Xinu の OS としての改良余地アセスメント

AIPL 駆動フィジカル AI（アーム型ロボット）RTOS 化に向けたコードレベル検討

—— rpi3(A53) / rpi4(A72) / rpi5(A76) 実カーネルの精査——

児玉靖司
法政大学経営学部

2026 年 7 月 20 日

Abstract

AIPL でプログラミングし、最終的にアーム型ロボットを **12 台の Xinu ノード**で制御するフィジカル AI 向け RTOS を作る構想の前段として、**現行 Xinu 3 ポートの実カーネルを精査**し、OS としての改良余地を**ファイル: 行**レベルで洗い出した。結論から言えば、**3 ポートは別物のカーネル**である。rpi3 は**クラシック Embedded Xinu**（プリエンプティブ・優先度スケジューラ、EDF/MLFQ/aging/RCU の RT 実験つき）で**唯一 RT の骨格を持つ**。rpi4/rpi5 は AIPL アクタ実行に最適化した**協調型**カーネルで、rpi5 は**プリエンプション皆無**・rpi4 は既定オフのラウンドロビンのみ。本書は 6 機構（スケジューラ／タイマ／割込み／ IPC ／メモリ／ AIPL GC）ごとに「現状の実装」「RTOS 上の限界」「改良案」を対にして示し、最後に**優先順の改良ロードマップ**（rtos_plan の P1–P4 をコード実態で肉付け）と**アーム型 12 ノードへの指針**を与える。

Contents

1 総括

アーム型ロボットの関節制御（例：1 kHz サーボループ）は、**高優先度タスクが即座にプリエンプトして走り、締切を守る**ことを要求する。現行 Xinu の到達度は表1 の通り。**rpi3 のみが RT 型スケジューラ**を持ち、RT 化の自然な基盤である。rpi4/rpi5 は **協調型・非ブロッキング同期・非有界 ISR** であり、そのままでは硬い実時間保証を満たさない。

Table 1: RTOS 観点の 3 ポート比較（実コード根拠つき）

機構	rpi3（クラシック Xinu）	rpi4（独自）	rpi5（独自）
スケジューリング	プリエンプティブ優先度 + EDF/MLFQ/aging `resched.c:78-116`	協調 FIFO；RR プリエンプト既定オフ `proc.c:288-296`	協調 FIFO；**プリエンプション無**・`prio` 未使用 `proc.h:12`
プロセス上限	`NTHREAD 100`	`NPROC 2048`	**`NPROC 8`**
tick ／プリエンプト源	BCM clk **1000 Hz**→`resched()` `clkhandler.c:52`	CNTP 100 Hz→`resched_request()`	**CNTP 100 Hz→I/O のみ、resched 無**
セマフォ／ mbox	FIFO ブロッキング `wait.c:44`	**非ブロッキング stub**	**非ブロッキング stub** `xinu_compat.c:88`
優先度継承	**無（FIFO 待ち）**	**無（ロック横取り）**	**無**
アロケータ	first-fit	first-fit	first-fit `memory.c:32`
デマンドページング	無	無	**有・ページフォルト** `mmu.c:267`
AIPL GC	—	アクタ回収+ vheap ロック	**タイマ駆動 sweep** `main.c:99`

2 P1 実装成果：rpi5 プリエンプティブ RT 化（実機実測）

本書①スケジューラ・②タイマ/ISR の改良案を **rpi5 で実装し実機検証した**（2026-07、branch `manet-m14-udp`；commit `930a910` Stage 1 ／ `4a42061` Stage 2 ／ `5e0b267` 診断ルート）。要点は二つ。

- **固定優先度プリエンプティブ化**。`ready_pop` を優先度順(同格 FIFO)に、`proc_sleep_us`+`PR_SLEEP`+`proc_timer_tick`(tickless one-shot ＝ `timer_arm_before_us` で CNTP 比較値を次締切へ)、EOI 後 `proc_preempt` で最高優先度 ready を走らせる。
- **ネットワークスタックを ISR から追い出し**。`timer_tick_hook()` は `net_wake()` のみを行い、専用 `net_proc`（prio 1、唯一の非 NULL プロセス）が `genet_rx_tick()` ＝ RX ／プロトコル／HTTP を**プロセス文脈**で回して `proc_block()`。ISR は短く有界に。非再入の AIPL ランタイムは net_proc 単独実行ゆえ保護される。

常設診断ルート `/rtos-jitter` で 1 kHz 周期タスクの**解放ジッタ**を、idle とフルコア計算 hog 負荷下で実測した（表2）。**rpi5 はプリエンプト有効時に最大ジッタ 4 µs・締切ミス 0**。協調（preempt off）では hog が 5 ms ごとにしか譲らず **200/200 全ミス**——これが改良前の姿である。tickless one-shot により rpi5 の **ディスパッチ遅延は約 2 µs**（平均 1002 vs 目標 1000 µs）に収まり、3 世代中**最良の時間精度**に達した。

Table 2: 1 kHz 周期タスクの解放ジッタ（フルコア hog 負荷、実機 HTTP `/rtos-jitter`、samples=200）

ボード	スケジューリング	平均周期	最大ジッタ	締切ミス
rpi5（A76）	**プリエンプティブ+ tickless（P1）**	1002 µs	**4 µs**	**0/200**
rpi3（A53）	クラシック preemptive 優先度	1002 µs	53 µs	0/200
rpi4（A72）	プリエンプティブ（固定 100 Hz tick）	1494 µs	498 µs	0/200
rpi5（A76）	**協調（preempt off）＝改良前相当**	5018 µs	4044 µs	**200/200**

残る改良余地（ディスパッチ遅延）。 rpi5 は tickless で約 2 µs だが、**rpi4 は固定 100 Hz tick のため周期が次 tick へ丸められ約 500 µs のオフセット**を持つ（表2 の平均 1494 µs ／最大ジッタ 498 µs；締切ミスは 0）。rpi4 に rpi5 の tickless one-shot を移植すれば µs 級精度に到達可能で、これが次の P1 課題である。`/preempt` デモも両ボードで確認済み（preempt=1→`ABAB` インターリーブ、preempt=0→`AAAA..BBBB` ブロック実行）。

3 機構別の改良余地

3.1 ①スケジューラ／ディスパッチャ

【2026-07 更新：本項の改良案は rpi5 で実装・実測済み。→§2】 現状（監査時点）。 rpi5 の `system/proc.c:19-48` は**単一グローバル FIFO**（`ready_push` 末尾追加・`ready_pop` 先頭取得）で、`proc_resched()`(`proc.c:177-198`)は先頭を取り出して `ctxsw()` するだけ。`prio` は書かれるが(`proc.c:63,96,142`)**一度も読まれない**。`proc.h:12`「No preemption yet」。rpi4 も同じ FIFO で、プリエンプトは `g_preempt_on` 既

定オフかつアクタポンプ中は抑止（proc.c:293、非リエントラントな AIPL/LLM ランタイム共有のため意図的）。対して rpi3 の system/resched.c:51–121 は **最高優先度 ready を選ぶ**（dequeue(readylist)、キー順キュー）、しきい値 resched.c:78 で同格は FIFO ＋量子 RR、さらに EDF 締切ヒント（s3_pick_deadline）・aging・MLFQ・RCU 非プリエンプト区間まで実装済み。

RTOS 上の限界。 rpi4/rpi5 は「最高優先度 ready が走る」を実現できない。制御タスクが下位の計算/LLM タスクをプリエンプトできず、自発 yield 待ち。rpi5 の NPROC 8 は多関節アームに過少。

改良案。 rpi4/rpi5 に**固定優先度プリエンプティブ**スケジューラを導入（ready を優先度キー付きキュー化、tick で最高優先度をディスパッチ）。基盤は **rpi3 の resched を移植**するのが最短。AIPL 非リエントラント問題は §7 の AMP ／領域分離で解く（RT コアは AIPL 共有状態に触れない）。

3.2 ②タイマ tick ／プリエンプト源

【2026-07 更新：本項の改良案（プリエンプト源化・ネット処理の ISR 追い出し・tickless one-shot）は rpi5 で実装・実測済み。→§2】現状（監査時点）。rpi5 は ARM generic timer 100 Hz（timer.h, PPI 30）だが、device/timer/timer.c:48–56 の timer_irq_handler() は再武装と tick_count++ とフックのみで **resched を呼ばない**。しかも loader/main.c:250–260 の timer_tick_hook() が genet_rx_tick() を呼び、**イーサ RX ドレイン＋ TCP/HTTP スタックを割込みコンテキストで実行**する。rpi3 は BCM clk **1000 Hz**（clock.h:20）で clkhandler.c:52–59 が毎 tick resched() ＝真のプリエンプション。

RTOS 上の限界。 rpi5 は周期タイマを持ちながらスケジューリングに使わず、暴走計算区間が全タスクを止める。**ネットワークスタック全体を ISR 内**で回すため ISR 実行時間が非有界—RTOS が求める「短く有界な ISR」の正反対。

改良案。 タイマ IRQ を**プリエンプション源**に接続（rpi4 は resched_request 済み、有効化＋優先度対応）。**ネットワーク処理を ISR から底半分／スレッドへ**追い出し ISR を短縮。精密締切には **tickless one-shot**（CNTP 比較値を次締切に設定）を併設。tick 安定性（rpi3 の clkhandler.c:52 は sleepq 非空時に resched を飛ばす癖）も是正。

3.3 ③割込み処理と遅延

現状。 GIC（system/irq_dispatch.c、IRQ_TABLE_MAX 256）。スケジューラ臨界区は irq_save/restore（rpi4 proc.c:233–338、短い）。**非有界スピンが真の遅延源：**system/smp.c:30 SMP_WAIT_LIMIT 2000000000（約 2×10^9 反復＝**秒**オーダ）で core0 がワーカ完了を待つ（smp.c:205–216）。rpi4 proc.c:63 AIPL_SPIN_LIMIT 300000 の vheap ロック待ち。

RTOS 上の限界。 SMP ディスパッチ経路に**数秒の最悪スピン**、ISR は重処理（②）。割込み遅延が短くも有界でもなく、スレッド化 IRQ ／優先度も無い。

改良案。 全待ちを**有界・イベント駆動**へ（SMP_WAIT_LIMIT を時間予算＋フォールバックに、スピンを条件変数/セマフォ待ちに）。**スレッド化 IRQ** で ISR 最小化、割込み遅延（latency ＋ jitter）を**実測・上限提示**するハーネスを常設（P1）。

3.4 ④ IPC ／同期

現状。rpi4/rpi5 のセマフォ／ mbox は非ブロッキング stub：rpi5 xinu_compat.c:88–97 の wait() は count-- して即 OK、負でも**ブロックしない**; signal() は count++ のみ。mbox もその上に構築。AIPL アクタの実 IPC は cc/cc.c:527–532 の**単一ロックフリーリング**で、cc.c:532「mailbox full: drop」＝ **溢れると黙って捨てる。優先度継承はどこにも無い：**rpi4 の vheap ロックは AIPL_SPIN_LIMIT 超で**ロックを横取り**（proc.c:369–382、相互排他を破る）、rpi3 セマフォは待ちを **FIFO** で並べる（wait.c:44）。

RTOS 上の限界。 **非有界の優先度逆転**が構造的に可能。rpi4/rpi5 の「セマフォ」はブロックしないので RT の相互排他・条件同期に使えない。制御用途で**メッセージをドロップ**するのは不可。

改良案。 RT パスに**ブロッキング・優先度対応キュー＋優先度継承**（PIP）または **優先度上限プロトコル**（PCP/immediate ceiling）を導入。アクタ mailbox に**有界＋バックプレッシャ**（満杯時は送信側をブロック or 明示エラー、決して黙ドロップしない）。rpi3 セマフォ待ち行列を**優先度順**へ。

3.5 ⑤メモリ管理

現状。 rpi5 `mem/memory.c:32-62` の `getmem()` は **first-fit**（自由リストを走査、`memory.c:40`）で O(n)。**rpi5 はデマンドページング：**`system/mmu.c:207-286` が VA 窓 `[0x8_0000_0000,+4MiB)` を L3 無効のまま置き、初回タッチで `vm_fault()`（`mmu.c:267-286`）がフレーム確保＋ **4 KiB ゼロ埋め**＋ `dsb/tlbi/isb`、`exception.c:55-74` で再実行。プール 512 フレーム、枯渇で致命。RT メモリを**ピン留め／事前フォルトする API は無い。**

RTOS 上の限界。 first-fit は**データ依存・断片化依存の非決定的遅延**。デマンドページングは**ページフォルト＋ゼロ埋め＋ TLB 保守**を制御ループに注入。硬い制御には致命的。

改良案。 RT 用に**固定ブロック／プール（O(1)）**アロケータ、RT タスクメモリの **ピン留め・事前フォルト（フォルト禁止）**、RT 領域を**デマンド窓の外**に置く。アクタ・メッセージは**事前確保**（RT パス無割当）。

3.6 ⑥ AIPL ランタイムと GC

現状。アクタ実行は `cc/cc.c`（`g_alive[]`/`g_protect[]`/`g_nact`）。**周期 GC が存在：**rpi5 `loader/main.c:92-106` の `gc_drive()` が `genet_rx_tick()`（＝ 100 Hz タイマ／ wm ループ）から約 8 s 毎に呼ばれ、`cc_actor_gc` で idle>20 s のアクタを回収。`cc.c:563-577` の `gc_sweep_core()` は**全アクタを走査**（O(actors)、stop-the-world）。値メモリは `g_vheap[32768]` の **bump アリーナ**（`cc.c:185-192`）で**実行中は回収されず**、32 KiB 枯渇で `vheap_alloc` 失敗。

RTOS 上の限界。 制御ループが同一ランタイムを共有すると、約 8 s 毎の **O(actors) sweep** に割り込まれ得る。rpi4 は cc 実行を vheap ロックで囲む（`proc.c:340-386`）ため GC/ヒープ競合がスピン/横取りを誘発。bump vheap は**漸進回収なし・32 KiB 上限**で非決定的。

改良案。 **RT 制御パスを AIPL 値ヒープ／ GC ドメインの外**に出す（RT アクタは無割当・事前確保）。または**インクリメンタル／リージョン GC** で停止を有界化。best-effort 領域と RT 領域の**ヒープ分離**。GC を**タイマ ISR 経路から外す**。

4 改良ロードマップ（優先順、コード実態版）

rtos_plan の P1–P4 を、上記の実コード所見に対応づける。

P1 プリエンプティブ固定優先度スケジューラ＋タイマ堅牢化＋ µs ジッタ計測（①②③）。**rpi3 の `resched` を基盤**に、rpi4/rpi5 の tick をプリエンプト源へ接続、ネットワークを ISR から追い出す。**受入：**1 kHz 周期タスクの release jitter を µs 計測し上限内、割込み遅延を実測提示。

P2 決定性の確保（④⑤⑥）：ブロッキング優先度キュー＋**優先度継承**、非有界スピン（`SMP_WAIT_LIMIT` 等）撲滅、RT メモリのプール化＋ピン留め、RT パスの AIPL GC ドメイン分離。**受入：**RT パスで GC/フォルト/非有界待ち/優先度逆転が 0。

P3 AMP パーティショニング＋安全：制御コアを裸で隔離（非リエントラント AIPL 問題を根本回避）、ウォッチドッグ＋締切超過 → セーフトルク、Capability ＋ MMU 領域でアクタ隔離。

P4 決定的ノード間通信＋時刻同期・分散スケジューラ・AIPL の timing/effect 型を整備し 12 ノードへ。WiFi は非 RT 副系、制御は有線フィールドバス（CAN/EtherCAT/TSN）＋ PTP。

5 アーム型 12 ノードへの指針

非リエントラントな AIPL ランタイムと硬い制御を両立する鍵は **AMP（役割固定）**である。「12 Xinu」構想は**関節／サブシステムごとに独立 RT パーティション**と自然に一致する：各関節ノードは **RT 制御コアを AIPL 共有状態から隔離**し、無割当・事前確保のサーボループを固定優先度で回す。計画・知覚など best-effort は別コア／別ノードで AIPL 駆動。ノード間は **決定的フィールドバス＋ PTP 時刻同期**、締切ミスは**局所セーフ状態**に封じ込める。AIPL 側は**周期・締切・WCET を型／効果で宣言**し、コンパイラが優先度付き RT タスクへ写像・スケジュール可能性を静的判定する（差別化点）。

6 結論

現行 Xinu を実コードで精査した結果、**rpi3 が唯一 RT 型スケジューラを持つ**が優先度継承と優先度順待ち行列を欠き、**rpi4/rpi5 は協調型・非ブロッキング同期・非有界 ISR・デマンドページング・タイマ駆動 GC** で、そのままでは硬い実時間保証を満たさない。改良の投資対効果が最も高いのは **P1（プリエンプティブ固定優先度＋タイマ堅牢化＋ジッタ実測）**で、以降のすべての基礎になる。基盤としては **rpi3 の resched を出発点**に、AMP による RT ／ best-effort 分離で AIPL 非リエントラント問題を回避する道筋が最も現実的である。