Basics of Drivers

- Devices live on a device tree
  - each node is a device
  - tree shows physical location, eg /devices is “on this machine”,
    /devices/pci is “on this machine is a pci bus”, etc
  - prtconf(1M)

- Files
  - /etc/driver_aliases
  - /etc/path_to_inst
  - /etc/name_to_major
  - /etc/minor_perms
  - /kernel/drv/<driver>.conf
  - /kernel/drv/amd64

- Utilities
  - add_drv / rem_drv / update_drv
  - modinfo / modload / modunload
  - devfsadm
General Driver Classes

- Character
- Block
- Net
- USB
- SCSI Target
- HBA
- Filesystem
Char / Block Drivers

- Every char or block driver **must** include:
  - `_init(9E)`
  - `_info(9E)`
  - `_fini(9E)`
  - `attach(9E)`
  - `detach(9E)`
  - `getinfo(9E)`

- The DDI/DDK also defines several standard functions, eg:
  - `open(9E)`
  - `close(9E)`
  - `read(9E)`
  - `write(9E)`

- Mandatory headers:
  - `#include <sys/modctl.h>  /* used by _init, _info, _fini */`
  - `#include <sys/cmn_err.h> /* used for errors */`
  - `#include <sys/ddi.h>    /* used by all entry points */`
  - `#include <sys/sunddi.h> /* used by all entry points */`
Driver Support Functions

- **Memory**
  - `kmem_alloc(9f) & kmem_free(9f)`
  - `kmem_cache_(create|destroy|alloc)(9f)`
  - `ddi_dma_mem_alloc(9f)`

- **Mutex**
  - `kmutex_t`
  - `mutex_(enter|exit|init|destroy)`
    - `mutex_init – ddi_intr_get_pri(9F)`
More Driver Support

- **Registers**
  - `ddi_regs_map_setup(9f) ddi_get/ddi_put`

- **DMA**
  - `ddi_dma_attr(9s)`
  - `ddi_dma_(alloc|free)_handle(9f)`
  - `ddi_dma_(addr|buf)_bind_handle(9f)`
  - `ddi_dma_unbind_handle(9f)`
Interrupt handling

- OpenSolaris's 4 interrupt types
  - Legacy
  - MSI
  - MSI-X
  - Software
    - useful to lower interrupt priority
Legacy Interrupts

- `ddi_intr_get_supported_types(9F)`
- `ddi_intr_get_nintrs(9F)`
- `kmem_zalloc(9F)`

- `ddi_intr_get_pri(9F)`
- `ddi_intr_set_pri(9F)`
- `mutex_init(9F)`
- `ddi_intr_add_handler(9F)`
- `ddi_intr_enable(9F)`

- `ddi_intr_disable(9F)`
- `ddi_intr_remove_handle(9F)`
- `mutex_destroy(9F)`
- `ddi_intr_free(9F)`
- `kmem_free(9F)}`
MSI Interrupts

- ddi_intr_get_supported_types(9F)
- ddi_intr_get_nintrs(9F)
- **ddi_intr_alloc(9F)**

- ddi_intr_get_pri(9F)
- ddi_intr_set_pri(9F)
- mutex_init(9F)
- ddi_intr_add_handler(9F)

- ddi_intr_block_enable(9F) / ddi_intr_enable(9F)
Interrupt Handler Functions

- Triage/reject interrupt
- Inform device
- I/O request processing
- Prevent further interrupts if possible
- Return DDI_INTR_CLAIMED
High level interrupts

- How do you know you're high-level?
  - `ddi_intr_get_pri(9F) >= ddi_intr_get_hilevel_pri(9F)`

- What next?
  - fail to attach
  - be a high-level interrupt handler
    - be careful with high-level mutexes
    - you can call a software interrupt at lower priority
Network Device Drivers

- Implement the mandatory features of char/block drivers (DDI/DDK specs)
- STREAMS based
- Communicate via DLPI with protocol stacks

- Generic Lan Driver (GLD) vastly simplifies network driver development
  - multi-threaded, clonable, LKM for LAN drivers
  - Implements most STREAMS and DLPI functionality for LAN drivers
Using the Generic Lan Driver (GLD)

Compile Reqs:
- `#include <sys/gld.h>`
- Link with `"misc/gld"` to register GLD driver dependency

qinit(9S) structures:
- `qinit_read.qi_putp = NULL`
- `qinit_read.qi_srvp = gld_rsrv`
- `qinit_read.qi_qopen = gld_open`
- `qinit_read.qi_qclose = gld_close`

- `qinit_write.qi_putp = gld_wput`
- `qinit_write.qi_srvp = gld_wsrv`
- `qinit_write.qi_qopen = NULL`
- `qinit_write.qi_qclose = NULL`
Using the GLD (contd...)

- For your convenience GLD implements:
  - open(9E), getinfo(9E) & close(9E)
  - put(9E) & srv(9E) (both required for STREAMS)

- you must implement:
  - _init(9E) _info(9E) _fini(9E), attach(9E), detach(9E) (like all modules)

- for GLD:
  - gldm_reset(9E)
  - gldm_start(9E)
  - gldm_stop(9E)
  - gldm_set_mac_addr(9E)
  - gldm_set_multicast(9E)
  - gldm_set_promiscuous(9E)
  - gldm_send(9E)
  - gldm_intr(9E)
  - gldm_get_stats(9E)
  - gldm_ioctl(9E)
  - gld_mac_info(9S)
USB Drivers

- Can be block, char, or STREAMS
- Difference is in the USBA framework
  - USBA abstracts devices
  - Your driver calls the USBA instead of the driver hardware
- Device ID's can be compatible, not necessarily fixed
- Devices can have multiple interfaces
USB Drivers (contd.)

- USB passes requests through pipes
  - Control
  - Bulk (data)
  - Interrupt
  - Isochronos (time & rate sensitive data)
- Every device has a default pipe, retrieved from 
  `usb_get_dev_data(9F)`
- `usb_pipe_open(9F)` , `usb_pipe_close(9F)` `usb_get_dev_data(9F)` and `usb_lookup_ep_data(9F)` to create, close & get endpoints
- Requests initialized and freed with `usb_(alloc|free)_(ctrl|bulk|intr|isoc)__req(9F)`
- Requests transferred with `usb_pipe_(ctrl|bulk|intr|isoc)_xfer(_wait)?(9F)`
- Pipes can be cleaned after errors, eg with `usb_pipe_reset(9F)`
USB Drivers (contd.)

- Hotplugging
  - `usb_register_hotplug_cb(9fF)`

- Power
  - `power(9E) & usb_create_pm_components(9F) (in attach(9E))`
  - `pm_busy_component(9F) & pm_raise_power(9F)`
  - `pm_idle_component(9F)`
  - `usb_handle_remote_wakeup(9F)`
SCSI Target Drivers

- Can be char or block
- SCSA framework separates the SCSI command from the transport & HBA
  - As easy as scsi_transport(9F)
SCSI Target Drivers (contd)

- Must be built with -N"misc/scsi" to register the dependency
- Allocate and initialize scsi_device(9S) before probe(9E) or attach(9E)
  - *ie. in _init(9E)*
- Implement probe(9E) to autoconfigure
  - Must call scsi_probe(9F) & scsi_unprobe(9F)
- attach(9E) & detach(9E) must call scsi_probe(9F) & scsi_unprobe(9F) respectively again

- SCSI drivers pass commands through scsi_pkt(9S) structures
  - Passed to scsi_transport(9F)
  - Contains a callback, HBA calls once it's done everything
SCSI HBA Drivers

- HBA driver is responsible for:
  - Managing HBA hardware
  - Accepting SCSI commands from the target driver
  - Transporting commands to target device
  - Performing data transfers (by command)
  - Collecting status
  - Auto-request sense (optional)
  - Informing the target driver of command completion (or failure)

- HBA drivers are nexus drivers
SCSI HBA Drivers (contd)

- **Key data structures:** scsi_hba_tran(9S), scsi_address(9S), scsi_device(9S), scsi_pkt(9S)

- **Functions:**
  - scsi_hba...
    - init, fini, attach_setup, detach
    - tran_alloc, tran_free pkt_alloc, pkt_free
    - probe, lookup_capstr
  - tran_tgt....
    - init, probe, free
    - init_pkt, destroy_pkt, sync_pkt
  - tran...
    - dmafree
    - start, abort, reset, reset_notify
    - getcap, setcap
    - bus_reset quiesce unquiesce

- Each Command in its own data structure
Filesystem Drivers

- Boy are they weird...
- Interface stability (there is none)
Files & Directories:

- /kernel/fs/fsname
- /kernel/fs/${ISA64}/fsname
- /usr/kernel/fs/fsname
- /usr/kernel/fs/${ISA64}/fsname
- /usr/lib/fs/fsname/
- /etc/fs/fsname/
Filesystem Drivers (contd)

- no cb_ops or dev_ops as in char / block drivers

- static mntopt_t myfs_mntopttbl[] = { ... };

- static mntopts_t myfs_mntopt_prototype = { sizeof (myfs_mntopttbl) / sizeof (mntopt_t), myfs_mntopttbl };

- static int myfsinit(int, char *); /* initializes everything else */

- static vfsdef_t vfw = { VFSDEF_VERSION,
  "myfs", myfsinit, VSW_HASPROTO VSW_CANREMOUNT| VSW_STATS, &myfs_mntopt_prototype };

- static struct modlfs modlfs = { &mod_fsops, "my fs", &vfw };

- static struct modlinkage modlinkage = { MODREV_1, (void *)&modlfs, NULL };
Filesystem Drivers: mount options

typedef struct mntopt {
    char* mo_name;    /* option name */
    char** mo_cancel; /* list of options cancelled by this one */
    char* mo_arg;     /* argument string for this option */
    int mo_flags;     /* flags for this mount option */
    void* mo_data;    /* filesystem specific data */
} mntopt_t;

The VFS framework parses & validates a set of generic mount options, listed in
/usr/src/uts/common/fs/vfs.c. Some more are listed in <sys/mntent.h>.
FS Drivers: VFS/Vnode Interfaces

- Two sets of interfaces for FS drivers
  - affect one instance (i.e., filesystem)
    - mount(2), umount(2), sync(2), etc.
    - vfs ops
  - operations on “nodes”
    - files, directories, pipes, devices, etc
    - vnode ops
    - open(2), close(2), read(2), etc
VFS Ops

- VFS_MOUNT() / mount(2)
- VFS_UNMOUNT() / umount(2)
- VFS_STATVFS() / statvfs(2)
- VFS_SYNC() / sync(2)
- VFS_FREEVFS()
- VFS_ROOT() / mount(2)
- VFS_MOUNTRoot
- VFS_VGET()

- struct vfsops
Vnode Ops

- VOP_
  - open, close, read, write, getpage, putpage, create, remove, mkdir, rmdir, getattr, setattr, map, unmap, delmap, link, readlink, symlink, readdir, lookup, setfl, frlock, etc, etc

- vnodeops_t

- **not** a static table

- you can define several vnode op structures
Filesystem Drivers

- Two main tasks
  - Bmap
  - Alloc
- quota, permission, etc checks are done in the write glue code, not the framework
- segmap ( VOP_GETPAGE/PUTPAGE ) is recommended for all disk I/O
  - segmap_getmap()
  - segmap_pagecreate()
  - uiomove()
  - segmap_pageunlock
  - segmap_release
Vnodes

- **VFS_VGET()**: create a node on this filesystem
  - create, lookup directly call vfs_vget()
- **vop_lookup()** makes internal representation
  - calls vfs_vget()
- **vn_alloc()/vn_free()** alloc/free the actual vnode data structures. *The framework does not.*
- **vn_exists()/vn_invalid()** let you show/hide the data structure
- **VOP_CREATE() → VFS_VGET() → vn_alloc() → vn_setops() → vn_exists()**
  - now you have a Vnode
    - Vnode has: fsops, data pointer, etc
Vnodes

- A vnode lives as long as it is locked.
- The vnode is cached after unlock (until it's discarded manually or automatically).
- Vnodes are locked with VN_HOLD().
- After a VOP, call VN_RELE().
- VOP_INACTIVE() notified when the vnode isn't in use.
  - VOP_INACTIVE() calls vn_invalid & vn_free.
Links

- Device Driver Tutorial: http://docs.sun.com/app/docs/doc/817-5789
- Writing Device Drivers: http://docs.sun.com/app/docs/doc/816-4854
- Frank Hofmann: How to Write a Filesystem Driver
  - http://opensolaris.org/os/project/czosug/events_archive/czosug7_writing_filesystems.pdf
- Genunix: Writing Filesystems:
- OpenSolaris Kernel Source Code:
Questions?
Thank You

John Sonnenschein

<John.Sonnenschein@Sun.COM>