AI時代のサーバープログラミング

～ 2年間苦しんだバグの原因が 1週間でわかった話 ～

IIJ 技術研究所

日比野 啓

自己紹介

- **名前**: 日比野 啓
- **所属**: IIJ 技術研究所
- DNS の研究開発として、bowline というフルリゾルバを Haskell で実装
  - DNS の最新動向への追随と実証実験環境の整備
  - DNS 関連ソフトウェアの整理と部品化
  - 社内の要望に応じたカスタマイズ対応

調査やデバッグに AI エージェントを活用する事例紹介

フルリゾルバ bowline のトラブル調査やデバッグに AI エージェントを活用した

- 更新したら SIGILL で落ちるようになった - 軽めの調査
- 暴走してメモリを使いつくす - 難しいバグ その1
- worker が停止したままになる - 難しいバグ その2

AI エージェント(Claude Code)との対話をもとに紹介する

> **worker**: bowline の中でクエリを処理するスレッド

依頼から 8 分で原因を特定

私 VM の CPU は x86-64-v2-AES。バージョンを上げたら、起動時に SIGILL で落ちます。

私 更新前後のバイナリとソースを置きました。原因を特定してください。

claude 2 つのバイナリの、ライブラリのバージョンと含まれる命令を比べます。

claude 原因は依存ライブラリ crypton の更新です。C のコード全体が AVX2 前提でコンパイルされていました。

claude qemu で確かめました。v2 相当の CPU では起動時に SIGILL、v3 相当では起動します。

> qemu-user を起動して SIGILL の再現を確認

> **x86-64-v2-AES**: 仮想マシンに見せる CPU の型。AES-NI はあるが、AVX2 は無い
> **AVX2**: 新しめの CPU が持つ拡張命令。持たない CPU で実行すると SIGILL になる

分かったこと	原因はサーバー実装のコードではなく、依存ライブラリのビルド設定だった。
同定の証拠	VM で見た現象は、v2-AES の CPU で起動時に SIGILL。qemu の v2 相当でも、同じく起動時に SIGILL。v3 相当では起動する。
出力の理解	原因の説明と、それを確かめた実行の結果は別のもの。両方がそろっているかを見る。

暴走してメモリを使いつくす

> **症状**: 動いている途中で突然暴走し、急速にメモリを使いつくす。何も答えなくなる。

> **難しさ**: 頻度が低く、手元で再現が困難。再現方法が分からない。スタックの伸長から再帰ループを疑うが、該当コードがわからない。スタックの伸長が起きたスレッドのコードに原因は無い。(後から判明)

> **調査の期間**: 2026-10-13 から 10-18。

> **流れ**: 再現の試みと本番の記録の解析、リビジョンの確定、手元での再現、仮説の訂正、仕組みの理解、同定。

> **結論**: GHC の RTS が、ほぼ空のスタックチャンクを足し続ける無限ループに入る。必要な空きが 1 個のチャンクに収まらないため。引き金は RTS オプション -kc2k。bowline の実装が直接の原因ではなかった。

再現を試み、元の観測環境のイベントログを解析

私 暴走するバグがあって困っています。発現すると、急速にメモリを使いつくします。

私 クエリを流しながら reload を繰り返して、暴走の再現を試みてください。

> 環境を用意して実験してもらう / reload で暴走したことがあった

claude 60 分回しましたが、暴走は再現しませんでした。

私 メモリ 2 GB の VM で取れた、暴走 2 回分のイベントログを置きました。

claude メモリを使いつくしたのは、worker 1 本のスタックでした。伸長は 30 万件、600 MB です。

> イベントログから、スタック伸長が起きているスレッドを検出している

claude 無応答の原因は、そのスタックをコピーする 37〜101 秒の GC です。この 2 回は reload が引き金ではありません。

> **イベントログ**: GHC の RTS が出す、スレッドや GC の動作の記録
> **スタック伸長**: スレッドのスタックが足りなくなったときに、広げる処理

分かったこと	クエリと reload の組み合わせでは、60 分では再現しない。メモリを使いつくしたのは worker 1 本のスタック。観測データでの事象は「reload が引き金」ではない。
同定の証拠	観測した現象が数字で確定した。伸長 30 万件、600 MB、長い GC。以後はこれを比較の基準にする。仮説と再現はまだ無い。

イベントログの内容からリビジョンを逆算する

claude イベントログに出ている文字列が、いま読んでいるソースにありません。暴走したバイナリは、どのリビジョンのものですか？

私 リビジョンはタグ pre-release-20250916 か 20251105 のはずです。

claude どちらでもなさそうです。ログの書式が一致するのは、調査用ブランチのコミットだけです。

私 そのブランチは、この検証のために作ったものです。合っています。

> claudeのアドバイスで記憶違いが修正できた

私 ほかにも圧縮したイベントログが数十 GB あります。調べられますか？

claude 圧縮したまま読む道具を作ります。

> 合計 168 億件を調べた結果、暴走は 3 回。複数のリビジョンにまたがっていた

分かったこと	暴走時のイベントログに対応するリビジョンが分かった。暴走は 3 回あり、複数のリビジョンにまたがる。
同定の証拠	観測した側の記録が 3 回分に増えた。2 回はリビジョンが確定、1 回はリビジョンを限定。
出力の理解	予想と違う答えを、すぐに誤りと決めつけない。自分だけが知っている事実と照らして判断する。

手元で再現、原因の推定は仮説段階

私 正常な worker と暴走した worker の、記録の違いから場所を推定したいです。

claude 評価の途中のスレッドが非同期例外で中断されると、待っていた別のスレッドが続きを引き継ぎます。ここが関係しそうです。

> **直接は実装していないものが、結果として実行される！**

私 非同期例外が関係するなら、reload は必要だと思います。

claude クエリのタイムアウトでも、非同期例外は起きます。

私 たしかに。reload は外してよいです。

> 原因の推定から、よりよい方針を提案してくれる

claude 暴走を再現しました。原因は、タイマーの表に未評価のサンクが積み重なることです。

> **非同期例外**: ほかのスレッドから投げ込まれる例外。タイムアウトやスレッドの停止に使う
> **サンク**: 評価途中のプログラム
> **タイマーの表**: 登録中のタイムアウトを管理する表。GHC のライブラリが持つ

分かったこと	reload なしでも、クエリのタイムアウトだけで暴走を再現できる。
同定の証拠	再現したバグが初めて手に入った。本番と同じ現象かは未確認。
出力の理解	筋の通った説明だが、「積み重なっている」ことを測っていない。断定の形をした仮説。

証拠から仮説を修正

私 小さいプログラムで実験して、再現条件を特定してください。

claude 訂正します。サンクは積み重なっていません。深さは暴走中も 2 のままでした。

claude スタックは 9 万 5 千個のチャンクで、ほぼすべてが空でした。深い再帰ではありません。

claude RTS が、ほぼ空のチャンクを足し続ける無限ループです。原因は RTS オプション -kc2k です。

> 原因らしきものを見つける / RTS が無限ループしてしまう

私 結局、bowline のリビジョンは無関係だったということですね。

> **RTS**: GHC のランタイム。スレッド、メモリ、GC を管理する
> **チャンク**: スタックを構成する断片。スタックは複数のチャンクをつないで作る
> **-kc2k**: チャンクの大きさを 2 KB にする RTS オプション

分かったこと	正体は RTS の無限ループで、引き金は -kc2k。bowline のリビジョンが直接の原因ではない。
同定の証拠	仮説を測定で確かめた。サンクの深さは 2 のまま、チャンクはほぼすべて空。bowline を外した小さなプログラムでも起きた。
出力の理解	仮説は測れば否定できる。測定の結果を出させて、初めて結論として受け取る。

暴走 図

空のチャンクが増え続ける

① 別スレッドが、中断されたサンクを引き継ぐ

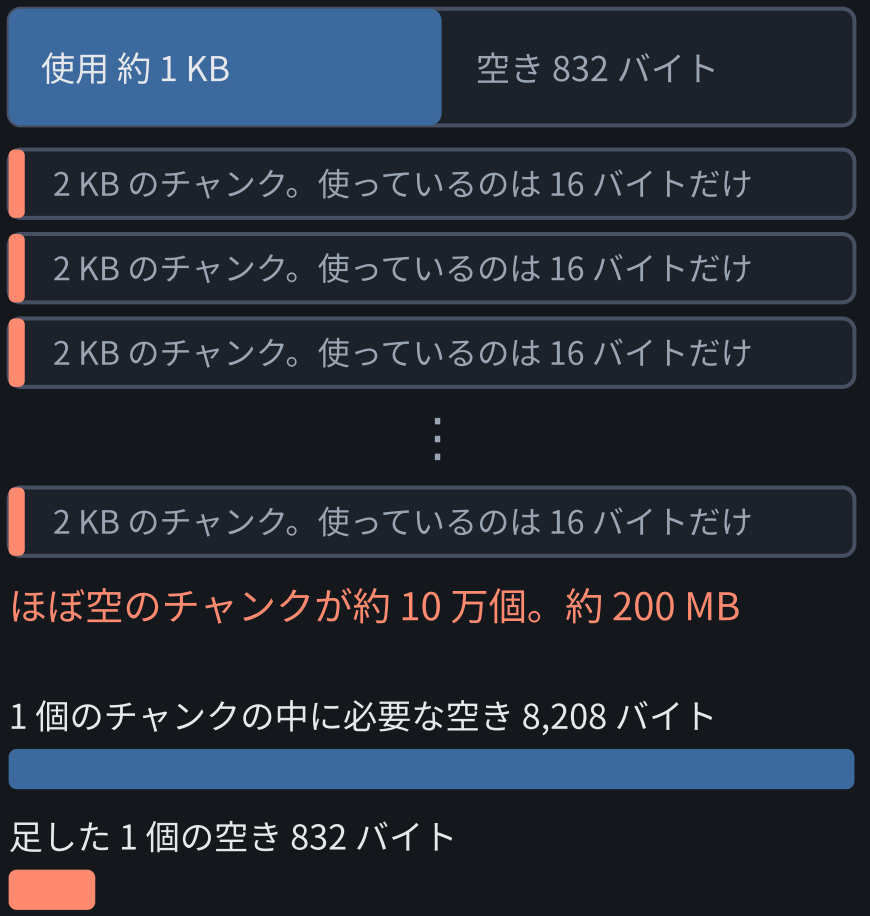
② 今のチャンクの空きを検査する
1 個のチャンクの中に 8,208 バイト以上の空きが要る

③ 足りないので、新しいチャンクを足す
大きさは 2 KB (-kc2k)。1 KB 分を移す (-kb1k)

④ 足した 1 個の空きは 832 バイト。足りない
空のチャンクが何個あっても、空きは**合算されない**

繰り返し
終わらない

1 本のスレッドのスタック



ほぼ空のチャンクが約 10 万個。約 200 MB

1 個のチャンクの中に必要な空き 8,208 バイト

足した 1 個の空き 832 バイト

プロファイル無しでも再現するか

私 チャンクを足しているのに、空きが増えない理由が分かりません。

claude 再開には、1 個のチャンクの中に 8 KB の空きが要ります。足すチャンクは 1 個 2 KB で、何個足しても空きは**合算されません**。チャンクを大きくする判定も、この大きさの組み合わせでは働きません。

私 大きさの比で決まるなら、-kc4k -kb2k でも起きそうに見えます。

claude 実測でも起きました。

私 今の再現はプロファイル付きのビルドです。本番に近い形でも、証拠を集められますか？

claude プロファイル無し、本番と同じ RTS オプションで、約 2 時間後に再現しました。スタックは 10 万個の空のチャンクでした。

> 元の観測環境と同じようにプロファイルのスイッチを外しても再現するか確認する

> **-kb**: チャンクを足すときに、古いチャンクから新しいチャンクへ移す量
> **プロファイル付きのビルド**: 性能解析用の情報を埋め込んだビルド

分かったこと	必要な空きは 1 個のチャンクの中に要る。小さいチャンクを何個足しても、空きは**合算されない**。プロファイル付きのビルドだけで起きる現象でもない。
同定の証拠	仮説から立てた予測が実測で当たった。本番と同じ RTS オプションの再現でも、同じ形のスタックが取れた。
出力の理解	理解できない説明は受け取らず、分かるまで聞く。理解できれば、自分で予測を立てて確かめられる。

同定: 元の観測環境と同じ暴走の現象が再現できたか

私 RTS の無限ループでスタックが伸びる。GC でコピーされてメモリを使う。OOM になる。この順でよいですか？

claude ほぼそのとおりです。伸びるのは、ほぼ空のチャンクの数です。

私 元々の話は、スタックが伸びた後の GC の間、何も答えなくなる、だったはずです。

> 元の無応答の現象と一致するか確認している

claude そのとおりです。説明には、その『後』が抜けていました。空のチャンクの山を GC のたびにコピーするので、GC が長くなります。

> 元の観測は、worker 1 本に伸長 30 万件、600 MB。再現は、1 本のスレッドに 10 万個の空のチャンク

分かったこと	RTS の無限ループ、スタックの肥大、長い GC、無応答とメモリの枯渇が、1 本の流れでつながった。
同定の証拠	本番は worker 1 本の伸長 30 万件と 600 MB。再現は 1 本のスレッドに 10 万個の空のチャンク。RTS オプションも同じ。長い GC と枯渇は再現実験では見ておらず、本番の記録と組み合わせた説明。
出力の理解	最初に見た症状まで説明が届いているかを問い直す。最後に自分の言葉で言い直して、理解を確かめる。

worker が停止したままになる

> **症状**: worker が権威サーバへのクエリを行なった際の close が止まったままになる。生きている worker があれば全体としては止まらない。

> **難しさ**: バグを再現するのに数日かかる。IO manager のロック待ちなら、recv も止まるはずだが、止まるのは close だけ。最初の原因はロックとは無関係。(後から判明)

> **調査の期間**: 2026-10-17 から 10-23。

> **流れ**: 症状の説明と犯人探しの道具、負荷の向きの変更、core の解析、起点の確定、道具を外した再現と修正前後の比較、同定。

> **結論**: 三段階で問題の状況になる。
>
> > ① DoT の処理で起きる競合が IO manager の fd ごとの状態を壊す。② その状態破壊が原因となって不整合なロックの状態を作り出す。③ 不整合なロックに触った close が止まる。

IO manager: Haskell で多重 IO を管理するスレッド
ロック: IO manager が fd の表を守るためのロック

close だけが止まる理由と、原因を探す道具

私 権威サーバへのクエリに使った socket の close が、長時間止まるバグがあります。

私 ロックを持ち続ける処理があるなら recv も止まるはずなのに、止まるのは close だけです。

> monitor から止まることは観測できている

claude close は例外が届かない区間で待つので、抜けられません。recv の待ちはタイムアウトで抜けます。

claude 止まる close は被害者です。閉じた fd 番号で読み込み待ちを登録する犯人が、別にいます。

claude fd の操作を、呼んだスレッドの名前つきで記録する仕組み (shim) を作ります。

> shim 経由でシステムコール呼び出しの記録を取る

claude UDP と、異常な応答を返す権威サーバで負荷をかけました。犯人は見つかりません。

> **monitor**: bowline の状態を見るための管理用ポート
> **例外が届かない区間**: 非同期例外を受け付けないようにして実行する範囲
> **shim**: プログラムとライブラリの間に挟み、呼び出しを記録する小さな仕組み

分かったこと close だけが止まって見える理由に、説明がついた。症状が出る場所と、原因を作る場所は別。

出力の理解 「被害者と犯人」は仮説の枠組みで、まだ確かめられていない。見つからなかったという報告も、疑う先を絞る情報になる。

close 2/6

元の観測環境での条件を伝える

私 バグを観測した環境では、PC 3 台のブラウザの名前解決を DoH で行って、数日かけて再現させています。

claude 負荷を変えます。接続を長く保つ DoH を作ります。アイドル切れをまたぐ DoT の接続も加えます。

claude 初めて記録できました。DoT の接続が close した 0.3 ms 後に、reader スレッドが同じ番号へ登録しています。

> **最初の状態破壊**

claude アイドル切れの直後に送る DoT で、この形を狙います。

claude 5 分後の別の例です。同じ形の登録の 49 ms 後に、その番号を権威サーバへの UDP socket が再利用し、その close でロックが失われました。

> 不整合なロック状態が作られる

> **reader スレッド**: 接続ごとに、受信だけを担当するスレッド

分かったこと	犯人はクライアント側の DoT 接続の reader スレッド。閉じた番号への登録、番号の再利用、ロックの喪失がつながった。
同定の証拠	仮説の各段を、実験の記録で確かめた。観測した現象と同じものかは、まだ確かめていない。
出力の理解	AI が知らない現場の再現条件を伝えると、負荷の向きが変わった。2 つの数字は別の例で、一続きの記録として読まない。

close 3/6

止まっていることを AI エージェントが見逃す

私 monitor から見ようとしましたが、止まってませんか？

claude 止まっていました。気づいていませんでした。

> 止まっていることを検出する仕組みが甘かった

私 core を取って、各スレッドの止まり方を解析できませんか？

claude IO manager のスレッド 3 本が、ロック待ちで止まっています。

> 不整合なロック状態に触ると IO manager も止まる - 観測したバグとは別の現象

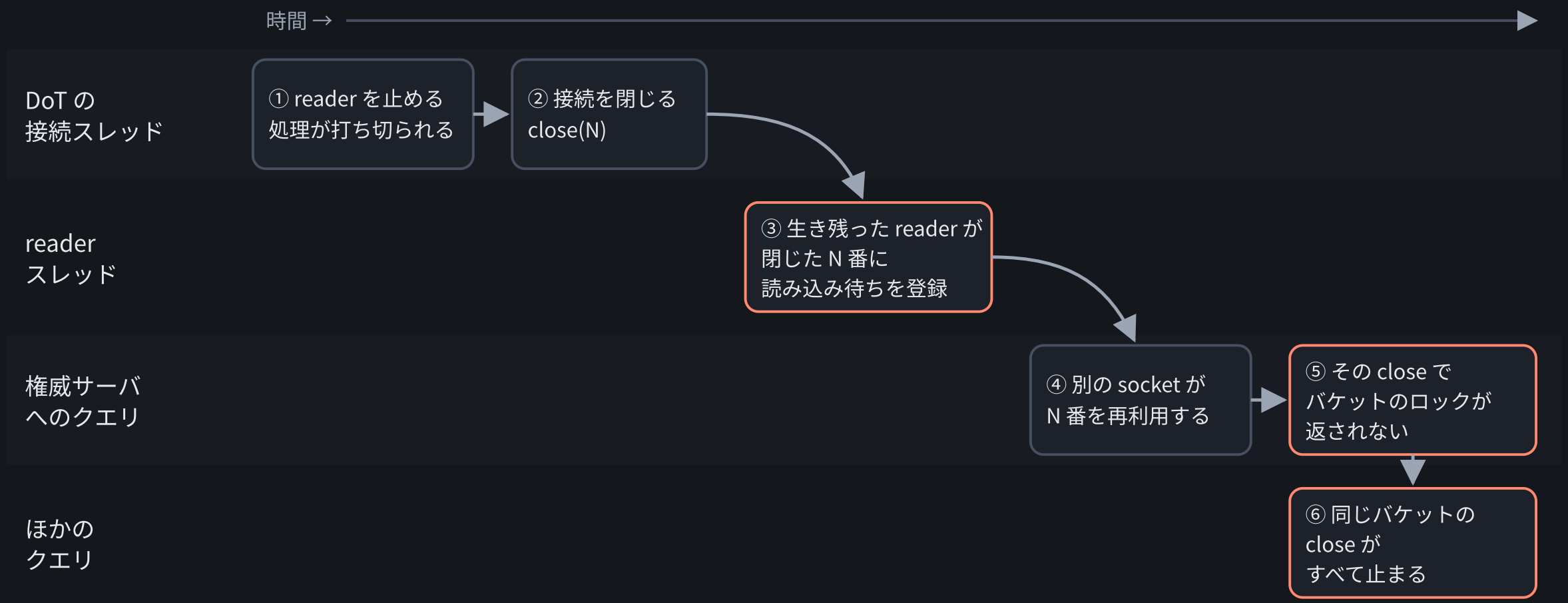
claude タイムアウトの例外を届けるスレッドも止まっています。タイムアウトでも抜けられない証拠です。

> gdb を利用して core を調べ、各スレッドの停止位置を特定している

分かったこと	実験の環境でも実際に止まっていた。止まり方はロック待ちで、タイムアウトでも抜けられない。
同定の証拠	core に残った各スレッドの状態。仮説どおりのロック待ちが、止まった現場から取れた。
出力の理解	AI の報告に書かれていないことも起きている。報告だけに頼らず、自分の目でも見る。

close 図

閉じた fd 番号が再利用されて止まるまで



バケットは fd 番号を 32 で割った余りで決まる。止まった close には例外が届かず、タイムアウトでも抜けられない。

期待と異なり reader が close 後も生き残る

claude reader が生き残った理由が、killThread が中断されたためかどうか確かめるにはイベントログが要ります。

私 『killThread が中断された』の意味が分かりません。説明してください。

claude reader を止める killThread が、ほかの例外で中断されることがあります。

私 イベントログを使ってかまいません。記録と突き合わせて、起きた状況を確定させてください。

claude 30 秒のタイマーが 2 つ同時に満了し、killThread が 3 µs で中断されて、reader が生き残りました。

> 『reader を中断する処理が中断する』ので reader が止まらない

claude 生き残った reader の読み込み待ちの登録が、接続の close と競合しました。登録は close の 37 µs 後で、閉じた番号への登録になりました。

> **killThread**: ほかのスレッドに非同期例外を投げて止める関数

分かったこと	起点は、30 秒のタイマー 2 つの同時満了。reader を止める killThread が中断され、reader が生き残る。生き残った reader の読み込み待ちの登録が、close と競合する。
同定の証拠	イベントログでスレッド単位の順序を確かめた。中断まで 3 µs。close の 37 µs 後に、閉じた番号へ登録。
出力の理解	分からない言葉はその場で聞く。記録と突き合わせて確定させる。

デバッグ用 shim 無しでも再現するか、修正の効果はあるか

私 shim を入れていない bowline でも再現するか、確かめたいです。

> 観測の仕組み(shim)を入れたことで、挙動が変わった可能性を排除したい

私 仮説どおりなら、ロックが失われた後は、1/32 の確率で close が止まるのでは。

claude shim なしで、8 分で再現しました。失われたロックも、閉じられずに残った fd も、7 番のバケットだけでした。

私 reader の別スレッドをやめる修正を入れました。問題が起きる確率が大きく減ったことを、確認したいです。

claude 修正前は 6〜8 分で停止しました。修正版は計 90 分 (60 分 + 30 分) で、停止 0 回です。

> **バケット**: IO manager の表の区切り。fd 番号を 32 で割った余りで決まる

分かったこと	観測の道具が原因を作っていたのではない。原因と見た構造を除くと、止まらなくなる。
同定の証拠	止まるバケットが、仮説の予測どおり 1 つにそろった。修正の前後で、停止までの時間がはっきり違う。観測した現象との同定は、まだ残っている。
出力の理解	「停止 0 回」は、確率が大きく減ったことの確認。競合が残っていないことの証明としては読まない。

同定: 元の観測環境での現象が再現できたか

私 結局、私が知っているバグの発現の証拠は、残せていないのでしょうか？

claude はい、残せていません。実験は、monitor に worker の停止が出ない、別の展開をたどっていました。

私 バグを観測した環境では、close も worker も数時間止まったままでも、クエリは解決できていました。そのときの記録です。

claude 仮説とよく合います。負荷を軽くして、その形を狙います。

私 この monitor の形と、そのときのイベントログと core が取れれば、完璧な証拠と納得できます。

claude 約 12 分で捕まえました。monitor から close 停止が観測できる状況で、同じ時点のイベントログと core がそろいました。

> 過去にバグを観測したのと同様の状況を再現して、証拠のイベントログと core を取得する

分かったこと	それまでの実験は、バグを観測した環境で見たものとは別の展開だった。負荷を軽くすると、monitor から close 停止が観測できる状況になる。
同定の証拠	monitor から close 停止が観測できる状況が、バグを観測した環境の記録と一致した。その同じ時点のイベントログと core で、仮説の仕組みを確かめた。観測、仮説、再現がここで初めて結びついた。
出力の理解	「再現しました」という報告を、自分の知っている症状と比べる。違いに気づけるのは、現場を見ている本人だけ。

まとめ 1/2

バグの同定にあたって注意したこと

- AI エージェントが「再現」した内容が、本当に元の観測したバグなのか確認する
  - 元のバグの観測内容と一致しているか
  - 再現実験用コードだけではなく、元の実装でバグを再現させて証拠を取る

AI エージェントは調査能力が高く、何らかの問題を見つけるが、それが本当に自分が調べている問題か

	観測した現象	仮説	再現	同定の決め手
SIGILL	v2-AES の VM で起動時に落ちる	起動時の初期化に AVX2 の命令	qemu の v2 相当で落ちる	CPU の型と落ちる時点の一致
暴走	worker 1 本のスタックが 600 MB	空のチャンクを足し続ける無限ループ	10 万個の空のチャンク	イベントログに、1 本のスレッドの大量のスタック伸長
close 停止	close と worker が止まり、クエリは解決できる	生き残った reader と fd 番号の再利用	monitor から close 停止が観測できる状況	同じ時点のイベントログと core

自分が観測できたバグの現象。仮説によるバグの説明。実験でバグが再現したときの証拠。 この 3 つが整合したときに、真の証拠が取得できたとする。

参考資料

- 暴走してメモリを使いつくす: https://github.com/iijlab/dnsext/issues/305
- worker が停止したままになる: https://github.com/iijlab/dnsext/issues/306
- Resuming an AP_STACK can make the RTS add stack chunks forever, depending on the RTS parameters
  - https://gitlab.haskell.org/ghc/ghc/-/work_items/27983
- bowline: https://iijlab.github.io/dnsext/bowline.html