An HTTP request on a hobby OS

Running Alpine's BusyBox HTTPd on Takibi

Kiwamu Okabe
kiwamu@metasepi.org
https://metasepi.org/



Demo setup on a Linux host

```
$ docker pull debian:stable-backports
$ docker run --rm -it --network host debian:stable-backports
```

Inside the container, as root, install the build and QEMU tools.

```
apt update
apt install git make opam cmake llvm-19 llvm-19-dev lld-19 \
    libzstd-dev qemu-system-arm qemu-user ipxe-qemu gdb-multiarch \
    curl mtools cpio e2fsprogs e2tools python3-serial openjdk-21-jre-headless
```

Host networking lets the host browser reach the bridge on loopback.



Build and boot Takibi

```
git clone https://github.com/takibi-lang/takibi.git
cd takibi
opam init --disable-sandboxing
opam switch create takibi 5.4.0
eval $(opam env --switch=takibi)
opam install --deps-only --with-test .
bash scripts/fetch_model_tools.sh
make kernelsh-qemu
```

Run setup ahead of the demo. After the listener is ready, open the printed URL in the host browser (by default, `http://127.0.0.1:18080/`).



Who served this page?

BusyBox HTTPd from Alpine Linux

- Unmodified AArch64 binary, dynamically linked with musl
- Started by BusyBox init inside Takibi
- Serves files from an ext2 root filesystem

Follow the links to `about.html` and `icon.png` to make separate requests.



Follow one request

Execution

```
TCP receive -> accept -> clone -> child reads and writes -> exit
```

Response bytes

```
ext2 image -> HTTPd child -> TCP/IP -> browser
```

QEMU mounts the ext2 image directly. On a Raspberry Pi 5, HTTPd can serve from USB ext2 over RP1 Ethernet.



Convert the length before comparing it

IPv4 stores its total length in network byte order. In Takibi, `struct packed be Ipv4Hdr` gives the field a big-endian integer type.

```
// Rejected: ordering on the wire-endian value.
ip_hdr.total_len <= ip_len_in_frame

// Accepted: convert to a host integer before comparing.
let total_len: u16 = ip_hdr.total_len as u16;
total_len <= ip_len_in_frame
```

The compiler requires explicit conversion before ordering or arithmetic on wire-endian integers. The receive code uses the converted length to validate the packet.



How far can the kernel read?

The TCP length comes from a received frame.

```
[ Ethernet: 14 ][ IPv4: 20 ][ TCP segment ... ]
                            ^
                            byte 34
```

```
let tcp_seg = frame[34..<34 + tcp_len_idx];
```

What bounds the end of this slice?



Validate the packet, then prove the slice fits

Received length. Check that the IPv4 total length fits in the received frame. Reject a malformed segment.

Buffer capacity. Once `20 <= tcp_len <= 1480`, the end of `frame[34..<34 + tcp_len_idx]` is at most `34 + 1480 = 1514` bytes.

Build gate. The compiler proves the slice falls within the buffer on that path. If it cannot prove the bound and a bounds-check trap remains, the kernel build fails.



BusyBox forks a worker for each connection

```
HTTPd parent
  accept connection
  clone(SIGCHLD)
    |
    +--> child serves this connection and exits
  accept next connection
```

The child needs a **separate address space** that inherits the parent's memory contents. Copy-on-Write defers physical page copies until a write.



At clone, share the physical page

```
Parent mapping --+
                 +--> physical page A
Child mapping ---+

A page that was writable starts shared and read-only.
```

Parent and child have separate address spaces and kernel stacks.
The read-only mappings make a store fault while the page is shared.



On write, give the child a private page

```
Before the write                 After the write fault

Parent --+                       Parent ---> page A
         +--> page A
Child ---+                       Child ----> page B
                                             copied from A
```

Takibi copies A to B, makes the child's mapping writable, and retries the store. The parent still sees A.



When the parent reaps the child

```
Parent ---> page A          Parent ---> page A
                    ---->
Child ----> page B          Child mapping removed
```

Reaping removes the child's mappings. A physical page can be reclaimed once its last reference is gone. The parent's mapping of A remains valid.



Can we mix up parent and child memory metadata?

The parent and child each have their own memory management metadata (`ProcessImageRecord`): active page count, current stack lower end, heap size, and more.

The setter requires **evidence that the target metadata exists**.

```
child root + child evidence  -> accepted
child root + parent evidence -> type error
```

`root` is an internal key identifying an address space.
Successfully creating or looking up the metadata yields evidence of type `ProcessImageRecordExists[root]`.



The setter's type requires matching roots

```
fn process_image_set_active_page_count(
    root: usize @ identity,
    evidence: borrow ProcessImageRecordExists[identity],
    value: usize
) !{unsafe} { /* ... */ }
```

The compiler checks that the update targets the same address space whose metadata was confirmed to exist. This prevents accidental writes to metadata whose existence has not been checked.

The lifetime of the metadata is currently managed at runtime.

Statically guaranteeing the lifetime of the physical page used as the COW copy source remains an open problem.



A file in, a socket out

```
ext2 file -- read --> HTTPd child -- write --> connected socket
```

The child assigns the connected socket to file descriptors 0 and 1 (standard input and output).

From BusyBox's perspective, the file and the socket are both file descriptors. The kernel routes the read to ext2 and the write to TCP.



What if the kernel ignores the send result?

TCP send outcome	System call return value
`Sent`	Return the number of bytes sent if cleanup succeeds
`SourceFault`	Return `EFAULT`
`Invalid`	Return `EINVAL`

`KernelTcpSendResult` is a **`must_use variant`**.
Each path must handle or pass on the result. Calling the send function and dropping the result is a compile error. The system call's `match` decides which Linux error to return.



What did one request exercise?

Stage	What Takibi provides	What the compiler checks
Start HTTPd	ELF loading for Alpine BusyBox and musl	
Receive and accept	IPv4/TCP parsing and a listening socket	Explicit byte-order conversion and slice bounds
Fork the worker	`clone(SIGCHLD)`, separate address spaces, COW	Existence check and update target match
Read the file	File descriptor I/O and ext2 (USB on Pi 5)	
Send the response	Socket file descriptors and TCP transmission	Handle or pass on the send result
Reap the worker	Release its mappings after exit	

This workload exercises the Linux ABI subset used by Alpine's BusyBox HTTPd.

github.com/takibi-lang/takibi

