# Running FreeBSD on WSL2 Without Modifying FreeBSD: A Working Implementation

Balaje Sankar
Chennai, India
balajesankar@gmail.com

***Abstract*— This paper demonstrates running unmodified FreeBSD on Microsoft's Open Source[1] Windows Subsystem for Linux version 2 (WSL2). While WSL2 is historically Linux-centric, its underlying VM architecture is built on the Host Compute Service (HCS), can run FreeBSD. We present a stock-only approach that requires zero kernel modifications, achieving full integration through minimal userland bridges. We explore how the early-boot silent console issue was solved using serial I/O redirection, demonstrate the high-performance interactive terminal leveraging FreeBSD's native hvsock implementation, and discuss ongoing work to bridge FreeBSD with the Guest Networking Service (GNS). This work proves that FreeBSD's architectural maturity enables seamless adaptation to modern Windows virtualization infrastructure without compromising system integrity.**

***Keywords*— *FreeBSD, WSL2, Virtualization, Host Compute Service, Hyper-V, Operating Systems Integration***

## I. Introduction

This research explores bringing FreeBSD to WSL2, enabling FreeBSD to run seamlessly within Windows environments with minimal overhead. Unlike approaches that compromise the guest operating system through extensive modifications, this work maintains FreeBSD's integrity—using unmodified FreeBSD kernels, standard base system components, and conventional installation procedures. Integration is achieved entirely through minimal userland bridges that interface with WSL2's infrastructure.

The motivation is compelling: developers gain access to FreeBSD's powerful toolset directly within Windows; system administrators can leverage FreeBSD's networking stack and ZFS without separate hardware; students and researchers can explore BSD systems without traditional virtualization overhead. More fundamentally, this work demonstrates FreeBSD's architectural flexibility—its ability to adapt to novel environments while preserving system design principles.

To achieve FreeBSD integration with WSL2, six core technical challenges must be addressed:

- Virtualization Layer: Orchestrating FreeBSD VMs through HCS APIs
- Guest Handshake Protocol: Implementing WSL2 initialization without modifying FreeBSD's boot process
- Boot Console: Providing early kernel boot visibility in WSL2's environment
- Interactive Terminal: Connecting FreeBSD's PTY infrastructure to WSL2's console
- Guest Networking: Bridging FreeBSD's network stack with WSL2's networking model
- File System Integration: Enabling FreeBSD access to Windows filesystems

Each challenge is solved by leveraging FreeBSD's existing capabilities—its robust serial console support, native Hyper-V socket (hvsock) implementation, flexible device driver framework, and clean separation between kernel and userland. Where Linux WSL2 distributions rely on custom kernels and Microsoft's proprietary mini_init system, FreeBSD's design allows integration through standard rc.d services and userland daemons.

This "stock-only" approach validates FreeBSD's portability and architectural soundness. By running unmodified FreeBSD in a new virtualization environment, we demonstrate that well-designed Unix systems can adapt through abstraction and modularity rather than invasive core modifications. The result is a FreeBSD installation that behaves identically to bare-metal or traditional VM deployments, with full access to ports, packages, and the complete FreeBSD ecosystem—all running efficiently within Windows.

## II. System Architecture

Figure 1 illustrates the complete integration architecture. When a user executes wsl.exe, the Windows WslService orchestrator spawns a FreeBSD VM via HCS. The FreeBSD kernel boots from a complete disk image containing bootloader, kernel, and root filesystem—maintaining system integrity unlike Linux's split kernel/userland model.

Two userland components enable WSL2 integration:

hvinit (Initialization Client): Connects to WSL2 services on port 50000, implements the initialization handshake protocol, and signals VM readiness to the Windows host.

hvbridge (Console Server): Listens on port 60000, accepts multiple connections from WSL2, and bridges FreeBSD's PTY infrastructure to WSL2's hvsock-based console for interactive terminal access.

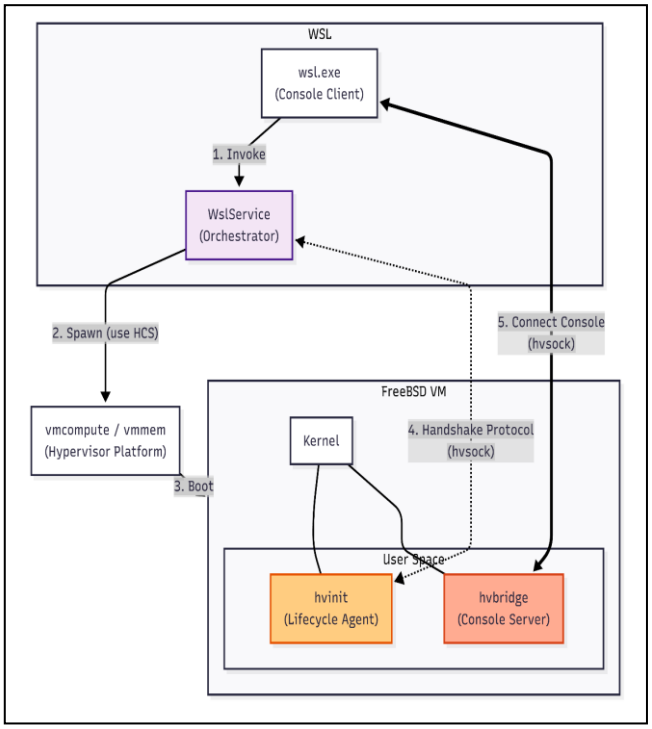


Fig. 1. FreeBSD on WSL2 System Architecture. Flow: (1) User command via wsl.exe, (2) WslService spawns VM through HCS, (3) FreeBSD boots, (4) hvinit connects and performs handshake, (5) hvbridge accepts connections and provides console. Boxes in bottom indicate FreeBSD-specific userland components—the only modifications required.

TABLE I. LINUX VS FREEBSD WSL2 INTEGRATION

| Component | Linux on WSL2 | FreeBSD on WSL2 |
|---|---|---|
| Kernel | Custom Microsoft | Stock |
| Boot Method | Custom Kernel + Userland | Self-contained disk Image |
| Init System | Custom mini_init | Standard rc.d + hvinit + hvbridge |
| Boot Console | virtio-conosle | serial-console |
| WSL2 Console | hvsock based console | hvsock based console |

## III. Implementation Environment

This implementation uses FreeBSD 14.3-RELEASE[2], the latest stable release. FreeBSD 14.3 includes features critical to WSL2 integration:

- Native hvsock support: Hyper-V socket implementation (hvsock(4))
- Enhanced Hyper-V drivers: Para-virtualized device drivers optimized for Hyper-V hypervisor
- Serial console stability: Robust uart(4) and comconsole support for early boot visibility

## IV. Hvsock Interface

Communication between FreeBSD and WSL2 relies on Hyper-V sockets (hvsock), designed for host-guest communication in Hyper-V environments. FreeBSD's kernel provides hvsock support through the AF_HYPERV address family.

### A. Socket Address Structure

The socket address structure required for hvsock programming is defined in FreeBSD's kernel (sys/dev/hyperv/hv_sock.h) but not exposed in standard userland headers. We recreated this structure in userspace:

```
struct sockaddr_hvs {
    unsigned char   sa_len;
    sa_family_t sa_family;
    unsigned int    hvs_port;
    unsigned char   hvs_zero[sizeof(struct sockaddr) -
                sizeof(sa_family_t) -
                sizeof(unsigned char) -
                sizeof(unsigned int)];
};
```

This approach maintains binary compatibility with the kernel's internal representation while avoiding kernel header dependencies, keeping the implementation self-contained and portable across FreeBSD versions.

### B. Port-Based Addressing Model

FreeBSD uses a simple port-based addressing model for hvsock communication. Applications specify only the port number without needing GUID knowledge:

- hvinit: Connects to port 50000 for initialization handshake
- hvbridge: Listens on port 60000 for console connections

Windows/WSL2 handles the GUID-based service identification internally. Microsoft defines a template GUID[3] (00000000-facb-11e6-bd58-64006a7986d3) for Linux-style guests where the first 32 bits represent the port number. When Windows connects to FreeBSD services, it constructs the appropriate service GUID by embedding the port number, but FreeBSD code remains simple and Unix-like using only integer port numbers.

### C. Client vs Server Patterns

The two components use different socket patterns:

hvinit (Client): Creates sockets and connects to host services.

```
int s = socket(AF_HYPERV, SOCK_STREAM, 0);
addr.hvs_port = 50000;
connect(s, (struct sockaddr*)&addr,
sizeof(addr));
```

hvbridge (Server): Binds to port and accepts incoming connections:

```
int s = socket(AF_HYPERV, SOCK_STREAM, 0);
addr.hvs_port = 60000;
bind(s, (struct sockaddr*)&addr,
sizeof(addr));
listen(s, 10);
int client = accept(s, NULL, NULL);
```

This asymmetric design reflects the initialization flow: hvinit actively connects to WSL2 services during boot, while hvbridge passively waits for WSL2 to establish console connections.

## V. Virtualization Layer

WSL2 orchestrates VMs through HCS API calls rather than traditional Hyper-V management interfaces. HCS accepts JSON[5] documents describing VM configuration including CPU count, memory allocation, device assignments, and boot parameters. Our implementation leverages HCS's full disk image support. FreeBSD resides on a virtual disk (VHD/VHDX) containing the complete base system: kernel, bootloader, and root filesystem as a unified image. This approach provides:

- System Integrity: No kernel modifications required
- Upgrade Path: Standard FreeBSD procedures should work unchanged
- Consistency: Boot process identical to bare-metal installations
- Debugging: Standard diagnostic tools should function normally

The virtualization layer treats FreeBSD as a first-class guest. The Host Compute Service (HCS) attaches the FreeBSD disk image as a SCSI device, configures serial ports for console access, and provides hvsock transport devices for host-guest communication

## VI. Boot Console

Initial boot attempts produced complete silence—no kernel messages or errors. WSL2's default configuration expects virtio-console for early boot visibility, which Linux distributions use with kernel support. However, FreeBSD lacks virtio-console support for early boot messages, making the system appear completely hung with no output during the critical kernel initialization phase.

FreeBSD's mature serial console support provided the solution. We configured HCS to attach a serial COM port and pipe output to the Windows console, then configured FreeBSD bootloader with standard settings:

console="comconsole"

comconsole_speed="115200"

No kernel modifications required—FreeBSD's existing serial console infrastructure, designed for headless servers and embedded systems, works identically in WSL2. The serial console provides complete boot transparency: initialization, bootloader menu, kernel device probe, rc.d service startup, and login prompt. This visibility proved invaluable for debugging initialization issues during development.

## VII. Initialization Protocol (hvinit)

WSL2 uses a "Utility VM" model for lightweight virtual machines. When launching a Utility VM, the Windows host needs to discover where guest services are available. Linux handles this through Microsoft's proprietary mini_init. For FreeBSD, our hvinit implements this protocol as a userland daemon running as a standard rc.d service.

### A. The Utility VM Handshake

Unlike traditional VMs with predetermined network endpoints, WSL2 Utility VMs announce their service ports during initialization. hvinit's role is to:

- Signal VM readiness to Windows
- Announce that console services are available on port 60000 (where hvbridge listens)
- Exchange configuration information with the host
- Maintain communication channels throughout VM lifetime

### B. Three-Socket Connection Sequence

hvinit establishes three sequential connections to the Windows host on port 50000:

- Capability Socket: Announces FreeBSD's identity and kernel version
- Notify Socket: Maintains notification channel for events
- Init Socket: Handles primary initialization and configuration exchange.

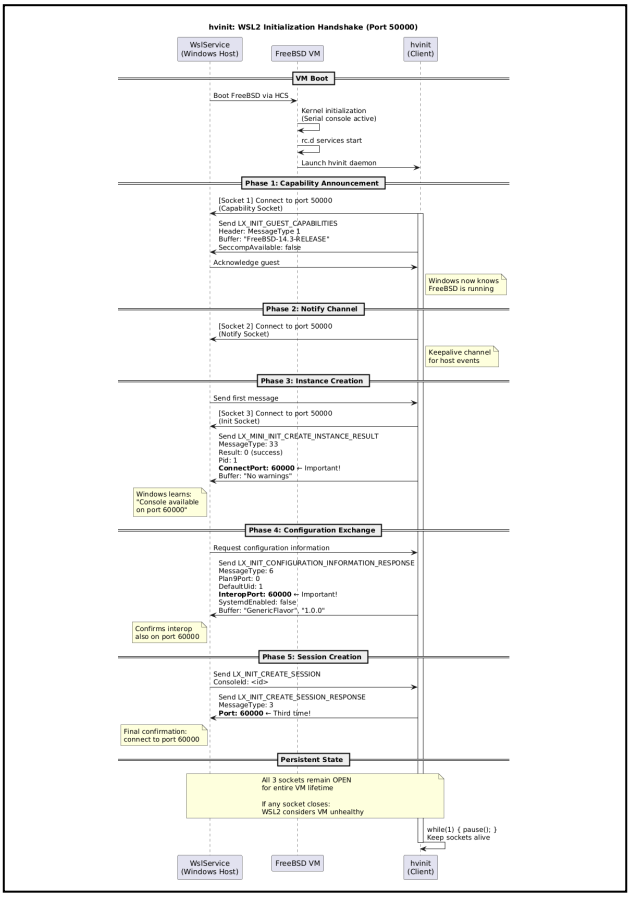


Fig. 2. hvinit handshake

## VIII. Interactive Terminal (hvbridge)

hvbridge provides WSL2's interactive console by listening on port 60000 and accepting seven sequential connections from the Windows host: an initial handshake connection, a message connection that receives LX_INIT_CREATE_PROCESS_UTILITY_VM containing terminal dimensions (rows/columns) and environment parameters, three console I/O connections for stdin/stdout/stderr, and two auxiliary connections for communication and interop channels. After storing the initialization message globally, hvbridge creates a pseudo-terminal using forkpty() which spawns /bin/sh in the child process while the parent implements a select()-based bidirectional I/O relay loop. This loop monitors both the PTY master file descriptor and the stdin hvsock connection, forwarding user input from Windows to the shell and shell output back to Windows Terminal, with both file descriptors configured as non-blocking to prevent deadlocks. The interop channel serves the stored initialization message (with modified MessageType=8) back to Windows tools querying FreeBSD's configuration, while hvbridge continuously monitors the shell process via waitpid(WNOHANG) and terminates gracefully when the shell exits. This pure userland implementation leverages FreeBSD's standard PTY infrastructure—the same mechanism supporting SSH and physical consoles—without requiring any kernel modifications, demonstrating clean separation between terminal emulation and I/O transport.

## IX. Future Work

Several areas remain under active research to achieve full feature parity with Linux WSL2 distributions:

Networking: Implementing Guest Networking Service (GNS) to enable automatic network configuration via WSL2's Host Compute Network service, providing seamless connectivity between FreeBSD guests, Windows host, and external networks.

File System Integration: Developing a 9P client to mount Windows filesystems (e.g., /mnt/c) within FreeBSD, enabling bidirectional file access between the guest and host operating systems.

Upstream Collaboration: Looking forward toward upstreaming hvinit and hvbridge to FreeBSD ports, collaborating with the FreeBSD Project on official WSL2 documentation.

## X. Conclusion

This work demonstrates that FreeBSD's architectural maturity and adaptability allows it to integrate seamlessly with modern Windows virtualization infrastructure. The stock-only approach proves that well-designed Unix systems can adapt to new environments through minimal userland bridges rather than requiring invasive kernel modifications.

The six integration components—virtualization layer, guest handshake, boot console, interactive terminal, networking, and filesystem—each showcase different aspects of FreeBSD's robustness. By leveraging existing FreeBSD capabilities like serial console support and hvsock implementation, this project achieves full WSL2 integration without compromising the integrity of the base system.

Future work includes completing the Guest Networking Service implementation and adding Plan9 filesystem support to provide feature parity with Linux WSL2 distributions.

## XI. Acknowledgments

We acknowledge the FreeBSD Project and Microsoft for open-sourcing WSL2 components. This work is independent research, not sponsored by or affiliated with either organization.

## References

[1] Microsoft Corporation, "Windows Subsystem for Linux (WSL)," GitHub repository, 2024. [Online]. Available: https://github.com/microsoft/WSL/

[2] The FreeBSD Project, "FreeBSD 14.3-RELEASE," 2024. [Online]. Available: https://www.freebsd.org.

[3] Microsoft, "Make your own integration services," Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/windows-server/virtualization/hyper-v/make-integration-service.

[4] B. Sankar, "FreeBSD on WSL2 Demo," Jan. 2025. [Online]. Available: https://x.com/balajesankar/status/1970585411153207715.

[5] Microsoft Corporation, "Host Compute Network (HCN) service API," Microsoft Learn, 2025. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/networking/technologies/hcn/hcn-top.