SecHack365

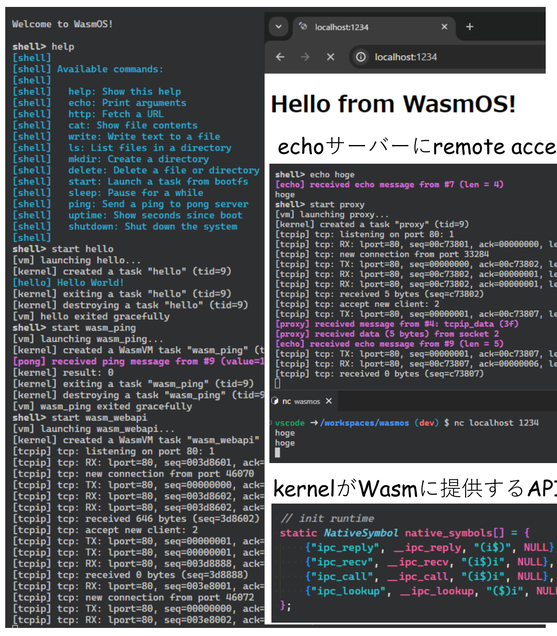
WasmOS

Wasmを実行するMicrokernel

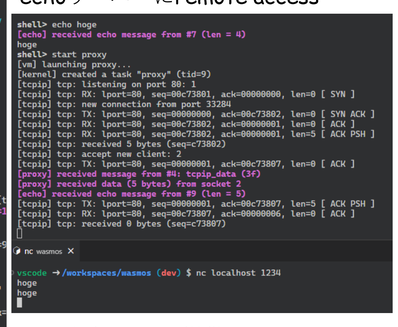
学習駆動コース 坂井ゼミ
尾田莉瑠

WasmOSとは

- Wasmを実行するMicrokernel
- Wasmを**kernel空間で**実行
- HinaOS をベースに開発
- 簡単なWebサーバーが動く



echoサーバーにremote access

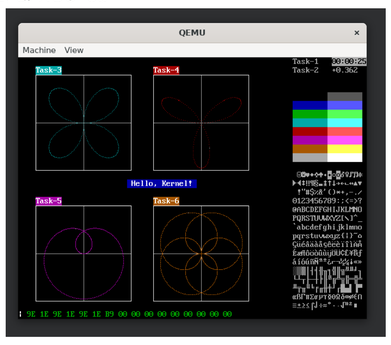


kernelがWasmに提供するAPI

```
// init runtime
static NativeSymbol native_symbols[] = {
    {"ipc_reply", __ipc_reply, "(i$)", NULL},
    {"ipc_recv", __ipc_recv, "(i$)i", NULL},
    {"ipc_call", __ipc_call, "(i$)i", NULL},
    {"ipc_lookup", __ipc_lookup, "($)i", NULL}
};
```

SecHack365での一年

6月
アセンブリ言語だけでOSを作る
https://github.com/r1ru/x86-os



7月
SecHack365 第2回イベント
WebAssembly, Microkernelとの出会い
https://github.com/nuta/microkernel-book

HinaOSをベースに開発したことで、CPUの初期化処理やデバイスドライバ等、"面倒な"部分をさぼることができた。

8月
セキュリティ・キャンプ全国大会参加 (専門C)
Mavis https://github.com/r1ru/mavis

- 全てのServerがWasmバイナリなMicrokernel
- Wasmバイナリを kernel空間で実行

```
shell> hello
Hello World!
task exited normally: tid = 00000003, code = 00000000
shell> exit
task exited normally: tid = 00000002, code = 00000000
PANIC kernel/kernel.c:28 switched to idle task
```

9月
WebAssembly night#11でMavisの発表
第35回 自作OSもくもく会でMavisの発表
10, 11月
自作Wasm Runtime
https://github.com/r1ru/tiny-wasm-runtime

- 公式のSpec Test をPass
- Wasm Specに貢献 (4 issue 1 PR)

```
80/85 Test #80: binary ..........   Passed    0.08 sec
      Start 81: binary-leb128
81/85 Test #81: binary-leb128 ...   Passed    0.06 sec
      Start 82: global
82/85 Test #82: global ..........   Passed    0.02 sec
      Start 83: elem
83/85 Test #83: elem ............   Passed    0.04 sec
      Start 84: data
84/85 Test #84: data ............   Passed    0.05 sec
      Start 85: exports
85/85 Test #85: exports .........   Passed    0.07 sec

100% tests passed, 0 tests failed out of 85

Total Test time (real) =   4.36 sec
```

12, 1, 2月
WasmOSの開発 https://github.com/r1ru/WasmOS

Wasmからtaskを生成するシステムコール

```
// create WASMVM task
static task_t sys_wasmvm(__user const char *name, __user uint8_t *wasm,
                         size_t size, task_t pager) {
    // get task name
    char namebuf[TASK_NAME_LEN];
    error_t err = strcpy_from_user(namebuf, sizeof(namebuf), name);
    if (err != OK) {
        return err;
    }

    // get pager task
    struct task *pager_task = task_find(pager);
    if (!pager_task) {
        return ERR_INVALID_ARG;
    }

    // validate size
    if (size > WASMVM_CODE_SIZE_MAX) {
        WARN("wasm too big: %u (max=%u)", size, WASMVM_CODE_SIZE_MAX);
        return ERR_INVALID_ARG;
    }

    // copy wasm binary
    uint8_t wasmbuf[WASMVM_CODE_SIZE_MAX];
    err = memcpy_from_user(wasmbuf, wasm, size);
    if (err != OK) {
        return err;
    }

    // create WASMVM task
    return wasmvm_create(namebuf, wasmbuf, size, pager_task);
}
```

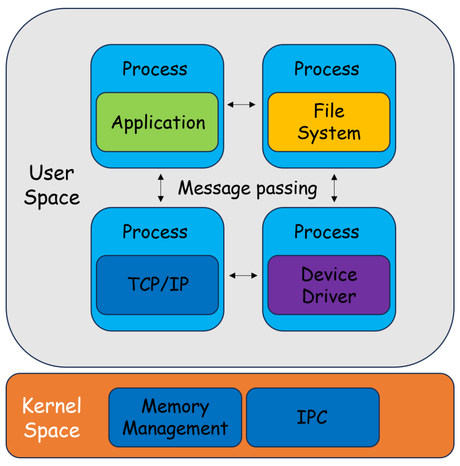
Pong serverのコード

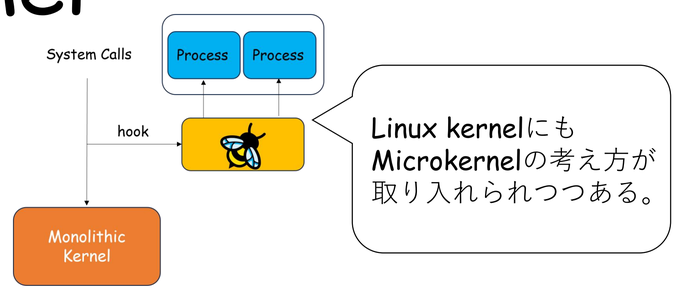
```
void main(void) {
    // pongサーバとして登録する
    ASSERT_OK(ipc_register("pong"));
    TRACE("ready");

    // メインループ
    while (true) {
        struct message m;
        ASSERT_OK(ipc_recv(IPC_ANY, &m));
        switch (m.type) {
            case PING_MSG: {
                DBG("received ping message from #%d (value=%d)", m.src,
                    m.ping.value);

                m.type = PING_REPLY_MSG;
                m.ping_reply.value = 42;
                ipc_reply(m.src, &m);
                break;
            }
            default:
                WARN("unhandled message: %s (%x)", msgtype2str(m.type), m.type);
                break;
        }
    }
}
```

Microkernelとは



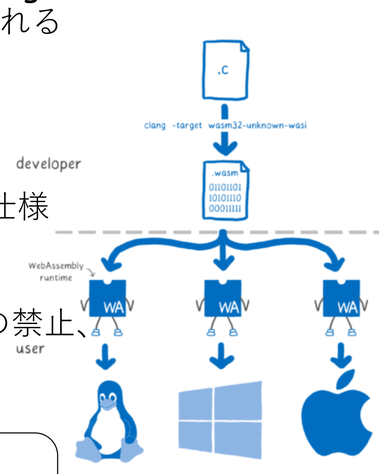


"eBPF is turning the Linux kernel into a microkernel" [9]

- kernelには必要最小限の機能だけを実装
- "OS"の機能をユーザーランドで実装
- IPCにMessage Passingが使われることが多い
- 安全性、拡張性に優れる
- **速度が課題**

WebAssemblyとは

- 仮想CPUの命令セットとバイナリフォーマットの仕様
- CPUやOSに依存しない汎用バイナリ
- 様々な言語をWasmにコンパイルできる。
- 高い安全性を持つ (型検査、未定義関数呼び出しの禁止、隔離されたメモリ、範囲外参照の検知等)
- WASIの登場でブラウザの外でも動かせるように



[6]

システムインタフェースの
標準化は険しい道のり…

Why Wasm in Kernel

- Microkernelの課題は速度
- Wasmをkernel空間で実行して安全性と効率性を両立。しかも汎用バイナリ
- 最近はWasm x kernelが熱い[1-4]

User空間の目的は隔離。もしソフトウェアでそれができるならkernel空間ですべてを動かしたって問題ない。

⇨ "User空間"の再定義

課題

1. Wasmは本当に安全？[12]

- バグがあると攻撃が可能
- MS-Wasm [13]という提案

これはWasmのバグというより、C/C++のバグといえる。Wasmが言語非依存をデザインゴールとする限り、この問題に対処する必要が生じる。

2. State spillという概念[7]

kernelの機能を細かく分けるだけでは不十分。大事なのは"うまく"分けること。

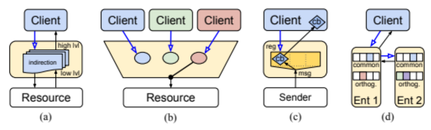


Figure 2: Common design patterns that cause state spill (→). (a) Indirection Layers cause state spill when converting between two representations of data/commands; (b) Multiplexers harbor state spill when serving multiple clients; (c) Dispatchers harbor state spill by holding registered callbacks (cb) for message/event delivery. (d) Inter-entity collaboration causes state spill when non-orthogonal states must be synchronized.

例) open後にfs serverがcrashしたら？

```
// ファイルを開く
rpc fs_open(path: cstr[256], flags: int) -> (fd: int);
// ファイルを閉じる
rpc fs_close(fd: int) -> ();
// ファイルの読み込み
rpc fs_read(fd: int, len: size) -> (data: bytes[1024]);
```

未来のOSはどうなる？

Theseus [5]:
Performance in Hardware, Isolation in Software

Ring Protection, Paging, eBPF, Wasm, Ownership…

Availability	Fault isolation & Recovery
Evolvability ← realize	Process Migration
Security	Live Update

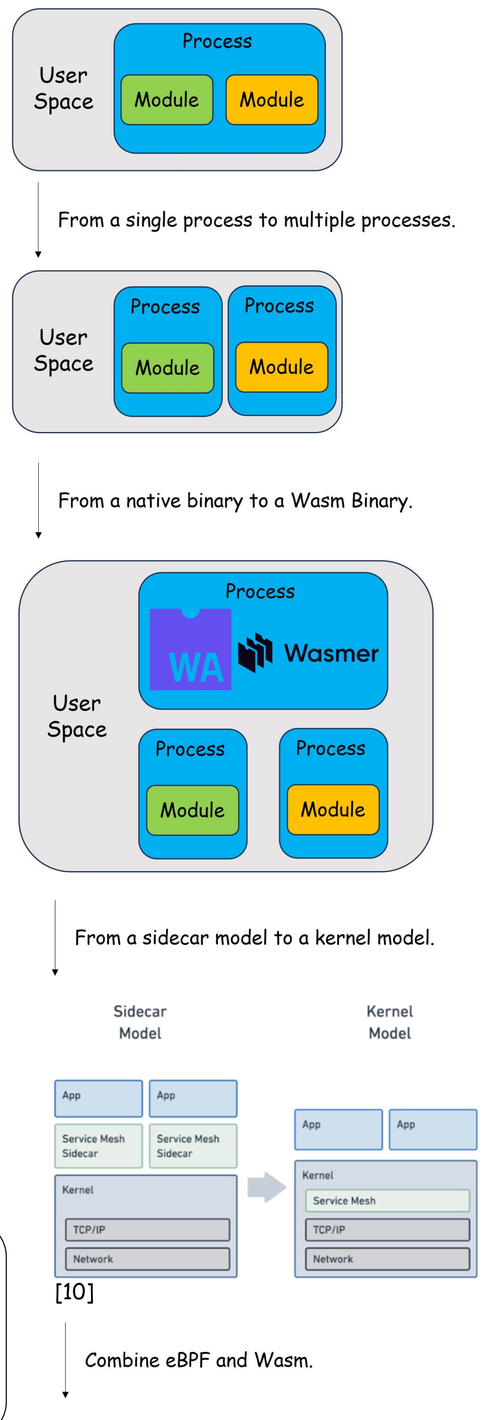
アプリケーションの変化によって、OSに要求されることも変化しつつある。未来のOSはどうなるのだろう？

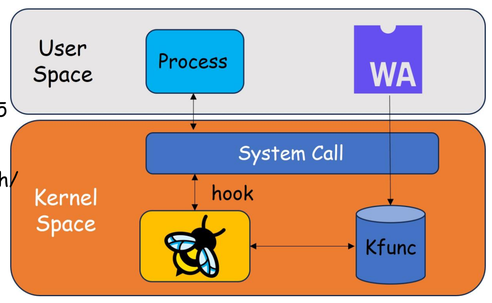
Related Projects

[1] kernel-wasm (https://github.com/wasmerio/kernel-wasm)
[2] Wasmachine (https://ieeexplore.ieee.org/document/9156135)
[3] Nebulet (https://github.com/nebulet/nebulet)
[4] Mewz (https://github.com/mewz-project/mewz)
[5] Theseus (https://github.com/theseus-os/Theseus)

Wasm Runtimeをkernelとして用いるというアイデア自体は実はそこまで新しくはない。

Applicationの進化





References

[6] Lin Clark. "Standardizing WASI". https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
[7] Kevin Boos, et al. "A characterization of State Spill in Modern Operating Systems". https://dl.acm.org/doi/10.1145/3064176.3064205
[8] Kevin Boos, et al. "Theseus: a State Spill-free Operating System". https://dl.acm.org/doi/10.1145/3144555.3144560
[9] Thomas Graf. eBPF- Rethinking the Linux Kernel. https://www.youtube.com/watch?v=f-oTe-dmfyI&t=8s
[10] Thomas Graf. "How eBPF will solve Service Mesh- Goodbye Sidecars". https://isovalent.com/blog/post/2021-12-08-ebpf-servicemesh/
[11] Martin Fowler. "Strangler Fig Application". https://martinfowler.com/bliki/StranglerFigApplication.html
[12] Daniel Lehmann, et al. "Everything Old is New Again: Binary Security of WebAssembly". https://dl.acm.org/doi/10.5555/3489212.3489225
[13] Craig Disselkoen, et al. "Position Paper: Progressive Memory Safety for WebAssembly" . https://dl.acm.org/doi/10.1145/3337167.3337171

NICT National Institute of Information and Communications Technology