Why is UNIX time usually represented as a signed as opposed to unsigned integer?

3 Answers

Richard Conto, Linux, Solaris, FreeBSD, Ultrix, OS/X developer and sysadmin

Answered 2 years ago · Author has **8K** answers and **6.9M** answer views

Unix time was for the Unix kernel and things managed by the Unix kernel (like file usage times.)

Unix didn't exist before November, 1971 - so picking a start-of-time of "Jan 1, 1970" was relatively easy - and allowed some low numerical values to be easily identified as being corrupted data.

The use of seconds (rather than milliseconds or microseconds) was also a choice based on the capabilities of the machines of the time. Other systems (even then) used 64-bit integers to represent microseconds since Jan 1, 1900 - or other time systems & epochs.

And as others have said, there was an enormous amount of development and experimentation going on in the field of operating systems back then - and Unix was designed to support running a word processing system that drove a type setter/printer. It was designed to run on a (relatively) cheap, spare machine available to the developers who were trying to satisfy a need that wasn't being addressed by the vendor.

The need for applications to represent time outside of the limits imposed the Unix file system was never a design criteria.

Later on, the need to measure time within the system to milliseconds (or even microseconds) resulted in various kludges to expose those clocks to user-level programs.

In general, any system that had a need to represent time more broadly would struggle with the inconsistent ways people have kept time over the years, and the inconsistencies of systems of astronomical time as well.

Comparing historical time stamps is fraught with peril - especially before time zones were invented (and even after as political decisions change what time zone a place is in.) Even comparing dates is tricky as the current Gregorian calendar took several centuries to be adopted, even within Europe.

A comprehensive solution to time is very difficult. So the Unix designers took a very simple approach by limiting it's scope.

1.1K views · View upvotes

2

Related Questions More Answers Below

How do I successfully compare signed and unsigned integers in C++?

What is the largest unsigned integer 64 bits can hold?

Why is entering an integer outside of range of unsigned short is difficult to understand and are there any rules?

Why does C++ use signed and unsigned integers?

How do you convert a signed integer to an unsigned integer?

Steven Angel, 30+ years of computer programming experience. Assembly, C,

Related Questions

How do I successfully compare signed and unsigned integers in C++?

What is the largest unsigned integer 64 bits can hold?

Why is entering an integer outside of range of unsigned short is difficult to understand and...

Why does C++ use signed and unsigned integers?

How do you convert a signed integer to an unsigned integer?

Why do we need a signed and unsigned integer?

This answer is really an accidental historic byproduct.

Native C calculates time as number of seconds since midnight, January 1 1970. A 32-bit number worked very nicely for this value as 16-bit would be uncomfortably small.

As for why a signed value was chosen, it was convenient for the architecture they were using and it was "good enough". This was the 1970's. The 32 bit signed value would roll over in January 2038. Why would they be concerned about something that happened over 60 years in the future? Computer languages and OS's were constantly being invented and changed at that time. It was a happy coincidence that C and UNIX became popular and let to a computer science revolution.

There have already been many changes to mitigate this problem. "time_t" is the type that C compilers expect time values to be declared as. Historically that is a 32 bit signed integer. But a particular compiler could declare it as 32 bit unsigned. Automatically you gain another 68 years before you hit the skids. Some compilers go ahead and declare time_t as 64 bit signed. This would fix the problem until long after the sun is expected to expand and destroy the world. Hence it becomes SOP. Somebody Else's Problem.

2.4K views · View upvotes

9

David Brower, C/UNIX from 80s; Database, networking; have S/W patents
Updated 6 months ago · Author has **13.7K** answers and **5.4M** answer views

Using a signed number makes it possible to do arithmetic with time that leaves a negative value as being indication it is in the past.

204 views · View upvotes

2

Related Answers

Related Answer

Sergey Bugaev, software developer
Answered 5 years ago · Author has **191** answers and **1.4M** answer views

Why does C++ use signed and unsigned integers?

By default, if you just use `int`, it is *signed* — it can have both positive and negative (and zero) values. So you don’t have to worry if you don’t care much.

However, sometimes you clearly know that a particular variable can never be `< 0` — think a string length, an index in the array, a file size, the number of open file descriptors, etc. You may want to use an unsigned type for that variable. There are

Continue Reading

10

Related Answer

Joe Zbiciak, I have been programming since grade school
Answered 2 years ago · Upvoted by Alan Mellor, Started programming 8 bit computers in 1981 and Litu Mihai, 10+ years of C++ · Author has **3.7K** answers and **16.5M** answer views

How do I successfully compare signed and unsigned integers in C++?