UNIX Systems have a Year 2000 Trap!

by

Richard A. Painter, P.E.

Painter Engineering, Inc.

8470 Swan Rd.
Black Forest, CO 80908
719 495 7054

painter@ieee.org

Abstract

The UNIX systems' standard timekeeping mechanism and library function calls contain a Year 2000 trap. It is not a bug but a discontinuity in the number of digits returned by specific functions for dates after 31 December 1999. Application programmers may not have accounted for this thus setting the trap. This paper reviews the details of the subject functions and demonstrates the trap.

UNIX Systems have a Year 2000 Trap!

Introduction

UNIX systems, deployed mostly at universities prior to 1980, were commercialized by a number of vendors in the early 1980s. Popular for high-end desk top computers, servers and more recently as World-wide Web servers for INTERNET Service Providers (ISP) their use has steadily grown. LINUX, a derivative operating system, is growing rapidly as a low-cost UNIX for many uses, especially running on PCs.

The basic internal time unit in UNIX is a 32-bit, signed integer representing *seconds* since the UNIX EPOCH. This EPOCH is defined as 1 January 1970. See Reference 1 for a complete analysis of the limitations and calendar issues for this time representation. Based upon this representation this will roll over on Tuesday, January 19 03:14:07 2038 to Friday, December 13 20:45:52 1901. Thus the limit and rollover point is not an issue for Year 2000. The trap lies elsewhere.

Problem Description

UNIX system functions are broken into two categories: system calls and library calls. All of the Year 2000 traps are in the library calls.

The following is a list of the time-related library calls and an indication if it has the trap.

ctime()	time()	timelocal()	asctime()
dysize()	gmtime() *TRAP*	localtime() *TRAP*	timegm()

Since the inception of UNIX (circa 1970) these time routines have returned 2 digits for the year. Given any time prior to 1970 that UNIX can represent (back to 1901) these functions return the last 2 digits of the year. This is not so after 1999!

The basic problem lies in the method that the time conversion functions return the year. The number of digits returned are discontinuous. Mathematically, they return (`year - 1900`). Hence for 1999 they return “99”. In 2000 these routines will return 3 digits! Hence for 2000 they return “100”. If an application was developed that strictly used the output year of these functions it will work fine until 2000. This is the trap.

Furthermore, some programmers may overlook how the C programming language (typically used in UNIX) handles output format field specifiers. Even though one specifies a 2-digit output field width, such as “%02d” for a 2-digit integer in C, if the integer is larger than 2-digits it will output all of the digits and effectively shift any remaining output! This can easily create additional problems when the output must be found in certain positions and this use has shifted subsequent output fields.

The current Year 2000 crisis lies mostly in application software where a 2-digit year is all that is used creating ambiguity around 2000. UNIX internal time representation does not suffer from this but not so with UNIX applications. Many applications likely suffer 2-digit-itis, possibly needing alteration to handle 4-digit years. Hopefully, when these applications are modified to use 4-digit years the programmer will correctly account for the digit discontinuity of these function outputs.

However, if windowing or other Year 2000 workarounds relying on 2-digit years are employed outside of this application the trap may still be there! This is very possible since some application source codes are not available. In this case the external, 2-digit workaround can fail when the application produces “100” for the expected 2-digit year.

The following sample C code was run on SunOS 4.x and demonstrates this:

```
/*
        Demo program for UNIX 2-digit year Trap on 1 Jan 2000

        (c) Copyright 1999 – 2005 Painter Engineering, Inc.
                8470 Swan Rd.
                Black Forest, CO 80908
                719 495 7054   painter@ieee.org      http://painterengineering.com
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/time.h>

int main(argc, argv)
    int argc;
    char argv[];
{
    time_t ourtime, jan1seconds, dec31seconds;
    struct tm current_tm, jan1_2000, dec31_1999;
    char current[26], jan1_2000str[26], dec31_1999str[26];

    /* obtain the current time */
    ourtime = time(0);
    current_tm = *gmtime(&ourtime);   /* struct copy */
    strcpy(current, asctime(&current_tm), sizeof(current));
    current[24] = '\0';       /* trash nl */

    printf("Current %s GMT time in seconds since EPOCH: %ld\n", current, ourtime);

    /* obtain the EPOCH seconds for 1 Jan 2000 */
    memset(&jan1_2000, 0, sizeof(jan1_2000));
    jan1_2000.tm_mday = 1;
    jan1_2000.tm_year = 100;
#ifdef DONT_HAVE_TIMEGM
    /* some systems don't have it so we just set it since we know the number... */
    jan1seconds = 946684800;
#else
    jan1seconds = timegm(&jan1_2000);
#endif /* DONT_HAVE_TIMEGM
    /* compute just prior... */
    dec31seconds = jan1seconds -1;
    dec31_1999 = *gmtime(&dec31seconds);     /* struct copy */

    /* confirm the date construction */
    strcpy(jan1_2000str, asctime(&jan1_2000), sizeof(jan1_2000str));
    jan1_2000str[24] = '\0';  /* trash nl */
    strcpy(dec31_1999str, asctime(&dec31_1999), sizeof(dec31_1999str));
    dec31_1999str[24] = '\0'; /* trash nl */

    printf("1 Jan 2000 GMT time in seconds since EPOCH: %ld\n", jan1seconds);
    printf("1 Jan 2000 GMT should be %s\n", jan1_2000str);
    printf("31 Dec 1999 GMT should be %s\n\n", dec31_1999str);

    /* now demonstrate the 2-digit and 3-digit effects */
    /* use the SAME output formats */
    printf("Typical 2-digit year date format for 1999 is %02d/%02d/%02d\n",
        dec31_1999.tm_mon +1,
        dec31_1999.tm_mday,
        dec31_1999.tm_year);
    printf("Typical 2-digit year date format for 2000 is %02d/%02d/%02d\t OOPS- Here is the
TRAP!\n",
        jan1_2000.tm_mon +1,
        jan1_2000.tm_mday,
        jan1_2000.tm_year);
}
/*
cc -O -g -o unix2digit unix2digit.c
cc -O -o unix2digit unix2digit.c
*/
```

Here is typical output from the program:

```
Current Thu Aug  5 00:55:36 1999 GMT time in seconds since EPOCH: 933814536
1 Jan 2000 GMT time in seconds since EPOCH: 946684800
1 Jan 2000 GMT should be Sat Jan  1 00:00:00 2000
31 Dec 1999 GMT should be Fri Dec 31 23:59:59 1999

Typical 2-digit year date format for 1999 is 12/31/99
Typical 2-digit year date format for 2000 is 01/01/100   OOPS- Here is the TRAP!
```

Potential Solutions

The primary solution is to modify the application code to account for the digit discontinuity. This is simple and can be performed at the same time conversion to 4-digit years is done.

The secondary solution is employed when application source code is not available. If possible an external application is used to modify the results of the subject application. In this case, one must determine through testing if the application accounted for the digit discontinuity or not. If it did not then the outputs will have 3-digit years obeying the formula (`year - 1900`) and can be accounted for appropriately. Otherwise, normal 2-digit to 4-digit methods can be employed.

Conclusions

Most people believe that there are no date-related problems in UNIX systems but they are mistaken.

Year 2000 traps exist in UNIX applications and must be located and corrected. They are not bugs but oversights by programmers implementing the applications, just as are the 2-digit problems. Testing will expose these but easy cures only exist when the source code, compiler, and external libraries are available.

References

1. *The UNIX Time will run out in 2038!*, December 1996, Richard A. Painter, P.E., Painter Engineering, Inc.
2. *Year 2000 Best Practices for Y2K Millennium Computing- UNIX Time*, 1998, Richard A. Painter, Edited by Dick Lefkon, ISBN 0136465064
3. UNIX manual pages for *ctime*(3), et al library calls.