xinu-rpi4 — OS としての改良余地アセスメント

Raspberry Pi 4 (BCM2711 / Cortex-A72, AArch64) 向け Embedded Xinu

2026 年 7 月 20 日

対象	Pi 4 カーネル (`compile/make pi4` → `kernel8.img`)
基準コミット	`ccb6437` (proc: harden tickless one-shot against total hang)
規模	`.c/.h/.S` 130 ファイル、約 31.4k 行、195 コミット (`extern/ obj/ references/` を除く)

本書は Pi 4 カーネルを「OS として」評価し、改良余地を影響度と工数で順位付けしたものである。スケジューラ／メモリ／ネットワーク・ドライバ／ファイルシステムとツーリングの 4 領域を精読した結果をまとめている。重大な所見は実際にソースで裏取りし、そのうち Tier 0 の 5 件と Tier 1 の 4 件は本アセスメントと同日に修正・実機検証した (第 5 章・第 6 章)。

行番号はアセスメント時点のもので、以後の編集でずれる。関数名を併記しているのでそちらを頼りにすること。Markdown 版は `docs/OS_ASSESSMENT_JA.md`。

目次

1 総評

上半分 (プロセス／ IRQ アーキテクチャ) は本物、下半分 (メモリ安全性・ネットワーク・ファイルシステム) はデモ級という非対称な成熟度にある。

評価できる点:

- **ISR の中で重い処理をしていない**。net プロセス + app ワーカー + aipl プロセスを park/kick で分離し (`loader/main.c:206–269`)、タイマ ISR はティック更新と RT スリーパの起床のみ、GENET ISR は計数・自己マスク・ack のみ。HTTP 処理は必ずプロセス文脈で走る。
- **フォルト後も生き延びる**。`recover_spin()` (`system/exception.c:61`) が AIPL ヒープロックを解放し IRQ を再開してスピンするので、同期例外のあとも HTTP が応答し `/fault` で原因を読める。
- **MMIO プローブが箱を殺さない**。`safe_mmio_read32/write32` (`system/exception.c:79–104`) が `__builtin_setjmp` でガードし、ゲートされた PCIe レジスタへの書き込みが −1 を返すだけで済む。
- ヘッダコメントの質が高い。`device/sd/sd.c:73–86` (74 クロックの電源投入ウィンドウ)、`fs/fat32.c:180–184` (クラスタ確保の off-by-2) など、根本原因が文章として残っている。

いずれも「1 回のハングが電源再投入 1 回」という開発コストへの対処として設計が筋道立っている。

弱いのは、**落ちないので気づけない類のバグ**が各層に残っていること。以下はその棚卸しである。

2 Tier 0 — 実害があり修正コストが小さいもの

すべてソースで裏取り済み。**本アセスメントと同日に全件修正した** (第 5 章)。

2.1 ヒープ分割の 8 バイト境界オーバーラン — `mem/memory.c getmem()`

```
if (curr->mlength > nbytes) {              /* <- 残り 8 バイトでも成立 */
    struct memblk *leftover = (curr + nbytes);
    leftover->mnext = ...; leftover->mlength = ...;   /* 16 バイト書く */
```

`struct memblk` は 16 バイト、`ROUNDMB` は 8 バイト粒度。残余がちょうど 8 バイトのとき、**返却したブロックの隣に 8 バイトはみ出して書き、偽の free ノードをリストに繋ぐ**。以後ヒープが静かに壊れる。

同型の欠陥が `freemem()` 側にもあった (8 バイト領域に 16 バイトのヘッダを書く)。こちらは目視では見落としており、ホスト側のランダム churn テストが検出した。

2.2 ヒープが IRQ／プリエンプション非安全 — mem/memory.c

getmem/freemem/kmalloc/kfree に critical section が一切ない。プリエンプションは実際に有効化される経路がある (/netpreempt, /rtos-jitter, preempt_demo)。proc_create は getmem を **irq_save の外**で、proc_kill は freemem を**中**で呼んでおり (system/proc.c:207 vs :422)、この非対称さ自体が症状である。

2.3 ICMP エコー応答からのメモリ情報漏洩 — system/net_responder.c:210–234

```
int total_len = ip[2]<<8 | ip[3];
if (icmp_len < 8 || total_len + 14 > 1518) return 0;   /* len と比較していない */
int reply_len = 14 + total_len;
for (int i = 0; i < reply_len; i++) tx_reply[i] = frame[i];
```

total_len を**実受信長 len と照合していない**。60 バイトの ICMP に total_len=1400 を書いて送ると、RX リング上の隣接メモリ (＝直前のパケット) 約 1.3 KB を攻撃者に送り返す。

同型の欠陥が system/tcp_server.c:2537–2541 (data_off も未検証。data がフレーム外を指し、偽の data_len が peer_seq を desync させる)。なお IP ／ TCP ／ UDP ／ ICMP のいずれもチェックサムを検証していない。

2.4 SYN 4 発で恒久 DoS — system/tcp_server.c alloc_conn()

struct tcp_conn にタイムスタンプがなく、状態は受信パケットでしか進まない。再送タイマもない。ACK を返さない SYN を NCONN(=4) 本打つと**全スロットが SYN_RCVD で永久に固まり、再起動まで HTTP が死ぬ**。alloc_conn() は LISTEN/CLOSED しか回収しない。

2.5 cat /microsd/xxx が物理アドレス 0 を読む — shell/fscmd.c:127

FAT32 のファイルを VFS に登録する際 node->size は入れるが node->data は NULL のまま (意図的、コメントあり)。cat/cp/mv はそれを無条件に deref する。アドレス 0 が RW 識別マップ (system/mmu.c:200) なのでフォルトせず、**低位 RAM の中身を表示・コピーする**。ls はもっともらしいサイズを出すので気づけない。

2.6 W^X が壊れている — system/mmu.c:191,214

kern_2mb = _start & ~2MiB = 0x0 なので、属性を分ける L3 (4 KiB ページ) 層が覆うのは 0x0–-0x200000 だけ。実際の _data = 0x268000。つまり **.rodata の約 416 KB と .data/.bss 全部 (＝全 DMA リング、proctab、smp_stack) が 2 MiB ブロックの RW+X** になっている。イメージが 2 MiB を超えた時点で静かに劣化したもので、ビルド時アサートもない。

3 Tier 1 — 設計上の本丸

3.1 優先度プリエンプションが実質ラウンドロビン — system/proc.c proc_resched()

proc_resched() が**走行中プロセスを ready に戻す前に最良候補を pop する** (:283 vs :292–295) ので、現在プロセスの優先度が比較に参加しない。prio=50 の RT タスクが prio=5 の hog に叩き落とされる。g_dbg_hi_dispatch カウンタ (:124) は、これが追われていた形跡である。

関連して、NULLPROC は決してプリエンプトされず (:403) ready リストにも載らない (:292)。NULLPROC は**シェル／ウィンドウマネージャのループそのもの**なので、対話文脈は純粋に協調的で、net_yield_tick の明示 proc_yield() でしか進まない。さらに AIPL アクター実行中はプリエンプションが全面的に無効化される (proc.c:401)。つまり主要ワークロードでは、このカーネルは協調型である。

修正は数行。P1 (RTOS 化) の看板そのものなので優先度は高い。

3.2 (8) ready リストが O(n)、タイマ ISR が毎回 proctab を 2 周

- ready_pop() は最大 prio の線形探索 (system/proc.c:107–126)。NPROC=2048、N-Queens で約 1300 アクター、しかも **D-cache は意図的に OFF** (system/mmu.c:263) なので 1 ノード＝ 1 DRAM 往復。
- proc_next_delay_us() (:342–355) と proc_timer_tick() (:377–390) が同一 ISR で 2048 エントリを 2 周。最大 5 kHz。PROC_TICK_MIN_US のコメント (:337–341) 自身が、この ISR がシステムを餓死させうると認めている。

→ 優先度別 FIFO + ビットマップ (CLZ) で O(1) 化、スリープはソート済みデルタキューへ。

3.3 (9) タイムベースが壊れている

timer_ticks() は**可変長のワンショット発火回数**であって時計ではない。にもかかわらず 4 つのサブシステムが固定周期として扱っている: actorproc.c:150,282,335 (∗ 10 = 100 Hz 前提)、cc/cc.c:514、device/usb/uspi_glue.c:95–100、device/usb/xhci/xhci.c:1119–1476。アクター GC の経過時間もキーリピートも約 10 倍ずれ、しかも非線形。CNTPCT_EL0 由来の真の now_ms() は既に 2 箇所にある。

3.4 (10) D-cache OFF が最大の性能レバー、ただし解放には DMA 規律が要る

SCTLR.C=0 は SMP コヒーレンシを「全アクセスが RAM に届く」で担保する設計判断 (include/smp.h:10–12) で理屈は通っているが、4 コア分の D-cache を捨てている。DCACHE_ON を有効にする前提として:

ドライバ	cache maintenance
mailbox (device/mbox/mbox.c:51–53,85–87)	正しい
video (device/video/video.c:247–249)	正しい
smp mailbox (system/smp.c:47–50,110)	正しい
JIT (cc/cc.c:723–735)	正しい (dc civac + ic iallu)
wifi (device/wifi/wifi.c:2138–2141)	部分的
genet (device/genet/genet.c)	**dc 命令ゼロ** (dsb sy のみ)
xhci (device/usb/xhci/xhci.c)	**dc 命令ゼロ** (dsb sy のみ)

規律は存在するのに 2 大 DMA ドライバだけ抜けている。DCACHE_ON は現状どのターゲットも定義しておらず、実質デッドコードである。

3.5 (11) セマフォが偽物、network/ が丸ごとデッドコード

system/xinu_compat.c:86–102 の wait()/signal() はカウンタの増減のみで、**キューなし・ブロックなし**。struct sement.queue は宣言されて未使用。sleep() は即 return、yield() は yield しない、create() はスレッドを作らない。

その結果、network/** (ARP/IPv4/ICMP/netaddr 約 1500 行) は**完全にデッドコード**である。netUp/netRecv/ipv4Recv/arpRecv の呼び出しは network/ の外にコメントとしてしか存在しない。それでも compile/Makefile:47–49 はビルドし続けている。生かす (本物の netif + ブロックするセマフォ) か消すかの決断が要る。

なお、ネットワークスタックは現在 **4 系統が並存**している: system/net_responder.c (ARP+ICMP)、system/dhcp_client.c、system/tcp_server.c、device/wifi/wifi.c:1238–2100 (ARP+ICMP+DHCP+DNS+NTP を独自に再実装)。

3.6 (12) スタックガードが皆無

カナリアも guard page も高水位マークもない。一方 IRQ エントリは **768 バイトのフレーム** (GPR 31 本 + q0-q31 全部、system/exception_vectors.S:97–131) を割り込まれたプロセスのスタックに積み、proc_create は 1024 バイトのスタックを受け付ける (proc.c:192)。NULLPROC に至っては sp = _start = 0x80000 で下方向に firmware spin table へ伸びる、長さ未知のスタックである。

3.7 (13) 2 GB のうち 1 GB が使えない

HEAP_END=0x40000000 が compile/Makefile:90 にハードコードされ、しかも 0x40000000–0x7FFFFFFF (実 DRAM) を **Device-nGnRnE でマップ**しているので (system/mmu.c:221–228) 触ることすらできない。mailbox の GET_ARM_MEMORY か、既にパース済みの DTB /memory から取るべき。

3.8 (14) TCP に再送も分割もない

- **再送なし**。struct tcp_conn に RTO タイマもタイマフィールドもない。

- **ウィンドウなし**。送出ウィンドウは `0x2000` 固定、相手の広告ウィンドウは読まない。
- **順序外の再組み立てなし**。`seq == peer_seq` のときだけ追記し、他は黙って捨てる。
- **分割なし**。`tcp_send()` は 1 フレーム超を拒否するのに、`tcp_app_flush()` は最大 16384 バイトを 1 回で渡す。**約 1464 B を超える HTTP 応答は丸ごと落ちる**のに `g_app_served++` は回り FIN も出るので、クライアントには「0 バイトの正常終了」に見える。
- **ISN が `0xDEADBEEF` + 接続ごとに `0x100`** で完全に予測可能、かつ ESTABLISHED で ACK のウィンドウ検査をしないので、経路外からの注入が容易。

3.9 (15) GENET TX が 50 ms のビジーループ

`genet_tx_frame()` (`device/genet/genet.c:902–952`) は `TDMA_CONS_INDEX` を最大 50 ms ポーリングする。yield しない。ARP 応答も ICMP 応答も ACK も HTTP 応答もこのループを通る。256 個ある TX ディスクリプタのうち、常に 1 個しか使っていない。しかも送信バッファは `tx_buf_pool` 1 枚を全呼び出し元で共有し、各呼び出し元も自前の `tx_frame[1518]` を持つので**パケットごとに 2 回コピー**している。

加えて `genet_rx_poll()` は**ディスクリプタのエラー／ status ビットも DMA_OWN も SOP/EOP も検査せず**、`length` (12 ビット、最大 4095) を `RX_BUF_LENGTH`(2048) にクランプもしない。`CMD_PROMISC` が常時 ON で、フィルタはソフトウェアの IP 比較のみ。

3.10 (16) DHCP が飾り

`dhcp_client.c:340–352` は BOUND まで到達してリースをログに出すが、`tcp_set_ip()` の呼び出し元がゼロで、`net_responder.c` には IP セッタすら存在しない。IP は `192.168.3.100` のハードコード。T1/T2 更新もリース失効も DECLINE/RELEASE もない。

3.11 (17) ユーザ空間が存在しない

全てが EL1 の単一アドレス空間で動く。`TCR.EPD1=1` で TTBR1 は無効 (`system/mmu.c:240`)、ASID なし、`SP_EL0` 不使用、**`svc` 命令はツリー内に 1 つもない**。シェルは 51 個の C 関数ポインタの線形探索 (`shell/shell.c:926–978`) で、ハンドラはカーネル内部を直接呼ぶ。`cc/` が JIT したコードも EL1 特権のまま実行可能ヒープ上で走る。

Tier 0 の (1)(5)(6)(12) が全て「EL0 がないことの緩和策」である以上、いずれ避けられない分岐点である。ただし `/mmio-write` や `/chainload` が**無認証で任意ハードウェア書き込み・任意コードロード**を提供している現状では、まずネットワーク面を締めるほうが先である。

4 Tier 2 — ファイルシステムとエンジニアリング実践

4.1 (18) FAT32 にキャッシュがなく、共有 static が無防備

- `fat32_next_cluster()` (`fs/fat32.c:96–107`) は **1 エントリごとに 512 バイト読む**。N クラスタのファイルを辿るのに FAT 読みだけで N 回の SD アクセス。

- `fat_find_free()` (`:290-302`) はクラスタ 1 個試すごとに 1 セクタ読む → 書き込みは O(クラスタ数^2)。
- `static unsigned char scratch[512]` (`:19`) を 5 つの関数が共有。プリエンプティブなスケジューラの下で `sdmount` プロセス・`kexec_tick`・AVM ストリーミングから到達しうるが、**ロックは一切ない**。
- **一貫性の保証がない**。`fat32_write_file_full` は新クラスタを確保する**前に**旧チェーンを解放するので、途中で電源が落ちるとディレクトリエントリが解放済みクラスタを指す。FSInfo も更新しないので、Linux/macOS でマウントすると空き容量が狂う。
- 削除・mkdir・truncate・rename・LFN なし。ルートディレクトリのみ (パス解決がない)。

4.2 (19) VFS が抽象化になっていない

`include/vfs.h:22-32` は単一の具象 `vfs_node_t`。`file_ops` の関数ポインタもマウントテーブルも inode もファイルディスクリプタも `open/close/seek` もない。`vfs_write` は**オフセット引数を持たず**ファイル全体を置き換える。FAT32 は完全に別 API で、VFS には配管されていない。

4.3 (20) 自動テストが実質ゼロ

これが最大の構造的ギャップである。31k 行・195 コミットに対して:

- `make qemu-smoke` は出力を `tee` するだけで**何もアサートしない**。**しかも現在リンクエラーでビルドすら通らない** (`sd_last_int` / `xhci_msd_*` が未定義。`ccb6437` 時点で既に破綻)。
- `examples_http/test.sh` は `192.168.3.100` 固定の curl 7 発、期待値比較なし。
- CI なし。`panic()` も `assert()` もない。`-Wall` のみ。

`fs/fat32.c` は read/write コールバックを受け取るので、FAT32 イメージファイルに対する**ホスト側テストが新しい抽象化なしで今すぐ書ける**。`fs/vfs.c` と `cc/` も同様。ここが投資効率の最も高い領域である。

4.4 (21) 重複と腐敗

- `str_eq`/`puts_dec`/`puts_hex` の near-duplicate が約 21 箇所。
- アクターランタイムが 3 実装並存 (`system/actor.c` / `system/actorproc.c` / `avm.c` 内蔵)。
- 陳腐化したコメント: `fat32.h:1`「read-only FAT32 reader」(write 実装済み)、`proc.h:12`「No preemption yet」(実装済み)、`shell.c:966`「no scheduler yet」。
- デッドコード: `device/usb/dwc2.c`、`include/qemu_repro_src.h`、`repro` コマンド、ビルドルールから参照されない `examples/` の 11 ファイル。
- **`make install_pi4` は地雷**。`config.txt` を上書きして `total_mem=2048` を落とす。この事実は `NEXT_SESSION_PI4.md` にしか書かれておらず、Makefile 自身は警告しない。

5 推奨着手順序と進捗

順	内容	効果	工数	状態
1	(1)(2) ヒープ分割 + irq_save	クリティカル	30 分	✓
2	(3)(4) パケット長検証 + half-open reap	情報漏洩 / DoS	半日	✓
3	(5) `node->data` NULL ガード	サイレント破壊の代表	半日	✓*
4	(7) 優先度プリエンプション修正	RTOS の看板そのもの	1 時間	✓
5	(8) ready キュー O(1) + sleep 整列リスト	アクター系の実測が動く	1–2 日	✓
6	(12) スタックカナリア	サイレント破壊の封じ込め	半日	✓
7	(14) TCP 分割・再送・ウィンドウ	HTTP が 1.4 KB で壊れない	1 日	✓
8	ホストテスト基盤 + CI	以降の全変更の安全網	半日	一部
9	(6) W^X 修正	`.data`/`.bss` が RWX	半日	✓†
10	(10) genet/xhci の DMA 規律 → `DCACHE_ON`	**最大の性能改善**	2–3 日	—
11	(15) GENET TX リング、50 ms スピン除去	スループット / レイテンシ	1–2 日	—
12	(9) タイムベース、(13)(16) 2 GB 開放・DHCP 適用、(11) `network/` の去就	中期	数日	—
13	(17) EL0/ユーザ空間分離、SVC 境界	OS としての次の段階	大	—

* `/microsd` が空でマウント未成立のため実機未検証（第 4 次時点では解消）。

† NX のみ。`.rodata` の RO 化は未了（第 4 次 参照）。

6　第 1 次改良 — Tier 0 の 5 件

コミット `e6973d5`。ビルド md5 `74d3ff7f` (修正前 `cfeeb18c`)。

#	修正	ファイル
1	`MEM_ALIGN` を 8 → **16** (= `sizeof(struct memblk)`)。分割の余りは 0 か 16 以上にしかならず、`getmem` と `freemem` 双方のはみ出しが**構造的に消える**。副次的に AArch64 が要求する 16 バイトスタックアラインメントも満たす	`include/memory.h` `mem/memory.c`
2	ヒープ操作を `irq_save`/`irq_restore` で保護	`mem/memory.c`
3	ICMP と TCP の長さ検証。`total_len`・`data_off` を実受信長と照合	`net_responder.c` `tcp_server.c`
4	アイドル接続リーパー。half-open 5 秒 / ESTABLISHED 60 秒でスロット回収	`tcp_server.c` `loader/main.c`
5	`cat`/`cp`/`mv` の `node->data` NULL ガード	`shell/fscmd.c`

6.1 ホストテスト基盤の追加 — `test/host/`

```
make -C test/host run
```

カーネルの純ロジックをビルドマシン上で直接走らせる枠組み。`mem/memory.c` をそのまま`#include` し、`critical.h` だけホスト用スタブで差し替える。インクルード順で実カーネルヘッダを使うので、テストと本番が同じ定義を共有する。

この枠組みは実際に仕事をした。目視では (1) の `getmem` 側しか見つけておらず、`freemem` 側の同型バグは 20000 回のランダム churn テストが検出した。歯があることの確認として、修正前のツリーに同じテストを掛けると **5 件 FAIL**、修正後は全 PASS になる。

6.2 実機検証結果

項目	結果
ICMP (リグレッション確認)	`ping` 5/5 応答、パケットロス 0 %
HTTP	`GET /`, `/ticks` 正常。`txfail=0`
リーパーの誤検出	正常接続 10 本で `reaped=0`、`drop_syn=0`
リーパーの DoS 解除	アイドル接続でスロット占有 → **`reaped=3`**、全スロット LISTEN に復帰
ヒープ	free 977733 KiB、**free blocks = 1** (完全合体)
`cat` ガード	**未検証** (`/microsd` が空でマウント未成立)

7 第 2 次改良 — Tier 1 の (7)(8)(12)(14)

コミット `ce106cc` (スケジューラ) と `256dbe9` (ネットワーク)。ビルド md5 `becaf060`。実機検証済み。

7.1 スケジューラ

修正	実測
(7) 走行中プロセスを pop の**前**に requeue し、自分の優先度を比較に参加させる。同優先度のラウンドロビンも自然に得られる	`/rtos-jitter` 10 ms 周期・CPU hog 下で **max jitter 10 µs / ミス 0/100** (協調モード: 5162 µs / 46 ミス)、プリエンプション 1002 回発火
(8) 優先度別 FIFO + 64 bit ビットマップ (`CLZ` 1 発)、sleep は deadline 整列リスト	SMP ベンチ劣化なし (dining 3.99×、primes 3.01×、n-queens 1.80×、`agree=yes`)
(12) `stkbase` にカナリア、全スイッチ点で検査、`/ticks` に `stkbad=`	全負荷試験を通して `stkbad=0`

作業中に見つけた既存バグも同時に修正した:

- `shell.c` の `procdemo` が ready キューから外さずに `PR_FREE` を直書き (次のディスパッチが解放済みスタックへ ctxsw する)。
- `proc_ready()` が `PR_FREE`/`PR_TERM` を受け付ける。
- `proc_ready()` が `PR_SLEEP` を sleep リストから外さずに ready へ入れる (両リストが同じ `next` を共有)。
- `proc_block`/`proc_sleep_us` が NULLPROC を自分の古い SP へ ctxsw しうる。

`test/host/proctest.c` を追加 (14 テスト)。`ctxsw` をスタブ化して実物の `proc.c` を走らせ、毎回ビットマップ／ FIFO ／カウント／ sleep リストの不変条件を検査する。核心のテストは `ccb6437` の順序に対して FAIL する — そこでは prio-50 のタスクが prio-5 の hog に叩き落とされ、実機と同じ挙動になる。

7.2 ネットワーク (14)

- **分割**: コネクションごとの送信バッファ + MSS 単位の `tcp_pump()`。FIN は全バイト ACK 後。実測 `GET /shell?cmd=help` が **2618 バイト = Content-Length 2618** (従来は 0 バイト)、6 回連続で同一 md5、`retrans=0`。
- **再送**: RTO 500 ms の go-back-N、6 回で打ち切り。相手の広告ウィンドウを尊重。net プロセスは RX 割り込みでしか起きないため、wm ループが `tcp_needs_tick()` で必要時のみ起こす (送信は `tx_frame` 共有のため net プロセスに一本化)。
- **`http_build` の `max` 無視**: `body[16384]` + ヘッダを `out[16384]` に書いて約 120 バイト BSS を溢れさせていた。

7.3 `/chainload` が箱を落としていた真因

ステージングアドレス `0x4000000` が、走行中カーネル自身の `.bss` の内側にあった。

```
ステージング先   0x4000000 = 67,108,864
カーネル _end    0x4233B18 = 69,286,168   <- 2.08 MB 上
```

アップロードの全域が生きたカーネル状態を上書きするため、`go=1` を撃つ前、**転送中に必ず死ぬ**。本セッション中に 2 回落とした。チェーンロード実装当時カーネルは約 115 KB で、`.bss` が 69 MB に育つ過程で固定アドレスが飲み込まれた**設計の腐食**である。README とマニュアルはこれを「SD 交換不要の安全な高速反復パス」と謳い続けていた。

修正: ヒープから確保 (必ず `_end` より上)、オフセットの境界検査、`go=1` 時に走行中イメージと重なるステージングを拒否。`system/kexec.c:108` が既にヒープバッファを `kernel_chainload()` に渡しており、実績のあるパターンだった。

実測: 2,008,152 バイトを 23.1 秒で転送、全 123 チャンク ACK、ステージしたカーネルが起動。これで実機検証が SD カードの往復なしで回せるようになった。

7.4 アクター実機ベンチマーク — と、そこで見つかった自作の欠陥

改良の効果を測るため、アクター版 N-Queens (`examples/nqueens_with_gc_eval.py`) を実機で走らせた。N=8 で約 1500 回の spawn、同時生存約 1300 アクターという、`NPROC=2048` の根拠そのものの負荷である。

結果は最初、惨憺たるものだった。N=8 が 30 秒で完走せず、1695 アクターが全員ブロックし、スロットが枯渇した (`spawn_fails=1058`、`send_to_neg=2116`、`lheap_overflows=2946`)。

修理したチェーンロードで**カーネルを 6 種類 RAM 起動して二分探索**した。1 回 25 秒で載せ替えられるので、これがなければ成立しなかった調査である。

検証したカーネル	N=8	結論
旧 `ccb6437` (全変更前)	✓ 2171 ms	基準
HEAD − スケジューラ (旧 `proc.c`)	停止	**スケジューラは無罪**
HEAD − ヒープ (旧 `memory.c`)	停止	ヒープも無罪
HEAD − ネットワーク (旧 `tcp_server.c` 他)	✓ 1326 ms	**ネットワークが原因**
HEAD − 周期呼び出し (旧 `loader/main.c`)	停止	reaper / tick は無罪
HEAD − 受信長検証	停止	長さ検証も無罪
HEAD − 分割送信	✓ **1331 ms**	**分割送信が原因**

真因は `tcp_pump()` が **FIN を「全バイトが ACK されるまで」保留していた**こと。この Pi 4 への RTT は 30–60 ms なので、HTTP 応答ごとに丸 1 往復が上乗せされる。ベンチマークは 50 ms ごとに 3 回ポーリングするため全体が約 1.8 倍遅くなり、N=8 は 30 秒の制限を超えて崩壊していた。正しい TCP は FIN を最後のデータと同じフライトで送る (FIN は次のシーケンス番号を占めるだけで、再送タイマがデータごと面倒を見る)。条件を 1 つ外すだけの修正 (`f07f0a6`)。

修正後の実測 (elapsed ms、いずれも解は全て正解):

N	期待解	旧 ccb6437	欠陥版	修正後 HEAD
4	2	856	1301	**746**
5	10	909	1313	**751**
6	4	956	1353	**772**
7	40	1044	1530	**864**
8	**92**	**2171**	**30 s タイムアウト**	**1332**

測定条件の注記。上表はすべて**チェーンロード起動**での測定である (旧カーネルを RAM 起動して A/B するため条件を揃えた)。**SD から起動した同じ修正版はさらに速く、N=8 = 1028 ms** (N=4..7 は 611 / 607 / 622 / 685 ms)。チェーンロード起動が一貫して遅いのは、4 MB のステージングバッファがヒープに残ることと再配置後のヒープ断片化によるものと思われる。公平な比較 (同条件同士) は 2171 → 1332 ms の **1.63 倍**である。

N=8 で旧比 1.63 倍高速。ready キューの O(1) 化とヒープ修正の効果は、この TCP の欠陥を取り除いて初めて可視化された。ピークアクター数は全 N で 2 — ソルバが即座に自殺してスロットが回るので、50 ms 間隔のサンプラには 2 個しか見えない (崩壊時は 1695 個が滞留していた)。

教訓として記録しておく。最初、症状の見た目から「スケジューラ刷新の回帰」と断定して報告したが、**誤りだった**。当時の A/B は「全変更前」対「全変更後」で、間に 3 コミットが挟まっていた。二分探索して初めて結論が反転した。分割統治を最初に行うべきだった。

7.5 決着した懸案 — /preempt は既存の壊れ

/preempt が空ログを返し cpuA/cpuB が居残る件は、修理したチェーンロードで旧カーネル (ccb6437) を RAM 起動して A/B した結果、**旧カーネルでも同じ**と確認できた (ログは BA / B / AB / 空と毎回変動、期待値 20 文字。子プロセスは呼ぶたび 2 個ずつ蓄積)。スケジューラ変更による回帰ではない。

原因は preempt_demo 自身のコメントが書いている前提 —「shell/NULLPROC コンテキストから実行する」。HTTP 経由だと app ワーカー (prio 1) が呼び出し元になり、子と同格なのでラウンドロビンで先に戻ってしまう。NULLPROC は ready キューに載らないのでシリアルシェルからなら意図どおり動くはず。procdemo が途中で戻るのと同じ構図 (こちらはホスト側で新旧のディスパッチ順が同一と実証済み)。

8 第 3 次 — 3 ボードメッシュと xinu-rpi5 への横展開

本アセスメントは Pi 4 のツリーに対して書かれたが、指摘の多くは Pi 5 (xinu-rpi5, BCM2712 / Cortex-A76) にもそのまま残っていた。ここでは 2026-07-21 の作業を記録する。実施順は「3 ボードをメッシュに載せる」→「アセスメントに沿って rpi5 を改良する」である。

8.1 3 ボードメッシュの成立

メッシュの実体は WiFi ad-hoc (IBSS)、SSID MANET、ch 6、固定 BSSID 02:4d:41:4e:45:54、静的 10.0.0.<n>、HELLO は UDP:5000 の 2 秒周期 (device/wifi/wifi.c)。開始時点では 3 台とも有線/AP 側では生きていたが、**メッシュには 1 ノードも載っていなかった**。

ボード	制御アドレス	開始時点の状態
Pi 4	192.168.3.100 (有線)	HELLO は動作 (`hello_tx=952`) だが `rx=0 peers=0`。`wifi adhoc` 未実行で IBSS セルに入っていない
Pi 3	192.168.3.50:8080 (USB eth)	SD 上のカーネルが HELLO 導入前ビルド。`/manet` が無い
Pi 5	192.168.3.101 (RP1 有線)	走行中カーネルが古く `/manet` も `/shell` も無い

Pi 3 は HEAD をビルドして SD に焼き直し (md5 `6c923e1e`)、Pi 5 は USB スティックを書き換えた。各ノードで `wifi adhoc MANET 6 <n>` 相当を実行した結果、まず **Pi 4 — Pi 3 — Pi 5 の線形トポロジ**が成立した。Pi 3 は両者を見ていたが、Pi 4 と Pi 5 は互いに届かない。実測で裏を取っている:

```
Pi5 -> 10.0.0.3 (Pi3)  ping rc=3   成功
Pi5 -> 10.0.0.2 (Pi4)  ping rc=-1  失敗
```

Pi 3 が両方と通じている以上チャネル/BSSID の不一致ではなく、電波の届き方の問題である。その後ボードを再起動して入れ直したところ**完全メッシュ**になった (Pi 4 `peers=2 ids= 5 3`、Pi 3 `peers=2 ids= 2 5`、Pi 5 から両者へ ping 3/3)。設定は変えていないので、原因は設置・向きだったことになる。

記録すべき副作用。疎通確認のつもりで叩いた `GET /wifi-adhoc` (パラメータ無し) が既定 `n=2` で**即座に実行**され、Pi 3 が Pi 4 と同じ `10.0.0.2` に入った。読み取り専用のつもりの URL が副作用を持つ設計だった。

8.2 Tier 0 の横展開 — rpi5 にも全て残っていた

コミット `08128bf`。ビルド md5 `2a6a81ce`。

項	rpi5 での状況	対応
(1)	`ROUNDMB` が 8、ヘッダは 16 — 同型	`MEM_ALIGN` 16 化
(2)	critical section 皆無	`irq_save/irq_restore`
(3)	`total_len` を実受信長と未照合。TCP は `data_off` も未検証	実受信長 `len` と照合
(4)	**Pi 4 より悪い**。接続スロットが単一で、SYN 1 発で HTTP が永久停止	`last_ms` + `tcp_conn_reap()`
(20)	自動テストゼロ	`test/host` を移植

ホストテストは修正後は全 PASS、`MEM_ALIGN` を 8 に戻した修正前の allocator に掛けると **5 件 FAIL** する。歯があることを確認済み。実機検証は ping 5/5、HTTP、JIT (0..999 の和 = 499500)、メッシュ復帰まで通った。

8.3 TCP 分割・再送 (14) と、そこで見つかった 2 つの別バグ

コミット `1d436da`。rpi5 の応答は `http_resp[1400]` の 1 セグメント固定で、`/fb` には「1 セグメント (≤1464 B) しか送れないのでチャンクに割った」というコメントまで残っていた。Pi 4 の実装を単一接続モデルに合わせて移植したが、**移植しただけでは動かなかった**。検証で判明した 2 つは、いずれもアセスメントに載っていない。

half-close (CLOSE_WAIT) が未実装。相手の FIN は「もう送らない」の意味であって「もう受け取らない」ではない。しかし ESTABLISHED ハンドラは、送信途中の応答が残っていても即座に自分の FIN を返して接続を畳んでいた。リクエスト送信後に書き込み側を閉じるクライアント (`nc`、および HTTP/1.0 一般) では、転送が **in-flight 上限ちょうどの 5840 バイト**で切れる。さらにその FIN がシーケンス番号を 1 消費するため、相手は 5840 送出に対して 5841 を ACK する。結果 `snd_una` が `snd_nxt` を追い越し、ポンプの in-flight 計算が負になり、再送タイマのガードも誤成立して**両方が黙る**。停止時に `retrans=0` だったのはこのためである。

`FIN_WAIT_1` の閉じ条件。ACK と FIN が同一セグメントに載っている場合しか LISTEN へ戻らなかった。実際のクライアントは別々に送るので、単一スロットは 60 秒のリーパーまで塞がったまま — 事実上**1 分に 1 リクエスト**しか捌けない。両イベントを独立に追跡し、自分の FIN が ACK された時点でスロットを解放するようにした。

診断が長引いた一因は自分のミスである。移植時にポンプの送信失敗カウントを落としており、`txfail=0` が「送信は成功している」と読めてしまった。最終的に、接続が閉じた後も直前の送信状態を読める `prev:` ／ `calls:` を `/tcpstat` に足して初めて `una=5841 > nxt=5840` が見え、原因が一意に決まった。

測定	修正前	修正後
1 応答の上限	1400 B	**8324 B** (実測、6 セグメント、retrans=0)
サイズ掃引 1532/2132/4132/8324	1532 のみ	**全て期待値と一致**
8 KB 応答の連続リクエスト	1 分に 1 回	**10/10 連続成功**
8 KB 応答の所要時間 (curl)	—	**54 ms**
小応答 20 リクエスト	—	**1.2 s** (1 件 60 ms)

8.4 `/chainload` — 誤った断定と、その訂正

第 2 次改良で Pi 4 のチェーンロードを直したので、rpi5 でも同じ経路が使えると考えた。staging アドレスが `0x4000000` 固定で、これが走行中カーネルの `.bss` (`0x167000`–`0x43BA680`) の内側にあるという**同一の欠陥**も見つかり、Pi 4 と同じくヒープ確保に直した。

しかしその後、チェーンロードするとボードがネットワークから消える現象が続いた。私はこれを「起動に失敗している」と判断し、セカンダリコアが上書きされる `.text` を実行し続けているという仮説を立てて `kernel_chainload()` にコア退避 (PSCI `CPU_OFF`) を実装した。外れた。次に EL2/EL1 の入り方を疑った。これも `boot.S` が `CurrentEL` を見ているので外れだった。

この判断が誤りだった。ユーザに画面を確認してもらったところ、**HDMI にはデスクトップが出ていた**。カーネルは正常に起動しており、死んでいたのは **RP1 PCIe / Cadence GEM の暖かい再初期化**だけである。Pi 5 の `192.168.3.101` は WiFi ではなく RP1 経由の有線であり (MAC は `device/genet/rp1eth.c:393` のハードコード)、それが唯一の遠隔経路なので、箱は死んだのではなく**黙っていた**。しかも `device/genet/rp1pcie.c:198--204` には、前のセッションが同じ問題に「skip if link already up」で対処を試み、*never made networking survive a chainload* と失敗を記録していた。

コミット `08128bf` のメッセージには「`/chainload` はこのボードでは全く動かない」「README の主張は誤り」と書いたが、**どちらも誤りである**。訂正はコミット `1d436da` に記載した。教訓は技術的なものではない — **目の前にある観測手段 (画面) を使わずに、ネットワークの沈黙だけから「死んだ」と結論した**ことである。

8.5 rpi5 では素直に移植できない項目

- **(6) W^X**。rpi5 の MMU は低位 4 GB を **1 GiB ブロック**でマップしている (`system/mmu.c:112--116`)。カーネル `.text` も `.data`/`.bss` もヒープも同一ブロック内で、**属性を分ける粒度が存在しない**。さらに `cc/cc.c` の JIT は書いた領域をそのまま実行するため、素朴に W^X を掛けると JIT が死ぬ。Pi 4 のような一行修正では済まない。
- **(13) 1 GB 制限**。Pi 4 と違い低位 4 GB は Normal でマップ済みで、`HEAP_END=0x40000000` が保守的に決め打ちされているだけ (`compile/Makefile:109`)。ただしフレームバッファはファームウェアが確保した RAM 上にあり (`device/video/video.c:93`)、拡張には衝突回避が要る。失敗しても静かにメモリが壊れるだけで症状が出るとは限らない。
- **(16) DHCP** は rpi5 では**解消済み**だった。`loader/main.c:78--79` が実際に `tcp_set_ip`/`net_responder_set_ip` を呼んでいる。ただし `main.c:1808` で固定 IP も書いており二重である。

8.6 デプロイ経路について

rpi5 の実機反復は**全て USB スティックの物理書き換え**で行った。1 回の試行に「ビルド → USB を抜いて Mac に挿す → 焼く → 戻す → 電源投入 → 起動待ち」が必要で、得られる情報は多くの場合「起動したか否か」の 1 ビットしかない。この条件下で仮説を当てずっぽうに潰すのは高くつく。実際、チェーンロードの原因究明では 3 回の往復を誤った仮説に費やした。転機は**修正ではなく計測を焼いた**ときで、`prev:` ／ `calls:` を入れた 1 枚が原因を一意に決めた。**シリアルコンソールがあれば、この往復の大半は不要だった**。

9 第 4 次 — 3 ボードすべての改良と、外した仮説の記録

第 3 次に続く同日の作業。3 ボードそれぞれに実測で裏付けた改良が入り、併せて**外した仮説を 5 つ**記録する。外れの記録が本節の主眼である — どれも「確認すれば分かることを確認しなかった」型で、実機往復の費用を直接押し上げた。

9.1　Pi 4 — W^X をイメージ全域へ (項目 6)

コミット `118d5ff`。L3 テーブルが 1 枚しかなく、覆うのは `_start` を含む 2 MiB ブロックだけだった。イメージはとうに超過しており、`_data = 0x26a000`、`_end = 0x4263b18` に対し **0x200000 より先は L2 の 2 MiB ブロックに落ちて RW+X** — `.rodata` の末尾と `.data`/`.bss` 全部、約 65 MB である。DMA リング・`proctab`・`smp_stack` がそこに載っている。

`KIMG_2MB` 枚の L3 で `[kern_2mb, heap_start)` を覆い、イメージが配列を超えたら `g_wx_uncovered` で報告する（従来は黙って劣化した）。ヒープは意図的に RW+X のまま — `cc/cc.c` の JIT はヒープに書いて分岐する。

設定ではなく強制を測る `/wx` を追加した。既存のトラップガード (`safe_call()`) の下で 3 つ試験するので、フォールトしても箱は落ちない。SD からのコールドブート:

```
uncovered=0 bytes
write .rodata : succeeded (RO NOT enforced)
exec  .bss    : FAULTED (NX enforced)
exec  heap    : succeeded (JIT still works)
```

SMP 4 コア・N-Queens N=8 で 1.79 倍、ping 5/5、ヒープ完全合体を確認。

`.rodata` の RO 化は入れていない。一度試すとネットワークが落ちたが、何が書いているかを特定できていない（該当範囲は組込み LLM の `model_data` が大半）。フォールトすると `/fault` を読む経路自体が死ぬので、リモートでは追えない。**検証できた半分だけを出し、推測の半分は出さない**という判断である。

9.2　Pi 3 — HTTP 失敗率 14 % → 0 % (項目 3、4、9)

コミット `2bb076a`、`7e15393`。

項目 3 の最も重い形。`tcpRecv()` が `tcp->offset` を一切検証しておらず、`offset2octets()` は 0–60 を返す一方ヘッダのみのセグメントは 20 バイト。`tcpSeglen()` は `ushort` での素の減算なので、過大な offset で**セグメント長が約 64 KB に巻き上がり**、`tcp->data` もフレーム外を指す。歯止めは `TCP_IBLEN`(65528) との窓クランプだけ＝実質無し。**リモートからの約 64 KB 範囲外読み出し**である。入口 1 箇所の検証で `tcpRecvOpts()` の経路も同時に塞がる。

項目 9。`clkticks` はミリ秒でも単調でもない — `clkhandler()` が `CLKTICKS_PER_SEC` で 0 に戻し `clktime` を進める、秒内カウンタである。アクターランタイムはこれを生で読み、さらに「100 Hz だから」と 10 倍していた。実際は 1000 Hz なので**経過時間が 10 倍ずれ、毎秒の境界で破綻**し、GC の掃引周期はさらに `/10` していた。`abcl_now_ms()` を導入して 14 箇所の `clkticks * 10` 等を置換。**このランタイムから取った過去のベンチマーク値はすべて無効であり、取り直しが要る。**

そして失敗率の真因。詳細は次節。`NTCP` 7 → 16、`TCP_TWOMSL` 5 s → 1 s。

200 要求 / 10 ms 間隔	**修正前**	**修正後**
失敗率	28/200 (14 %)	**0/200 (0 %)**
中央値	14.6 ms	12.5 ms
p95	1020 ms	1014 ms

約 1 秒の裾は残る。これは TIME_WAIT の値そのもので、要求間隔 200 ms では消える (p95 17 ms)。負荷時に消すには HTTP keep-alive が要る。

9.3 外した仮説 — 5 件

対象	仮説	実測と結末
Pi 3	半開き接続の再送が長い (約 160 秒) ので詰まる	短縮しても**失敗率は不変** (28/200)。p99 は 3 s→1 s に改善
Pi 3	passive open が枯渇し SYN に RST が返る	リスナを 3 スレッド化しても**まったく不変** (28/200)
Pi 3	conf.h の NTCP を書き換えれば増える	**ボードがクラッシュ**。1 行目に GENERATED FILE; DO NOT EDIT
Pi 4	WiFi/MANET がレイテンシを食っている	無効化しても **79–84 ms のまま**
Pi 4	GENET TX の 50 ms ビジーループ (項目 15)	計測すると **tx__wait__avg 1 μs / max 2 μs**。送信は無実

Pi 3 の真因は失敗インデックスを出した瞬間に分かった。失敗は 6, 14, 21, 28, …と**周期がちょうど 7**、そして NTCP は 7。1 接続が TIME_WAIT 満了までデバイスを保持するので持続レートは NTCP/TWOMSL = 1.4/s に制限され、8 本目が確保できずタイムアウトしていた。要求間隔を 400 ms に広げると失敗が消えることでも裏が取れた。

3 つのカーネルで失敗数がちょうど 28 と一致していたのに、私は「何回失敗したか」だけを 3 回続けて見ていた。どの要求がなぜ失敗したかを最初に見ていれば、SD 物理往復 2 回とクラッシュ 1 回は不要だった。

9.4 Pi 4 のレイテンシ — 原因未特定、結論を撤回

コミット f61c710、4d9c2b5。ICMP 2.6 ms に対し HTTP は connect だけで 54–90 ms。送信 1 μs・WiFi 無関係と潰した上で、RX 割り込みから net_proc が走るまでを計測し**平均 36 ms** を得て「これが原因」と報告した。

その結論は誤りである。平均の正体は 130 件中 1 件の 2.67 秒だった。同じ測定をバケットに直すと:

```
wake_hist(ms) <1:91  <2:29  <4:12  <8:4  <16:0  <32:38  <64:7  >=64:1
```

4 分の 3 は 4 ms 未満。16–32 ms に第二の山は実在するが、1 トランザクション 3–4 交換で期待値は約 19 ms にとどまり、ICMP との 77 ms の差を説明しない。誤った平均を根拠に wm へ入れた譲り処理 3 箇所は**すべて取り消した**（36→27 ms の「改善」も外れ値が動いただけだった）。

次の仮説（未検証）：genet の ISR は自己マスクするので、パケットが RX リングに届いても割り込み再有効化まで通知されない。**現在の計測は「割り込みが上がった時刻」を起点にしているため、この待ちが構造上まったく見えない** — 平均で分布を隠したのと同じ形である。

同日 2 度目の同型の誤りだった。Pi 3 は失敗回数、Pi 4 は平均。どちらも集計値が答えを隠していた。**まずインデックスかヒストグラム、平均は後。**

9.5 デプロイ経路が調査の質を決める

同じ日の 2 つの調査が、経路の差でこれだけ違った。

対象	経路	1 試行の費用	結果
Pi 4	**`/chainload` が動く**	数十秒、SD を抜かない	仮説 3 つを数分で潰した
Pi 3	SD 物理焼き	抜く → 焼く → 戻す → 電源	仮説 4 つに往復 4 回、3 つ外れ

Pi 5 のチェーンロードは起動するが RP1 Ethernet が暖かい再初期化に耐えないため遠隔では使えない（第 3 次の訂正どおり）。

9.6 設計側 — `act` 効果の分離

aios-claude `25b7c02` / aipl-line-simulator `bad8ab9`。`ServoMotor` と `ShelfActor` がどちらも `!{mut, net}` で、一方は実機サーボを回し他方は自分の在庫表を書くだけなのに**効果集合から区別できなかった**。`mut`（自分の状態を書く）と **`act`**（外部世界へ作用する）を分けると、配置と複製の規則が効果から機械的に導ける — `act` を持つものは**その機器に繋がったノードで、同時に厳密に 1 つだけ**。フェイルオーバは `act` の排他移譲として定義できる。

型検査は**実際に駆動する 3 メソッドだけ**を検出した (`ServoMotor.write` / `ConveyorActor.feed` / `ConveyorActor.eject`)。`[type] no issues`、`test/aipl_sync.mjs` も申告漏れ 0 件。

ここでも型検査が私の誤りを捕まえた。`dofbot_servo_writes` を名前から「6 軸まとめ書き」と推測して `act` を付けたところ、`Coordinator.run` が `act` を持つという不自然な結果が出た。実装は `return backend().writes` — **発行済み指令数を返すカウンタ**であった（“servo を write” ではなく “servo write の回数”）。

9.7 `/actor/load` の実測

Pi 5 中央値 **37.9 ms** / Pi 4 **114.8 ms**（各 20 回、1743 B の `ServoMotor` を投入し `1 actor live` を確認）。その大半は JIT ではなく HTTP 自体の遅延で、各ボードの通常 HTTP とほぼ一致する。**アクター移動の費用は無視できる** — メッシュのフェイルオーバ見積りは 4–8 秒から 3–5 秒に下がり、支配するのはリース期限である。

最初の測定は無効だった。トップレベル文の無い `.abcl` を使ったため `main()` が空になり、`0 actor(s) live` が返っていた。時間はもっともらしく見え、そのまま報告すれば「アクター投入の費用」として通っていた。**経過時間だけでなく応答本文を見た**のが救いである。

10 未解決の課題

- **`/microsd` は解消した。**第 4 次時点で実機が `OVERLAYS/`・DTB 群・`WIFI.CFG` を列挙する。FAT32 経路が実機でテスト可能になった。
- **メッシュ (MANET) でのアクター実行は未測定。**3 ボードの完全メッシュは 2026-07-21 に成立した (第 3 次、Pi 4/Pi 3/Pi 5 いずれも `peers=2`) が、複数ノードに跨るアクター分散実行のベンチマークはまだ取っていない。なお M14 のアプリ層 (`manet fire/pull/cfg`) は **rpi5 にしか入っていない**ので、案 B の密度実験には Pi 4・Pi 3 への移植が要る。
- `/preempt` は HTTP 経由では動かない (既存。上記参照)。
- `make qemu` / `qemu-smoke` がリンクしない (`sd_last_int`、`xhci_msd_*` が未定義)。`ccb6437` 時点で既に破綻していた。
- 残る優先課題は表の 9–13 — W^X、DMA キャッシュ規律 → `DCACHE_ON`、GENET TX リング、そして EL0 分離。

10.1 3 ボードの残懸念 — rpi5 17 件 / rpi4 11 件 / rpi3 5 件

第 4 次時点の数え上げ。**rpi4 は (6) W^X が解消して 12 → 11 件、rpi3 は upstream Embedded Xinu をそのまま使っているため素性が良く 5 件** (+ 部分該当 4 件) で、ヒープ・スケジューラ・セマフォ・TCP スタックはいずれも本物である。TCP に至っては rpi5 より真っ当で、rpi5 で修正した half-close と `FIN_WAIT_1` の欠陥はこちらには存在しない。

rpi4 に残る 11 件: (9) タイムベース、(10) D-cache OFF、(11) 偽セマフォと `network/`、(13) 1 GB 制限、(15) GENET TX リング、(16) DHCP 未適用 (rpi5 では解消済)、(17) ユーザ空間、(18) FAT32、(19) VFS、(21) 重複、そして **Pi 4 のレイテンシ (原因未特定)**。

rpi3 に残る 5 件: (4) 単一 accept (失敗率は解消、裾は残る)、(6) W^X、(10) D-cache OFF、(17) ユーザ空間、(18) FAT32、(21) 重複。

10.2 xinu-rpi5 側に残る懸念 — 17 件

第 3 次の作業後、rpi5 に残っている懸念を数え上げると 17 件になる。

アセスメント由来 (10): (6) W^X、(9) タイムベース二重管理 (`device/usb/uspi_glue.c:95` が 100 Hz 固定仮定、`cc/cc.c:136` は `CNTFRQ` 由来)、(10) D-cache OFF、(11) 偽セマフォ (`system/xinu_compat.c:86--101` はカウンタ増減のみ) と `network/` の残存、(13) `HEAP_END` 決め打ち、(15) 相当として `rp1eth_tx_frame()` の最大 100 万回ポーリング、(17) ユーザ空間なし、(18) FAT32、(19) VFS、(21) 重複と腐敗。

今回新たに判明 (5):

1. RP1 PCIe/GEM が暖かい再初期化に耐えず、`/chainload` が実用にならない。
2. 接続スロットが単一で、同時 1 リクエストしか捌けない。検証中も繰り返し詰まった。
3. `http_build()` が `max` を `(void)max` で捨てており、`out` に直接書くルート (`/fb` 等) に境界検査が無い。
4. WiFi の再初期化が不安定 (`op 23 status -17` の反復、`IBSS cell up` が出ない)。
5. `/tcpstat` の `prev:` ／ `calls:` は診断のために入れた足場で、暫定。

未検証 (2): リーパーの half-open (SYN_RCVD) 5 秒経路は未実証で、確認できたのは ESTABLISHED 60 秒側だけ。アセスメント (5) の `node->data` NULL 相当が rpi5 の `/fs/cat` 経路にあるかも未確認。

本書は *`docs/OS_ASSESSMENT_JA.md`* の *LaTeX* 清書版である。数値はすべて実機 *(Raspberry Pi 4 / 192.168.3.100)* での測定値。