%loch: A Hardware Control Vane

## Rationale

External device interfaces will require an Urbit ship to know about and speak several device languages, much as HTTP and UDP must be spoken by %ames and the runtime. From Internet-of-Things connected devices, to cryptographic hardware keys and dongles, to embedded devices (including GPGPUs), Urbit ships will need to be able to communicate via the runtime with external devices that may not present an HTTP (%eyre/%iris) or jammed-noun (%khan) application interface. POSIX-compliant host OSs provide device access via file handles and device drivers: device files, sockets, FIFO files, and so forth. This proposal recommends the creation of an explicit hardware control vane for Urbit to coordinate device operations. The runtime side of %loch acts as a hardware abstraction layer, which the Urbit/Hoon side complements by exposing an interface for the various expected communications types to devices.

## Overview

This project (%loch) aims to produce a hardware control vane which yields fundamental functionality for device-driven applications. One or more specific applications should be identified and targeted as benchmarks of task completion.

## Requirements

The Urbit/Hoon vane exposes an interface for registering and communicating with an external device using the host OS system calls such as POSIX IOCTL. In essence, a desk can register device drivers included in a /dev folder on the desk polled by %l. The %gall agent speaks to %loch while the bridged runtime speaks the communications protocol via IOCTL to the physical or logical device.

For instance, given a USB-attached device, a %gall agent pokes %loch to pass communications to a Hoon driver (under the aegis of %loch) which then communicates via IOCTL to the actual device (handling reads/writes, baud rate, and so forth). Any operations taking place with respect to external devices are still specified in Hoon.

Aspects of device communication are similar to networking, in that a device (like a remote server) may or may not be available at any given moment. There are precedents for synchronizing Earth-side operational environments to Mars, such as %ames, %clay, and %eyre. Storing device IDs in the pier is one way of solving the ongoing device connectivity challenge.

%loch (or rather its runtime complement) queries hardware characteristics in a standardized way, much as OpenCL exposes hardware specifications for heterogeneous device computing.

In particular, one challenge is to decide what the relationship of the runtime to device drivers and libraries should be. This problem will be less like jetting and more like networking, since devices may become unavailable or change specifications over time. We envision %loch providing some standard device templates for registering possible attached devices.

We want to avoid turning Hoon into C with explicit memory management, and the line we recommend is no pointers in Hoon—the runtime and IOCTL can handle device memory operations. We also want Earth problems to remain Earth problems, such as user/group membership for device access.

Arvo and Vere Interface

* Loch heartbeat to vere which requests a list of available devices
  + Vane then checks any incoming pokes for validity
  + Loch Should be able to manually request a heartbeat
  + Interval heartbeat can be set at command-line
    - Default is 5 minutes with possibly increasing when no devices are reported (similar to wifi spec)
  + ALTERNATIVE: Vere sends device changes to arvo (This works better and simpler)
* Commands from arvo to vere are %read %writ this reads and writes to an ioctl device
  + The noun that gets passed (either direction) is an atom with a magic-number header (i.e 0xDEADBEEF) to help it preserve leading zeros
* Vere will handle the device mounting and opening form host OS
  + If a read or writ command is sent it checks if device is open and opens it
  + On shutdown it should close the device(s)
  + Unit unit unit response
    - Failure to locate device
    - Failure to Open device
    - Failure from device
  + %error respond
    - %find - locate failure
    - %open - open failure
    - %vice - device failure
* Moves
  + Agent -> Loch
    - [%writ %i2c0 some\_noun] - pass
  + Loch -> Arvo
    - UNKNOWN solved problem though needs r&d
  + Arvo -> Vere
    - [ ] - Unix Move - UNKNOWN
  + Vere -> Arvo
    - [ ] - Unix Mode - UNKNOWN
  + Arvo -> Loch
    - Gift
  + Loch -> Agent
    - Gift

Drivers will need to be implemented in Hoon. This has the disadvantage of requiring driver implementation, but the advantage of preventing the possibility of pressure for binary blobs to be included in the runtime in the future.

* How does the runtime know to pack a noun into the correct structure for the device? We could pass a cell [%bits noun] to the runtime and let it pack the values into a device-appropriate struct.
* How should event loops be handled, e.g. for device control (as opposed to device reads)?
* How should device matching be handled by the runtime? (One can think of analogues on globbing and in jetting.)
* How can device permissions be handled with respect to the host OS?

## Applications

The most significant device protocols we have considered are:

1. UART (USB)
2. I2C
3. PCIe
4. Bluetooth

Third-party libraries considered for inclusion must have a compatible license for the Urbit kernel. This argues in favor of an IOCTL-based approach that doesn’t require compiling third-party libraries into the Urbit runtime.

Potential device applications include:

1. Internet-of-Things connected devices on various firmware.
2. Security tokens for secure applications (as Yubikeys or one-time tokens).
3. Hardware wallets.
4. GPGPU devices

Without device registration, no device is accessible to the runtime.

A target sample application to validate this process would be controlling a serial device.

## Team

* Amadeo Bellotti
* Neal Davis

## To-Do

1. Figure out pattern for device registration.
2. Build interface spec for both sides.
3. Construct vane.