ID Allocation

Author: Matthew Wilcox

Overview

A common problem to solve is allocating identifiers (IDs); generally small numbers which identify a thing. Examples include file descriptors, process IDs, packet identifiers in networking protocols, SCSI tags and device instance numbers. The IDR and the IDA provide a reasonable solution to the problem to avoid everybody inventing their own. The IDR provides the ability to map an ID to a pointer, while the IDA provides only ID allocation, and as a result is much more memory-efficient.

IDR usage

Start by initialising an IDR, either with DEFINE_IDR() for statically allocated IDRs or idr_init() for dynamically allocated IDRs.

You can call idr_alloc() to allocate an unused ID. Look up the pointer you associated with the ID by calling idr_find() and free the ID by calling idr_remove().

If you need to change the pointer associated with an ID, you can call idr_replace(). One common reason to do this is to reserve an ID by passing a `NULL` pointer to the allocation function; initialise the object with the reserved ID and finally insert the initialised object into the IDR.

Some users need to allocate IDs larger than `INT_MAX`. So far all of these users have been content with a `UINT_MAX` limit, and they use idr_alloc_u32(). If you need IDs that will not fit in a u32, we will work with you to address your needs.

If you need to allocate IDs sequentially, you can use idr_alloc_cyclic(). The IDR becomes less efficient when dealing with larger IDs, so using this function comes at a slight cost.

To perform an action on all pointers used by the IDR, you can either use the callback-based idr_for_each() or the iterator-style idr_for_each_entry(). You may need to use idr_for_each_entry_continue() to continue an iteration. You can also use idr_get_next() if the iterator doesn't fit your needs.

When you have finished using an IDR, you can call idr_destroy() to release the memory used by the IDR. This will not free the objects pointed to from the IDR; if you want to do that, use one of the iterators to do it.

You can use idr_is_empty() to find out whether there are any IDs currently allocated.

If you need to take a lock while allocating a new ID from the IDR, you may need to pass a restrictive set of GFP flags, which can lead to the IDR being unable to allocate memory. To work around this, you can call idr_preload() before taking the lock, and then idr_preload_end() after the allocation.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\[linux-master][Documentation][core-api]idr.rst, line 66)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/idr.h
   :doc: idr sync
```

IDA usage

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\[linux-master][Documentation][core-api]idr.rst, line 72)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: lib/idr.c
   :doc: IDA description
```

Functions and structures

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\[linux-master][Documentation][core-api]idr.rst, line 78)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/idr.h
   :functions:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\core-api\[linux-master][Documentation][core-api]idr.rst, line 80)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: lib/idr.c
   :functions:
```