Microsoft

Replacing the Radix Tree

MATTHEW WILCOX

LINUX.CONF.AU 2018-01-22

The two hardest problems in computer science

- Cache invalidation
- Naming things
- Off-by-one errors

Phil Karlton & Leon Bambrick

What is a Radix Tree?

- Wikipedia says Radix Trees are all about strings
  - Used for string compression and inverted indices of text documents
- The Linux Radix Tree is unlike any tree I have found in the computer science literature
  - "We always always always must engineer the solution first. In the wise words of Peter Denning, in this field 'Theory follows practice'" – Paul McKenney
- Implicit keys (like a trie), but not a bitwise trie
- Grow/shrink, but never rebalanced
- RCU-safe

Keys in a tree?

Keys in a tree!

Why don't more people use it?

- The API is bad
  - (mostly) Not in the Rusty API Scale sense
  - http://ozlabs.org/~rusty/index.cgi/tech/2008-03-30.html
  - http://ozlabs.org/~rusty/index.cgi/tech/2008-04-01.html
- It doesn't fit how users want to use it

Tree-Insert

- Adding a key to a tree has been called Insert since at least 1968
- Classical trees only store keys
  - Storing additional information is left as an exercise for the reader
  - Literally in the case of Cormen, Leiserson & Rivest
- What happens when you try to store a duplicate key?
  - CLR's algorithm stores two copies of the same key
  - Linux Radix Tree returns –EEXIST
- What is the value of a key not in the tree?
  - Linux Radix Tree returns NULL
- What is the difference between storing a NULL value and removing a key?

Fixing the interface

- Keep the Radix Tree data structure unchanged
- Change the metaphor to an array
- Provide locking by default
- Remove memory preloading
- Make memory allocation flags explicit
- Hide details of implementation from users
- Provide two levels of API – Normal and Advanced

XArray concept

- An XArray is an array of pointers indexed by an unsigned long
- Entries may have up to three tag bits (get/set/clear)
- You can iterate over entries
- You can extract a batch of entries

XArray Normal API (part 1)

```
void xa_init(struct xarray *);

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

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

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

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

int xa_insert(struct xarray *, unsigned long index,
                        void *entry, gfp_t);

bool xa_empty(struct xarray *);

bool xa_tagged(struct xarray *, xa_tag_t);
```

XArray Normal API (part 2)

```
bool xa_get_tag(struct xarray *, unsigned long index, xa_tag_t);

void xa_set_tag(struct xarray *, unsigned long index, xa_tag_t);

void xa_clear_tag(struct xarray *, unsigned long index, xa_tag_t);

int xa_extract(struct xarray *, unsigned long start, unsigned long max,
                        void **dst, unsigned int n, xa_tag_t filter);

void *xa_find(struct xarray *, unsigned long *index,
                        unsigned long max, xa_tag_t filter);

void *xa_find_after(struct xarray *, unsigned long *index,
                        unsigned long max, xa_tag_t filter);

xa_for_each(struct xarray *, void *entry, unsigned long index,
                        unsigned long max, xa_tag_t filter) { }
```

brd example

```
1.  if (radix_tree_preload(GFP_NOIO))
2.      return NULL;
3.  spin_lock(&brd->brd_lock);
4.  if (radix_tree_insert(&brd->brd_pages, idx, page)) {
5.      __free_page(page);
6.      page = radix_tree_lookup(&brd->brd_pages, idx);
7.  }
8.  spin_unlock(&brd->brd_lock);
9.  radix_tree_preload_end();
```

brd reimplementation

```
1.  cur = xa_cmpxchg(&brd->brd_pages, idx, NULL, page,
                     GFP_NOIO);
2.  if (cur) {
3.      __free_page(page);
4.      page = cur;
5.  }
```

XArray Advanced API

- Based on XArray Operation State (xa_state) containing:
  - Pointer to XArray
  - Index we're operating on
  - Cached pointer into tree (or error)
  - Freshly allocated node
  - Callback function pointer
  - Other miscellaneous state
- Explicit locking

xa_cmpxchg (xa, index, old, entry, gfp)

```
1.  XA_STATE(xas, xa, index);
2.  void *curr;
3.  do {
4.      xas_lock(&xas);
5.      curr = xas_load(&xas);
6.      if (curr == old)
7.          xas_store(&xas, entry);
8.      xas_unlock(&xas);
9.  } while (xas_nomem(&xas, gfp));
10. return xas_result(&xas, curr);
```

Current status

- Page Cache converted to use XArray
  - Will plead for inclusion in 4.16
- IDR converted to use XArray
  - IDR test suite found some great bugs
- Radix tree test suite adapted to the XArray API
- Dozens of radix trees converted, but limited testing
  - Typically involves changing locking

Future projects

- Convert remaining radix trees to XArray
- Replace IDA with XBitmap (in development)
- Add the XQueue (documented but no code)
- Add support for an Array of XArrays
- Expand the number of tags
  - Five or six possible for non-page-cache uses
  - Dozens possible with the XBitmap and Array of XArrays ideas combined
- Reduce the memory consumed by smaller XArrays
- Represent sparse indices more efficiently
- Allow index to be an unsigned long long

Lessons learned

- Simple to use and simple to implement are not the same
  - Work on both sides of an interface
- Performance and simplicity can also be in conflict
- Hide irrelevant things from users, but expose relevant things
- Make common tasks easy
- Always write documentation
- Fix other people's bugs as you notice them
- Get as many reviewers as you can stand
  - Be prepared to iterate on your design
- Look around for inspiration
- Short names are better than long

Microsoft ❤ Linux

Bibliography

- Gödel, Escher, Bach: An Eternal Golden Braid (Douglas Hofstadter)
- Nineteen Eighty-Four (George Orwell)
- The Jargon File (Finkel, Crispin, Steele, Woods, Raymond, et al)
- The Art of Computer Programming (Donald Knuth)
- Introduction to Algorithms (Cormen, Leiserson & Rivest)
- Algorithms + data structures = programs (Niklaus Wirth)
- The C Programming Language (Kernighan & Ritchie)

Reclaiming nodes

- Slab allocator can notify slab owner when it wants to defragment a slab page
  - Experimental patches available
- Can't be done with Radix Tree because slab owner doesn't know what locking protects each radix tree
- XArray patches available to replace reclaimed nodes with fresh

XArray Advanced API (part 1)

```
int xas_error(struct xa_state *);
void xas_set_err(struct xa_state *, long err);
bool xas_invalid(struct xa_state *);
bool xas_valid(struct xa_state *);
void xas_reset(struct xa_state *);
bool xas_retry(struct xa_state *, void *entry);
void *xas_load(struct xa_state *);
void *xas_store(struct xa_state *);
void *xas_create(struct xa_state *);
void *xas_find(struct xa_state *);
```

XArray Advanced API (part 2)

```
bool xas_get_tag(struct xa_state *, xa_tag_t);
void xas_set_tag(struct xa_state *, xa_tag_t);
void xas_clear_tag(struct xa_state *, xa_tag_t);
void *xas_find_tag(struct xa_state *, unsigned long max, xa_tag_t);
void xas_init_tags(const struct xa_state *);
bool xas_nomem(struct xa_state *, gfp_t);
void xas_pause(struct xa_state *);
void xas_create_range(struct xa_state *, unsigned long max);
void *xas_reload(struct xa_state *);
void xas_set(struct xa_state *, unsigned long index);
```

XArray Advanced API (part 3)

```
void xas_set_order(struct xa_state *, unsigned long index,
                   unsigned int order);
void xas_set_update(struct xa_state *, xa_update_node_t);
void *xas_next_entry(struct xa_state *, unsigned long max);
void *xas_next_tag(struct xa_state *, unsigned long max, xa_tag_t);
void *xas_prev(struct xa_state *);
void *xas_next(struct xa_state *);
xas_for_each(struct xa_state *, void *entry, unsigned long max) {}
xas_for_each_tag(struct xa_state *, void *entry, unsigned long max,
                 xa_tag_t) {}
```

vmalloc example

```
1   struct page **pages;
2   unsigned int nr_pages = get_vm_area_size(area) >> PAGE_SHIFT;
3   unsigned int array_size = (nr_pages * sizeof(struct page *));
4   /* Please note that the recursion is strictly bounded. */
5   if (array_size > PAGE_SIZE) {
6       pages = __vmalloc_node(array_size, 1, nested_gfp,
7                              PAGE_KERNEL, node, area->caller);
8   } else {
9       pages = kmalloc_node(array_size, nested_gfp, node);
10  }
```