

Re: realloc(): invalid next size

You've got that wrong. The last case is `memcpy(new_pointer,old_pointer,0);`
i.e., `new_pointer = my_realloc(old_pointer, 0); /*`
`memcpy(new_pointer,old_pointer,0); */`

If size is zero, s1 is freed (which doesn't set s1 to NULL) and s2 is returned which is NULL.

Right. Let's run "my_realloc(valid_old_pointer, 0);" step by step ...

```
void *my_realloc (void *s1, size_t size)
{
void *s2=NULL;
/* s1 --> valid_old_pointer
* s2 --> NULL
* size --> 0 */

if (size!=0||s1==NULL)
s2 = malloc(size);
else
free(s1);
/* s1 --> free'd valid_old_pointer
* s2 --> NULL
* size --> 0 */

if (s1!=NULL)
memcpy(s2, s1, size);
/* And here we go ...
* s2 is NULL, s1 is the free'd valid_old_pointer, size is 0
* so
*
* ...
*
* memcpy(NULL, ????, 0); */

return(s2);
}
```

In the last example, will the call to `memcpy()` invoke UB? Or, because size is 0 (zero), it doesn't matter what the pointers point to?

As I said, you've got the last one wrong. No NULL is ever passed to `memcpy()`, but a size zero is.

Where did I go wrong in my step-by-step run of `my_realloc()`?

Re: realloc(): invalid next size

To eliminate the size zero problem, you need to know the old size, and check that it isn't zero. Which as Chris Torek's example shows, it's based on compiler secrets.

Yes, I understand this is nothing more than an exercise in second guessing the compiler secrets ... but I feel it's a good exercise to aid in developing my knowledge of C.

--

If you're posting through Google read <<http://cfaj.freeshell.org/google>>

.