怎样解决应用程序的 2000 年问题

王晓武 （北京复兴路 22 号甲 1 号一局 100842）

以往开发的计算机应用软件，运行到 2000 年时，将发生时间混乱。这个问题近来被许多新闻媒体所关注。笔者认为应该从不同的角度来看待这个问题。从全世界及全国的宏观角度分析，以往开发的应用软件若要全部顺利地适应新世纪日历记年，的确要花费巨大的人力和资金，其损失和影响不可低估。然而，仅从某个应用软件系统的开发和应用单位的角度来看，解决这个问题并不会费多少周折，只需对原有的程序进行适当的修改即可。

以往的计算机应用系统从它们的应用范围和程序规模可归类为以下三类：

1.**通用的支撑软件**。如文字排版编辑软件。通用表格处理软件等这类软件一般均采用 C 语言开发。

2.**大、中型数据库管理软件**。如大中型企业的数据库管理系统，通用的财务数据处理系统等。这类软件很多是采用 C 语言嵌套 SQL 查询语句（如：ORACLE 数据库系统）的方法开发。

3.**小型的数据统计软件**。这类软件主要包括应用单位的软件开发者，根据本单位的需求，结合实际开发的数据统计报表软件，这种小型应用软件大都采用 DBASE、FOXBASE、FOXPRO 数据库系统编程。

上述三类应用软件若在当时开发过程中没有考虑如何适应新世纪日历记年的特殊要求。程序员就必须对其进行适当的修改，因为大多数应用软件含有以日历为基础的用户注册、备忘录、安全检测以及系统日历时钟显示功能。若不对其进行修改，到二十一世纪这些应用软件将产生难以予测的混乱和错误。笔者在此以用 C 语言和 FOXPRO 系统开发的软件为例，介绍如何修改原有应用程序解决 2000 年问题的有关技术和方法，仅此抛砖引玉，供读者参考。

首先分析 C 语言（以 MS-C6.0 为例）的日历时间函数是如何处理日历记年信息的？

C 语言编译系统提供了有关系统日历时钟调用的一组函数，程序员正确地使用这些函数和嵌入文件并做适当的调整，就能够准确地获取和利用系统日历时钟信息并可适应二十一世纪的记年要求。

获取系统日历时钟编程（参见例程 PROCTIME.C 清单），应注意调用编译系统提供的嵌入文件 TIME.H，该文件中包含有关系统日历时钟调用函数以及结构数组 tm 的说明，tm 结构中定义有：年（tm-year）、月（tm-mon）、日（tm-mday）、时（tm-hour）、分（tm-min）、秒（tm-sec）以及星期标志（tm-wday）等存储单元。应当特别注意的是其中 tm-year 的定义说明："years since 1900"，其含义就是 time()及 aclock()等函数在处理记年计数时，只考虑二十世纪内的年计数而不处理世纪的更迭。

调用有关函数前必须对 TIME.H 文件中说明的 tm 结构缓冲区予以重定义，然后适时地调用 time()及 aclock()等函数将系统日历，时钟信息读出后存放到结构缓冲区 newtime 中备用。

显示日历、时钟信息前，程序员应对结构缓冲区的日历星期信息进行适当的中英文转换和顺序排列，并对记年变量 newtime－＞tm-year 作特殊处理（YEAR＝newtime－＞tm-year＋1990）即可解决 2000 年的问题。若不如此处理，到 2000 年，C 程序中 tm-year 变量的记年值会变成 100。

本实例在应用程序的主循环体内，根据用户的按键值以 1 秒钟为周期，将系统日历，时钟信息显示到屏幕适当的座标上。

PROCTIME.C 程序用 MS-C 6.0 编写，在 286 以上档次 PC 机的 DOS 环境下运行通过，详细说明参阅程序清单注释。

用 FOXPRO（包括 FoxPro for DOS 和 FoxPro for windows 以及 FOXBASE）编写的程序为了适应新世纪记年要求，只需做简单的修改。在原有程序的首部增加一条设置语句（SET CENTURY ON）即可，否则到 2000 年程序中所有日期型字段、变量以及 date()的记年值都会变成 00 年。

PROCTIME.C 源程序清单：

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

```
#include <bios.h>
void goto-xy(int x, int y) /* 设置光标位置 */
{
  union REGS r;
  r.h.ah=2; r.h.bh=0;
  r.h.dh=(char)x; r.h.dl=(char)y;
  int86(0x10, &r, &r);
}
get-key() /* 读键盘按键值 */
{
  union REGS r;
  r.h.ah=0;
  return int86(0x16, &r, &r);
}
int wherex() /* 读当前光标位置 X 座标值 */
{ int x;
  union REGS r;
  r.h.ah=3; r.h.bh=0;
  int86(0x10, &r, &r);
  (char)x=r.h.dh;
  return x;
}
int wherey() /* 读当前光标位置 Y 座标值 */
{ int y;
  union REGS r;
  r.h.ah=3; r.h.bh=0;
  int86(0x10, &r, &r);
  (char)y=r.h.dl;
  return y;
}
void write-video(int x, int y, char *p, int attrib) /* 显示字符串 */
{
  register int i;
  union REGS r;
  for (i=y; *p;i++) {
    if (*p=='\n')
      { goto-xy(x+1,2);break; }
    goto-xy(x,i);
    r.h.ah=9; r.h.bh=0;
    r.x.cx=1; r.h.al=*p++;
    r.h.bl=(char)attrib;
    int86(0x10, &r, &r);
  }
}
void disptime(int x, int y) /* 显示系统日历、时钟 */
{
  struct tm *newtime; /* 定义时间结构指针 */
  time-t aclock;
  long startsec, currsec;
  int dqx, dqy, YEAR;
  char sj[50], week[3];
  time(&aclock);
  newtime=localtime(&aclock); /* 将系统时钟读入结构缓冲区 */
  YEAR=newtime->tm-year+1990 /* 记年变量处理 */
  startsec=newtime->tm-sec;
  switch(newtime->tm-wday) /* 转换星期 */
    {
    case 0: strcpy(week,"日"); break;
    case 1: strcpy(week,"一"); break;
    case 2: strcpy(week,"二"); break;
    case 3: strcpy(week,"三"); break;
    case 4: strcpy(week,"四"); break;
    case 5: strcpy(week,"五"); break;
    case 6: strcpy(week,"六"); break;
    }
  while(! kbhit()) /* 循环读取和组装系统时钟字符串 */
    {
    time(&aclock);
    newtime=localtime(&aclock);
    sprintf(sj,"%d.%2d.%2d 星期%s %d:%2d:%2d",YEAR,
      newtime->tm-mon+1, newtime->tm-mday, week, newtime->tm-hour,
      newtime->tm-min, newtime->tm-sec);
    currsec=newtime->tm-sec;
    if(startsec! =currsec) /* 以 1 秒为周期显示系统日历时钟 */
      {
```

```
        dqx = wherex(); dqy = wherey();
        write-video(x, y, sj, 0x0a);
        goto-xy(dqx, dqy);
        return;
        }
    }
}
main ()
{
  union inkey { char ch[2]; int i; }c;
  for (;;) {
    for(;;) {
      if (kbhit()) { c.i = get-key(); break; } /* 无按
键时显示系统时钟 */
      disptime(0,54);
      if (c.ch[0] = = 27) exit(0); /* 按 ESC 键时退
出 */
    }
  }
}
```

(来稿时间:1997 年 3 月)