Writing Oracle Solaris 11 Device Drivers

Bill Knoche
Oracle Systems ISV Engineering
# Solaris 11 for Developers
## Webinar Series

<table>
<thead>
<tr>
<th>Webinar Series Topic</th>
<th>Date</th>
<th>Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modern Software Packaging for Enterprise Developers</td>
<td>03-27-12 @ 9am PT</td>
<td>Eric Reid</td>
</tr>
<tr>
<td>Simplify Your Development Environment with Zones, ZFS &amp; More</td>
<td>04-10-12 @ 9am PT</td>
<td>Eric Reid &amp; Stefan Schneider</td>
</tr>
<tr>
<td>Managing Application Services – Using SMF Manifests in Solaris 11</td>
<td>04-24-12 @ 9am PT</td>
<td>Matthew Hosanee</td>
</tr>
<tr>
<td>Optimize Your Applications on Solaris 11: The DTrace Advantage</td>
<td>05-06-12 @ 9am PT</td>
<td>Angelo Rajadurai</td>
</tr>
<tr>
<td>Maximize Application Performance and Reliability on Solaris 11</td>
<td>05-22-12 @ 9am PT</td>
<td>Vijay Taktar</td>
</tr>
<tr>
<td>Writing Solaris 11 Device Drivers</td>
<td>06-05-12 @ 9am PT</td>
<td>Bill Knoche</td>
</tr>
<tr>
<td>Publishing IPS Packaging</td>
<td>06-19-12 @ 9am PT</td>
<td>Eric Reid &amp; Brock Pytlík</td>
</tr>
<tr>
<td>The World of IPS, Part 2: Tools and Commands</td>
<td>07-17-12 @ 9am PT</td>
<td>Eric Reid &amp; Brock Pytlík</td>
</tr>
<tr>
<td>Oracle Solaris Remote Lab</td>
<td>07-31-12 @ 9am PT</td>
<td>Ron Larson &amp; Angelo Rajadurai</td>
</tr>
</tbody>
</table>
Fun Fact for today

• Transit of Venus occurs today in the evening before sunset.
• It is rare. The next is on December 11, 2117
• They happen twice, 8 years apart, every 100 years or so.
• Please pay attention to safe viewing
• For more information: http://www.transitofvenus.org/
Goal

- A introduction to how devices and drivers work in Solaris
- Basic device driver construction
- Where to go for documentation and help
Disclaimer

- The kernel is a dangerous place
  - You can see and touch everything
  - Some things will break if you touch them
- Drivers are difficult to get right
  - Attention to details
  - Be rigorous about debugging your code
- Drivers can do really bad things
  - Panics, hangs, destroy an installed OS and even break hardware
What we will talk about

- What are drivers?
- Major driver types
- Devices and device tree
- Driver anatomy
- Compilation/linking
- Installation/loading
- Testing
What are drivers?

Drivers implement a set of standardized functions used by the kernel and user programs to manage and interact with devices, real and imagined.

- There is no main()
- In kernel address space – linked only to kernel routines
- Interrupt-able, pre-emptable
Major driver types

- Character – a sequential stream of bytes
- Block – a buffer at a time, addressable
- Streams – a subset of character device using the streamio package
- Drivers can be both Character and Block
Devices

- Devices are initially identified by the firmware (OpenBoot, BIOS)
- Solaris boots and reads the device tree – attempts to associate a driver for each device
- Solaris descends each branch (nexus) attempting to discover devices
- Solaris can add new devices at any time
Device tree

- libdevinfo, prtconf, /devices represent the device tree
Device identification

- Vendor ID and Device ID
  - PCI, PCIx, PCIe
- `/etc/driver_aliases` provides a mapping from device name/compatible property to a driver name
- `/etc/name_to_major` lists major device numbers which correspond to a driver
- Each device in `/devices` has a major and minor device number and a name
  - Minor number managed by the driver
Device name, number, id

```
# ls -l /dev
/dev/dsk/c0t0d0s7 -> ../../devices/pci@1c,600000/scsi@2/sd@0,0:h

# ls -l /devices/pci@0,0:devctl
crw------- 1 root sys  86,255 date time /devices/pci@0,0:devctl

ID - Persistent and unique device identifier (dev_t)
    Use libdevid
```
Solaris kernel APIs

- Solaris has a stable device driver interface
- DDI - Device Driver Interface
- DKI - Device/Kernel Interface
- GLD – Generic LAN Driver
Anatomy of a simple driver

- Need some kernel specific header files
- Define some data structures
  - A handle
  - Device configuration operations
  - Character/block operations
  - Module load/unload operations
- Define some routines for the entry points
  - routines from cb_ops and dev_ops
Driver entry points

- Described in man pages 9e
- Loadable module interface (modlinkage, modldrv)
  - init, fini, info
- Auto configuration (dev_ops)
  - attach, detach, getinfo, probe, power
- Character
  - open, close, read, write, segmap, ioctl, chpoll
- Block
  - open, close, aread, awrite, strategy, print, etc
Driver entry points, cont.

- Memory mapped
  - devmap, devmap_access, devmap_map
- Streams
  - put, srv
- System
  - dump, quiesce
Kernel vs User device context

- Some entry points are callable by a user program: open, close, read, write, ioctls, etc
- Remember the drive is in kernel address space – do not attempt to access user space directly.
- Use ddi_copyin and ddi_copyout
Some header files

```c
#include <sys/modctl.h>   /* used by modlinkage, modldrv, _init, _info, */
    /* and _fini */
#include <sys/types.h>    /* used by open, close, read, write, prop_op, */
    /* and ddi_prop_op */
#include <sys/file.h>     /* used by open, close */
#include <sys/errno.h>    /* used by open, close, read, write */
#include <sys/open.h>     /* used by open, close, read, write */
#include <sys/cred.h>     /* used by open, close, read */
#include <sys/uio.h>      /* used by read */
#include <sys/stat.h>     /* defines S_IFCHR used by ddi_create_minor_node */
#include <sys/cmn_err.h>  /* used by all entry points for this driver */
#include <sys/ddi.h>      /* used by all entry points for this driver */
    /* also used by ddi_get_instance and */
    /* ddi_prop_op */
#include <sys/sunddi.h>   /* used by all entry points for this driver */
    /* also used by ddi_create_minor_node, */
    /* ddi_get_instance, and ddi_prop_op */
```
Writing Device Drivers
Oracle Corporation

Data Structures, dev_info_t

/*
 * The entire state of each xx device.
 */
typedef struct {
    dev_info_t *dip;
    /* my devinfo handle */
} xx_devstate_t;

/*
 * An opaque handle where our set of xx devices live
 */
static void *xx_state;
extern struct mod_ops mod_driverops;

static struct modldrv modldrv = { /* see modldrv(9s) */
    &mod_driverops,
    "xx driver v1.0",
    &xx_ops
};

static struct modlinkage modlinkage = { /* see modlinkage(9s) */
    MODREV_1,
    &modldrv,
    0
};
Data Structures, dev_ops

static struct dev_ops xx_ops = { /* see dev_ops(9s) */
    DEVO_REV,
    0,            /* refcnt */
    xx_getinfo,   /* identify */
    nulldev,      /* probe */
    xx_attach,    /* reset */
    xx_detach,    /* power */
    nodev
};
static struct cb_ops xx_cb_ops = { /* see cb_ops(9s) */
    xx_open,
    nulldev,    /* close */
    nodev,      /* strategy */
    nodev,      /* print */
    nodev,      /* dump */
    xx_read,
    xx_write,
    nodev,      /* ioctl */
    nodev,      /* devmap */
    nodev,      /* mmap */
    nodev,      /* segmap */
    nochpoll,   /* poll */
    ddi_prop_op,
    NULL,       /* streamtab */
    D_NEW | D_MP,
    CB_REV,
    nodev,      /* aread */
    nodev       /* awrite */
};

Data Structures, cb_ops
static struct cb_ops xx_cb_ops = { /* see cb_ops(9s) */
    xx_open,
    nulldev, /* close */
    nodev, /* strategy */
    nodev, /* print */
    nodev, /* dump */
    xx_read,
    xx_write,
    nodev, /* ioctl */
    nodev, /* devmap */
    nodev, /* mmap */
    nodev, /* segmap */
    nochpoll, /* poll */
    ddi_prop_op,
    NULL, /* streamtab */
    D_NEW | D_MP,
    CB_REV,
    nodev, /* aread */
    nodev /* awrite */
};
/* functions can be found in man pages 9e */
/* and header file /usr/include/sys/devops.h */

static int xx_open(dev_t *devp, int flag, int otyp, cred_t *cred);
static int xx_read(dev_t dev, struct uio *uiop, cred_t *credp);
static int xx_write(dev_t dev, struct uio *uiop, cred_t *credp);
static int xx_getinfo(dev_info_t *dip, ddi_info_cmd_t infocmd, void *arg,
                      void **result);
static int xx_attach(dev_info_t *dip, ddi_attach_cmd_t cmd);
static int xx_detach(dev_info_t *dip, ddi_detach_cmd_t cmd);
Routine, _init

```c
static void **xxstatep;
int
_init(void)
{
    int error;
    const int max_instance = 20; /* estimated max device instances */

    ddi_soft_state_init(&xxstatep, sizeof (struct xxstate), max_instance);
    error = mod_install(&xxmodlinkage);
    if (error != 0) {
        /* Cleanup after a failure */
        ddi_soft_state_fini(&xxstatep);
    }
    return (error);
}
```
Routine, _fini

```c
int
_fini(void)
{
    int error;
    error = mod_remove(&modlinkage);
    if (error != 0) {
        return (error);
    }
    /*
     * Cleanup resources allocated in _init()
     */
    ddi_soft_state_fini(&xxstatep);
    return (0);
}
```
Routine, _info

```c
int
_info(struct modinfo *modinfop)
{
    return (mod_info(&xxmodlinkage, modinfop));
}
```
Routine, attach

static int /* called for each device instance, see attach(9e) */
xx_attach(dev_info_t *dip, ddi_attach_cmd_t cmd)
{
    int instance;
    xx_devstate_t *rsp;
    switch (cmd) {
        case DDI_ATTACH:
            instance = ddi_get_instance(dip);
            if (ddi_soft_state_zalloc(xx_state, instance)
                != DDI_SUCCESS) {
                cmn_err(CE_CONT, "%s%d: can't allocate state\n", ddi_get_name(dip), instance);
                return (DDI_FAILURE);
Routine, attach  continued

} else
    rsp = ddi_get_soft_state(xx_state, instance);
    if (ddi_create_minor_node(dip, "xx", S_IFCHR,
        instance, DDI_PSEUDO, 0) == DDI_FAILURE) {
        ddi_remove_minor_node(dip, NULL);
        goto attach_failed;
    }
    rsp->dip = dip;
    ddi_report_dev(dip);
    return (DDI_SUCCESS);

default:
    return (DDI_FAILURE);

}  
attach_failed:
    (void) xx_detach(dip, DDI_DETACH);
    return (DDI_FAILURE);
Routine, detach

```c
static int /* see detach(9e) */
xx_detach(dev_info_t *dip, ddi_detach_cmd_t cmd)
{
    int instance;
    register xx_devstate_t *rsp;
    switch (cmd) {
        case DDI_DETACH:
            ddi_prop_remove_all(dip);
            instance = ddi_get_instance(dip);
            rsp = ddi_get_soft_state(xx_state, instance);
            ddi_remove_minor_node(dip, NULL);
            ddi_soft_state_free(xx_state, instance);
            return (DDI_SUCCESS);
        default:
            return (DDI_FAILURE);
    }
    return (DDI_FAIL);
}
```
/*ARGSUSED*/
static int /* called on open(2), see open(9e) */
xx_open(dev_t *devp, int flag, int otyp, cred_t *cred)
{
    cmn_err(CE_NOTE, "Inside xx_open");
    if (otyp != OTYP_BLK && otyp != OTYP_CHR)
        return (EINVAL);
    if (ddi_get_soft_state(xx_state, getminor(*devp)) == NULL)
        return (ENXIO);
}
Routine, read

static int
xx_read(dev_t dev, struct uio *uiop, cred_t *credp)
{
    int instance = getminor(dev);
    xx_devstate_t *rsp = ddi_get_soft_state(xx_state, instance);
    return(0);
}
/*ARGSUSED*/
static int
xx_write(dev_t dev, register struct uio *uiop, cred_t *credp)
{
    int instance = getminor(dev);
    xx_devstate_t *rsp = ddi_get_soft_state(xx_state, instance);
    return(0);
}
Compiling

For a 64-bit SPARC architecture using Sun Studio 12.1, use the -m64 option:

% cc -D_KERNEL -m64 -c mydriver.c
% ld -r -o mydriver mydriver.o

For a 64-bit x86 architecture using Sun Studio 12, use the -m64 option, and the -xmodel=kernel option:

% cc -D_KERNEL -m64 -xmodel=kernel -c mydriver.c
% ld -r -o mydriver mydriver.o
Compiling, continue

For a 64-bit SPARC architecture, use the following build commands for gcc:

```bash
% gcc -D_KERNEL -m64 -mcpu=v9 -mcmodel=medlow -fno-pic -mno-fpu \  
   -ffreestanding -nodefaultlibs -c mydriver.c
% ld -r -o mydriver mydriver.o
```

For a 64-bit x86 architecture, use the following build commands for gcc:

```bash
% gcc -D_KERNEL -m64 -mcmodel=kernel -mno-red-zone -ffreestanding \  
   -nodefaultlibs -c mydriver.c
% ld -r -o mydriver mydriver.o
```
Driver.conf

• If a device is not self identifying you may also need a driver.conf.

• It is located in the same directory along with the driver and is named xx.conf for a driver named xx.

• It contains any properties you might want to set property=value
Packaging

• IPS for Solaris 11 and later
• SVR4 packages for previous releases
Installation

# cp mydriver.conf /usr/kernel/drv
# add_drv mydriver

- _info(9E), _init(9E), and attach(9E) entry points are called in order.
- The driver is added to the /devices directory.
- The driver is the most recent module listed by modinfo(1M).
- The driver is the most recent module listed in the file /etc/name_to_major
Driver loading

modload(1M)
modunload(1M)
Testing

- Start simple, build incrementally
- cmn_err – similar to printf
- Use alternate boot environment
- kmdb, mdb - debuggers
- Watch for memory leaks, addressing errors
- Use ASSERT
To get help

• Join OPN

• Join Oracle Solaris Development Initiative
  http://www.oracle.com/partners/secure/support/oracle-solaris-development-494690.html

• Send me email:
  bill.knoche@oracle.com
To Learn More…

Download Solaris 11

Learn more about Solaris 11 – the documentation
http://docs.oracle.com/cd/E23824_01/

Writing Device Drivers
http://docs.oracle.com/cd/E23824_01/html/819-3196/

Device Driver Tutorial
To Learn More…

• man section 9e: DDI and DKI Driver Entry Points

• man section 9f: DDI and DKI Kernel Functions

• man section 9s: DDI and DKI Properties and Data Structures

• Writing FCode 3.x Programs
  http://download.oracle.com/docs/cd/E19253-01/806-1379-10/
To Learn More…

• PCI special Interest group
  http://www.pcisig.com/

• DLPI spec:
  http://pubs.opengroup.org/onlinepubs/009618899/toc.htm

• TPI spec:
  http://pubs.opengroup.org/onlinepubs/009618999/toc.htm

• OpenSolaris Device Driver Community
  http://hub.opensolaris.org/bin/view/Community+Group+device_drivers/
Oracle Solaris 11
Developer Webinar Series

Join us on April 24th for a complimentary webcast. In just one hour you will learn how to streamline your development process to maximize the performance of your application with Oracle Solaris 11. Access the Slides here.

- Learn why there is no more patching!
- Understand Oracle Solaris 11 cloud features
- Review how to maximize application reliability in Oracle Solaris 11
- Live Chat with Oracle Solaris 11 Engineers

Who should attend?

Application developers and administrators wanting a deep-dive on key features of Oracle Solaris 11 which you can exploit to make your applications superior to your competitors and easier to use.

Please note: (after registering) You will receive details on how to attend the meeting in a separate confirmation e-mail shortly before the webcast.

Next Webinar June 19th

Agenda - Next Sessions

- Click on event to register
- All webinars on Tuesday’s 9-10am PT (Event will support VOIP)

<table>
<thead>
<tr>
<th>Webinar Series Topic</th>
<th>Date</th>
<th>Speaker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing Oracle Solaris 11 Device Drivers</td>
<td>06-05-12 @ 9am PT</td>
<td>Bill Kroche (Principal Software Engineer)</td>
</tr>
<tr>
<td>Publishing IPS Packages</td>
<td>06-19-12 @ 9am PT</td>
<td>Eric Reid (Principal Software Engineer) and Brock Pytlík (Senior Software Engineer)</td>
</tr>
<tr>
<td>The World of IPS 2: Tools and Commands</td>
<td>07-17-12 @ 9am PT</td>
<td>Eric Reid (Principal Software Engineer) and Brock Pytlík (Senior Software Engineer)</td>
</tr>
<tr>
<td>Oracle Solaris Remote Lab</td>
<td>07-31-12 @ 9am PT</td>
<td>Ron Larson (Project Manager, ISV Engineering) and Angelo Riaadural (Principal Software Engineer)</td>
</tr>
</tbody>
</table>


Questions
Hardware and Software
Engineered to Work Together