1 Работа с календарным временем и интервалами времени

В данном разделе рассматриваются функции работы с календарным временем и интервалами времени в операционных системах, удовлетворяющих стандарту POSIX.

1.1 Работа с календарным временем (ANSI C)

Типы данных

Календарное время включает в себя год, месяц, день месяца и текущее время суток. Для хранения календарного времени используется тип `time_t`. В POSIX-системах календарное время хранится как количество секунд, прошедшее от «начала эпохи», причём для простоты считается, что каждый 4-й год всегда високосный. Эпоха началась (для UNIX) в полночь 1 января 1970 г. по Гринвичу. Время отсчитывается по часовому поясу Гринвичского меридиана без перехода на летнее время (так называемое «Универсальное координированное время» UTC — Coordinated Universal Time).

В настоящее время в POSIX-системах тип `time_t` эквивалентен типу `long`. Диапазона значений типа `long` (32 бита) достаточно для представления дат, лежащих в отрезке длиной примерно 136 лет. Поскольку отрицательные числа используются для представления дат до 1970 года, переполнение текущего представления типа `time_t` может произойти в 2038 году, если, конечно, до этого момента представление не будет изменено.

Для более удобного представления календарного времени используется тип `struct tm`, определённый следующим образом.

```
struct tm
{
        int     tm_sec;     /* секунды [0-60] */
        int     tm_min;     /* минуты [0-59] */
        int     tm_hour;    /* час [0-23] */
        int     tm_mday;    /* день месяца [1-31] */
        int     tm_mon;     /* месяц [0-11], 0 = январь */
        int     tm_year;    /* число лет от 1900 г */
        int     tm_wday;    /* день недели [0-6], 0 = воскр. */
        int     tm_yday;    /* день года [0-365] */
        int     tm_isdst;   /* летнее время */
};
```

Поле `tm_sec` обычно хранит значения в интервале от 0 до 59, но может хранить и 60, когда для коррекции времени в календарь вставляется дополнительная секунда.

Поле `tm_year` хранит год, отсчитываемый от 1900 года, то есть, например, значение 100 в поле `tm_year` означает 2000 год.

Поле `tm_wday` хранит день недели, отсчитываемый от воскресенья (значение 0) и до субботы (значение 6).

Поле `tm_isdst` хранит информацию о том, действует ли в описываемый момент летнее время. Этот флаг больше нуля, если летнее время действует, нуль, если летнее время не действует, и отрицателен, если информация о действии летнего времени отсутствует.

Оба описанных выше типа определены в заголовочном файле `<time.h>`, определённом стандартом ANSI C, то есть должны присутствовать в любой достаточно современной систе-

ме программирования на Си. Стандарт ANSI не определяет внутреннее представление для типа time_t.

Функции

В стандарте ANSI C также определяется набор функций для манипулирования с календарным временем. Прототипы этих функций определены в заголовочном файле <time.h>.

Функция **time** позволяет получить текущее системное календарное время. Функция имеет следующий прототип:

```
time_t time(time_t *pval);
```

Функция возвращает текущее календарное время. Кроме того, если указатель pval ненулевой, функция записывает текущее время по указанному адресу.

Функции **localtime**, **gmtime** преобразовывают дату из секундного представления time_t в структурное представление struct tm. Они имеют следующие прототипы:

```
struct tm *gmtime(const time_t *timep);
struct tm *localtime(const time_t *timep);

extern char *tzname[2];
long int timezone;
```

Функция gmtime конвертирует календарное время timep в развёрнутое структурное представление, выражающее универсальное координированное время (UTC), т. е. время Гринвичского меридиана без перехода на летнее время. Функция возвращает указатель на статическую область памяти, которая переписывается при каждом вызове функций работы с календарным временем. Поэтому при необходимости эту структуру нужно скопировать в другое место.

Функция localtime конвертирует календарное время timep в развёрнутое структурное представление в определённом пользователем или системой часовом поясе. Функция устанавливает переменную tzname в имя текущего часового пояса (tzname[0] — имя часового пояса зимнего времени, tzname[1] — имя часового пояса летнего времени), переменную timezone в разницу в секундах между поясным временем и универсальным координированным временем (UTC). Функция возвращает указатель на статическую область памяти, которая переписывается при каждом вызове функций работы с календарным временем. Поэтому при необходимости эту структуру нужно скопировать в другое место.

Функция **mktime** конвертирует календарное время местного часового пояса в развёрнутом структурном формате в секундное представление. Функция имеет следующий прототип:

```
time_t mktime(struct tm *timeptr);
```

Функция игнорирует содержимое полей tm_wday, tm_yday, tm_isdst и перевычисляет их, используя оставшиеся поля структуры. Если значение какого-либо поля структуры находится за пределами допустимых значений, оно нормализуется, например, 40 октября становится 9 ноября. Вызов mktime заполняет переменную tzname информацией о местном часовом поясе. Если заданное структурное представление не может быть преобразовано в посекундное представление, функция возвращает значение (time_t)(-1) и не изменяет значения полей tm_wday и tm_yday.

Функции **ctime** и **asctime** конвертируют календарное время в символьную строку. Они имеют следующие прототипы:

```
char *asctime(const struct tm *timeptr);
char *ctime(const time_t *timep);
```

Функция `ctime` преобразует календарное время `timep` в строку вида

```
"Tue Nov  7 11:32:11 2000\n"
```

Дни недели сокращаются до 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'. Имя месяца сокращается до 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'. Возвращаемое значение указывает на статическую область памяти, которая переписывается при вызовах функций `ctime` и `asctime`. Функция заполняет переменную `tzname` информацией о местном часовом поясе.

Функция `asctime` преобразовывает развёрнутое структурное представление `timeptr` в строку того же самого формата, что функция `ctime`. Возвращаемое значение указывает на статическую область памяти, которая переписывается при вызовах функций `ctime` и `asctime`. Функция заполняет переменную `tzname` информацией о местном часовом поясе.

Функция **strftime** позволяет получать строковое представление времени согласно заданной форматной строке. Прототип следующий:

```
size_t strftime(char *s, size_t max, const char *format,
                const struct tm *tm);
```

Функция форматирует время в развёрнутом структурном представлении `tm` в соответствии со спецификацией формата `format` и помещает результат в символьный массив `s` размера `max`.

Обычные символы копируются из форматной строки в строку `s` без преобразования. Спецификаторы формата начинаются со знака `'%'` и приведены в таблице.

Функция возвращает количество символов, записанных в массив `s`, не считая последнего символа `'\0'` (то есть, длину строки `s`), при условии, что длина получившейся строки `s` меньше `max`. В противном случае функция возвращает 0, а содержимое массива `s` не определено.

Пример программы

```
#include <time.h>
#include <stdio.h>

#define SIZE 256

int
main (void)
{
  char       buffer[SIZE];
  time_t     curtime;
  struct tm *loctime;

  /* Get the current time. */
  curtime = time(NULL);

  /* Convert it to local time representation. */
  loctime = localtime(&curtime);
```

%a	сокращённое имя дня недели (например, Tue)
%A	полное имя дня недели (например, Tuesday)
%b	сокращённое имя месяца (например, Nov)
%B	полное имя месяца (например, November)
%c	дата и время (например, Nov 7 11:32:11 2000)
%d	день месяца (например, 07)
%H	час в 24-часовой шкале (от 00 до 23)
%I	час в 12-часовой шкале (от 01 до 12)
%j	день года (от 001)
%m	месяц года (от 01)
%M	минуты в часе (00–59)
%p	индикатор AM/PM
%S	секунды в минуте (00–59)
%U	неделя (считая от воскресенья) в году (от 00)
%w	день недели (0 — воскресенье)
%W	неделя (считая от понедельника) в году (от 00)
%x	дата (например, Nov 7 2000)
%X	время (например, 11:32:11)
%y	год в веке (00-99)
%Y	год (например, 2000)
%Z	имя часового пояса (если определено) (например, MSK)
%%	символ %

Таблица 1: Спецификаторы формата функции strftime

```
    /* Print out the date and time in the standard format. */
    fputs(asctime(loctime), stdout);

    /* Print it out in a nice format. */
    strftime(buffer, SIZE, "Today␣is␣%A,␣%B␣%d.\n", loctime);
    fputs(buffer, stdout);
    strftime(buffer, SIZE, "The␣time␣is␣%I:%M␣%p.\n", loctime);
    fputs(buffer, stdout);

    return 0;
}
```

1.2 Дополнительные функции POSIX

Функции, описанные в предыдущем разделе, позволяют получать время с точностью до секунды. Часто требуется получить время с более высокой точностью. Для этого стандарт POSIX определяет дополнительные функции.

Функция **gettimeofday** описана следующим образом.

```
#include <sys/time.h>
#include <unistd.h>

int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Тип struct timeval определён следующим образом.

```
#include <sys/time.h>

struct timeval
{
        long tv_sec;     /* секунды */
        long tv_usec;    /* микросекунды */
};
```

Второй аргумент этой функции не должен использоваться, он должен иметь значение NULL. Функция заполняет поле tv_sec структуры tv текущим календарным временем, а поле tv_usec микросекундами в секунде. Функция возвращает 0 при успешном завершении и -1 при ошибке. В этом случае переменная errno содержит код ошибки (единственная причина возможной ошибки — недопустимый адрес tv).

1.3 Приостановка выполнения программы

Функции, описываемые в данном разделе, позволяют приостановить выполнение программы на заданный промежуток времени. Функции завершают работу в двух ситуациях: либо истечёт промежуток времени, либо процессу поступит сигнал, который процесс не игнорирует.

Функция **sleep** позволяет приостановить выполнение программы на заданное количество секунд.

```
#include <unistd.h>

unsigned int sleep(unsigned int seconds);
```

Выполнение программы приостанавливается на указанное количество секунд. Выполнение программы продолжится, когда либо истечёт указанный интервал времени, либо в программу поступит неигнорируемый сигнал. Функция возвращает количество секунд, оставшееся до истечения интервала времени (0, если интервал времени истёк).

Функция sleep может использовать в своей работе сигнал SIGALRM, поэтому не рекомендуется смешивать вызовы alarm() и sleep().

Функция **usleep** позволяет приостановить выполнение программы на заданное количество микросекунд.

```
#include <unistd.h>

void usleep(unsigned long usec);
```

Функция usleep может реализовываться через системный вызов nanosleep (см. ниже). Обсуждение использования этих функции находится ниже.

Системный вызов **nanosleep** приостанавливает выполнение программы по крайней мере на интервал времени req.

```
#include <time.h>

int nanosleep(const struct timespec *req,
              struct timespec *rem);
```

Структура timespec используется для задания интервалов времени с наносекундной (10^{-9}) точностью. Структура определена в файле <time.h> следующим образом.

```
struct timespec
{
        time_t  tv_sec;         /* секунды */
        long    tv_nsec;        /* наносекунды */
};
```

Значение поля `tv_nsec` должно лежать в пределах 0–999 999 999.

`nanosleep` приостанавливает выполнение программы по крайней мере на время, заданное в `*req`. Функция может завершиться раньше, если процесс получил неигнорируемый сигнал. В этом случае функция возвращает `-1`, устанавливает переменную `errno` в значение `EINTR` и записывает оставшееся время в структуру, на которую указывает `rem`, если `rem` не равен `NULL`. Значение `*rem` может быть использовано для вызова `nanosleep` для завершения паузы.

В отличие от `sleep` и `usleep`, `nanosleep` не использует никаких сигналов, стандартизован POSIX, позволяет задавать время с большей точностью и позволяет легко продолжить паузу после прерывания от сигнала.

Для отсчёта времени система использует свой внутренний таймер, имеющий на некоторых машинах (например, i386) частоту 100 Гц, а на других машинах — 1000 Гц. Частота таймера определяется константой `HZ`, определённой в файле `<time.h>`. Это значит, что система не может приостановить выполнение пользовательского процесса на время, меньшее чем `1/HZ` (на i386 это — 10 миллисекунд). Время выравнивается в сторону увеличения до ближайшего значения, кратного `1/HZ`. Соответственно, оставшееся время выравнивается в сторону увеличения до значения, кратного `1/HZ`.

Если процесс имеет приоритет реального времени (политика планирования `SCHED_FIFO` или `SCHED_RR`, тогда паузы на интервал времени, меньший `1/HZ` могут выполняться с помощью циклов задержки.

Функции приостановки выполнения программы должны использоваться, когда программа постоянно опрашивает какой-нибудь объект (например, каталог) и отслеживает изменения в нём. Например, программа, которая отслеживает появление в каталоге нового файла и сразу же переносит этот файл в другой каталог.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <dirent.h>

int main(int argc, char **argv)
{
  DIR             *dd;
  struct dirent   *de;
  struct timespec  ts;
  char             in_path[PATH_MAX+1];
  char             out_path[PATH_MAX+1];

  if (argc != 3) {
    fprintf(stderr, "too␣few␣arguments\n");
    exit(1);
  }
```

```
  while (1) {
    if (!(dd = opendir(argv[1]))) {
      perror("opendir");
      exit(1);
    }
    while ((de = readdir(dd))) {
      if (strcmp(de->d_name, ".") && strcmp(de->d_name, ".."))
        break;
    }
    if (!de) {
      closedir(dd);
      ts.tv_sec = 0;
      ts.tv_nsec = 1;
      nanosleep(&ts, NULL);
    } else {
      strcpy(in_path, argv[1]);
      strcat(in_path, "/");
      strcat(in_path, de->d_name);
      strcpy(out_path, argv[2]);
      strcat(out_path, "/");
      strcat(out_path, de->d_name);
      if (rename(in_path, out_path) < 0) {
        perror("rename");
        exit(1);
      }
      closedir(dd);
    }
  }
  return 0;
}
```

Если бы вызов `nanosleep` в строке 34 отсутствовал, программа бы постоянно опрашивала заданный каталог, бесполезно трача процессорное время. Вызов `nanosleep` приостановит выполнение программы до следующего кванта времени, позволив нормально работать другим процессам.