Unix domain socket APIのポータビリティ問題

田中　哲
産業技術総合研究所 情報技術研究部門
2016-07-02

注意

- 2013年くらいに調べた話なので、変化していることもあるかもしれません。

趣旨

- Unix domain socket をさまざまな環境でテストした
- とてもとても多様な振る舞いが観測できた
- そもそも API が腐っている
  - API をデザインする人はそうならないように気をつけましょう
  - API を使う人は罠にはまらないよう気をつけましょう

Unix domain socket をテストした

- Debian GNU/Linux x86_64
- Debian GNU/Linux ARM
- Debian GNU/kFreeBSD
- Debian GNU/Hurd
- NetBSD
- FreeBSD
- OpenBSD 5.1
- OpenBSD 5.2
- DragonFly BSD
- MirOS
- Darwin (PureDarwin)
- SunOS (OpenIndiana)
- Haiku
- Minix
- Cygwin

いろいろなカーネルを試すのが
興味深い結果を得るポイント
(ふつうソケットはカーネルで
実装されるため)

Unix domain socket

- パス名をアドレスとして通信
- アドレスは struct sockaddr_un 構造体で表現
- あとは TCP/IP とだいたい同じ

Unix domain socket のクライアントとサーバ

サーバのアドレス

クライアントのアドレス

サーバ:

- serv = socket(AF_UNIX, SOCK_STREAM, 0);
- bind(s, &saddr, saddrlen);
- listen(s, SOMAXCONN);
- c = accept(serv, &caddr, &caddrlen)

クライアント:

- c = socket(AF_UNIX, SOCK_STREAM, 0);
- connect(c, &saddr, saddrlen);

アドレスを扱うシステムコール

- アプリケーションからカーネルへ
  struct sockaddr *addr と socklen_t len を渡す
  bind, connect, sendto, sendmsg
- カーネルからアプリケーションへ
  struct sockaddr *addrbuf と socklen_t *lenp を渡す
  *lenp は addrbuf に確保したバッファの長さ
  カーネルは *lenp を実際の長さに書き換える
  accept, getsockname, getpeername, recvfrom, recvmsg

struct sockaddr_un

- Unix domain socket のアドレスはパス名
  struct sockaddr_un を使う
- struct sockaddr_un は以下のフィールドを持つ
  - sa_family_t sun_family Address family. (POSIX)
  - char sun_path[] Socket pathname. (POSIX)
    実際の長さは決まっていない
- 実装依存で他のフィールドがあるかもしれない

sun_family	sun_path

sockaddr_un のバリエーション

- 4.4BSD: sun_len フィールドを追加
  - Darwin, Hurd, Haiku も追随
  - Debian GNU/kFreeBSD にもある
- sun_family フィールドのサイズ
  - 1: sun_len フィールドがあるOS および Minix
  - 2: その他
- sun_path のサイズ
  - 104: 4.4BSD (NetBSD, FreeBSD, OpenBSD, DragonFly BSD, MirOS)
  - 108: GNU/Linux, Hurd, SunOS, Cygwin
  - 126: Haiku
  - 127: Minix
  - Debian GNU/kFreeBSD (squeeze):
    user land は 108, kernel は 104 (wheezy では両方 104)

ソケットアドレスの終端

パス名は可変長なので終端を示す必要がある

- C言語の文字列としてのNUL文字による終端
- bind() や connect() の長さ引数による終端
- sun_path フィールドのサイズによる終端
- (sun_len が存在すれば、その長さによる終端)

これで混乱が起きないわけがない

バリエーション豊かな挙動

getsockname と getpeername でソケットの名前を調べる

- Minix は相対パスで bind しても、getsockname などでは絶対パスが返る
- Hurd は getsockname, getpeername を未サポート。getsockname は sun_path に "\0" という1byte が返り、getpeername は EOPNOTSUPP で失敗する
- OpenBSD 5.2 は短いパスで bind すると、getsockname などでは \0 で残りが塗りつぶされてパスは常に 104バイトになる。
- 他の環境は、Minix と Hurd 以外、bind に与えたパスが NUL 終端されている限り、そのままの長さで返す。

バリエーション豊かな挙動 (2)

ひとつのファイルを示すパスは複数ある。
サーバで bind したアドレスと
クライアントでconnectしたアドレスが違ったらどうなるか?

- connect した後 getpeername すると、SunOS と Cygwin は connect に与えたアドレスが返る。
- 他はサーバの bind に与えたアドレスが返る。

バリエーション豊かな挙動 (3)

ソケットを bind せずに getsockname

- 空の sockaddr (family もなし): DragonFly BSD, OpenBSD, MirOS, SunOS
- sun_path が空になる: Linux
- sun_path が空になり、その直後に NUL: Cygwin
- sun_path に 1byte の NUL: Hurd
- sun_path に 14byte の NUL: FreeBSD, Darwin, Debian GNU/kFreeBSD
- sun_path に 104byte の NUL: NetBSD
- ENOTCONN: Haiku
- EINVAL: Minix

バリエーション豊かな挙動 (4)

ソケットを bind せずに connect したとき、accept が返すアドレスは

- だいたい getsockname した結果と同じ、でも
- DragonFly BSD, OpenBSD, MirOS, SunOS は 空の sockaddr ではなく sun_path に 14byte の NUL
- Haiku は ENOTCONN でなく "\0002b1" というような通し番号
- Minix は EINVAL でなくサーバソケットのアドレス
- Cygwin は終端の後の NUL を書き込まない

バリエーション豊かな挙動 (5)

パスの NUL 終端の後にゴミを書き、そのゴミも含めた長さで bind して getsockname すると

- ゴミも含めてそのまま返ってくる: 4.4BSD, SunOS
- ゴミは NUL に書き潰されるが長さはそのまま: OpenBSD 5.1
- ゴミは NUL に書き潰され、長さは sizeof(sun_path) に伸ばされる: OpenBSD 5.2
- ゴミが捨てられ、最初の NUL までに切り詰められる: Haiku, Linux, Cygwin
- 絶対パスになり、ゴミは捨てられる: Minix

バリエーション豊かな挙動 (6)

- パスの NUL 終端の後にゴミを書き、そのゴミも含めた長さで connect した後、getpeername すると
  - SunOS: ゴミも含めて返ってくる
    (Cygwin は NUL までに切り詰めるのでゴミは返ってこない)
- sizeof(sun_path) よりも長いパスを bind 可能だった場合、getsockname などの結果は
  - バッファには切り詰められた結果が書き込まれ、長さは本来の長さが返る: Darwin, SunOS, Linux
  - 長さはバッファの長さになる: DragonFly BSD, NetBSD, MirOS

バリエーション豊かな挙動 (7)

bind に NUL を含まない長さを与えた場合、getsockname すると

- そのままの長さ、内容が返ってくる:
  FreeBSD, Debian GNU/kFreeBSD, Darwin, DragonFly BSD, NetBSD, OpenBSD 5.1, MirOS
- NUL がひとつ追加されて返ってくる:
  Linux, SunOS
- sun_path の残りが NUL で埋め尽くされて返ってくる: OpenBSD 5.2
- Buffer over read してる?
  Haiku, Minix, Cygwin

バリエーション豊かな挙動 (8)

- sun_len についてはとくにバリエーションは無い模様:
  - アプリケーションが指定した値をカーネルは無視する
  - カーネルがアプリケーションに渡すときは長さを書き込む
- Haiku は SOCK_DGRAM だと socket が EAFNOSUPPORT
- Minix は SOCK_DGRAM では connect が EINVAL

どこまで長いパスを受け入れるか

- ccccc\0 というように c を繰り返した場合 (NUL 終端も含めた長さを指定した場合)
  - 104: FreeBSD, OpenBSD
  - 108: Linux, Cygwin
  - 111: Minix
  - 126: Haiku
  - 23?: MirOS (kernel panic)
  - 253: Darwin, DragonFly BSD, NetBSD
  - 256: SunOS, Hurd (ファイル名の限界)

どこまで長いパスを受け入れるか

- ./././././.../ab\0 というように ./ を繰り返した場合 (NUL 終端も含めた長さを指定した場合)
  - 104: FreeBSD, OpenBSD
  - 108: Linux, Cygwin
  - 126: Haiku
  - 128: Minix
  - 235: MirOS
  - 253: Darwin, DragonFly BSD, NetBSD
  - 1024: SunOS
  - 1027: Hurd (./ の繰り返し 512 回の限界)

どこまで長いパスを受け入れるか

- NUL終端をしない長さを与えた場合
- 疲れたので省略

Buffer Over Read

- Cygwin: bind() が引数の長さを無視して NUL文字を探す
- Hurd: connect() が引数の長さを無視して NUL 文字を探す。(bind() は引数で与えた長さまでしか見ない)
- Haiku: bind に NUL終端されていないパスを与えると、指定した長さだけ kernel 空間にコピーした後、その長さを無視して NUL文字を探す
- Minix 3.2.1: NUL 終端の後にゴミがあると bind が失敗する
- Minix 3.2.1: 上記の修正後でも引数で指定した長さを無視して sizeof(sun_path) だけコピーする。

その他の問題

- Cygwin: connect と accept が同期する
- Minix: connect しただけでは accept が終わらない。クライアントが終わってもサーバに EOF が伝わらない
- NetBSD: クライアントが close した後にサーバが accept すると、クライアントのアドレスがくるべきところにサーバのアドレスとゴミがくる
- MirOS: 234バイト(NUL込み)のパスで bind して、getsockname すると kernel panic (もっと条件がある?)

その他の問題 (2)

- Cygwin: Unix domain の SOCK_STREAM ソケットで accept や getpeername が空のアドレスを返す
- Cygwin: Unix domain の SOCK_DGRAM ソケットで recvfrom が AF_INET なアドレスを返す
- Debian GNU/kFreeBSD: 長すぎるパスを与えてもエラーにならず、単に 104バイトで切り落とされる

その他の問題 (3)

- Minix: すでに存在するファイルやディレクトリに Unix domain socket を bind できる

エピソード

- Cygwin の Buffer over read を指摘した
- Cygwin 開発者は Buffer over read を直すついでに NUL 終端を必須にした
  (POSIX は NUL 終端必須)
- スナップショットが出たら、D-Bus の開発者がバグレポートを送ってきた
  NUL終端していなかったのだろう
- (おそらく) NUL終端を必須にはしなくなった

現実と仕様の乖離

- POSIX では sun_path はパス名で、パス名は 定義としてNUL終端するもの (POSIX2001)
  a pathname consists of, at most, {PATH_MAX} bytes, including the terminating null byte.
- POSIX では構造体の長さを指定することを想定
  傍証1: bind() の項のサンプル(POSIX2008)
  傍証2: sockaddr_un のフィールドの順序が決まっていない(現実的には変えられないけれど)
  The <sys/un.h> header shall define the sockaddr_un structure, which shall include at least the following members:
- 4.3BSD のドキュメントではNUL直前までの長さを指定すると明確に書いてある (1986)
- きれいな仕様を決めたが現実には勝てなかった?

IPv4 もちょっとテストした

- struct sockaddr_in にはパディングがある
- getsockname などでカーネルからアドレスを得たとき、パディング部分にゴミが入る OS がある: Hurd, Haiku

教訓とまとめ

- 仕様が仕様を決めるわけではない
  APIデザインは重要
- デファクト仕様を見抜くのは難しい
  ひどいAPIを使用するときは気をつけないといけない
- いくら仕様がひどくても Buffer over read は許されないと思う (情報漏洩になるかも?)
  しかし終端の定義の違いと言われると困る
- 寛大な仕様で多様性を育てるのは不幸
- いくつか開発元にレポートした:

  Cygwin, Hurd, Haiku, Minix, NetBSD, MirOS, Debian GNU/kFreeBSD