The XArray data structure

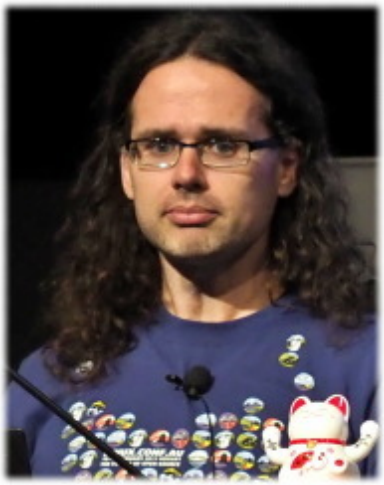
By **Jonathan Corbet**
January 24, 2018

linux.conf.au

Sometimes, a data structure proves to be inadequate for its intended task. Other times, though, the problem may be somewhere else — in the API used to access it, for example. Matthew Wilcox's presentation during the 2018 linux.conf.au Kernel miniconf made the case that, for the kernel's venerable radix tree data structure, the latter situation holds. His response is a new approach to an old data structure that he is calling the "XArray".

The kernel's radix tree is, he said, a great data structure, but it has far fewer users than one might expect. Instead, various kernel subsystems have implemented their own data structures to solve the same problems. He tried to fix that by converting some of those subsystems and found that the task was quite a bit harder than it should be. The problem, he concluded, is that the API for radix trees is bad; it doesn't fit the actual use cases in the kernel.

Part of the issue is that the "tree" terminology is confusing in this case. A radix tree isn't really like the classic trees that one finds in data-structure texts. Addition of an item to a tree has been called "insertion" for decades, for example, but an "insert" doesn't really describe what happens with a radix tree, especially if an item with the given key is already present there. Radix trees also support concepts like "exception entries" that users find scary just because of the naming that was used.



So Wilcox decided to fix the interface. He has kept the existing radix-tree data structure unchanged; there are, he said, few problems with it. But the metaphor describing its operation has been changed from a tree to an array. It behaves much like an automatically resizing array; fundamentally, it is an array of pointer values indexed by an unsigned long. This view better describes how the structure is actually used.

The radix tree requires users to do their own locking; the XArray, instead, handles locking itself by default, simplifying the task of using it. The "preload" mechanism, which allows users to pre-allocate memory before acquiring locks, has been removed; it added significant complexity to the interface for almost no real value.

The actual XArray API has been split into two pieces, the normal API and the advanced API. The latter provides much more control to the caller; it can be used to explicitly manage locking, for example. This API will be used at call sites with special needs; the page cache is one example where it is needed. The normal API is entirely implemented on top of the advanced API, so it serves as a demonstration of how the advanced API can be used.

The page cache has been converted to use the XArray, he said, and there are no bugs remaining that he knows of. His plan is to "plead" for inclusion during the 4.16 merge window.

A quick look at the XArray API

The current version of the XArray patch set, as of this writing, is version 6, posted on January 17. It is a 99-patch series and, thus, not for the faint of heart, but an introduction to its operation can be found in the documentation patch in the series. One starts by defining an array with:

```
#include <linux/xarray.h>

DEFINE_XARRAY(array_name);
/* or */
struct xarray array;
xa_init(&array);
```

Storing a value into an XArray is done with:

```
void *xa_store(struct xarray *xa, unsigned long index, void *entry,
               gfp_t gfp);
```

This function will store the given `entry` at the requested `index`; if memory must be allocated, the given `gfp` flags will be used. The return value on success is the previous value (if any) that was stored at `index`. An entry can be removed from the array by storing `NULL` there, or by calling:

```
void *xa_erase(struct xarray *xa, unsigned long index);
```

Other variants include `xa_insert()` to store without overwriting an existing entry, and `xa_cmpxchg()`:

```
void *xa_cmpxchg(struct xarray *xa, unsigned long index, void *old,
                 void *entry, gfp_t gfp);
```

In this case, `entry` will be stored at `index`, but only if the current value stored there matches `old`. Either way, the current value stored at `index` is returned.

Fetching a value from an XArray is done with `xa_load()`:

```
void *xa_load(struct xarray *xa, unsigned long index);
```

The return value is the value stored at `index`. In an XArray, an empty entry is the same as an entry that has had `NULL` stored into it, so `xa_load()` will not behave specially for empty entries.

Up to three single-bit tags can be set on any non-null XArray entry; they are managed with:

```
void xa_set_tag(struct xarray *xa, unsigned long index, xa_tag_t tag);
void xa_clear_tag(struct xarray *xa, unsigned long index, xa_tag_t tag);
bool xa_get_tag(struct xarray *xa, unsigned long index, xa_tag_t tag);
```

The `tag` values here are one of `XA_TAG_0`, `XA_TAG_1`, and `XA_TAG_2`. A call to `xa_set_tag()` will set the given `tag` on the entry at `index`, while `xa_clear_tag()` will remove that tag. `xa_get_tag()` returns true if the given `tag` is set on the entry at `index`.

As a general rule, an XArray is sparsely populated; that means that looping through all of the possible entries looking for the non-null ones would be rather inefficient. Instead, this helper macro should be used:

```
xa_for_each(xa, entry, index, max, filter) {
    /* Process "entry" */
}
```

Before entering this loop, `index` should be set to the beginning of the range to be iterated over, while `max` indicates the largest index that should be returned. The `filter` value can specify tag bits which will be used to filter out uninteresting entries. During loop execution, `index` will be set to match the current entry. It is possible to change the iteration by changing `index` within the loop; it is also allowed to make changes to the array itself.

There are many other functions in the normal API that provide other ways to access an XArray; there is also the entire advanced API for the special cases. The API as a whole is reasonably large and complex, but it would appear to be rather easier to work with than the radix-tree API. The current patch set converts a number of

radix-tree users to XArrays, but some still remain. If all goes according to Wilcox's plan, though, those will be converted in the near future and the radix-tree API will head toward removal.

[Your editor would like to thank the Linux Foundation, LWN's travel sponsor, and linux.conf.au for assisting with his travel to the event.]

Index entries for this article

Kernel	Radix tree
Kernel	XArray
Conference	linux.conf.au/2018

(Log in to post comments)

The XArray data structure

Posted Jan 24, 2018 12:37 UTC (Wed) by **cborni** (subscriber, #12949) [Link]

The preloading is used in some cases with tricky locking. Looking at the patches, preloading is not removed, it is now replace by a different scheme with a call names xa_reserve.

Reply to this comment

The XArray data structure

Posted Jan 24, 2018 13:54 UTC (Wed) by **Paf** (subscriber, #91811) [Link]

Yeah, the Lustre filesystem (possibly not in the upstream kernel version) has a tricky interaction with a radix tree where it has to do a preload or it can deadlock. I didn’t see how it was possible to not have the preload in some form. Relieved to hear it’s still there.

Reply to this comment

The XArray data structure

Posted Jan 29, 2018 9:54 UTC (Mon) by **willy** (subscriber, #9762) [Link]

You'd be amazed how few places actually need the preloading functionality. We had 42 calls to various forms of the preload functions.

I've entirely removed the radix tree preload from my tree and I have only seven calls to xa_reserve() (in four files). Until a few weeks ago, I hadn't found anywhere with sufficiently complex locking requirements to need xa_reserve(). The general pattern that needs xa_reserve() is:

```
radix_tree_preload();
spin_lock(some_other_lock);
spin_lock(tree_lock);
radix_tree_insert();
...
```

Sometimes you can invert the order of the locks, or combine the two locks, or resort to the advanced API in order to explicitly handle unlocking before allocating memory, but converting it to:

```
xa_reserve();
spin_lock(other_lock);
xa_store();
```

is easier.

If the out-of-tree lustre code has developed new and exciting ways to use the radix tree API, then they'll get to fix it up ... maintaining out of tree patches is expensive and drivers/staging/ is there to lower the cost.

Reply to this comment

Preallocation

Posted Jan 24, 2018 21:16 UTC (Wed) by **corbet** (editor, #1) [Link]

Indeed, preallocation is not entirely removed. It's not visible in the normal API, though, and has become something that most callers need not worry about.

Reply to this comment

The XArray data structure

Posted Jan 24, 2018 20:48 UTC (Wed) by **meyert** (subscriber, #32097) [Link]

What is the reason for the maneki-nekos at conferences these days?

Reply to this comment

The XArray data structure

Posted Jan 24, 2018 21:20 UTC (Wed) by **LawnGnome** (subscriber, #84178) [Link]

Apparently there have been issues with video recording lockups at past conferences not being noticed until a talk has already been underway, so there needs to be movement in the frame for a quick sanity check as the speaker is preparing to begin. Some conference venues have clocks, but the linux.conf.au venue doesn't this year, so they've deployed a small clowder of maneki-neko instead.

Reply to this comment

The XArray data structure

Posted Jan 25, 2018 14:42 UTC (Thu) by **skitching** (guest, #36856) [Link]

Isn't this equivalent to a map (aka dictionary) structure keyed by an integer? That metaphore works better for me than a "dynamic array".

Reply to this comment

The XArray data structure

Posted Jan 25, 2018 17:20 UTC (Thu) by **excors** (subscriber, #95769) [Link]

Yes, but I think "a map structure keyed by an integer" is essentially the definition of "array" (at least as an abstract data type), and it's generally easier to use the shorter name.

(Except in Javascript, where an array is actually a map structure keyed by the decimal string representation of an integer.)

The concrete implementation of that data type is a separate issue; it could be a series of consecutive memory addresses, or a hash table, or a red-black tree, or a radix tree, or something that changes depending on the density of its contents, etc. But users of an API shouldn't care about those implementation details - they just

care that it behaves like an abstract array with certain performance characteristics, so it seems sensible to call it an array.

Reply to this comment

The XArray data structure

Posted Jan 25, 2018 18:00 UTC (Thu) by **bof** (subscriber, #110741) [Link]

> (Except in Javascript, where an array is actually a map structure keyed by the decimal string representation of an integer.)

Or in PHP, where it can be both, at the same time...

> The concrete implementation of that data type is a separate issue; it could be a series of consecutive memory addresses, or a hash table, or a red-black tree, or a radix tree, or something that changes depending on the density of its contents, etc. But users of an API shouldn't care about those implementation details - they just care that it behaves like an abstract array with certain performance characteristics, so it seems sensible to call it an array.

In kernel space, users probably do care about such details very much. Otherwise the kernel would have been rewritten in PHP a long time ago...

Reply to this comment

The XArray data structure

Posted Feb 2, 2018 9:25 UTC (Fri) by **Wol** (subscriber, #4433) [Link]

No. The USERS don't give a monkeys. The implementers care deeply ...

Cheers,
Wol

Reply to this comment

The XArray data structure

Posted Feb 2, 2018 11:07 UTC (Fri) by **andresfreund** (subscriber, #69562) [Link]

Sorry to be that blunt: Could you perhaps consider commenting a little less frequently? You comment on nearly everything - not infrequently the majority of new comments are yours. Even ignoring them reduces the usability of the "unread comments" page noticeably.

Reply to this comment

The XArray data structure

Posted Jan 25, 2018 20:19 UTC (Thu) by **rweikusat2** (subscriber, #117920) [Link]

> Yes, but I think "a map structure keyed by an integer" is essentially the definition of "array" (at least as an abstract data type), and
> it's generally easier to use the shorter name.

Abstractly, an 'array' is an ordered set of values which can be accessed by position in constant time. That's something entirely different from a map associating members from a set of keys with members from a set

of values. Eg, they key set of a map using integer keys might be {-60, -1, 1434, 131072}.

Reply to this comment

The XArray data structure

Posted Feb 2, 2018 9:24 UTC (Fri) by **Wol** (subscriber, #4433) [Link]

That's a sparse array ... :-)

Cheers,
Wol

Reply to this comment

The XArray data structure

Posted Jan 25, 2018 21:27 UTC (Thu) by **zblaxell** (subscriber, #26385) [Link]

Everyone working in a new programming environment, even if the language is familiar, has to figure out whether each thing with the word "array" in its name is some flavor of btree, dict, hash table, map, rbtree, set, vector, constant pointer + index * size, "other", or "not implemented here." Once that's known for "array", one then has to learn the local names for all the other things too. They all have different capabilities, limitations, performance, and unimaginable quirks that make the wrong choice wrong for at least one situation.

The wrong word was used here, but that was inevitable, as there's always someone who joins the party late and complains "but in my tribe we always called that by another name." There could never have been a correct name for this structure--only a previously unused one, preferably something with a short unique acronym to use as a prefix.

(That said, it's totally a std::map<unsigned long, std::tuple<void*, bool[3]>>, Alloc> with internal locking, stable iterators, and atomic value operators...except for the parts where it's different from that)

Reply to this comment

The XArray data structure

Posted Jan 29, 2018 8:58 UTC (Mon) by **willy** (subscriber, #9762) [Link]

Hi Zygo, great to hear from you again

The XArray is intended to have performance and usage characteristics as close as possible to a C native array. Unlike, say, a hash table. I don't want anyone to be surprised by a mismatch between the naming and the behaviour.

You're right that there's no right name for this data structure. The closest thing I've found in the literature is the Judy Array. So I sidestepped the question of what the _data structure_ is called and focused on what it provides (an array abstraction). That enables us to change the underlying implementation as we see fit. There were a number of ways the old API leaked the internal details of the data structure to the callers, and I've eliminated those.

I'm not a C++ programmer, nor a theoretician. I'm happy for the API to not match C++, as long as it matches people's expectations.

Reply to this comment

(XArray) C++ → C naming conversions

Posted Mar 31, 2019 0:49 UTC (Sun) by **zenaan** (guest, #3778) [Link]

>> (That said, it's totally a std::map<unsigned long, std::tuple<void*, bool[3]>>, Alloc> with internal locking, stable iterators, and atomic value operators...except for the parts where it's different from that)

> I'm not a C++ programmer, nor a theoretician. I'm happy for the API to not match C++, as long as it matches people's expectations.

How would "std::map<unsigned long, std::tuple<void*, bool[3]>>, Alloc> with internal locking, stable iterators, and atomic value operators..." even map to C as a name?

Discluding the behaviour characteristics, and taking the first letter of each C++ 'word' → smulstvbA_array perhaps?

Reply to this comment

(XArray) C++ → C naming conversions

Posted Mar 31, 2019 16:22 UTC (Sun) by **nybble41** (subscriber, #55106) [Link]

> How would "std::map<unsigned long, std::tuple<void*, bool[3]>>, Alloc> …" even map to C as a name?

This is a solved problem. After all, G++ already generates C-compatible names which encode their C++ argument types.

Just call it St3mapImSt5tupleIJPvA3_bEESt4lessImESaISt4pairIKmS3_EEE_array. :-)

Reply to this comment

The XArray data structure

Posted Jan 26, 2018 22:09 UTC (Fri) by **ncultra** (subscriber, #121511) [Link]

The api is similar to the existing, though sparsely used, flex_array library, which has pre-allocation but not locking. I have found it useful but am concerned that no upstream kernel code seems to be using flex_arrays.

Reply to this comment

The XArray data structure

Posted Jan 29, 2018 4:27 UTC (Mon) by **willy** (subscriber, #9762) [Link]

Thanks for writing up my talk, Jon!

I have a quibble with your article,

> The return value on success is the previous value (if any) that was stored at index.

This is the kind of radix tree thinking that I'm trying to eradicate. Every index has a value stored at it. An "empty array" has a NULL pointer stored at every index; and a freshly-initialised array is empty. Just deleting the parenthetical would make me happy :-)

If you can suggest an improvement to the documentation that would help people get to this mindset, I'd be more than happy to incorporate it. It may just be a matter of time and people getting used to the new way of thinking.

Reply to this comment

The XArray data structure

Posted Jan 29, 2018 13:50 UTC (Mon) by **nix** (subscriber, #2304) [Link]

That's exactly what I was thinking. This is not a tree. It's an array that happens to have a tree as its internal storage representation currently. You can't have an array element with no value! (Sure, it's sparse, but all that means is that null elements are cheaper than non-null ones).

Reply to this comment