

Memory hotplug

Memory hotplug event notifier

Hotplugging events are sent to a notification queue.

There are six types of notification defined in `include/linux/memory.h`:

MEM_GOING_ONLINE

Generated before new memory becomes available in order to be able to prepare subsystems to handle memory. The page allocator is still unable to allocate from the new memory.

MEM_CANCEL_ONLINE

Generated if MEM_GOING_ONLINE fails.

MEM_ONLINE

Generated when memory has successfully brought online. The callback may allocate pages from the new memory.

MEM_GOING_OFFLINE

Generated to begin the process of offlining memory. Allocations are no longer possible from the memory but some of the memory to be offlined is still in use. The callback can be used to free memory known to a subsystem from the indicated memory block.

MEM_CANCEL_OFFLINE

Generated if MEM_GOING_OFFLINE fails. Memory is available again from the memory block that we attempted to offline.

MEM_OFFLINE

Generated after offlining memory is complete.

A callback routine can be registered by calling:

```
hotplug_memory_notifier(callback_func, priority)
```

Callback functions with higher values of priority are called before callback functions with lower values.

A callback function must have the following prototype:

```
int callback_func(
    struct notifier_block *self, unsigned long action, void *arg);
```

The first argument of the callback function (self) is a pointer to the block of the notifier chain that points to the callback function itself. The second argument (action) is one of the event types described above. The third argument (arg) passes a pointer of struct `memory_notify`:

```
struct memory_notify {
    unsigned long start_pfn;
    unsigned long nr_pages;
    int status_change_nid_normal;
    int status_change_nid;
}
```

- `start_pfn` is start_pfn of online/offline memory.
- `nr_pages` is # of pages of online/offline memory.
- `status_change_nid_normal` is set node id when `N_NORMAL_MEMORY` of `nodemask` is (will be) set/clear, if this is -1, then `nodemask` status is not changed.
- `status_change_nid` is set node id when `N_MEMORY` of `nodemask` is (will be) set/clear. It means a new(memoryless) node gets new memory by online and a node loses all memory. If this is -1, then `nodemask` status is not changed.

If `status_change_nid* >= 0`, callback should create/discard structures for the node if necessary.

The callback routine shall return one of the values `NOTIFY_DONE`, `NOTIFY_OK`, `NOTIFY_BAD`, `NOTIFY_STOP` defined in `include/linux/notifier.h`

`NOTIFY_DONE` and `NOTIFY_OK` have no effect on the further processing.

`NOTIFY_BAD` is used as response to the `MEM_GOING_ONLINE`, `MEM_GOING_OFFLINE`, `MEM_ONLINE`, or `MEM_OFFLINE` action to cancel hotplugging. It stops further processing of the notification queue.

`NOTIFY_STOP` stops further processing of the notification queue.

Locking Internals

When adding/removing memory that uses memory block devices (i.e. ordinary RAM), the `device_hotplug_lock` should be held to:

- synchronize against online/offline requests (e.g. via sysfs). This way, memory block devices can only be accessed (`.online/.state`

attributes) by user space once memory has been fully added. And when removing memory, we know nobody is in critical sections.

- synchronize against CPU hotplug and similar (e.g. relevant for ACPI and PPC)

Especially, there is a possible lock inversion that is avoided using `device_hotplug_lock` when adding memory and user space tries to online that memory faster than expected:

- `device_online()` will first take the `device_lock()`, followed by `mem_hotplug_lock`
- `add_memory_resource()` will first take the `mem_hotplug_lock`, followed by the `device_lock()` (while creating the devices, during `bus_add_device()`).

As the device is visible to user space before taking the `device_lock()`, this can result in a lock inversion.

onlining/offlining of memory should be done via `device_online()/device_offline()` - to make sure it is properly synchronized to actions via sysfs. Holding `device_hotplug_lock` is advised (to e.g. protect `online_type`)

When adding/removing/onlining/offlining memory or adding/removing heterogeneous/device memory, we should always hold the `mem_hotplug_lock` in write mode to serialise memory hotplug (e.g. access to global/zone variables).

In addition, `mem_hotplug_lock` (in contrast to `device_hotplug_lock`) in read mode allows for a quite efficient `get_online_mems/put_online_mems` implementation, so code accessing memory can protect from that memory vanishing.