# XArray

**One data structure to rule them all**
**One data structure to find them**
**One data structure to bring them all**
**And in the darkness bind them**

Matthew Wilcox
Linux Plumber’s Conference
2018-11-14

ORACLE

# What is the XArray?

- Automatically resizing array of pointers
- Indexed by *unsigned long*
- All pointers initially `NULL`
- Contains spinlock
- Loads under RCU read lock

# Normal API - Fundamentals

- Some users only need load, store and (maybe) iterate:

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

# Normal API - Marks

- Three auxiliary bits per non-NULL entry

```
void xa_set_mark(struct xarray *,
                  unsigned long index, xa_mark_t);
void xa_clear_mark(struct xarray *,
                  unsigned long index, xa_mark_t);
bool xa_get_mark(struct xarray *,
                  unsigned long index, xa_mark_t);
```

# Normal API – Less used

- Some users need something a little more complex:

```
int xa_insert(struct xarray *, unsigned long index,
              void *entry, gfp_t);
void *xa_cmpxchg(struct xarray *, unsigned long index,
              void *old, void *entry, gfp_t);
void *xa_find(struct xarray *, unsigned long *index,
              unsigned long max, xa_mark_t filter);
int xa_reserve(struct xarray *, unsigned long index,
              gfp_t);
```

# Advanced API

- If you need something a little unusual, the previous functions are all built on smaller building-block functions.  Here’s xa_cmpxchg():

```
XA_STATE(xas, xa, index);
void *curr;
do {
    xas_lock(&xas);
    curr = xas_load(&xas);
    if (curr == old)
        xas_store(&xas, entry);
    xas_unlock(&xas);
} while (xas_nomem(&xas, gfp));
return xas_result(&xas, curr);
```

# Normal API - Allocation

- The XArray can track free entries for you:

```
int xa_alloc(struct xarray *, u32 *id,
              u32 max, void *entry, gfp_t);
int xa_alloc_cyclic(struct xarray *, u32 *id,
              u32 min, u32 max, void *entry, gfp_t);
```

- Storing NULL does not free the entry; now have to use **`xa_erase()`**

# What should it be used for?

- All radix tree users replaced here: http://git.infradead.org/users/willy/linux-dax.git/shortlog/refs/heads/xarray-conv
- Some of the IDR users also converted
- Replace custom implementations of resizing arrays
- Some linked lists can be replaced

# What shouldn’t it be used for?

- Sparse arrays (yet)
- Hashtables (yet)
- Ranges (API exists, has one user, don’t add more yet)
- Replacing rbtrees (yet)
- The fd table

# Defend against Spectre!

```
        if (get_user(id, &ring→id))
                return NULL;

        rcu_read_lock();
-       table = rcu_dereference(mm->ioctx_table);

-       if (!table || id >= table->nr)
-               goto out;

-       ctx = rcu_dereference(table->table[id]);
+       ctx = xa_load(&mm->ioctx, id);
```

# Convert linked lists to allocating XArrays

- If you just need to keep a list of objects and iterate over them, you can just delete the list_head from your data structure
- If you need to be able to remove objects from the middle of the list, you may need to store the ID in the object (16 bytes → 4 bytes)
- If an object may be on one of several lists, you may also need to store the XArray pointer in the object (16 bytes → 12 bytes)
  - But maybe you can use marks to avoid having multiple lists
- If order matters, use a cyclic allocator
  - We might need a cyclic iterator too

# Convert linked lists to allocating XArrays

- For lists with high turn-over (eg LRU), this is not ideal yet
- If a driver is keeping track of its devices, this is perfect
- Antipattern: IDA to allocate device number, store devices in linked list
  - Worse antipattern: Driver searches linked list to find device with matching ID
- Must be able to allocate memory at list add/move time
- Do we need a new API for this usage?

# Example linked list conversion

```
+unsigned long index;
-list_for_each_entry(sdev, &starget->devices, same_target_siblings) {
+xa_for_each(&starget->devices, sdev, index, UINT_MAX, XA_PRESENT) {
-struct scsi_device *sdev, *tmp;
+struct scsi_device *sdev;
+unsigned long index;
-list_for_each_entry_safe(sdev, tmp, &starget->devices,
-      same_target_siblings) {
+xa_for_each(&starget->devices, sdev, index, UINT_MAX, XA_PRESENT) {
-INIT_LIST_HEAD(&sdev->same_target_siblings);
-INIT_LIST_HEAD(&starget→devices);
+xa_init_flags(&starget->devices, XA_FLAGS_ALLOC);
-list_del(&sdev->same_target_siblings);
+xa_erase(&sdev->sdev_target->devices, sdev->pertarget_id);
-list_add_tail(&sdev->same_target_siblings, &starget→devices);
+xa_alloc(&starget->devices, &sdev->pertarget_id, UINT_MAX, sdev, GFP_ATOMIC);
-struct list_head    same_target_siblings; /* devices sharing same target id */
+u32 pertarget_id;      /* index into target's device list */
-struct list_head    devices;
+struct xarray       devices;
```

ORACLE®