Системное время

Время в вычислительном устройстве определяется при помощи счетчика тактов.

Для привязки тактов к реальному времени существует специальный счетчик — RTC (Real-Time Clock). Точность RTC, как правило, составляет 1 секунду, но при этом обеспечивается абсолютное позиционирование во временной шкале.

Для вычисления времени с большей точностью применяют счетчики тактов процессора, но такое позиционирование всегда относительное.

RTC

- Подключен к отдельному элементу питания.
- Считает такты отдельного кварцевого резонатора, настроенного на 32768 Гц.
- Имеет дрейф от 1 до 8 секунд в сутки.
- Ведущему устройству доступны команды чтения и установки текущего времени в виде целого количества секунд.

Точное время

- Источник питания совпадает с источником питания системы
- Источник колебаний — кварцевый резонатор, задающий частоту системной шины.
- Счетчик может быть настроен на произвольный модуль счета. Интервал срабатывания таймера настраивается программно.
- Точность временной привязки определяется характеристиками резонатора.
- Система получает прерывание на каждое срабатывание счетчика.

Типы времени

- Локальное время. Определяется астрономическими методами в конкретном месте.
- Универсальное координированное время (UTC) — локальное время на нулевой широте.

Локальное время отличается от UTC на целое число часов (часовой пояс).

Конкретный часовой пояс в каждой точке пространства не стандартизирован.

Время Unix

- Является стандартным способом представления времени для большинства систем секундной точности.
- Записывается как целое число секунд с 1 января 1970 года UTC.
- Не имеет географической привязки. Трактовка времени как локальное время или UTC остается на стороне пользователя.
- Программный интерфейс для доступа к времени в формате Unix входит в стандартную библиотеку любого языка программирования.

Время Unix

Проблема времени Unix — изначальный формат представления: 4 байта, знаковое число. Диапазон представления — от 13 декабря 1901 г до 19 января 2038 г.

Современные системы используют 64 бита для хранения времени Unix и лишены этой проблемы. Тем не менее, до 2038 года могут дожить старое встроенное ПО и многие форматы файлов (например zip).

Время NTP

- Используется в протоколе NTP для синхронизации часов.
- Отводит 64 бита на представление времени. При этом цена младшего разряда составляет 2^{-32} с (0.238 нс).
- Старшие 32 бита определяют количество секунд.
- Одна эпоха NTP длится 138 лет. Перед началом синхронизации часов, приложение должно узнать эпоху NTP, получив ее от других средств. Например, от часов реального времени. Текущая эпоха NTP начинается 1 января 1900 года UTC.

Время FILETIME

- Используется в системе MS Windows для представления времени.
- Отводит 64 бита на хранение времени. Цена младшего разряда — 100 нс. Начальная точка — 1 января 1601 года UTC.
- Период отображения времени порядка 58 тысяч лет.
- Старшие и младшие разряды хранятся в виде отдельных 32-битных переменных.

Время time_t

Тип данных **time_t** является типом для представления времени в стандартной библиотеке языка C.

```
#include <time.h>
time_t t = 0;
```

Для получения текущего времени Unix в виде time_t необходимо вызвать функцию time:

```
t = time(NULL);
time(&t);
```

Время time_t

Стандартная библиотека содержит ряд функций для преобразования времени.

Функция **ctime** преобразует время в строку, отформатированную в соответствии с текущей локалью:

```
#include <time.h>
time_t t = time(0);
printf("%s\n", ctime(&t));   // Mon Sep 28 22:29:16 2020
```

Функция **localtime** позволяет преобразовать время Unix в структуру **tm**, которая содержит отдельные составляющие времени.

```
struct tm* tt = localtime(&t);
```

Функция **mktime** позволяет преобразовать структуру **tm** обратно во время Unix.

Время time_t

```
struct tm
{
    int tm_sec;         // Секунды      (0-60)
    int tm_min;         // Минуты       (0-59)
    int tm_hour;        // Часы         (0-23)
    int tm_mday;        // День месяца  (1-31)
    int tm_mon;         // Месяц        (0-11)
    int tm_year;        // Год          0 = 1900
    int tm_wday;        // День недели  (0-6, 0 — воскр.)
    int tm_yday;        // День в году  (0-365)
    int tm_isdst;       // Признак зимнего времени
};
```

Точное время в C

Получение точного времени в C — платформозависимая задача.

В Unix, а также в mingw, существует заголовочный файл **sys/time.h**. Он содержит функцию **gettimeofday**, возвращающую время в виде структуры **timeval**. Структура timeval содержит время Unix в секундах и миллисекунды.

```
int a=0;
struct timeval tv;
printf("%d sec, %d usec", tv.tv_sec, tv.tv_usec);
```

Время FILETIME можно получить системным вызовом **GetSystemTimeAsFileTime**.

```
FILETIME ft;
GetSystemTimeAsFileTime(&ft);
printf("%u, %u", ft.dwLowDateTime, ft.dwHighDateTime)
```

Точное время в C++

Начиная с C++11, функции для работы с точным временем вынесены в стандартную библиотеку C++ в заголовочный файл **chrono**.

```
#include <chrono>
using namespace std;

auto t1 = chrono::system_clock_now();
time_t tt = chrono::system_clock::to_time_t(t);
```

Файл chrono содержит средства для преобразования интервала времени в необходимые единицы измерения

```
auto t1 = chrono::steady_clock::now();
// <...>
auto t2 = chrono::steady_clock::now();
int ms = chrono::duration_cast
            <chrono::microseconds>(t2-t1).count();
```

Время и дата в Qt

Qt содержит следующие классы для работы с датой и временем:

- **QTime** — позволяет работать со временем в пределах одних суток.
- **QDate** — позволяет работать с датой.
- **QDateTime** — позволяет работать одновременно с датой и временем, но не содержит некоторых специфических функций.

```
#include <QDateTime>
QDateTime dt = QDateTime::currentDateTime();
time_t t = dt.toSecsSinceEpoch();
```

Время и дата в Qt

Все классы Qt для работы со временем поддерживают форматированный вывод при помощи метода **toString**. Аргументом метода является строка с форматом преобразования. Символы, не совпадающие с маркерами, выводятся как есть.

Маркер	Назначение	Маркер	Назначение
d	День месяца (1-31)	h	Часы (0-23, либо 1-12 AM/PM)
dd	День месяца (01-31)	hh	Часы (00-23, либо 01-12 AM/PM)
ddd	День недели ("Пн", "Вт")	H	Часы (0-23)
dddd	День недели ("понедельник")	HH	Часы (00-23)
M	Месяц (1-12)	m	Минуты (0-59)
MM	Месяц (01-12)	mm	Минуты (00-59)
MMM	Месяц ("Янв", "Февр")	s	Секунды (0-59)
MMMM	Месяц ("Январь")	ss	Секунды (00-59)
yy	Год (0-99)	z	Миллисекунды (0-999)
yyyy	Год (2020)	zzz	Миллисекунды (000-999)
t	Часовой пояс	a, ap / A, AP	am, pm / AM, PM

Время и дата в Qt

Класс QTime позволяет измерять временные интервалы с миллисекундной точностью.

Метод **start** запускает отсчет, фиксируя момент времени. Метод **restart** перезапускает отсчет.

Метод **elapsed** возвращает количество миллисекунд с последнего вызова start или restart.

```
#include <QTime>
#include <QtDebug>
QTime t1;
t1.start();
// <...>
qDebug() << t1.elapsed() << "мс затрачено";
```

Время и дата в Python

Python содержит модуль **time**, который предоставляет функции по работе со временем.

Одноименная функция **time** возвращает время в виде вещественного числа секунд, прошедших с 1 января 1970 UTC.

```
import time
print(time.time())    # 1601324255.975494
```

Такой формат представления времени позволяет легко измерить временной интервал:

```
t1 = time.time()
# <...>
t2 = time.time()
elapsed = t2 – t1     # Разница в секундах
```

Время и дата в Python

Для представления времени в виде отдельных компонентов используется функция **localtime**. Она возвращает именованный тьюпл **time.struct_time**.

```
import time
tt = time.localtime(time.time())
```

Доступ к полям struct_time осуществляется как по индексу, так и по имени:

```
print(tt.tm_year)     # 2020
print(tt[1])          # 9
```

Обратное преобразование происходит при помощи функции **mktime**. Вместо struct_time можно передать тьюпл.

time.struct_time

Индекс	Имя	Назначение
0	tm_year	Полный год (например 2020)
1	tm_mon	Месяц (1-12)
2	tm_mday	День месяца (1-31)
3	tm_hour	Часы (0-23)
4	tm_min	Минуты (0-59)
5	tm_sec	Секунды (0-61)
6	tm_wday	День недели (0 — понедельник)
7	tm_yday	День года
8	tm_isdst	Признак зимнего времени

Таймеры

Таймер — это специальный программный объект, порождающий событие через заданный промежуток времени.

Таймер может быть одноразовым и повторяющимся.

Таймеры в Qt представлены классом **QTimer**. Он порождает сигнал **timeout()** через заданный интервал времени.

Метод **start** запускает таймер, а метод **stop** останавливает. Не обязательный параметр метода start — интервал срабатывания в миллисекундах.

Таймер может функционировать только в рамках приложения Qt с очередью событий (например, графическое приложение).

Таймер Qt **не повторяет** вызов слота, если за время обработки слота таймер сработал еще раз.

Таймеры

```
class Window : public QWidget
{
    QTimer tmr;
public:
    Window(QWidget* parent = nullptr) : QWidget(parent)
    {
        connect(&tmr, SIGNAL(timeout()),
                this, SLOT(onTimer()));
        tmr.start(500);
    }
private slots:
    void onTimer()
    {
        qDebug() << "timeout";
    }
};
```