Conreality Driver Development Kit (DDK) Manual

Release 2016-03-09

Arto Bendiken

March 09, 2016

This is the device driver development reference manual for the Conreality live-action wargame.

The latest version of this manual can be browsed online at ddk.conreality.org, and it is also available for download in the PDF and EPUB formats.

CHAPTER

ONE

INTRODUCTION

The Conreality Driver Development Kit (DDK) is a toolkit meant for hackers & makers adding support in Conreality for new hardware devices or porting the Conreality platform to new hardware architectures and development boards.

Conreality device drivers are userspace abstractions that are implemented primarily in the OCaml programming language, with minor bits of lower-level C/C++ language glue as needed.

The device drivers at this level do not generally deal with raw hardware itself, but rather low-level system interfaces exposed by the kernel. For example, the V4L2 video drivers for Linux are implemented using the `ioctl(2)` facility to communicate with the kernel-mode, device-specific drivers that implement the V4L2 interface.

CHAPTER

TWO

SOFTWARE ARCHITECTURE

This chapter explains where Conreality device drivers fit in with the rest of the system and application layers.

2.1 Process Model

(To be written.)

2.2 Execution Environment

Device drivers are executed by and as subprocesses of the Conreality daemon (`conreald`), which provides the following execution environment for the driver process:

- The process has three pre-opened file descriptors (0, 1, and 2), for the standard input, output, and error streams.
- The process's user and group privileges are adjusted (depending on configuration).
- The process's scheduling policy and priority are adjusted (depending on configuration).
- The process's CPU affinity mask is set so as to pin it to specific CPU cores (depending on configuration).

When executing drivers manually as standalone processes–in particular, during driver development–it may be necessary to emulate some aspects of this execution environment by using utilities such as *`su(1)`*, *`chrt(1)`*, *`taskset(1)`*, and *`nice(1)`* when executing the standalone driver process.

2.2.1 Process Privileges

(To be written.)

2.2.2 Process Scheduling

(To be written.)

2.2.3 Signal Handling

It is the responsibility of the driver process to set up signal handlers and correctly handle signals delivered by the kernel. The following signal-handling behaviors are normative for driver processes:

- `SIGHUP`: reload configuration.
- `SIGINT`: exit the process.
- `SIGPIPE`: exit the process.
- `SIGTERM`: exit the process.

2.2.4 Command-Line Arguments

Even though driver binaries are not intended to be directly executed by end users, they should nonetheless implement standard command-line argument processing so as to handle at least the following common options:

- `-h`, `--help`: show a help message and exit
- `-q`, `--quiet`: suppress superfluous output
- `-v`, `--verbose`: increase the verbosity level (could be given multiple times)
- `-d`, `--debug`: enable debugging output

Driver binaries may accept additional positional and/or optional arguments, but must be able to execute using sane defaults when none are given.

2.2.5 Error Logging

Drivers must log any warnings and errors to the system log using *`syslog(3)`* interfaces. They may also log additional informational messages and notices to the system log, using the appropriate lower priority levels.

Logging should be configured with the program identification `"conreality"`, the option flags `LOG_PID | LOG_CONS | LOG_NDELAY`, and the facility `LOG_DAEMON`.

The log mask shall be set as follows:

- Default: `LOG_NOTICE`
- `-q`, `--quiet`: `LOG_WARNING`
- `-v`, `--verbose`: `LOG_INFO`
- `-d`, `--debug`: `LOG_DEBUG`

2.2.6 Exit Codes

Driver processes must terminate using standard *`sysexits(3)`* exit codes, as per the following guidelines:

Code	Label	Summary
0	`EX_OK`	successful termination
64	`EX_USAGE`	command line usage error
65	`EX_DATAERR`	data format error
66	`EX_NOINPUT`	cannot open input
67	`EX_NOUSER`	addressee unknown
68	`EX_NOHOST`	host name unknown
69	`EX_UNAVAILABLE`	service unavailable
70	`EX_SOFTWARE`	internal software error
71	`EX_OSERR`	system error (e.g., can't fork)
72	`EX_OSFILE`	critical OS file missing
73	`EX_CANTCREAT`	can't create (user) output file
74	`EX_IOERR`	input/output error
75	`EX_TEMPFAIL`	temp failure; user is invited to retry
76	`EX_PROTOCOL`	remote error in protocol
77	`EX_NOPERM`	permission denied
78	`EX_CONFIG`	configuration error

CHAPTER

THREE

HARDWARE ABSTRACTIONS

3.1 Physical Memory Access

CHAPTER

FOUR

DEVICE REGISTRY

This chapter describes the device registry.

4.1 Examples

CHAPTER

FIVE

DEVICE INTERFACES

The Conreality DDK defines a number of abstract device interfaces for device drivers to implement. This chapter describes the major interfaces.

5.1 Video Camera

5.2 General-Purpose I/O (GPIO)

5.2.1 GPIO Modes

5.2.2 GPIO Chips

5.2.3 GPIO Pins

5.3 Pulse Width Modulation (PWM)

CHAPTER

SIX

CONREALITY DDK FOR OCAML

(To be written.)

CHAPTER

SEVEN

CONREALITY DDK FOR PYTHON

(To be written.)

7.1 Tutorial

(To be written.)

```
from conreality import ddk
```

7.1.1 Defining the Driver Event Loop

(To be written.)

```
class MyDriver(ddk.Driver):
  def init(self):
    pass # TODO

  def exit(self):
    pass # TODO

  def loop(self):
    while True:
      self.pause() # TODO
```

7.1.2 Parsing Command-Line Arguments

(To be written.)

```
class MyArgumentParser(ddk.ArgumentParser):
  def init(self):
    self.add_argument('id', nargs='?', default='default',
      help='the ID of the camera to attach to')
    self.add_argument('-r', '--fps', type=int, default=30,
      help='set the frames per second (FPS) rate (default: 30)')
```

```
if __name__ == '__main__':
  MyDriver(argparser=MyArgumentParser).run()
```

7.2 Reference

(To be written.)

7.2.1 Module conreality.ddk.camera

(To be written.)

7.2.2 Module conreality.ddk.driver

(To be written.)

class conreality.ddk.driver.**Driver**(*argv=sys.argv*, *argparser=ArgumentParser*)

Driver.**init**()

Driver.**exit**()

Driver.**loop**()

Driver.**log**(*priority*, *message*, **args*)

Return type None

Driver.**panic**(*message*, **args*)
LOG_EMERG

Driver.**alert**(*message*, **args*)
LOG_ALERT

Driver.**critical**(*message*, **args*)
LOG_CRIT

Driver.**error**(*message*, **args*)
LOG_ERR

Driver.**warning**(*message*, **args*)
LOG_WARNING

Driver.**notice**(*message*, **args*)
LOG_NOTICE

Driver.**info**(*message*, **args*)
LOG_INFO

Driver.**debug**(*message*, **args*)
LOG_DEBUG

class conreality.ddk.driver.**ArgumentParser**

class conreality.ddk.driver.**DataDirectory**(**path*)

exception conreality.ddk.driver.**DataDirectoryException**(*error*)

exception conreality.ddk.driver.**SignalException**(*signum*)

7.2.3 Module conreality.ddk.sysexits

(To be written.)

```
EX_OK          = 0  # successful termination
EX_USAGE       = 64 # command line usage error
EX_DATAERR     = 65 # data format error
EX_NOINPUT     = 66 # cannot open input
EX_NOUSER      = 67 # addressee unknown
EX_NOHOST      = 68 # host name unknown
EX_UNAVAILABLE = 69 # service unavailable
```

```
EX_SOFTWARE    = 70 # internal software error
EX_OSERR       = 71 # system error (e.g., can't fork)
EX_OSFILE      = 72 # critical OS file missing
EX_CANTCREAT   = 73 # can't create (user) output file
EX_IOERR       = 74 # input/output error
EX_TEMPFAIL    = 75 # temp failure; user is invited to retry
EX_PROTOCOL    = 76 # remote error in protocol
EX_NOPERM      = 77 # permission denied
EX_CONFIG      = 78 # configuration error
```

CHAPTER

EIGHT

FREQUENTLY ASKED QUESTIONS (FAQ)

(To be written.)

8.1 Technical Questions

8.1.1 What user privileges do the device drivers require?

Drivers generally require superuser privileges. The reason varies per driver: for example, the Sysfs drivers for GPIO manipulate virtual files in the `/sys/class/gpio` VFS hierarchy on Linux, which requires `root` privileges; meanwhile, the BCM283x drivers for GPIO require access to `/dev/mem` in order to perform *DMA* operations, which also requires `root`.

CHAPTER

NINE

REVISION HISTORY

9.1 2016

9.1.1 2016-03-09

(To be written.)

9.1.2 2016-02-29

- Added material about the driver process model and execution environment.

CHAPTER

TEN

GLOSSARY

API Application Programming Interface.

DDK (Device) Driver Development Kit.

DMA Direct Memory Access.

GPIO General-Purpose I/O.

I/O Input/Output.

PWM Pulse Width Modulation.

SDK Software Development Kit.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- search

C