学習用TCP/IPプロトコルスタックのIPv6対応

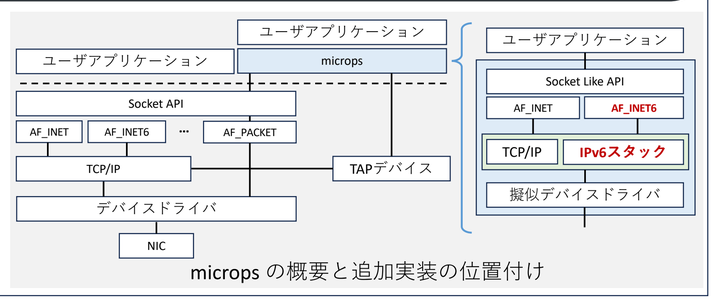
～アプリケーションプロトコルスタックの利活用～

学習駆動コース 横尾和真

背景: 学習用TCP/IPスタック "microps[1]"

- TCP/IPスタックの自作やソケットプログラミングは、IPv4を前提にした実装が多い
- IPv6の普及率は年々少しずつ増加しており、知らないうちにIPv6を使っている
- カーネルによるTCP/IPの処理は高性能だがブラックボックス状態
  → 自作することで理解を深める

IPv6スタックを追加実装し、公開することで今後IPv6を学習する人にとっても有用なものにしたい！

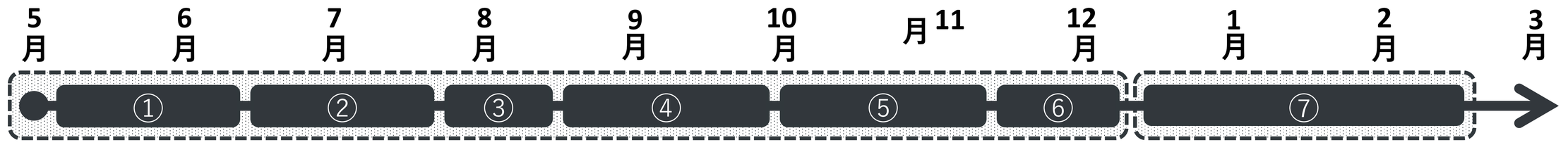


microps の概要と追加実装の位置付け

実用性の検討

- 実装を進めているうちに、アプリケーションとして動作するTCP/IPプロトコルスタックを「**学習用**」としてのみ消化するのは勿体無いと感じるように..
- **実用的**なユースケースを考えてみることに！

SecHack365で年間通して行った活動



IPv6 実装

① IPv6 基本部の実装

- FreeBSD の IPv6 実装（KAME[2]）などを参考に実装
  - 128 ビットアドレスを扱う
    - 標準で 128 ビットを扱える型がないので定義
    - IPv6 アドレスの省略表記に対応した ntop・pton の実装
  - IPv6 ヘッダ構造体の定義
  - IPv6 パケットの入出力・マルチキャストの受信
    - ip6_input / ip6_output

```
typedef struct {
    union {
        uint8_t __u6_addr8[16];
        uint16_t __u6_addr16[8];
        uint32_t __u6_addr32[4];
    } __addr_un;
#define addr8   __addr_un.__u6_addr8
#define addr16  __addr_un.__u6_addr16
#define addr32  __addr_un.__u6_addr32
} ip6_addr_t;
```



```
struct ip6_hdr {
    union {
        uint32_t ip6_un_flow;   /* ver(4) tc(8) flow-ID(20) */
        uint8_t ip6_un_vfc;
    } ip6_un;
    uint16_t ip6_plen;          /* payload length */
    uint8_t  ip6_nxt;           /* next header */
    uint8_t  ip6_hlim;          /* hop limit */
    ip6_addr_t ip6_src;
    ip6_addr_t ip6_dst;
#define ip6_vfc  ip6_un.ip6_un_vfc
#define ip6_flow ip6_un.ip6_un_flow
};  __addr_un.__u6_addr16
```

2001:00db:0000:0000:1234:5678:9abc
↓（省略）
2001:db8::1234:5678:9abc

② ICMPv6（NDPなど） の実装

- Echo Request/Reply，Destination Unreachable，Packet Too Big，Time Exceeded
- NDP（Neighbor Discovery Protocol）
  - MACアドレス解決
  - IPv6 特有の on-link / off-link に苦戦

```
01:03:57.750 [I] ip6_device_init      : created, link-local address=fe80::200:5eff:fe00:5301, dev=net1 (ip6.c:921)
01:03:57.750 [I] slaac_run            : start, SLAAC, dev=net1 (slaac.c:86)
01:03:57.751 [D] nd6_rs_output        : fe80::200:5eff:fe00:5301 => ff02::2, type=(133), +msg_len=8 (nd6.c:409)
```

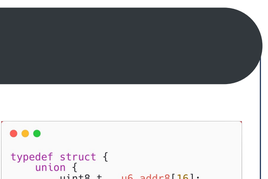
Link Local Address を生成し、SLAAC を開始している様子

```
01:03:57.764 [D] nd6_ra_input         : fe80::148a:f2ff:fe6b:dc86 => fe80::200:5eff:fe00:5301, type=(134), len=56 (nd6.c:486)
01:03:57.764 [I] ip6_route_add        : network=2001:db8::/64, nexthop=::, iface=2001:db8::200:5eff:fe00:5301, dev=net1 (ip6.c:455)
01:03:57.765 [I] ip6_iface_register   : registered: dev=net1, unicast=2001:db8::200:5eff:fe00:5301, prefixlen=64, scope=14 (ip6.c:572)
01:03:57.765 [I] ip6_route_add        : network=::/0, nexthop=fe80::148a:f2ff:fe6b:dc86, iface=2001:db8::200:5eff:fe00:5301, dev=net1 (ip6.c:455)
01:03:57.766 [I] slaac_ra_input       : created, global address by slaac=2001:db8::200:5eff:fe00:5301, dev=net1 (slaac.c:63)
```

Router Advertisement を受け取って、Global Unicast Address を自動生成している様子（DAD は未実装）

③ SLAAC の実装

- Stateless Address Auto Configuration: アドレスの自動設定
  - MAC ドレスを元に拡張 EUI-64 形式でインターフェース識別子を生成

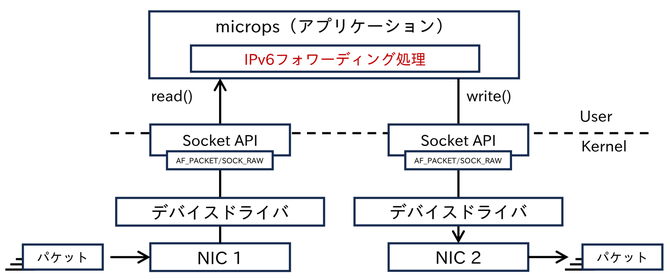


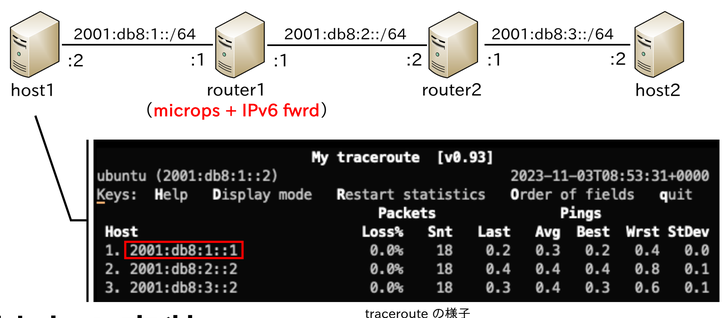
```
/* error messages */
#define ICMPV6_TYPE_DEST_UNREACH    1
#define ICMPV6_TYPE_TOO_BIG         2
#define ICMPV6_TYPE_TIME_EXCEEDED   3
#define ICMPV6_TYPE_PARAM_PROBLEM   4

/* informational messages */
#define ICMPV6_TYPE_ECHO_REQUEST    128
#define ICMPV6_TYPE_ECHO_REPLY      129
#define ICMPV6_TYPE_ROUTER_SOL      133
#define ICMPV6_TYPE_ROUTER_ADV      134
#define ICMPV6_TYPE_NEIGHBOR_SOL    135
#define ICMPV6_TYPE_NEIGHBOR_ADV    136
#define ICMPV6_TYPE_REDIRECT        137
```

④ IPv6 Forwarding の実装

- アプリケーション実装の IPv6 ルータと Network Namespace による動作確認






```
                My traceroute  [v0.93]
ubuntu (2001:db8:1::2)                    2023-11-03T08:53:31+0000
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                          Packets               Pings
 Host                  Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 2001:db8:1::1       0.0%    18    0.2   0.3   0.2   0.4   0.0
 2. 2001:db8:2::2       0.0%    18    0.4   0.4   0.4   0.8   0.1
 3. 2001:db8:3::2       0.0%    18    0.3   0.4   0.3   0.6   0.1
```

traceroute の様子

⑤ Socket Like API の AF_INET6 対応の実装

- microps では，アプリケーション開発がしやすいように Socket Like な API が用意されている（sock_open，sock_close，sock_bind，sock_listen，sock_accept，sock_connect，sock_send，sock_recv）
  - 「AF_INET6」を扱えるように追加実装
    - sockaddr 構造体の定義（ntopやptonを含む）
    - アドレスファミリーとアドレスをセットで扱う

→ `soc = sock_open(AF_INET6, SOCK_STREAM, IPPROTO_TCP);`

アドレスファミリーをセットで保持することIPv4/IPv6アドレスのどちらを扱っているのかが常にわかる

```
typedef struct {
    unsigned short family;
    union {
        ip_addr_t __u_addr4;
        ip6_addr_t __u_addr6;
    } __ip_un;
#define s_addr4 __ip_un.__u_addr4
#define s_addr6 __ip_un.__u_addr6
} ip_addr_storage;
```

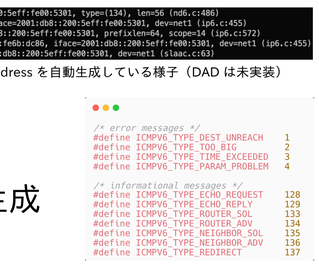
【エンドポイントを表すフォーマット】
[2001:db8:::1234:5678:9abc]:8080

⑥ 動作確認（アプリケーションの実装など）

- Web サーバの実装
  - カーネルとは独立した IPv6 プロトコルスタック上で実装したアプリケーションの一例
- ISIC（IP Stack Integrity Checker）[3] を用いたテスト
  - TCP/IP プロトコルスタックの整合性を確認するツール
  - ランダムなデータを持つ IPv6 パケットを生成し、それらに対して適切な処理ができているかをチェック
    - ヘッダに含まれる Length Field の値と実際のパケット長の整合性の確認不足による Segmentation Fault を確認・修正

```
18:30:55.449 [I] httpd_thread         : 533 bytes received (acc=1) (app/httpd6.c:212)
18:30:55.449 [I] http_handler         : GET: /test.pdf (app/httpd6.c:138)
18:30:55.449 [D] tcp_output_segment   : [fe80:cafe:dead:1::4]:80 => [2001:2f8:1c1:1816::1]:60758, len=62 (payload=42) (tcp.c:352)
18:30:55.449 [D] ip6_output_core      : dev=net0, iface=fd80:cafe:dead:1::4, len=62 +hdr_len=40 (ip6.c:762)
18:30:55.449 [D] nd6_resolve          : resolved, ip6addr=fe80::1234:5678:9abc, lladdr=12:34:56:78:9a:bc (nd6.c:343)
18:30:55.449 [D] net_device_output    : dev=net0, type=IPV6(0x86dd), len=102 (net.c:169)
18:30:55.449 [D] ether_transmit_helper : 80:22:a8:66:bf:03 => 12:34:56:78:9a:bc, dev=net0, type=IPv6(0x86dd), len=116 (ether.c:133)
```

Webサーバ over 自作IPv6スタックのログ



Webサーバ over 自作IPv6スタックにブラウザでアクセスした様子

```
2001:db8::1     2001:db8::2     ICMP      599                  Unknown ICMP (obsolete or malformed?)
2001:db8::1     2001:db8::2     IPv6      611                  Unknown IP Protocol: Unassigned (155)
2001:db8::1     2001:db8::2     IPComp    841                  IPComp (CPI=0x5095)
2001:db8::1     2001:db8::2     OSPF      857                  Unknown (30)
2001:db8::1     2001:db8::2     SCTP      828    11294   39573 RESERVED [Malformed Packet]
2001:db8::1     2001:db8::2     IPComp    90                   IPComp (CPI=0xe895)[Malformed Packet]
2001:db8::1     2001:db8::2     IPv6      417                  Unknown IP Protocol: Unassigned (196)
```

ISIC がさまざまな IPv6 パケットを生成している様子

```
09:13:47.639 [E] ip6_input            : ip version error: v=8 (ip6.c:640)
09:13:47.639 [E] udp6_input           : length error: len=1238, hdr->len=53953 (udp.c:234)
09:13:47.639 [E] icmp6_input          : checksum error: sum=0xc6ea, verify=0xA241 (icmp6.c:167)
09:13:47.640 [E] ip6_input            : ip version error: v=1 (ip6.c:640)
09:13:47.640 [W] ip6_hbh_input        : Hop-by-Hop option is not supported: ignored (ip6.c:600)
09:13:47.640 [E] tcp6_input           : checksum error: sum=0x6770, verify=0x1f69 (tcp.c:874)
09:13:47.641 [E] ip6_input            : ip version error: v=4 (ip6.c:640)
```

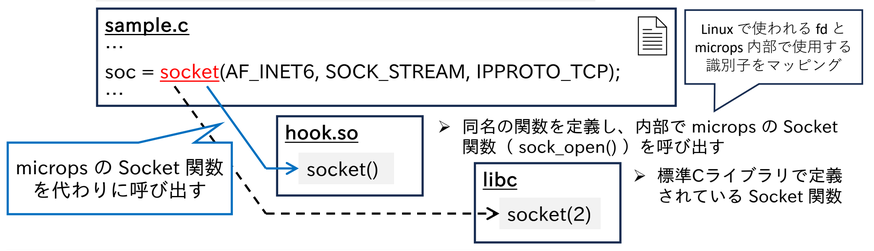
不完全なパケットを受け取ったときのログ

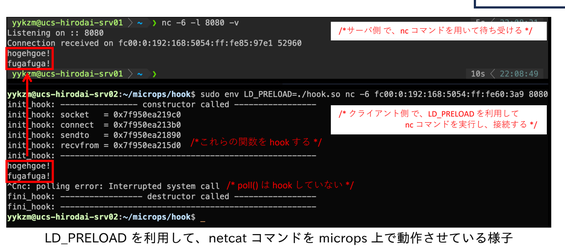
実用性の検討

⑦ 実用性の検討＆プロトタイプの実装

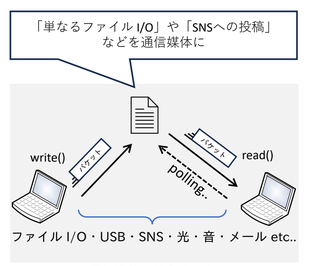
- アプリケーションスタック（microps ）の特徴
  - ライブラリ化されており、Socket Like な API も提供されているため、プロトコルスタックを含めて1つのアプリケーションとして動作可能
  - あくまでもアプリケーションとして動作するので、カーネル保護機構を使える
  - カーネルをビルドせずに、アプリケーションごとにネットワーク機能の追加やアップデートが可能
  - 通信媒体の抽象化が比較的容易に実現できる
  - microps 上で動くキラーアプリケーションも含めて実装する必要がある
- 既存アプリの再利用
  - 既存のアプリケーションを自作スタック上で動かすことができれば、キラーアプリケーションを実装することなくアプリケーションスタックの恩恵を受けられる

  → 環境変数 LD_PRELOAD を利用
    - 置き換えたい関数を定義した共有オブジェクトを指定して実行することで 優先してリンクさせられるため、標準ライブラリの特定の関数を自前のものに置き換えられる
    - `$ LD_PRELOAD=./hook.so ./sample`



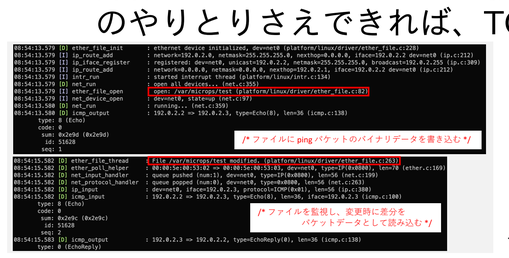


LD_PRELOAD を利用して、netcat コマンドを microps 上で動作させている様子



通信媒体の抽象化の例

- 通信媒体の抽象化
  - 媒体ごとにプロトコルを定義する必要がなく、データのやりとりさえできれば、TCP/IPで通信できる



← ファイル I/O で TCP/IP 通信をしている様子

```
yykzm@ucs-hirodai-srv01 ~ hexdump -C /var/microps/test
00000000  ff ff ff ff ff ff 00 00  5e 00 53 02 08 00 01 00  |........^.S.....|
00000010  08 00 06 04 00 01 00 00  5e 00 53 02 c0 00 02 02  |........^.S.....|
00000020  00 00 00 00 00 00 c0 00  02 03 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 5e 00  |..............^.|
00000040  53 02 08 00 01 06 04 00  01 00 00 5e 00 53 02 c0  |S..........^.S..|
00000050  00 02 02 00 00 5e 00 53  03 c0 00 02 03 00 00 5e  |.....^.S.......^|
00000060  53 02 c0 00 02 02 00 00  00 00 00 00 00 00 00 00  |S...............|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 53 b3 00 00  |............S...|
00000080  5e 00 53 02 00 00 45 00  00 30 00 01 00 00 ff 01  |^.S...E..0......|
```

パケットのバイナリデータが書き込まれたファイル

BLOG での手順公開

- micropsは、作者の方が日本語のドキュメントを無料で公開しており、誰でも取り組みやすい内容になっている

→ IPv6 対応を Next Step として取り組めるように手順を公開

- https://blog.hatena.ne.jp/y_kzm

Contact: kzm-yokoo@hiroshima-u.ac.jp / yykzmnet@gmail.com
作品のURL: https://github.com/y-kzm/microps

【参考文献】
[1] microps, https://github.com/pandax381/microps [2] KAME, https://www.kame.net [3] ISIC, https://github.com/IPv4v6/isic