EMBEDDED REAL-TIME AND
OPERATING SYSTEMS PROGRAM
HTTP://ERTOS.NICTA.COM.AU

Darbat Release 0.2 Notes

http://www.ertos.nicta.com.au/software/darbat

Charles Gray, Geoffrey Lee and Tom Birch

Document Number: ERTOS 10023:2006
Software Version: Release 0.2

Contact Details:

National ICT Australia Limited
Embedded, Real-Time and Operating Systems

Locked Bag 6016
The University of New South Wales
Sydney NSW 1466

email: **ertos@nicta.com.au**
web: **http://ertos.nicta.com.au/**

Contents

1 Release Information

Darbat, the L4/Darwin project is an experimental port of Darwin to the L4 microkernel to study the characteristics of a large-scale microkernel-based system. It includes a port of I/O Kit to L4, a modified libc to communicate with the Darbat server, and of course XNU. Many of the machine-dependent parts of XNU have been heavily modified (pmap, thread/task creation, etc) but much left unchanged (most of BSD, and large parts of OSFMK work without modification).

This release is a binary & hardware compatibility and performance demonstrator. This version of Darbat boots natively on Apple Intel EFI hardware. It can also load binary I/O Kit drivers from Intel Mac OS X 10.4.6 and user binary applications unmodified. Performance of the system call path is comparable to that of 10.4.6.

Darbat is based on publicly available Darwin and I/O Kit source code. Where incompatibilities were found between the public source code and release binaries, we have reverse-engineered a compatibility solution using available headers, documentation and lots of hard work. :)

Specifically, Darbat is based on the Darwin 8.2 source code (8C46), available from opendarwin.org. XNU is based on version 792.2.4. For compatibility, the dyld version used is 44.17, from the Darwin 8.6 release.

This release is intended to be just a demonstrator. There are still many bugs and stability problems. Don't expect to use Darbat for any real work. Be surprised if Darbat has more than a few minutes of up-time.

1.1 Features

The summary of interesting features for this release is as follows:

- Apple Intel Hardware

  Darbat boots natively on all classes of Apple Intel-based hardware (Mac mini, iMac, MacBook, MacBook Pro) released so far. There is no requirement for Boot Camp or any other legacy BIOS emulation code. To achieve this, EFI support has been added to the L4 microkernel.

- Kernel Extensions

  The I/O Kit component of Darbat now also supports loading of native binary kernel extensions. I/O Kit can dynamically link to extensions from Mac OS X.

  Currently, we only load enough drivers to get the USB keyboard and the internal hard disk working. With regards to other peripherals on the systems, we have tried to load the network and video card drivers, but they further work is required to utilise these devices.

- Binary Applications

  Our user-land library support is now compatible with release binaries from 10.4.6. Most text-mode binary applications off an Intel Mac will run. Applications known to run include gcc, as, make, mount, vim, emacs and python, among others. There are others which do not currently work due to some yet unimplemented features in Darbat. If in doubt, we encourage you to give it a spin and see how well it works.

- Mach-O Toolchain

  We build Darbat executables with the native toolchain as shipped with Xcode. This means Darbat builds with standard tools under OS X.

- Performance

  This release also demonstrates the performance of L4 IPC for the system call path. Table 1.1 shows cycle counts for various short syscalls. It is not possible to compare these numbers directly because Darbat and Mac OS X are based on different sources. This does demonstrate, however, that the Darbat approach is competitive, even at the "null" system call level.

Syscall	Darbat	Mac OS X 10.4.6
mach_msg (null args)	2393.86	2523.19
pid_for_task	4011.82	4327.65
flock (null args)	3515.86	3678.16
mach_msg (atomic send & recv)	5956.32	5897.97

Table 1.1: Cycle counts of various syscalls on a 1.83GHz iMac Core Duo (17")

1.2 Limitations

This release has a number of limitations which prevent it from being used for serious work yet.

- Text console

  Darbat only supports a keyboard and text console interface for now. There is no graphics support.

- Apple ICH7 Only

  Currently we only boot on release Apple Intel hardware with ICH7 chipsets. See Section 2.1 for a list of specific machines we have tested on. A computer from a different manufacturer with an ICH7 chipset most likely won't work. Computers we've not tested on from Apple also may not work. This limitation will be removed some time in the near future.

- ACPI

  Directly related to the ICH7 chipset is that we currently only have limited support for ACPI in L4 and I/O Kit. Because we do not have a full ACPI driver we directly program the ICH7 interrupt routing registers to suit our static assignment. Apple has a functional ACPI I/O Kit driver, and in theory, it should be possible to make use of it, however, such a configuration is not yet supported. We are currently working on support for this.

- Network

  Networking support in Darbat currently does not work. There is experimental infrastructure for it, and we have previously had a driver ported from Linux. We cannot release the driver due to GPL licensing problems and we have not yet updated the networking support code to work with proper drivers.

- Stability

  This release is definitely more about features and performance than stability. Sorry! :)

- SMP

  This release of L4 and Darbat are currently uni-processor only. Booting on a dual-processor machine works, however only one CPU is used.

- Large RAM sizes

  Darbat only statically allocates a fixed sized pool of physical memory. It won't make a difference to Darbat if you're booting on 256MB or 2GB of RAM.

- Signals

  BSD Signal support is very limited to some very specific cases of signal delivery. For example, Control-C does not work.

2 Building a Boot Image

This section how to build an image you can boot on real hardware.

2.1 Requirements

So far we have successfully booted darbat on the following machines:

- Intel Mac mini (Core Solo)
- Intel iMac (17”, 1.83GHz Core Duo)
- Intel iMac (20”, 2GHz Core Duo)
- MacBook (White, 1.83GHz Core Duo)
- MacBook Pro (15.4”, 1.83GHz Core Duo)

The 17” MacBook Pro video configuration is known not to work, however. Currently we have no ability to boot on either a simulated machine or inside a virtualised environment. This is because we have found no simulators or virtualisation tools that support EFI or the full CPU feature set used by Darwin and OS X. If you have one or know of one, please let us know.

To boot Darbat you need an Intel Mac machine to boot on, called the *victim*. You also need another machine, the *host*. The host can be either another Intel Mac or a x86 Linux machine with a FireWire port. The host is used to download Darbat onto the victim machine at boot time and to interact with the L4 kernel debugger.

2.2 Downloading and Building

If you would like to boot Darbat on real hardware there are two options for doing so. The quicker and easier way is to assemble a boot image from binary modules. If you would like to tinker with the code and see what makes it go, you can compile from the full sources.

Either way you will need binary files only available on an Apple Intel OS X computer. We cannot provide these files due to licensing issues, however, this should not be a problem since if you wish to test Darbat you should already have a copy of Mac OS X for Intel.

It is also possible to build the user-land libraries for Darbat, however this is incredibly difficult and we do not recommend it. Binaries are provided in the default build. Source is also available, however. If you would like more information about building Darbat and user libraries, contact the Darbat team [1].

2.2.1 Binary Modules

Assembling the binary modules is relatively straight-forward. The steps are as follows and should be done on an Intel 10.4.x Apple.

1. Get the binaries

   Download and unpack the binary distribution from the Darbat webpage.

   ```
   http://www.ertos.nicta.com.au/downloads/darbat/darbat-0.2-bin.tar.bz2
   ```

2. Setup the root disk

   Darbat uses a few default OS X binaries on the root disk, eg. launchd. To create a root disk, execute the `make_disk.sh` program. This will copy some binaries off the hard disk into a disk image.

3. Assemble the boot image

   Darbat also uses binary drivers which must be placed into the boot image. To do this, execute the `make_bootimg.sh` command.

4. Boot the image

   The binary tarball comes with the dbg1394 program in the tools directory. The host-side of booting will be as easy as:

   ```
   ~/darbat-0.2-bin % ./tools/dbg1394
   ```

   See Chapter 3 for more details.

2.2.2 Full Source

Building darbat from source will requires OS X 10.4.x and Xcode 2.3. Make sure you have 'gcc version 4.0.1 (Apple Computer, Inc. build 5341)' or later. Anything earlier will cause a reboot early in the boot process. You will also need the SCons build system [8]. We suggest that you use the latest pre-release, as we have discovered that the latest stable version is rather slow.

The steps to build from the source are as follows. Binary drivers are copied in during the build, however you must first configure a disk with applications. This must be done on an Intel OS X machine.

1. Download the binaries

   Download and unpack the binary distribution from the Darbat webpage.

   ```
   http://www.ertos.nicta.com.au/downloads/darbat/darbat-0.2-bin.tar.bz2
   ```

2. Setup the root disk

   To create a root disk, execute the `make_disk.sh` program from within the darbat-0.2-bin directory. This will copy some binaries off the hard disk into a disk image, disk.img.

3. Download the source

   Download and unpack the source distribution from the Darbat webpage.

   ```
   http://www.ertos.nicta.com.au/downloads/darbat/darbat-0.2-src.tar.bz2
   ```

4. Copy in the disk image

   Copy in the disk image generated in Step 2. eg.

```
~/darbat-0.2-src % cp ../darbat-0.2-bin/disk.img .
```

5. Build the source

   Run `scons` to build the file bootimg.dite

6. Boot the image

   The dbg1394 program comes as part of the darbat-0.2-bin package. You can do the host-side of the boot as follows:

   ```
   ~/darbat-0.2-src % ../darbat-0.2-bin/tools/dbg1394
   ```

   See Chapter 3 for more details.

2.3 Tools

2.3.1 dite

The *dite* [2] utility is used to bundle multiple files into a single ELF binary. The resulting *dite file* can be loaded to the correct addresses by a bootloader, which can jump to the registered entry point. It should be pointed out that Dite is merely a tool for us, a convenience that helps us merge the Mach-O binary executables and support files into a single file that can be conveniently downloaded over FireWire.

Darbat is loaded over FireWire from the *bootimg.dite*. This file contains the L4 microkernel, iguana, init and timer servers, I/O Kit Darbat, an HFS+ rootdisk image and a number of plists and binary files for I/O Kit.

The dite utility comes pre-built in the binary distribution. It is automatically built from source in the source code distribution.

2.3.2 dbg1394

The *dbg1394* program is an application for booting and remote debugging of a darbat machine over a FireWire bus. This program can be compiled on both Linux and OS X. There are pre-built binaries in the binary distribution. Dbg1394 also supports a gdb remote interface via a TCP socket. This can be used to inspect the victim machine's physical memory using gdb or some graphical utility like ddd or Xcode.

2.3.3 fwload

The *fwload* program is a bootloader component which works with dbg1394. Fwload acts as a bridge between the *elilo* loader and the L4 microkernel. Fwload is compiled from source on a Linux machine. The *fwload.dmg* disk image contains the files necessary to load fwload on an Intel Mac. Simply burn this image to CD or copy the contents to a USB thumb drive. Together dbg1394 and fwload provide a bi-directional serial communication over the FireWire bus.

3 Booting

As mentioned in Section 2.1, booting Darbat requires two machines, linked via a FireWire cable. Darbat bootstrap is started using elilo, the EFI version of the Linux loader *lilo*.

Elilo is used to load the fwload program. The fwload program parses some of the information provided by elilo, sets up information for the L4 boot and communicates with dbg1394.

Dbg1394 on the host machine loads the resulting ELF boot image, named bootimg.dite into physical memory on the victim machine. It then hands the entry point to fwload to jump to L4's startup code.

3.1 Configuring the Victim

The victim machine needs to boot with elilo and the fwload program. This can be done simply with a USB key or CD-R and the default Apple EFI bootloader. For more permanent development we use the very handy *rEFIt* [7] utility.

With the USB key or CD inserted you made with instruction in Section 2.3.3, holding the 'Option' key down during early boot will allow you to select the USB or CD device for boot.

3.2 Configuring the Host

On the host machine, simply execute the binary for your host operating system, either Linux or Intel OS X. On Linux ensure you have the *raw1394* module loaded. Execute the dbg1394 program from inside the directory with the bootimg.dite file.

3.3 A boot

Booting darbat is relatively straightforward. Connect the host and victim machines via a FireWire cable. Insert your boot CD or plug in your USB key to the victim. Reboot the victim and hold the 'Option' key down from early boot. This should enter a boot option menu allowing you to select the boot device. Depending on your firmware version you may have to wait a few sections for the CD option to appear. Select your boot device and press Enter. The screen should turn grey and stay there while fwload waits for a connection.

On the host machine, execute the dbg1394 program. This should perform a FireWire bus scan and locate the victim machine. If the victim is not found, make sure they are connected correctly via FireWire, and both your machines are 32-bit little endian.

Once fwload and dbg1394 has established a connection, fwload outputs some text to dbg1394 and then requests a boot. At the boot request dbg1394 loads the bootimg.dite file into the remote physical memory, also writing out information about what segments it is loading. Dbg1394 then tells fwload the entry point for L4 from the bootimg.dite file. Fwload then jumps to this address and starts L4.

When L4 starts, it once again initialises the FireWire controller and performs a bus reset, forcing dbg1394 to re-scan the bus. Once they have once again established a connection L4's kernel debugger input and output is over the FireWire connection.

You should see a lot of debug output from various OS components. Early during Darbat's startup it will prompt for the video output of the victim machine. Type the number of the type of victim machine into the dbg1394 program on the host machine and the boot should continue.

After much more output the display on the victim machine will clear and you should be greeted with single-user mode launchd startup and a bash prompt.

Some early sample output is as follows:

```
~/darbat-0.2-bin % ./tools/dbg1394
dbg1394 test starting
==========>  dbg1394 running  <==========

Searching for victim...
Talking to victim...


==========>  dumping output  <==========

elilo-l4 bootloader started
Test Code: 0x1234abcd
Setting up async recv
 Setting up RX desc...
Setting up AR DMA...
Async setup done

...
```

The prompt for you screen size is as follows:

```
Initial servers started

**** Please select your video mode:
1) Mac Mini
2) 17" iMac
3) 20" iMac
4) Macbook
5) Macbook Pro 15"
6) Macbook Pro 17"
Number:
```

Once the system is booted the screen should look as follows:

```
Singleuser boot -- fsck not done
Root device is mounted read-only

If you want to make modifications to files:
        /sbin/fsck -fy
        /sbin/mount -uw /

If you wish to boot the system, but stay in single user mode:
        sh /etc/rc
-sh-2.05b#
```

The boot disk has a script, *setup.sh* which you can source in order to set some environment variables and mount the local hard disk (read only). This might look as follows:

```
-sh-2.05b# source ./setup.sh
re-mounting / read-write
mounting internal disk as HFS
setting up environment variables
Done.
```

```
-sh-2.05b# ls
.DS_Store       Desktop DF      dev             sbin            usr
.Trashes        bin             etc             setup.sh        var
Desktop DB      build           mnt             tmp
```

The path is now set so you can execute binaries off the hard disk. Some programs may or may not work depending on your exact OS X revision.

If you are having problems booting, there may be a number of reasons why the system does not start correctly. In some instances the dbg1394 client will freeze. This may require a FireWire bus reset (i.e. unplug & replug the cable). Otherwise, something may have gone wrong during initialisation. Reboot the client and try again! Anecdotally it seems the faster machines (MacBook Pro, iMac) are more reliable booting darbat than the MacBook and Mac minis. Occasionally the hard-disk or USB keyboard do not initialise fully, even though you get to a shell prompt. For the keyboard, this can be fixed by unplugging and re-plugging the USB keyboard. For a laptop is is possible to attach an external keyboard. If in doubt, reboot.

4 System Architecture

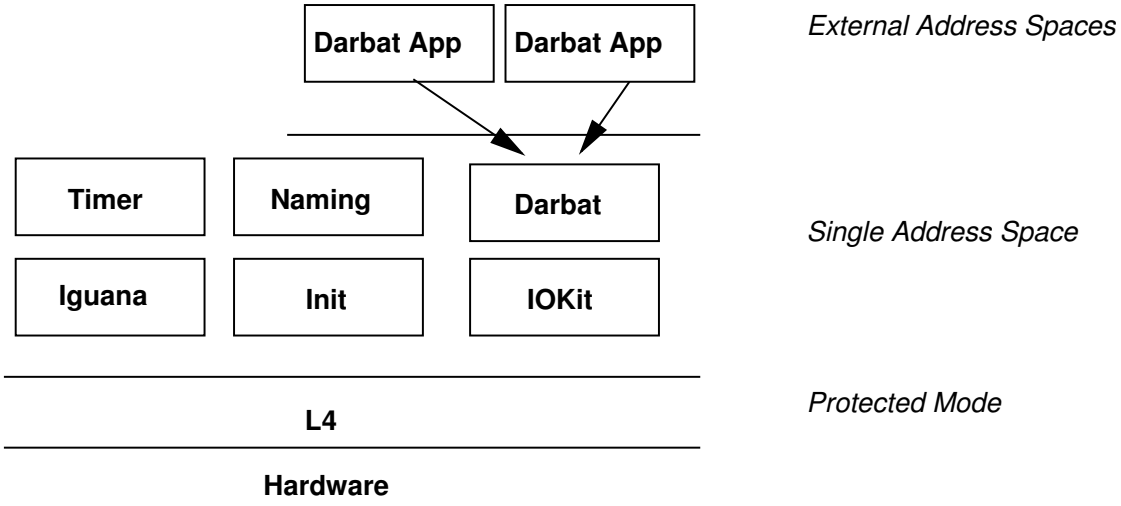

Figure 4.1: Darbat Architecture

The name *Darbat* is actually used to refer to both the darbat system as a whole, as well as the L4 port of the XNU component of Darwin.

Figure 4.1 illustrates the component layout in the Darbat system. Each component is isolated from one another using hardware memory protection for improved security. Components typically communicate using L4 IPC and shared memory.

The system can be logically divided into two sections. The first section is the Iguana OS [3], including the Iguana server, the Iguana init program, the naming server and the timer driver. The other section is Darbat and I/O Kit.

4.1 Components

A brief description of each component is as follows:

4.1.1 L4

L4 [6] is the secure, second-generation microkernel at the core of Darbat's performance and security. A full discussion of L4 is far beyond the scope of this document, and there are already documents which discuss it. Our version of L4 is a variant of L4 Pistachio.

In Darbat, L4 provides fast messaging between applications and secure sharing of memory.

4.1.2 Iguana

Iguana is an OS personality for L4. Iguana runs as the *privileged task* and manages the policy in the system for resource allocation. A full discussion of Iguana is also beyond the scope of this document.

Iguana provides a *single address space* for native L4 applications. While these applications share a single address space, that is, a single mapping from virtual to physical addresses, it does not mean there is no protection. Each Iguana application executes in its own hardware address space. Therefore, each Iguana application can have different, or no permissions, on each page of virtual memory.

4.1.3 Init

The *init* program is responsible for starting Iguana applications.

4.1.4 Naming

The *naming* program provides a shared namespace through which Iguana applications can publish and discover services published by other applications.

4.1.5 Timer

The timer program is a hardware timer driver for the system. Iguana programs needing timers and wake-ups communicate with the timer application via L4 IPC. The Darbat server and I/O Kit both use the timer driver.

On x86 the timer driver uses the 8254 timer chip and can provide periodic and one-shot timers.

4.1.6 I/O Kit

I/O Kit is the device driver subsystem for the Darbat system. It is, however, independent of Darbat and can operate without it. For example, it has been used with Wombat [5], our L4/Linux port. I/O Kit is predominantly C++ and runs purely in user-mode in Darbat.

I/O Kit is a framework suited for device drivers. It supports dynamic loading of binary Apple OS X device drivers. As well as device drivers and device families, I/O Kit provides enough kernel support code to manage threads, memory, interrupts and other abstractions device drivers need.

4.1.7 Darbat

The Darbat server is the XNU component of Darwin running de-privileged as an L4 task. Darbat manages external address spaces for darbat user-land, and communicates with I/O Kit for device driver access. Operation of I/O Kit and Darbat server is completely isolated. A failure in either should not extensively affect the other.

Darbat, being XNU, includes both the Mach and BSD components of Darwin. Darbat handles system call requests from user-land applications and manages their address spaces.

4.1.8 Darbat Applications

Darbat applications are regular Darwin applications. Applications such as launchd, bash, ls, etc. are binary compatible, however they rely on a modified libc (libSystem) and dynamic linker to support using L4 for system calls.

4.2 Bootstrap

There are a lot of steps involved in the bringing up of the whole Darbat system. This section provides a brief overview.

When L4 starts it is in a 32-bit direct-mapped mode inherited from EFI. L4 configures some bootstrap page-tables (also direct-mapped). L4 then performs a PCI bus scan in order to try and find a FireWire device for the kernel debugger. Once the FireWire device has been found and initialised, L4 can set itself up.

The first user application to start is Iguana, the privileged task. The details of the Iguana binary are stored in L4's *kernel interface page* when the boot image is created.

Iguana starts and configures itself and discovers resources such as physical memory. Iguana then starts the init program based on information in the *bootinfo* records. Init executes a compile-time generated script to startup the timer, naming, Darbat and I/O Kit.

The I/O Kitstartup sequence is no different from how it would be started in the traditional Mach-based I/O Kit. The I/O Kitis initialised by calling the StartIOKit() function in the IOStartIOKit.cpp. This is normally done by the Platform Expert. In the L4-based version of the I/O Kit, this is done from within the XNU emulation library, which contains a stripped-down version of the Platform Expert.

Darbat startup is similar to that of normal XNU. Mach is bootstrapped and started, then BSD is initialised. During BSD initialisation the rootdisk built into the dite image is mounted. Darbat then locates and begins talking to I/O Kit to locate disk and network resources.

Darbat finishes startup by starting a user-land process, launchd. launchd is run as any other process and dynamically linked to libSystem. launchd ultimately forks and execs a bash prompt for user interaction.

4.3 Darbat Server Internals

Darbat provides the XNU functionality of the Darwin kernel, isolated into its own hardware address space running as a user-level server. A number of modifications have been made to XNU to support efficient execution on L4. The main abstractions XNU is concerned with are virtual and physical memory, tasks and threads and handling system calls and user-threads. Darbat also has a minimal *platform expert*.

The virtual memory system of XNU is modified at the pmap layer. Darbat provides pagetable and physical-to-virtual lookup tables. These are hooked into the appropriate L4 calls to manage address spaces of userland applications and Darbat itself.

The various sections in the Darbat address space are shown in Figure 4.2. This address space layout is currently only experimental and will change significantly.

The physical section is an allocated region backed by real physical memory. This section is used to back both user-land application address spaces and Darbat's own virtual section. The darbat text and data sections are in a different section from physical and virtual because they are inherited from where they bootloader placed them.

The I/O Kit shared region is a buffer shared with the I/O Kit server. This model allows shared memory communication to reduce copying overhead. The ramdisk is also provided by the bootloader and gives Darbat its root filesystem.

The Darbat internal thread and message structure is shown in Figure 4.3. Each physical CPU in the system is represented by two L4 threads, although currently only a single CPU is supported. The dashed lines represent IPC communication.

All proper mach kernel threads are multiplexed onto the *darbat main thread*. The main thread switches stacks itself to provide multi-threaded execution. The *darbat exception thread* is a higher priority thread used for generating clock notifications to the darbat main thread.

The Darbat clock comes from a periodic timer in the Timer server. This notification is delivered (currently synchronously) to the darbat main thread. The darbat main thread also receives messages from user-land

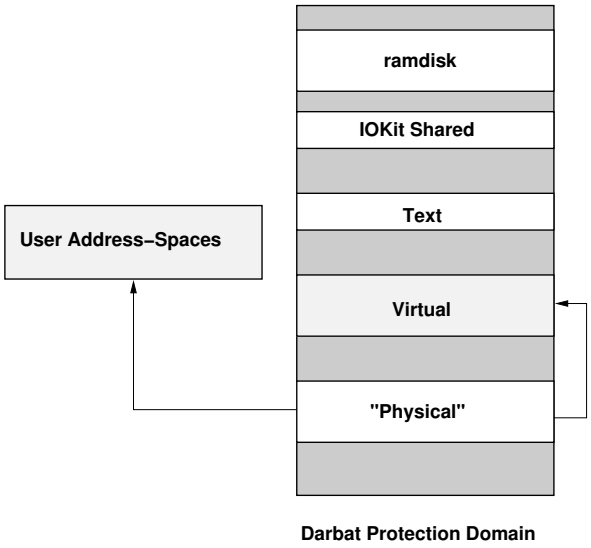

Figure 4.2: Darbat Address Space

applications requesting system calls and handling replies. I/O Kit interaction is also handled through the darbat main thread.

For the most part the XNU kernel is unmodified. Mach IPC is still used on top of the L4 IPC mechanisms. Because Darbat implements the pmap interface, all Mach VM operations are supported.

4.4 Darbat user-land

Darbat provides a library compatible user-land to OS X. With a modified dyld and libSystem native binaries can execute on Darbat. The modifications change the system call mechanism to use L4 IPC instead of direct system call instructions. This version of darbat marshals system call arguments into L4 message registers to avoid unnecessary *copyin* calls.

4.5 I/O Kit Internals

I/O Kit in the Darbat system provides a stand-alone device driver subsystem on L4. It is compatible and can binary load I/O Kitkernel extensions from Apple Intel OS X computers. Interfacing with other subsystems is done by means of having a thread of execution within the I/O Kit to handle and service such requests. This interfacing layer is still somewhat primitive and we plan to improve this soon.

I/O Kit threads all execute as proper L4 kernel scheduled threads. They do not switch their own stacks. Function calls or data references out into Mach or BSD are handled by the XNU emulation layer library. This emulation library does not aim to emulate everything that is currently in Mach and BSD. The scope of what is emulated will likely be revisited at regular intervals.

All I/O Kit extensions currently execute in the single I/O Kit server and are not separated from each other. More details about the structure of I/O Kit are described by Geoffrey Lee [4].

4.5.1 KEXT Interface

Driver support is provided by the I/O Kit. Darbat currently only makes use of the ICH7 disk controller (for mounting the internal hard disk) and the USB controller (for keyboard input). Others will probably load, and have loaded before, but in general will not function yet without further work. The current implementation does not resolve KPI dependencies properly and these must be removed from the plists.

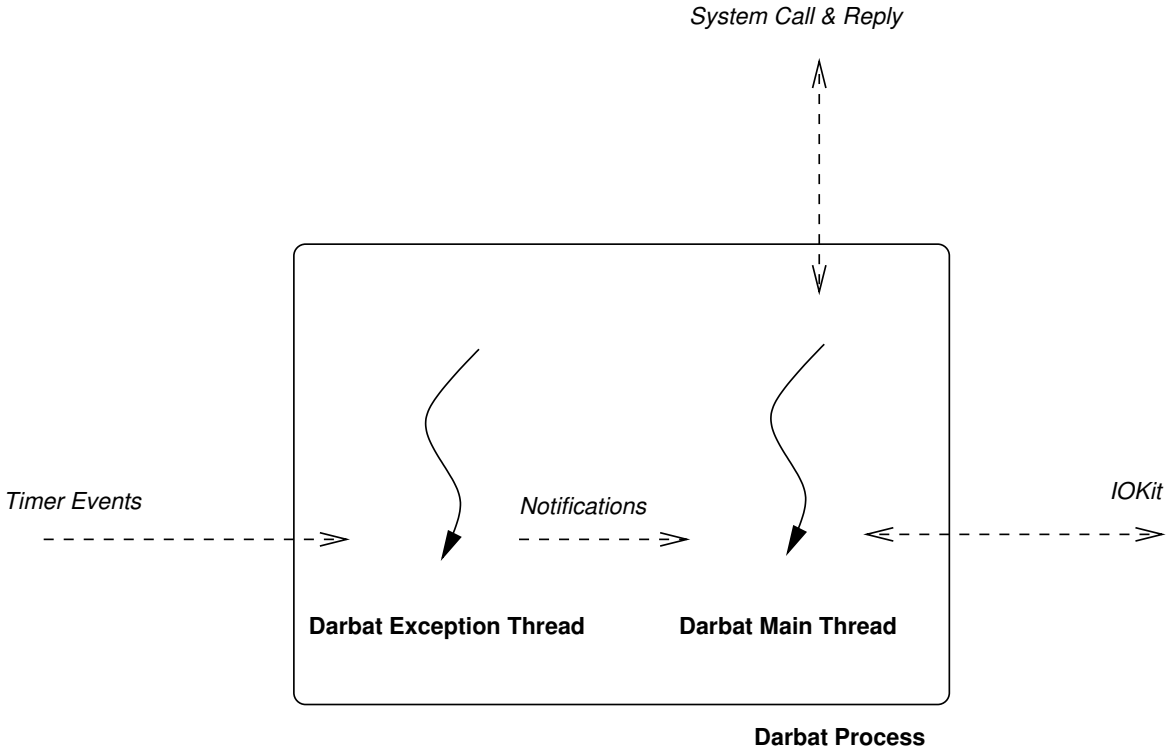

Figure 4.3: Darbat Threading and Messaging

4.6 I/O Kit/Darbat Interface

Darbat and I/O Kit currently have a very primitive interface for making use of various devices. These interfaces are just to get things working and will be replaced with a proper model in the future.

Console input is provided by I/O Kit in the form of a simple L4 IPC interface. Characters from the keyboard cause a notification IPC to be sent to Darbat where they are pushed into the bottom layers of the terminal code with *cons_cinput*. User applications can read the console by opening the `/dev/console` device as in normal Darwin.

The disk I/O interface provides a simple blocking read/write interface. At startup I/O Kit notifies Darbat of the disk geometry of the root partition on the disk device. Darbat internally creates a block device for talking to I/O Kit. This device is registered as `/dev/ikd0` (I/O Kit disk 0). Actual disk data is transferred via the I/O shared region of memory.

The network interface is somewhat more advanced, however it is currently disabled. The network interface uses a region of the I/O Shared segment to provide asynchronous transmit and receive queues. On each side of these queues (Darbat and I/O Kit), the actual data is moved into and out of mbufs.

4.7 Source-code layout

The main Darbat source release has a large number of files in it for building a large number of libraries etc. This section briefly summarises some of the more interesting directories to start looking at code.

- `pistachio`

  The pistachio directory contains the source code for the L4 microkernel.

- `iguana/iokit`

  This directory contains the entry point and main loop code for the I/O Kit server. The bulk of the code is in libraries, however.

- `libs/xnuglue`

  Xnuglue is the support code for running the necessary Mach infrastructure for I/O Kit on L4.

- `darbat`

  This directory contains the the XNU component of darwin with darbat changes. Notable sub-directories include `igcompat` and `osfmk/l4` for darbat support code.

Bibliography

[1] Darbat webpage. http://www.ertos.nicta.com.au/software/darbat.

[2] Dite utility. http://www.ertos.nicta.com.au/software/kenge/dite/devel/overview.pml.

[3] Iguana OS personality. http://www.ertos.nicta.com.au/software/kenge/iguana-project/latest/.

[4] Geoffrey Lee. I/O kit drivers for L4. BE thesis, University of New South Wales, November 2005.

[5] Ben Leslie, Carl van Schaik, and Gernot Heiser. Wombat: A portable user-mode Linux for embedded systems. In *Linux.conf.au*, Canberra, April 2005.

[6] Jochen Liedtke. On μ-kernel construction. In *Symposium on Operating Systems*, pages 237–250, Copper Mountain, CO, USA, December 1995.

[7] rEFIt bootloader. http://refit.sourceforge.net/.

[8] SCons build system. http://www.scons.org/.