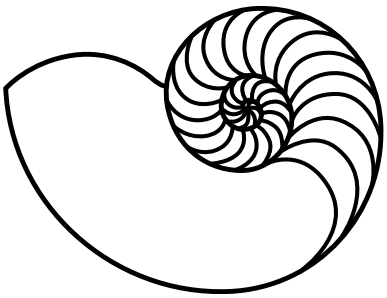

MUUGLines

The Manitoba UNIX User Group Newsletter

Volume 34 No. 8, April 2022 Editor: Trevor Cordes

Next Meeting: April 12th, 2022 (Online Video Meeting)

Cloud-init

Kevin McGregor will provide an overview of cloud-init, a widely-supported way to initialize instances of cloud-aware distros (i.e. pretty much all of them!) on your public, private or home cloud.

The latest meeting details are always at:
https://muug.ca/meetings/

Where to Find the Meeting:

This month we will continue to use the open source meeting software: Big Blue Button. If you haven't tried it yet, we recommend joining the meeting a little early to familiarize yourself with the controls.

The virtual meeting room will be open by 7:00 pm on April 12th, 2022 with the actual meeting starting at 7:30 pm. You do not need to install any special app or software to use Big Blue Button: you can use it via any modern web-cam-enabled browser by going to the website link above.

Please note that the meeting link will not be active until approx. 30 minutes before the actually meeting date and time.

Backups Versus EMPs

With the hugely decreasing cost of flash media (e.g. USB sticks) and external shingled hard drive storage, this author has been recalculating the costs of doing backups to various media.

It turns out optical media, mostly DVD-Rs or BD-Rs are still much cheaper per gigabyte; although still the least convenient due to their limited size. Strangely enough, it is starting to get slightly difficult to source 50- and 100-disc spindles of optical media, with some outlets not stocking any at all, and others with vastly reduced selection.

What do EMPs have to do with this? For various reasons, justified or not, EMP chatter has been increasing as of late. We all know optical media can survive an EMP, but can flash media? There seems to be a lot of confusion and disagreement. Flash backup, being somewhat cheap, and extremely convenient, would be slightly more attractive if it was known it could survive an EMP and/or strong electrical storms.

After much research the question remains unanswered. But one interesting idea is to put flash backup sticks into a small Faraday cage. In theory, a strong/thick enough Faraday cage should shield the contents from an EMP. Thus arises the next question: how would you obtain or build such a cage? Internet merchants have already been working on it.

If your answer to backup is "cloud": beware that the cloud storage vendors are also likely extremely vulnerable to EMPs, especially multiple, geographically distributed EMPs. Read the terms of storage: few guarantee against loss of data.

https://briantomasik.com/backing-data-geomagnetic-storms-emps/

ioctl() Insanity

Zack's Kernel News recently had some choice nuggets regarding ioctl():

> *Input/output controls (ioctls) are a nightmarish fantasy of one of the outer gods, possibly Nyarlathotep. Ioctls exist in the nether region between what you need the hardware to do, and what the system calls are able to provide. [...]*
>
> *Instead, the single ioctl() system call can take all of that malignant energy unto itself, growing darkly beneath the surface for all time. If you asked a kernel developer about documenting all the behaviors of ioctl(), they would begin to laugh, cry, and explode simultaneously. Try it and see. Or don't. They have suffered enough.*

https://www.linux-magazine.com/Issues/2021/248/Kernel-News

Did You Know? ... Rounding

We all learned about rounding decimal numbers in school. It seemed very straightforward. Did you know that there also exist some strange rules for rounding, and that many computer programs, libraries, languages, and standards differ in their behaviour in pretty common scenarios? If you're expecting a certain result or consistency, you might want to verify your tool is doing what you want.

The big problem is in some cases you'll get "Banker's rounding", or "round to nearest – ties to even" instead of the more typical and expected "round to nearest – ties away from zero". The latter states that you round 8.5 to 9; the former rounds 8.5 to 8. The difference between the two only applies when you have a perfect half value / tie, so rounding 8.51 will result in 9 with both methods. Still, to many of us, rounding 8.5 to 8 (ties-to-even) may seem strange.

Let's survey the real world:

PHP: `echo round(8.5)`
9

Perl:
```
use v5.10; use Math::Round;
say round(8.5);
```
9

C:
```
#include <stdio.h>
#include <math.h>
int main(){ printf(
"%f\n",roundf(8.5)); }
```
9.000000
**NOTE1*

JS: `console.log(Math.round(8.5))`
9

Mysql: `select round(8.5);`
9

Python3: `print(round(8.5))`
8

.NET (from docs, untested):
```
decimal.Round(8.5)
```
8

Any language/program using stdio's printf:
```
printf("%1.0f\n",8.5)
```
8

**NOTE1*: gcc doesn't provide a itoa function so only printf type functions can turn ints into strings; yet this causes problems for us because printf itself applies rounding rules. However, here we can be sure we are not seeing any printf rounding, and roundf()'s documentation declares it uses ties-away-from-zero anyhow. See below.

As we can see, languages and libraries cannot agree on this simple question. The majority use ties-away-from-zero, which is what you'd probably expect. But python3, and, very importantly, printf, use ties-to-even.

Since almost every language provides a printf function which just ties into the system stdio library, most languages will have different rounding behaviour between their internal rounding function and printf's rounding – within the same language!

The whole problem seems to stem from the IEEE 754 (1985) standard which governs floating-point arithmetic used by many (most?) FPUs and many libraries. This apparently includes stdio; or perhaps stdio just defers to what the FPU does. (Homework exercise: use the stdio source, Luke.)

IEEE 754 states "ties to even – rounds to the nearest value; if the number falls midway, it is rounded to the nearest value with an even least significant digit; this is the default for binary floating point and the recommended default for decimal." (wikipedia, emphasis ours)

It also states: "ties away from zero [...] intended as an option for decimal floating point."

It seems clear printf() is following IEEE 754's default. The manpage talks about rounding but not which method it uses or what standard, though *perhaps* another stdio manpage states in some manner that it mainly follows IEEE 754. In any event, printf() does not appear to provide any method for selecting the other rounding algorithm(s) the standard offers.

So what is this *Banker's* (ties-to-even) rounding and why does it even exist? It seems to have been championed by, well, bankers (or the Dutch or Gauss, whom both also lend their names to the method: no one really knows), because they thought that rounding the tie up (let's assume only positive numbers for this argument) was biased and that banker's rounding would give a more precise result. But does it?

https://www.sqlservercentral.com/articles/bankers-rounding-what-is-it-good-for

That article states that it gets closer to the actual unrounded sum, but disturbs the normal, even distribution of target digits. But what if the unrounded sum is not the correct goal?

Think of the case of a tax, like the PST. When you buy a product, you are charged PST in cents, never fractions of a cent. But since the tax rate is 7%, the tax on an item will, the vast majority of the time, have fractions of a cent. Any rounding that goes up will favour the government: they will get more tax than they should: think ceil(). Any rounding that goes down will favour the taxpayer: they will pay less tax than they should: think floor(). So perhaps the correct goal of rounding should be to see what algorithm best "meets in the middle", or is closest to the government (and/or taxpayer) winning precisely half the time.

Your author wrote a simulation program to determine which algorithm best approaches both goals. The common rounding method is vastly superior to "meet in the middle", whilst ties-to-even gets closer to the unrounded sum. So perhaps both are correct depending on your actual goal.

One moral of this story is read the documentation and run some simple tests with your tool to ensure it is doing what you desire. Many will state what they are doing with precise ties. Some provide an optional argument or different function to specify the algorithm. Another moral of the story is to do an audit of your code to see if you are using sprintf() anywhere to effect rounding. This author discovered he was using it in many places where it could be considered a bug in his personal perl programs.

At the end of the day, so many tools seem to agree on what the "right thing" is, so why did IEEE 754, and thus ubiquitous things like printf() decide to use Banker's rounding? IEEE 754 hints at a possible reason when it says "for binary floating point". Maybe there is some advantage or logic for it when dealing exclusively with binary numbers, as computers obviously do. Maybe it requires less CPU cycles. Maybe it is more straightforward. (Once again, homework exercise.)

As for taxes like the GST and PST, their regulations clearly state they use the normal ties-away-from-zero. Software programs like Quickbooks and MS Money appear to do so as well. That can allow many to emit a big sigh of relief.

Ironically enough, it was an amount on a MUUG invoice that resulted in precisely a half-cent in PST that triggered your MUUG Treasurer to initiate a trip down this particular rabbit hole with this author. The final word: this is one heck of a complicated question. Feel free to add to the discussion at or on the MUUG Round Table! (Paging M.Doob...)

https://en.wikipedia.org/wiki/Rounding

emacs Versus vi

Put your flameproof anorak on! This author is going to definitively answer the age old question: which is better, emacs or vi?

Actually, that's a lie. But here are some statistics to show that vi (in its modern, active vim form) is a couple of orders of magnitude more buggy and full of security holes than emacs. The number of CVEs issued for vim in 2021 and 2022 have been shocking, and not a week seems to go by without the Fedora Updates mailing list not having a new vim update with a CVE attached. Emacs hasn't seen a CVE since 2017. Even worse, about one third of vim's holes are of the "execute code" variety (very bad).

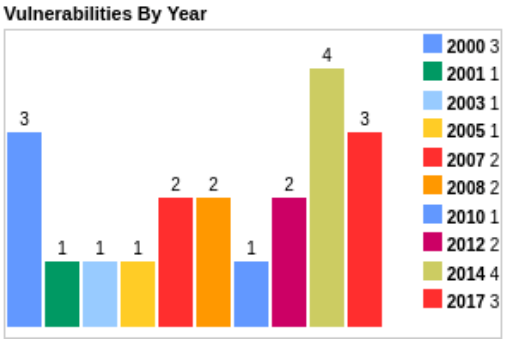

Figure 1: emacs

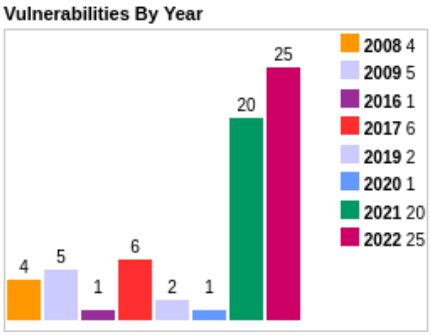

Figure 2: vim

A big thanks to Les.net for providing MUUG with free hosting and all that bandwidth! Les.net (1996) Inc. is a local provider of VoIP, Internet and Data Centre services. Contact **sales@les.net** by email, or +1 (204) 944-0009 by phone.

Thank You Michael W. Lucas

MUUG would like to thank Michael W. Lucas for donating one of his ebooks every month as a door prize. You can view and purchase his tech books here:

https://www.tiltedwindmillpress.com/product-category/tech/

Creative Commons License

Except where otherwise noted, all textual content is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

https://creativecommons.org/licenses/by-sa/4.0/

Help us promote this month's meeting, by putting this poster up on your workplace bulletin board or other suitable public message board:

https://muug.ca/meetings/MUUGmeeting.pdf