struct sigevent *evp,
                 timer_t *timerid);
  /* Create a per-process timer using the specified clock as the */
  /* timing base. evp points to a structure which contains */
  /* all the information needed concerning the signal */
  /* to be generated. */

int timer_delete(timer_t timerid);
  /* delete a per-process timer */
```

```
int timer_settime(timer_t timerid, int flags,
                  const struct itimerspec *value,
                  struct itimerspec *ovalue);
  /* Set the next expiry time for the timer specified. */
  /* If flags is set to TIMER_ABSTIME, then */
  /* the timer will expire when the clock reaches the
  /* absolute value specified by *value.it_value */
  /* if flags is NOT set to TIMER_ABSTIME, then the timer will */
  /* expire when the interval specified by value->it_value passes */
  /* if *value.it_interval is non-zero, then a periodic timer will */
  /* go off every value->it_interval after value->it_value has */
  /* expired */
  /* Any previous timer setting is returned in *ovalue. */


int timer_gettime(timer_t timerid, struct itimerspec *value);
  /* get the details of the current timer */

int timer_getoverrun(timer_t timerid);
  /* if real-time signals are supported, return the number of signals */
  /* that have been generated by this timer but not handled */


/* All the above functions, except timer_getoverrun, return 0 if */
/* successful, otherwise -1. timer_getoverrun returns the number */
/* of overruns.  When an error condition is returned by any of */
/* the above functions, a shared variable errno contains the */
/* reason for the error */
```