Docks, Bays, and Ports

Kristen Carlson Accardi

SSG Core Software Division

Agenda

- How do hotpluggable devices work generically?
- How do hotpluggable devices work with ACPI?
- How do we work with hotpluggable devices in userspace?


Tipsy Pookins
Tipsy Pookins
Tipsy Pookins

Hotplug Overview

- **Detect something has changed (insertion or removal)**
  - Via Interrupt or via polling
  - Usually requires that a bus be rescanned to detect the changes
- **Initialize the new device (device driver will do this)**
  - Including device memory allocation or free, interrupt allocation etc.
- **Create or Destroy Software representations of device**
- **Some buses require physical unlocking of the device to prevent unsafe removal or unlocking via software**

ACPI Basics (very, very basic...)

- ACPI is a specification which essentially allows system manufacturers to write configuration and power management code to be executed by the OS via the ACPI interpreter
- ACPI software will capture ACPI events and route them to the interested drivers
- ACPI software allows drivers to execute specific methods that are provided by the system manufacturer that are found in the system's DSDT table, which is read by the ACPI subsystem at boot time

Hot Plug Drivers and ACPI

- **At driver load time, search for devices defined in the DSDT which are hotpluggable.**
  - You can do this by looking for specific methods which must be defined for removable devices, such as _EJ0, or _EJD, or you can look for other indications that a removable device is present
- **Tell ACPI you are interested in Events occuring on a specific ACPI device – usually "Bus Check" or "Device Check".**
- **These events indicate that the bus should be rescanned.**
- **If "Eject", then usually call the provided ACPI method to perform the removal safely (will usually unlock or power down connector)**
- **If a new device, then configure, sometimes executing ACPI methods provided to do this**
  - PRT, DCK, - is device dependent what you would execute

How Docking works with ACPI

- System vendor defines an ACPI object for the dock that implements a _DCK method
- _DCK is a control method, and not only defines the object as a dock station, but is also executed to control the isolation logic on the connector, as well as perform any other system specific configuration
- After _DCK is completed, the OS must reenumerate all enumerable buses as well as add non-enumerable devices

Dock Station Example DSDT

Docking is controlled by ACPI

```
Device(DOCK1) {    // Pass through dock – DOCK1
                   Name(_ADR, …)
                   Method(_EJ0, 0) {…}
                   Method(_DCK, 1) {…}
```

If an Object contains a method called _DCK, then it is a docking station

The _DCK method definition

Arguments:

Arg0

1–Dock (that is, remove isolation from connector)

0–Undock (isolate from connector)

Return Code:

1 if successful, 0 if failed.

Docking Basics

- **Cold Docking/Undocking**
- Always supported by Linux
- **Warm Docking/Undocking**
- Suspend, then dock or undock. This may or may not work
- **Hot Docking/Undocking**
- Insert or remove laptop while completely powered on. Not supported until now.

A peek in the dock driver code

```
static int __init dock_init(void)
{
        int num = 0;

        dock_station = NULL;

        /* look for a dock station */
        acpi_walk_namespace(ACPI_TYPE_DEVICE,
                        ACPI_ROOT_OBJECT, ACPI_UINT32_MAX,
                        find_dock, &num, NULL);

            ...
}
```

Dock driver (cont...)

```
/**
 * is_dock - see if a device is a dock station
 * @handle: acpi handle of the device
 *
 * If an acpi object has a _DCK method, then it is by definition a dock
 * station, so return true.
 */

static int is_dock(acpi_handle handle)
{
        acpi_status status;
        acpi_handle tmp;

        status = acpi_get_handle(handle, "_DCK", &tmp);

        if (ACPI_FAILURE(status))
                return 0;

        return 1;
}
```

Dock driver (cont ...)

static void dock_notify(acpi_handle handle, u32 event, void *data)

```
{
        struct dock_station *ds = data;
        switch (event) {
        case ACPI_NOTIFY_BUS_CHECK:
             if (!dock_in_progress(ds) && dock_present(ds)) {
                  begin_dock(ds);
                  dock(ds);
                  if (!dock_present(ds)) {
                        printk(KERN_ERR PREFIX "Unable to dock!\n");
                        break;
                  }
                  atomic_notifier_call_chain(&dock_notifier_list, ....
         case ACPI_NOTIFY_DEVICE_CHECK:
         case ACPI_NOTIFY_EJECT_REQUEST:
                   handle_eject_request(ds, event);
                   break;
```

Dock driver (cont...)

- "Rescanning" is accomplished by other device drivers.
- The dock driver must execute the _DCK method upon docking and undocking as well as calling _EJ0 to eject safely

```
arg_list.count = 1;
 arg_list.pointer = &arg;
 arg.type = ACPI_TYPE_INTEGER;
 arg.integer.value = 1;

if (ACPI_FAILURE(acpi_evaluate_object(ds->handle, "_EJ0",
 &arg_list, NULL)))
        pr_debug("Failed to evaluate _EJ0!\n");
```

Removable Drive Bays

- **What is it?**
  - An ATA device that is listed as ejectable in ACPI namespace
- **How do we find it?**
  - Searching for the bay is a bit more difficult than finding the dock
- **How do we insert/remove?**
  - More complicated due to PATA

Bay defined in DSDT

ATA controller is identified by searching for ATA specific ACPI methods

```
Device (SCND)

{
        Name (_ADR, 0x01)
        Method (_GTM, 0, NotSerialized)
        Method (_STM, 3, NotSerialized)

}
```

Device is child of controller – Device contains the “Eject” routine

```
Scope (\_SB.PCI0.IDE0.SCND.MSTR)

{
        Method (_EJ0, 1, NotSerialized)
        Method (_STA, 0, NotSerialized)

}
```

Bay driver notification routine

```
switch(event) {
        case ACPI_NOTIFY_BUS_CHECK:
        case ACPI_NOTIFY_DEVICE_CHECK:
        case ACPI_NOTIFY_EJECT_REQUEST:
                kobject_uevent(&dev->kobj,
                        KOBJ_CHANGE);
                break;
        default:
                printk(KERN_ERR PREFIX "Bay:
                        unknown event %d\n", event);

}
```

Userspace notifications for drive bay and dock

- Drive bays and dock stations both present a simple user interface via sysfs. They are both “platform” devices, and have files located in /sys/devices/platform/
- To cause the driver to execute _EJ0 for bay, or _DCK for dock, you write “1” to the “eject” file.
- To check the status of the device (whether it is present or not) you can read the “present” or “docked” file.
- The drivers both indicate status changes via the CHANGED uevent.
- The bay driver will NOT eject until userspace writes the file, the dock driver WILL dock or undock without userspace intervention if the button is pressed.

UDEV rules can be created to deal with these events

Future Work?

- **SATA does not require _EJ0, and therefore does not require special ACPI drivers for hotplug activities.**
- **Unfortunately, most laptops still ship with CDROM as PATA.**
  - But, that will change eventually
- **Ideally, the bay eject could be done from the ATA subsystem rather than in a separate driver and would require (little) user intervention**
- **The Dock driver is feature complete in the kernel, and just needs someone to spiff up the user interface and "do stuff" during dock/undock**


intel®
Leap ahead™