Gate project

Timo Savola
FOSDEM 2020
Portable execution state

Migrate live programs between desktops, servers and devices - safely.

Gain control by repositioning the abstraction layer.

Distributed software architecture, or dynamic network architecture.

Disclaimer: not a blockchain.
Reposition the abstraction layer

---

**USER**

Indirection layer for portable code

---

**CODE**

**DATA**

Traditional indirection layer
Reduce external interface surface

Data is portable. Portable code can be bundled with it, dissolving the boundary.
Reposition communication interface for locality

API can be moved into the server. Network I/O patterns become a client detail.
Gate

Personal hobby research project.

In development for 5 years - or over 10 if counting previous experiments.

BSD license.

https://github.com/tsavola/gate
https://gate.computer
Three tiers

WebAssembly

Portable program format, and a tooling ecosystem to go along with it.

Runtime for untrusted code

Usual Linux containerization features, but with extreme decoupling.

Pluggable, discoverable services

Hosts can provide their own sets of APIs.
Portable snapshot and restore

No support needed from user programs.

A running instance can be suspended at any time. The effect is immediate (or at least the time is bounded).

Snapshots are WebAssembly binaries with Gate-specific custom sections. Other runtimes could load them, but they appear as modules without any export functions.

*Halted* instances have returned from their entry function. Such snapshots have export functions, which may be called to re-enter the program.
Internals

Go packages, including a WebAssembly compiler:

https://github.com/tsavola/wag

Runtime core implemented in C and assembly.

Implementation is currently Linux-specific. Supports x86-64 and ARM64.

Can also run on Android.
Safety

WebAssembly defines a logical sandbox.

Each program invocation has its own OS process.

Service interaction happens via IPC messages sent through pipes.

Linux syscalls restricted via seccomp filter:

   Whitelist: read, write, close, ppoll, mprotect, rt_sigreturn, exit_group.
   mprotect arguments are restricted.

Finally, employ all the Linux namespaces to protect the host system.
Services

Services are discovered and may disappear as the program migrates.

Implementations:

- catalog – explore available services.
- origin – I/O with the originator/owner of the instance (≈ stdio).
- gate.computer/localhost – access whitelisted HTTP endpoints.

Services are implemented in Go. State serialization has an important role.

Next step: Support communication among peers on a server.
User program APIs

Impossible to support standard APIs meaningfully. Limited WASI support; Gate services are accessible through a dedicated file descriptor.

No blocking system calls. Purely asynchronous programming model.

Primitive C API. Used for simple test programs.

Rust is ideal for lightweight WebAssembly programs:

Gain crate provided Gate support, but it's out of date. **Next step:** Update it, with std futures and async/await syntax support.
Demo

1. Start the Gate port of Doom on an x86-64 machine.
2. Suspend it (SIGQUIT).
3. Show stack trace at the suspension point.
4. Create a snapshot.
5. Inspect the snapshot using wasm-objdump.
6. Copy the snapshot to an ARM64 machine.
7. Resume the game from the snapshot.

https://github.com/tsavola/doom
https://gate.computer/raster
V_Init: allocate screens.
M_LoadDefaults: Load system defaults.
Z_Init: Init zone memory allocation daemon.
W_Init: Init WAD files.
adding DOOMWADDIR/doom1.wad

Shareware!

M_Init: Init miscellaneous info.
R_Init: Init DOOM refresh daemon - [..]
InitTextures
InitFlats........
InitSprites
InitColormaps
R_InitData
R_InitPointToAngle
R_InitTables
R_InitPlanes
R_InitLightTables
R_InitSkyMap
R_InitTranslationsTables
P_Init: Init Playloop state.
I_Init: Setting up machine state.
D_CheckNetGame: Checking network game status.
startskill 2 deathmatch: 0 startmap: 1 startepisode: 1
player 1 of 1 (1 nodes)
S_Init: Setting up sound.
HU_Init: Setting up heads up display.
ST_Init: Init status bar.
x86-64 $ gate call doom.wasm < /usr/share/games/doom/doom1.wad

DOOM Shareware Startup v1.10

V_Init: allocate screens.
M_LoadDefaults: Load system defaults.
Z_Init: Init zone memory allocation daemon.
W_Init: Init WADfiles.
  adding DOOMWADDIR/doom1.wad

Shareware!

M_Init: Init miscellaneous info.
R_Init: Init DOOM refresh daemon - [..]
  InitTextures
  InitFlats.......
  InitSprites
  InitColormaps
  R_InitData
  R_InitPointToAngle
  R_InitTables
  R_InitPlanes
  R_InitLightTables
  R_InitSkyMap
  R_InitTranslationsTables
  P_Init: Init Playloop state.
  I_Init: Setting up machine state.
D_CheckNetGame: Checking network game status.
  startskill 2 deathmatch: 0 startmap: 1 startepisode: 1
  player 1 of 1 (1 nodes)
S_Init: Setting up sound.
H_Init: Setting up heads up display.
ST_Init: Init status bar.
^\n
bc32807d-eee8-4775-b4dd-48abdee67bfc SUSPENDED

x86-64 $
x86-64 $ gate snapshot bc3207d-eee8-4775-b4dd-48abdee67bfc snapshot.wasm
M4nu1fwg81A-SHsc27CGt91jlyu67VDTev12N-s5VkCIPuCzwk8aUoF6IDbuJ1
x86-64 $ gate debug bc3207d-eee8-4775-b4dd-48abdee67bfc backtrace
#0 0x6e69 in NetUpdate at /home/user/doom/linuxdoom-1.10/d_net.c:320
#1 0x7e1e in TryRunTics at /home/user/doom/linuxdoom-1.10/d_net.c:655
    0 0000000000000004 0000000000000001 0000000000000000c6c 0000000000000000
    4 0000000000000000be8 00000000000000001 0000000000000000be8 0000000000000000
#2 0x5434 in D_DoomLoop at /home/user/doom/linuxdoom-1.10/d_main.c:386
    0 0000000000000000
#3 0x6aaf in D_DoomMain at /home/user/doom/linuxdoom-1.10/d_main.c:0
    0 00000000000000001922450 0000000000000000 000000000000000000000000000000000000000000000
    4 0000000000000000
#4 0x0896 in _start at /home/user/doom/linuxdoom-1.10/libc.c:262
x86-64 $
arm64 $ uname -a
Linux graviton 4.15.0-1057-aws #59-Ubuntu SMP Wed Dec 4 09:58:16 UTC 2019 aarch64 aarch64 aarch64 GNU/Linux
arm64 $ gate call snapshot.wasm
Gate components

gate

Command-line client for the local daemon and remote servers.

gated

D-Bus daemon running and managing programs for the local user.

gate-server

Web server serving the public, or just authenticated users.
Server highlights

Can be configured to serve anonymous drive-by execution requests.

Uses Ed25519 public keys for grouping persistent resources. Authentication is optional. Supports SSH keys and authorized_keys files.

Optional IPFS support for sourcing programs.

Remote WebAssembly debugging with breakpoints. Portable snapshots.
Program and instance image management

Stored in sparse files; snapshotting requires shared memory mappings.

Backends:

- memfd (or ashmem on Android).
- Regular files on a filesystem, optimized for zero-copy (relink).

Normally, programs and suspended instances would go on the filesystem, and running instances in memory. But instances can also be directly backed by the filesystem.
WebAssembly “microcode”

Additional safety layer. Written in WebAssembly text format for stability.

Trusted WebAssembly library between user code and low-level runtime functions (syscall wrappers) implemented in x86-64/ARM64 assembly.

Implements the Gate runtime ABI (including WASI). Pointer arguments of ABI functions need to be checked carefully before accessing memory.

The low-level functions avoid pointers so that the WebAssembly compiler can generate checked memory access code outside of hand-written assembly code.