MnemOS

an operating system for
building small computers

James Munns
OneVariable UG
contact@onevariable.com

RIOT Summit 2023
2023-09-18

MnemOS is a *hobby* operating system.

The name comes from *Mnemosyne*

The Greek goddess of memory, and mother of the nine muses

Antique mosaic of Mnemosyne, National Archaeological Museum of Tarragona

mnemos

You *will* be able to poke holes in the claims and details.

- "Well actually"
- "What about?"
- "Why didn't you?"
- "Why not..."

Don't overthink it.

It doesn't have to *make sense*, it has to *be fun*.

But first a little context...

Embedded systems typically come in one of two flavors today:

The "big" option: Embedded Linux

- Yocto
- Buildroot
- OpenWRT
- Raspbian

The "small" option: Bare Metal/RTOS

- Vendor HALs
- FreeRTOS
- Zephyr
- RIOT-OS
- Embassy
- RTIC
- Arduino
- Open Source HALs
- Hundreds of other options

Linux is better for some stuff...

- Networking
- Filesystems
- Portability
- Existing tools + SW
- Hiring developers
- Orchestration
- Isolation or Containerization
- Graphical interfaces

Bare Metal/RTOS is better for some stuff...

- Hard real-time
- MCUs
- Low power
- Custom hardware and drivers
- Auditability
- Customization

You *could* usually do any of this with either choice...

It just might suck.

I do **a lot** of projects "in the between"

I often need:

- Networking
- Observability
- Filesystems
- Custom drivers + hardware
- Soft real-time
- Low power

MnemOS is an operating system for the **Liminal Space** between other options.

MnemOS is designed for *Small Computers*.

- Network Connected
- User interfaces
- Dynamic applications
- Limited power and performance

It prioritizes my favorite things

- Willing to require non-minimal HW
- Must play nice with other computers
- Soft real-time is usually enough
- Relatively portable
- Relatively flexible

It is willing to steal any good idea from the last 55 years of computer science.

- Embedded systems
- Language design
- Backend servers
- Desktop OSs
- Server OSs

So what did we steal?

Or: What design choices did we make?

Async-first operational model

aka: "co-operative multitasking"

Stolen from:

- async/await in Rust
- Asyncio in Python
- NodeJS
- NGINX
- Protothreads

Why async?

- Hardware is usually event-driven
- Rarely CPU bound
- Smaller systems often only have one core
- Very power/resource friendly
- Userspace still preemptive, *kinda*

Message Passing as the primary interface style

Stolen from:

- Erlang
- Smalltalk
- Distributed Systems

Why Message Passing?

- Fewer "ABI" concerns
- Channels, Queues, etc. play **great** with async
- Messages can come from/go to:
  - Within the kernel
  - Userspace
  - External systems
- [De]serialization can be *very* fast.

io_uring or iocp for userspace API

Stolen from:

- Linux
- Windows

Why io_uring or iocp?

- Better fit for async, vs traditional syscalls
- Messages can easily be serialized to a ring buffer
- You only need **one** real system call: "yield".

Flexible Kernel Setups

aka: "make the OS a library, not a distribution"

Similar in effect to:

- BSD Rump Kernels
- C++ IncludeOS

Why Flexible Kernel Setups?

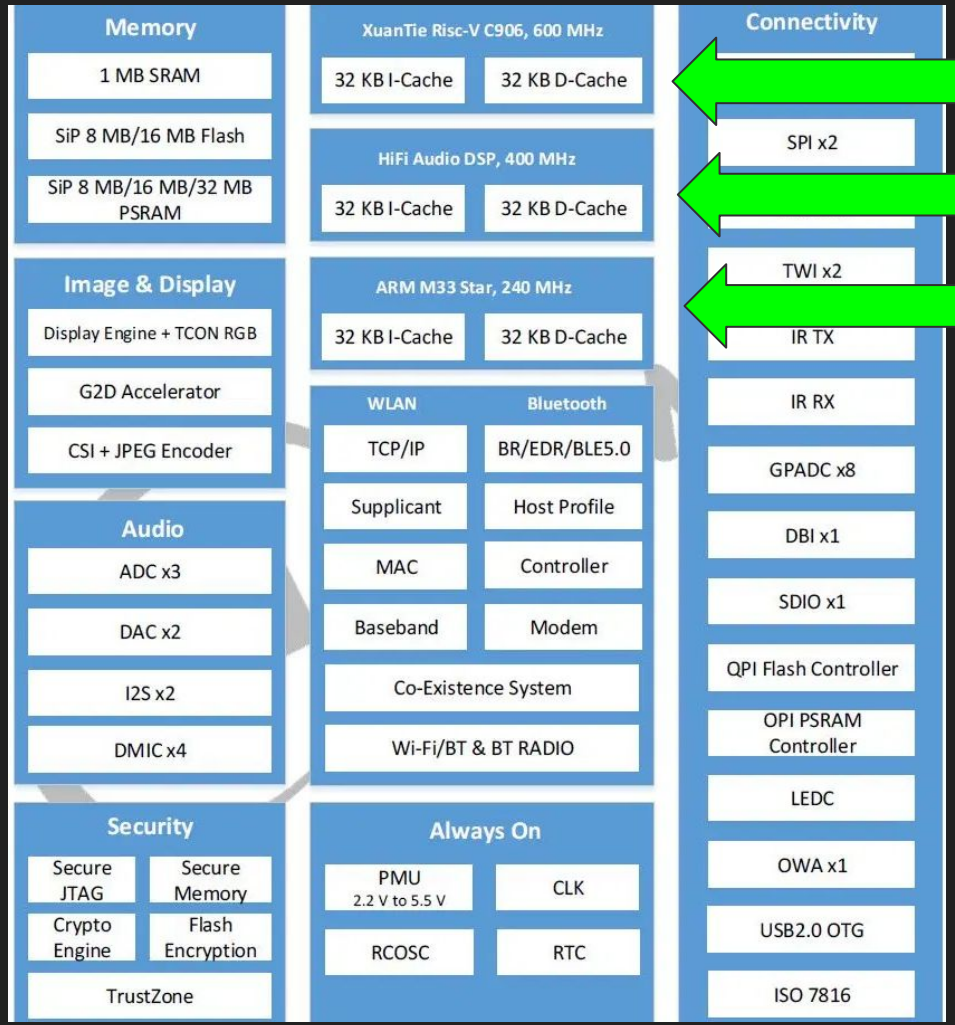
- Make it easier to run anywhere
- Let the integrator make "last mile" OS choices (with real code!)
- Easy to run on a 32-bit MCU or 64-bit CPU
- Reuse whatever HAL you already have today

Distributed-first system design

Stolen from:

- Backend Servers
- Transputers
- Erlang (again)

Many CPUs in
one package...



RISC-V 64

Xtensa LX7

Cortex-M

Allwinner R128 Block Diagram

Many CPUs on one board...

- Main CPU
- GPU
- Wifi controller
- Eth controller
- SSD/HDD controller

Why distributed first design?

- A "computer" is really lots of littler computers
- We should treat it like a real network
- What if we could run the same kernel and comms stack everywhere?

Okay but what actually works today?

Kernel Basics

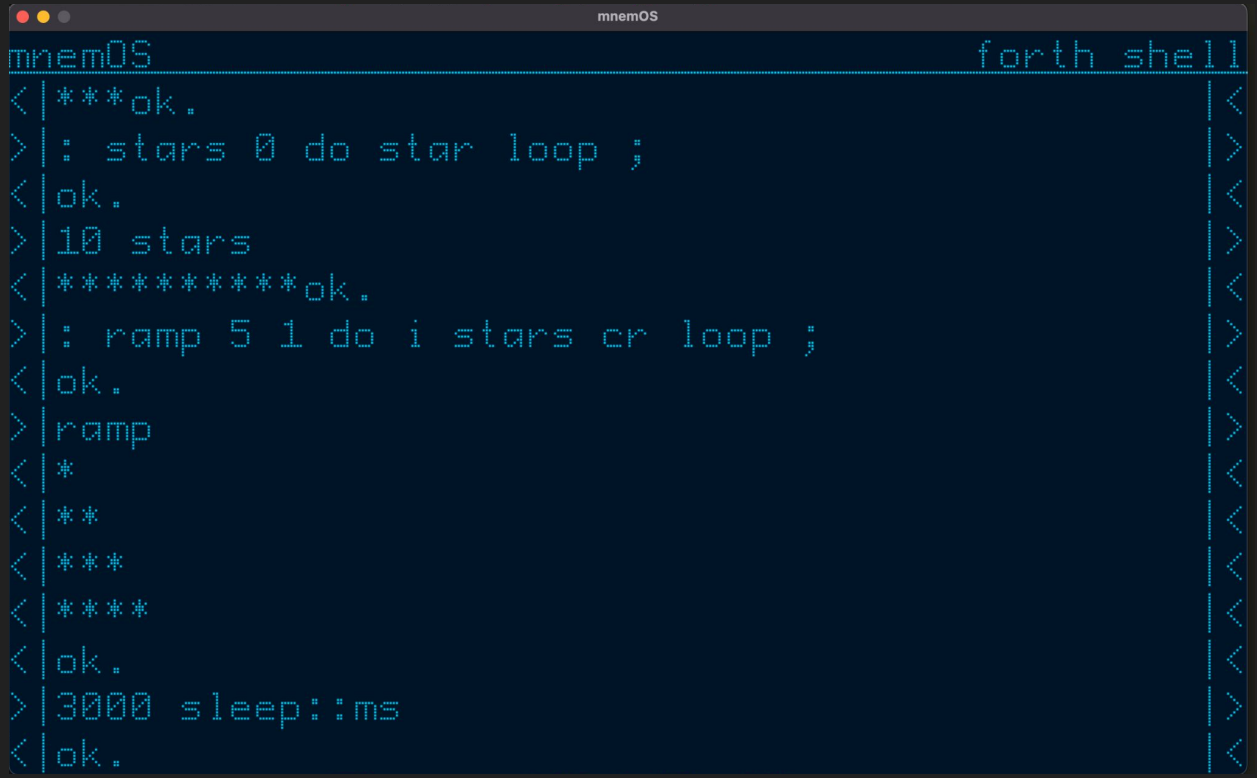
- Memory Allocation
- Kernel Async Scheduler
- Message Passing
- Service Discovery

```rust
/// Register the D1 Serial Port Service
pub async fn register(
    k: &'static Kernel,
    cap_in: usize,
    cap_out: usize,
    tx_channel: Channel,
) -> Result<(), registry::RegistrationError> {
    // Allocate serial FIFO
    let (fifo_a, fifo_b) = new_bidi_channel(cap_in, cap_out).await;

    // Register a socket with the OS to handle SimpleSerialService
    // messages
    let reqs = k
        .registry()
        .bind_konly::<SimpleSerialService>(4).await?
        .into_request_stream(4).await;

    // Spawn the server worker task, with the FIFO and socket
    k.spawn(D1Uart::serial_server(fifo_b, reqs)).await;

    let (prod, cons) = fifo_a.split();
    // Spawn the SENDING worker, which uses DMA to send serial data
    k.spawn(D1Uart::sending(cons, tx_channel)).await;

    // Store the RECEIVING end, for UART interrupts to fill
    let boxed_prod = Box::new(prod).await;
    let leaked_prod = Box::into_raw(boxed_prod);
    UART_RX.swap(leaked_prod, Ordering::AcqRel);

    Ok(())
}
```

Kernel Basics

Creature Comforts

- Basic User Interfaces
- Forth Scripting
- Multiplexed UARTs
- Kernel Tracing



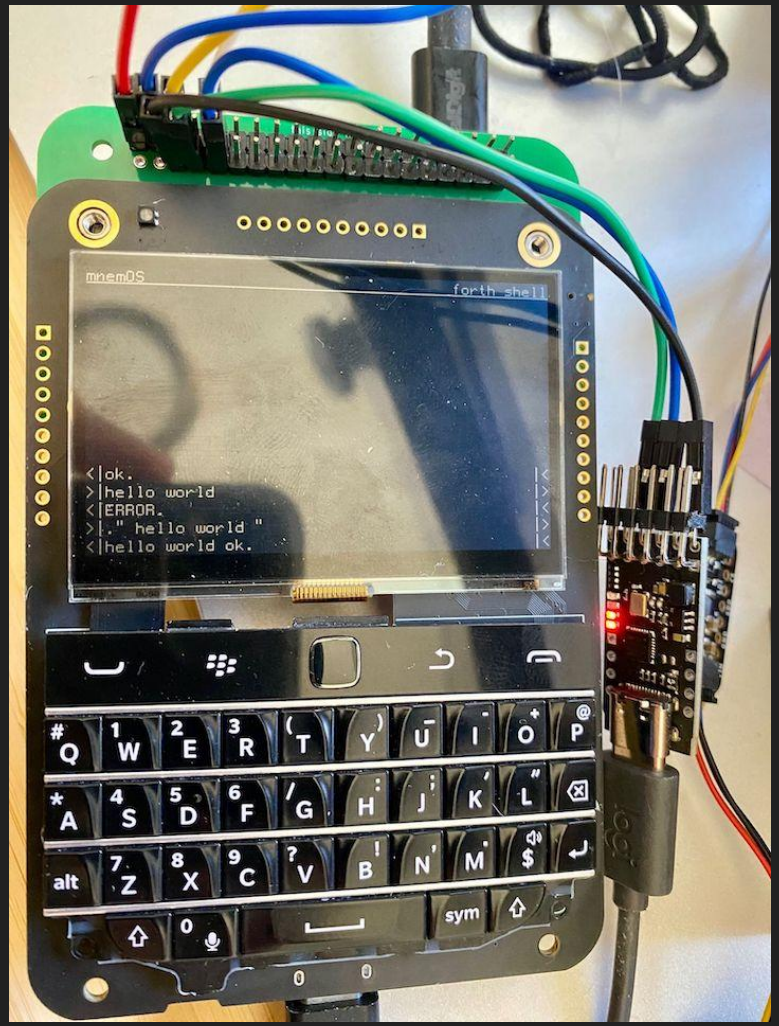
Creature Comforts

```
    0.001415625s  INFO Melpo:Kernel: kernel::comms::bbq: Creating new mpsc BBQueue channel capacity=4096
    0.001432292s  INFO Melpo:Kernel: kernel::comms::bbq: Channel created successfully
    0.001434375s  INFO Melpo:Kernel: kernel::comms::bbq: Creating new mpsc BBQueue channel capacity=4096
    0.001436667s  INFO Melpo:Kernel: kernel::comms::bbq: Channel created successfully
    0.001460125s  INFO Melpo:Kernel: kernel::registry: Registered KOnly uuid=f06aac01-2773-4266-8681-583ffe756554 service_id=0
    0.001500167s  INFO Melpo:Kernel: melpomene::sim_drivers::tcp_serial: TCP serial port driver listening on 127.0.0.1:9999
    0.001537625s  INFO Melpo:Kernel: melpomene: simulated UART (127.0.0.1:9999) initialized!
    0.001597709s  INFO Melpo:Kernel:register{settings=DisplayConfig { enabled: true, kchannel_depth: 2, frames_per_second: 20, scalin
g: 2 } width=400 height=240}: kernel::registry: Registered KOnly uuid=aa6a2af8-afd8-40e3-83c2-2c501c698aa8 service_id=1
    0.001647834s  INFO Melpo:Kernel:register{settings=DisplayConfig { enabled: true, kchannel_depth: 2, frames_per_second: 20, scalin
g: 2 } width=400 height=240}: melpomene::sim_drivers::emb_display: SimDisplayServer initialized!
    0.001721125s  INFO Melpo:Kernel: kernel::registry: Registered KOnly uuid=70861d1c-9f01-4e9b-89e6-ede77d8f26d8 service_id=2
    0.001735292s  INFO Melpo:Kernel: kernel::registry: Registered KOnly uuid=524d77b1-499c-440b-bd62-e63c0918efb5 service_id=3
    0.001807459s  INFO Melpo:Kernel: kernel::registry: Registered KOnly uuid=4ae4a406-005a-4bde-be91-afc1900f76fa service_id=4
    0.001823375s  INFO Melpo:Kernel: kernel::services::forth_spawnulator: ForthSpawnulatorService registered
    0.001949959s  INFO Melpo:Kernel: kernel::registry: Got KernelHandle from Registry svc=kernel::services::simple_serial::SimpleSeri
alService uuid=f06aac01-2773-4266-8681-583ffe756554 service_id=0 client_id=5
    0.001969667s  INFO Melpo:Kernel: kernel::registry: Registered KOnly uuid=54c983fa-736f-4223-b90d-c4360a308647 service_id=6
    0.002010542s  INFO Melpo:Kernel: kernel::registry: Got KernelHandle from Registry svc=kernel::services::serial_mux::SerialMuxServ
ice uuid=54c983fa-736f-4223-b90d-c4360a308647 service_id=6 client_id=7
    0.002014417s  INFO Melpo:Kernel: kernel::services::keyboard::mux: opening Serial Mux port 2
    0.002077417s  INFO Melpo:Kernel: kernel::comms::bbq: Creating new mpsc BBQueue channel capacity=8
    0.002080292s  INFO Melpo:Kernel: kernel::comms::bbq: Channel created successfully
    0.002089375s  INFO Melpo:Kernel: kernel::services::keyboard::mux: KeyboardMuxServer registered!
    0.002092209s  INFO Melpo:Kernel:loopback{settings=LoopbackSettings { enabled: true, port: 0, buffer_size: 128 }}: kernel::registr
y: Got KernelHandle from Registry svc=kernel::services::serial_mux::SerialMuxService uuid=54c983fa-736f-4223-b90d-c4360a308647 servi
ce_id=6 client_id=8
    0.002108000s  INFO Melpo:Kernel:hello{settings=HelloSettings { enabled: true, port: 1, buffer_size: 32, message: "hello\r\n", int
erval: 1s }}: kernel::registry: Got KernelHandle from Registry svc=kernel::services::serial_mux::SerialMuxService uuid=54c983fa-736f
-4223-b90d-c4360a308647 service_id=6 client_id=9
    0.002115625s  INFO Melpo:Kernel: kernel::comms::bbq: Creating new mpsc BBQueue channel capacity=128
    0.002125875s  INFO Melpo:Kernel: kernel::comms::bbq: Channel created successfully
    0.002127750s  INFO Melpo:Kernel: kernel::comms::bbq: Creating new mpsc BBQueue channel capacity=32
    0.002129709s  INFO Melpo:Kernel: kernel::comms::bbq: Channel created successfully
    0.002132459s  INFO Melpo:Kernel:loopback{settings=LoopbackSettings { enabled: true, port: 0, buffer_size: 128 }}: kernel::daemons
::sermux: SerMux Loopback running!
```

Creature Comforts

Platform Support

- **Allwinner D1** (64-bit RISC-V)
  - 1 GHz, 512MiB RAM
- ESP32C3 (32-bit RISC-V)
  - 160MHz, 400KiB RAM
- x86_64 (QEMU)
- Simulators:
  - **Melpomene** (native)
  - Pomelo (WASM)



Platform Support

Beepy by SQFMI + Beeper and Mango Pi MQ-Pro

What's Next?

- Message Passing Overhaul
- Inter-system communication protocol
- Reintroducing userspace and user programs

- Main Docs
  - https://mnemos.dev/
- GitHub
  - https://github.com/tosc-rs/mnemos
- Matrix Chat
  - https://matrix.to/#/#mnemos-dev:beeper.com

MnemOS

an operating system for
building small computers

James Munns
OneVariable UG
contact@onevariable.com

RIOT Summit 2023
2023-09-18