Chronos OS: Phases 11 & 12 Report

Hardware Enumeration and Network Driver Implementation

Development Log

January 20, 2026

Abstract

To transform Chronos into a daily-usable operating system, network connectivity is required. Phases 11 and 12 focused on discovering hardware via the PCI bus and implementing a device driver for the Realtek RTL8139 Fast Ethernet controller. Key achievements include enabling Bus Mastering, configuring Direct Memory Access (DMA) buffers, resolving Virtual-to-Physical memory mapping conflicts, and successfully transmitting and receiving raw Ethernet frames.

Contents

1 Phase 11: PCI Enumeration

1.1 The PCI Bus Architecture

The Peripheral Component Interconnect (PCI) bus allows the CPU to discover connected hardware dynamically. Communication is performed via two 32-bit I/O ports:

- **0xCF8 (CONFIG_ADDRESS):** Selects the Bus, Device, Function, and Register to access.
- **0xCFC (CONFIG_DATA):** Reads or writes data to the selected register.

1.2 Implementation

We implemented a brute-force scan of all 256 buses and 32 devices per bus.

```
let address = 0x80000000
            | ((bus as u32) << 16)
            | ((slot as u32) << 11)
            | ((func as u32) << 8)
            | ((offset as u32) & 0xFC);

address_port.write(address);
let data = data_port.read();
```

Listing 1: PCI Configuration Reading

1.3 Result

The scan successfully identified a device with **Vendor ID 0x10EC** (Realtek) and **Device ID 0x8139**, confirming the presence of the emulated Network Card.

2 Phase 12: The RTL8139 Driver

2.1 Initialization Sequence

The RTL8139 is a DMA-based device. To initialize it, we performed the following steps:

1. **Enable Bus Mastering:** Modified the PCI Command Register (Offset 0x04) to allow the device to write to RAM.
2. **Power On:** Wrote 0x00 to the Command Register (Offset 0x37).
3. **Software Reset:** Wrote 0x10 to the Command Register and polled until the bit cleared.
4. **Unlock Config:** Configured the Receive Configuration Register (RCR) to Promiscuous Mode (Accept All Packets).

3 The Memory Management Challenge

3.1 Virtual vs. Physical Addresses

A critical issue was encountered when setting up the Receive Buffer (RBSTART). The Kernel uses Virtual Addresses (e.g., `0xFFFF...`), but the Network Card requires Physical Addresses (e.g., `0x0020...`).

Initially, we allocated a buffer on the Heap using `Box::new()`. However, calculating the physical address of a Heap allocation is complex because the Heap resides in the Kernel's `.bss` section, which has a different offset than standard RAM.

3.2 The Fixed-Address Solution

To ensure stability, we bypassed the Heap allocator for the DMA buffers. We selected a safe region of Physical RAM at the 32MB mark (`0x02000000`).

```
const RX_BUFFER_PHYS: u32 = 0x0200_0000;

// Calculate Virtual Address for the CPU to read
let hhdm = state::HHDM_OFFSET.load(Ordering::Relaxed);
let rx_virt = hhdm + (RX_BUFFER_PHYS as u64);

// Send Physical Address to the Network Card
let mut rbstart_port = Port::<u32>::new(io_base + REG_RBSTART);
rbstart_port.write(RX_BUFFER_PHYS);
```

Listing 2: DMA Buffer Setup

4 Packet Transmission (TX)

4.1 Descriptor Rings

The RTL8139 uses 4 Transmit Descriptors (TSD0-TSD3). We implemented a round-robin scheduler to rotate through these descriptors.

To verify connectivity, we constructed a raw **ARP Request** packet.

- **Target:** 10.0.2.2 (The QEMU Gateway).
- **Payload:** "Who has 10.0.2.2? Tell Chronos."

5 Packet Reception (RX)

5.1 Polling Strategy

While interrupts were enabled on the card, we utilized a polling strategy for the initial driver to simplify debugging. We monitor the first byte of the Receive Buffer.

1. The CPU reads `RX_BUFFER_VIRT`.
2. If the value is non-zero, a packet has arrived.
3. The driver extracts the packet and manually clears the memory to "re-arm" the detector.

6 Verification Results

The driver was tested in QEMU with User Networking enabled.

1. **TX Success:** The Transmit Status Register returned `OK`.
2. **RX Success:** After broadcasting an ARP Request, the Receive Buffer populated with raw data.

```
[NET] Sending ARP Request...
[TX] Status OK.
[NET] RAM CHANGED! PACKET DETECTED!
RAW DATA: .. .. 52 54 00 12 34 56 .. ..
```

7 Conclusion

Chronos OS has successfully established a Physical Layer connection. The kernel can now control the network card to send and receive arbitrary data. The next phase will involve implementing a software Network Stack to parse these raw bytes into Ethernet, ARP, and IP protocols.