

トラブルシューティングから Linux カーネルに潜り込む

伊藤洋也 / GMO PEPABO inc.

2023.05.24 TechFeed Experts Night#19

GMOペパボ

GMOペパボ セキュリティ対策室

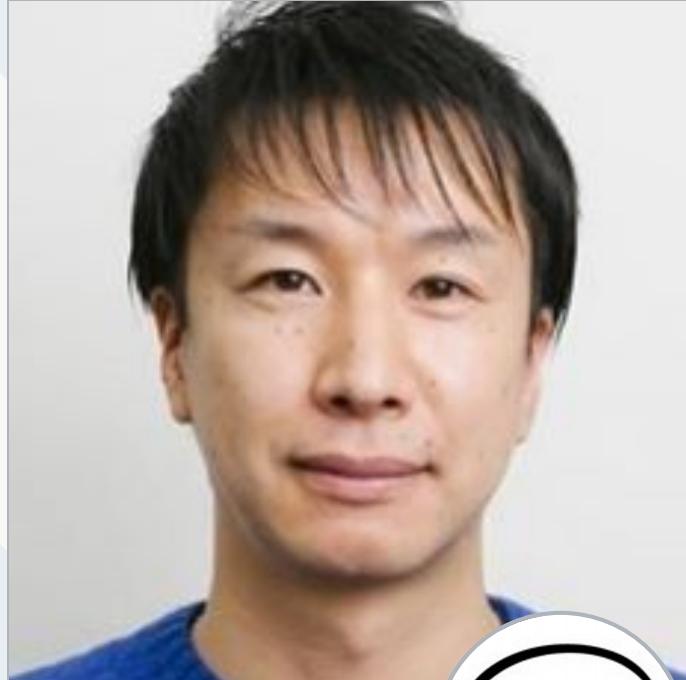
伊藤 洋也

Hiroya Ito

2007年 入社

ホスティングサービス、技術基盤チームを経て
現在はセキュリティ対策室に所属

- GitHub, Twitter : **@hiboma**
- <https://hiboma.hatenadiary.jp/>



トラブルシューティングの話をしよう

『達人に聞く、Linux カーネルコードの歩き方』

登壇の打診をいただいたが、私の場合どんな話をできるだろうか？🤔

- Linux カーネルコミッターではないしなあ ...
- Linux カーネルの開発業務も経験がないしなあ ... *
- Linux カーネルのトラブルシューティングは時々やってる💡

注) 検証や調査のカーネルモジュールやパッチを書くことはあります。でも production で動かすコードを書くことは全然無いです

Linux カーネルのトラブルシューティング

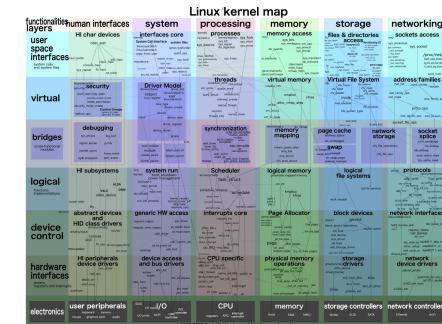
自社サービスの運用で Linux カーネルの問題に遭遇する 🔥

- パフォーマンスの劣化
- カーネルパニック
- メモリリーク
- レースコンディション
- デッドロック
- 脆弱性
- ハードウェアとの互換性問題 *
- ... etc

注) 業務でサーバ構築やハードウェアの保守に携わる機会が無く、私個人はハードウェア / ドライバ周りのトラブル対応経験が全然ありません

トラブルシューティングで Linux カーネルコードまで潜り込む動機

- 問題を解決(緩和)するため*
- 問題を再現するため
 - 再現のヒントを得たい
- 問題を理解するため
 - 理解したうえで同僚にも説明したい
- 問題の再発防止のため
 - 監視に使えるデータを探したり
- 単純に実装に興味がある 😊



注) カーネルのアップデート、ワークアラウンドの実施、サーバの構成変更して 「解決」するケースが多いです。パッチを送って upstream から解決! ... は未経験

トラブルシューティング事例

発表時間の都合、駆け足で説明します ご了承ください 🙏
数年前の話も混ざっております

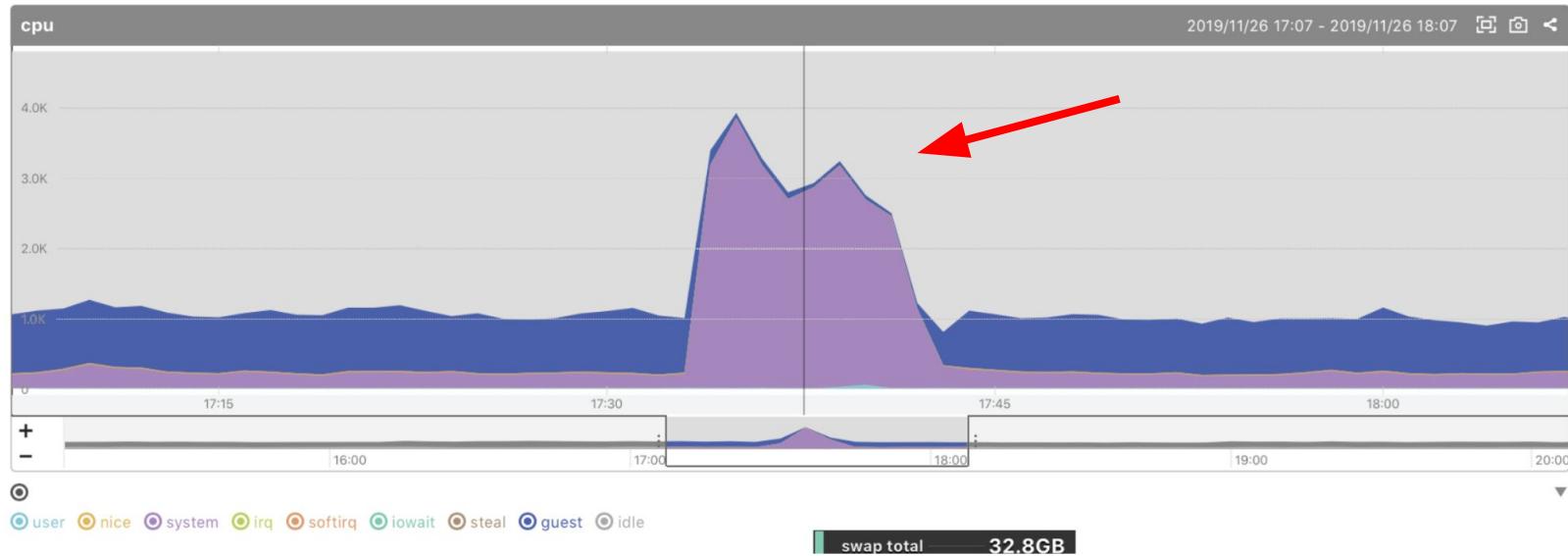
トラブルシューティング事例の舞台

GMOペパボの SaaS サービス / ホスティングサービスを提供する Linux サーバが対象です

- ・ オンプレミス環境のカーネル
- ・ プライベートクラウド (OpenStack / KVM) 環境
 - ・ ゲスト VM のカーネル
 - ・ ホストマシンのカーネル
- ・ パブリッククラウド環境のカーネル

トラブルシューティング事例 (1) CPU %system がスパイク

こういうのはよく遭遇する



トラブルシューティング事例 (1) CPU %system がスパイク

環境 (OpenStack)

- ・ ゲストに vCPU (仮想CPU) をたくさんはやした VM (mysqld) がいた
- ・ ホストで wazuh のプロセス* がセキュリティチェックで様々なファイルを read(2) してた
 - ・  ホストは Ubuntu Xenial で 4.4系カーネル

症状

- ・ ゲスト VM 内の mysqld の監視で、接続断を知らせるアラートが時々出ていた
- ・ 同時にホストの CPU %system がスパイクしていた

注) <https://wazuh.com/> ファイルの整合性監視などを備えた XDR SIEM を提供するセキュリティプラットフォーム (OSS もあります)

トラブルシューティング事例 (1) CPU %system がスパイク

問題

- 1. ゲスト VM でページFAULTが発生する
 - tdp_page_fault() で **struct kvm の mmu_lock** を取ろうとする
- 2. wazuh がファイルを read(2) まくり -> Free なメモリが減り slab の回収処理が走る
 - mmu_shrink_scan() で **struct kvm の mmu_lock** を取ろうとする
- 1, 2 が並行してスピンロックが競合 🔥

解決

- wazuh を cgroup のメモリ制限化に入れた
 - slab 回収処理が cgroup 内だけで走るようにしてページFAULTと競合しないようにした

トラブルシューティング事例 (1) CPU %system がスパイク

perf top でスピンドル周りを調べてた記録

```
Samples: 2M of event 'cycles:pp', Event count (approx.): 13572632562071532
Children      Self     Shared Object      Symbol
- 91.06%    91.06%  [kernel]          [k] _raw_spin_lock
- 78.00%    ioctl
  - entry_SYSCALL_64_fastpath
    sys_ioctl
    do_vfs_ioctl
    kvm_vcpu_ioctl
  - kvm_arch_vcpu_ioctl_run
    - vcpu_enter_guest
      - vmx_handle_exit
        - handle_eptViolation
          kvm_mmu_page_fault
        - tdp_page_fault
          _raw_spin_lock
+ 7.12%    ret_from_fork
+ 2.71%    0x1a5eb
+ 1.67%    0x447dac
+ 1.44%    0x82786
  0.97%    0x8c27
    0x1dbd4
    page_fault
    do_page_fault
    __do_page_fault
    handle_mm_fault
  0.89%    0x7f6c0976ac14
```

```
F 94.50% 0.00% [kernel]          [k] KVM_mmu_page_fault
- 89.72% 89.72% [kernel]          [k] _raw_spin_lock
  _GI_ioctl
  entry_SYSCALL_64_fastpath
  sys_ioctl
  do_vfs_ioctl
  kvm_vcpu_ioctl
  - kvm_arch_vcpu_ioctl_run
    - vcpu_enter_guest
      - vmx_handle_exit
        - handle_eptViolation
          kvm_mmu_page_fault
        - tdp_page_fault
          _raw_spin_lock
        - 0.19% try_async_pf
          _gfn_to_pfn_memslot
          _get_user_pages
          handle_mm_fault
          swapin_readahead
          read_swap_cache_async
          _read_swap_cache_async
          alloc_pages_vma
          _alloc_pages_nodemask
          _alloc_pages_slowpath_constprop_88
          try_to_free_pages
          do_try_to_free_pages
          shrink_zone
        - shrink_lruvec
          - 0.11% shrink_active_list
            page_referenced
            rmap_walk
            page_referenced_one
            _mmu_notifier_clear_flush_young
            kvm_mmu_notifier_clear_flush_young
            _raw_spin_lock
        - 0.07% shrink_inactive_list
          shrink_page_list
          try_to_unmap
          rmap_walk
          try_to_unmap_one
          + 0.07% _mmu_notifier_invalidate_page
          + 0.00% _mmu_notifier_clear_flush_young
  0.00% _raw_spin_lock
  + 0.00% apic_timer_interrupt
  + 1.90% kvm_check_async_of_completion
```

Xiao Guangrong
guangrong@pepper-fuji.com>

トラブルシューティング事例(2) CPU %user がスパイクする

カーネルのバグで CPU %user が高くなるケース。珍しい気がする



トラブルシューティング事例(2) CPU %user がスパイクする

2018年の Spectre / Meltdown 脆弱性対応の修正カーネルがどんどん出ていた頃の話

環境 (OpenStack)

- ゲスト CentOS 3.10.0-693.17.1.el7.x86_64  (問題のカーネル)
- ホスト Ubuntu 4.4.0-119-generic 

症状

- 3.10.0-693.17.1.el7.x86_64  の VM でなぜか CPU %user が高い 🔥
 - 冗長化したロールでカーネルバージョンが混在しており、このバージョンだけ異常を示していた
 - このカーネルの VM が起動すると、なぜか 同じホストの他 VM でも CPU %user が高くなる

トラブルシューティング事例(2) CPU %user がスパイクする

問題

- kernel-3.10.0-693.17.1.el7.x86_64 の **IBRS*** の扱いにバグがあった
 - 勝手に IBRS 機能を有効 (*sysctl kernel.ibrs_enabled=1* 相当)にしてしまう
 - 💡 **IBRS** で Specter V2 の脆弱性を緩和できたのだが、CPU パフォーマンスが低下する副作用 があった
 - 💡 後々、[RHSA-2018:2216](#) としてセキュリティアドバイザリも出ていた

解決

- kernel-3.10.0-693.17.1.el7.x86_64  を使っていた VM 全台でカーネルを更新
 - 1台でも起動すると問題が再発するので根絶しないといけなかった
 - 🥺 VM を別のホストに live-migration すると症状が <伝染> する ... という問題にもなっていたので <根絶> が必要

注) IBRS(Indirect Branch Restricted Speculation:間接分岐の投機的実行機能に対する制約の付加)

トラブルシューティング事例(2) CPU %user がスパイクする

当時の issue を発掘。コードをひたすら追って問題を深追いして仮説を立てた



hiboma commented on May 25, 2018 · edited

...

アセンブラで書いてる箇所を見落としていた

3.10.0-693.17.1.el7

`PER_CPU_VAR(spec_ctrl_pcp)` を `testl` 命令でみている。これは CPU ごとに持つ変数で、ブート時以下のコードで `true` に設定されている

```
// arch/x86/kernel/spec_ctrl.c

if (cpu_has_spec_ctrl()) {
    setup_force_cpu_cap(X86_FEATURE_IBPB_SUPPORT);
    if (!ibrs_enabled && !noibrs_cmdline) {
        set_spec_ctrl_pcp_ibrs(true); // ← ここの中で true にセットしている
        ibrs_enabled = IBRS_ENABLED
    }
    printk_once(KERN_INFO "FEATURE SPEC_CTRL Present\n");
    spec_ctrl_cpu_init();
}
```

このアセンブラのパスがゲストで実行されるタイミングでホストの IBRS に影響を及ぼしているのかなあ

トラブルシューティング事例(3) sysfs のファイルがリークする

cgroup を作ると slab の sysfs ファイルが生えるが、cgroup を消しても sysfs ファイルが残るバグに遭遇

```
root@bionic:~# find /sys/kernel/ -type d | grep hogehoge
/sys/kernel/slab/radix_tree_node/cgroup/radix_tree_node(987:hogehoge-1)
/sys/kernel/slab/dentry/cgroup/dentry(1029:hogehoge-4)
/sys/kernel/slab/dentry/cgroup/dentry(1027:hogehoge-3)
/sys/kernel/slab/dentry/cgroup/dentry(1025:hogehoge-2)
/sys/kernel/slab/dentry/cgroup/dentry(1039:hogehoge-9)
/sys/kernel/slab/dentry/cgroup/dentry(1023:hogehoge-1)

....
```

コンテナ環境を提供するホスティングサービスで遭遇。サービスに特に影響はしていなかったが不可思議で調べた。

トラブルシューティング事例(3) sysfs のファイルがリークする

詳細をまとめてコンテナ技術の情報交換会で発表しました



<https://speakerdeck.com/hiboma/cgroup-and-sysfs-files>

トラブルシューティング事例(4) : TCP: out of memory

dmesg に出てるログを調べるケース。これも よくありますね

```
Dec 17 07:27:02 proxy000 kernel: [ 9774.254080] TCP: out of memory -- consider tuning tcp_mem
Dec 17 07:27:02 proxy000 kernel: [ 9774.399771] TCP: out of memory -- consider tuning tcp_mem
Dec 17 07:27:02 proxy000 kernel: [ 9774.401927] TCP: out of memory -- consider tuning tcp_mem
Dec 17 07:27:02 proxy000 kernel: [ 9774.900174] TCP: out of memory -- consider tuning tcp_mem
Dec 17 07:27:03 proxy000 kernel: [ 9775.152904] TCP: out of memory -- consider tuning tcp_mem
```

トラブルシューティング事例(4) : TCP: out of memory

環境 (OpenStack)

- haproxy を載せた VM
 - OpenStack の L7 ロードバランサーサービス Octavia として動いていた

症状

- haproxy を介しての TCP のデータ送受信が極端に遅い/ 接続が切れる
- 同時に **TCP: out of memory -- consider tuning tcp_mem** の dmesg も出ていた

トラブルシューティング事例(4) : TCP: out of memory

解決

- `net.ipv4.tcp_mem` をチューニングして解決*

発展の課題

- 問題自体は解決したが... 気になることが多い 😕
 - `net.ipv4.tcp_mem` はどう作用するチューニングパラメータなのか?
 - TCP out of memoryになるとカーネル内部で何が起きるのか?
 - `/proc/net/*` 以下のどの数値を見たら監視できるのか?

トラブルシューティング事例(4) : TCP: out of memory

自分で調べ挙げてまとめた。たぶん日本語で一番まとまる記事(自信)…!

2020-06-26

ペパボ トラブルシュート伝 - TCP: out of memory -- consider tuning tcp_mem の dmesg から辿る 詳解 Linux net.ipv4.tcp_mem

トラブルシューティング

セキュリティ対策室の伊藤洋也です @hiboma

過去の障害対応中に遭遇した TCP: out of memory -- consider tuning tcp_mem という dmesg を端緒として、Linux カーネルがどのようにTCPのメモリを管理するのかを調べました。

本エントリのテーマ

- TCP: out of memory -- consider tuning tcp_mem の dmesg
- memory pressure モード
- sysctl net.ipv4.tcp_mem の詳細

を扱います。

最新記事

2023-05-19 メールシステムのリバースプロキシに Nginx を使っているのでご紹介

2023-05-10 Rails Girls Tokyo 15thに会場提供と T シャツスポンサーを行いました！

2023-04-28 PHPKaigi2023 レポート

2023-04-28 「Pepabo Tech Conference 20 春の SREまつり」開催レポート

2023-04-20 Debezium ServerによるChange Data Captureの事例紹介

2023-04-19 RubyKaigi2023にVideo Sponsorとして協賛します & Shibuya.rb RubyKaigi 2023 前夜祭 @ GMO Yours ・ フラスを開催します

2023-04-18 YAMLファイルのkey構造が環境ごとに

<https://tech.pepabo.com/2020/06/26/kernel-dive-tcp-mem/>

トラブルシューティング事例 more ...

- 足早に紹介しました
- ほかにもいろいろな事例があったのですが ...
 - 時間の都合で説明が複雑になりそうなものは取り上げず 

トラブルシューティングで Linux カーネルコードリーディング

- 問題のありそうな関数やログを端緒にコードを眺める
 - 素朴に grep * や GNU Global で読んでいる
- コードを 全部 理解しようとはしない。大変 😊
 - 細部は読み飛ばして、大枠をつかめば問題解決に辿り着けることも
 - たまに、読み飛ばした部分が重要だったりすることが ... ウワー

注) 素の grep を使うわけでなくパターンマッチさせて検索するという意味

Linux カーネルのトラブルシューティングで 見るもの / 使うもの

- dmesg のログ
- perf コマンド
- /proc/\$pid/wchan や /proc/\$pid/stack
- /proc/* とか /sys/* とかのファイル
- sysrq-trigger でダンプ
- strace , gdb
- メトリクス
- ... etc

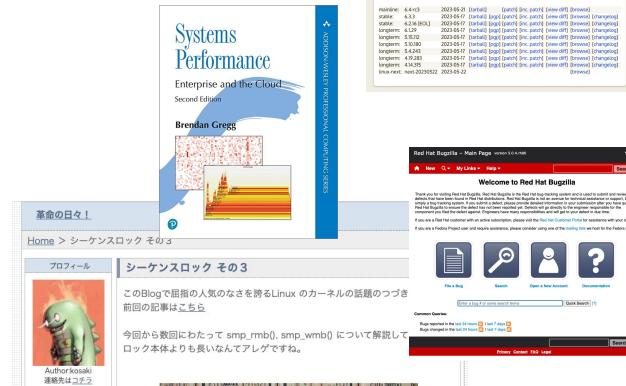
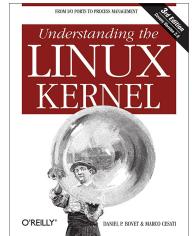
プロファイルとったり、コードを探るためのログやカーネルの関数や統計などを探していく

トラブルシューティング コードを追いつつ情報収集

コードを読みつつ、情報収集

- ・ 現行バージョンと upstream との差分を見る
- ・ パッチ、コミットログを漁る
- ・ 発表スライドを見る
- ・ バグトラッカーを見る
- ・ ベンダーのサポートページを見る
- ・ ブログを読む
- ・ 文献にあたる
- ・ ... etc

2023年なら ChatGPT も選択肢かな?



トラブルシューティング 難しいところ

いろいろ情報集めても確信を得られない時

- ・ 「ほんとに ここが問題のコードなのかな? 😕」
- ・ 「Web でそれっぽい解説や解決方法を見つけたけど、一緒に問題なんだろうか 😕」
- ・ 「パッチを見つけたけど、これでほんとに直るのかな ... 😕」

確信が得られるまでコード読み直したり情報収集をやり直す *

注) 開発環境で問題が再現できれば、確認を得るために試行錯誤できますね。本番環境でしか問題が起きないケースが難しい ... !

トラブルシューティング やりがい

- 問題を解決できるとうれしい
 - サービス / システムが安定するとうれしい
 - 「自分のアプローチや仮説は確かに合っていたんだな！」という充実感もある
- 教科書的でない角度から Linux カーネルの理解できる
- 調べたり試したことが自分の中で「Linux カーネルのモデル *」として定着する

注) メンタルモデルというか、脳内でのトイモデルというか

トラブルシューティングからのチャレンジ

- 既知の解決策やワークアラウンドを何とか探し当てるの 精一杯
- パッチも書いて upstream にコミットして解決する人たちはすごい !!! 達人 !!!
- いつか自分も～ 💪🐧

Thank You!

GMOペパボ

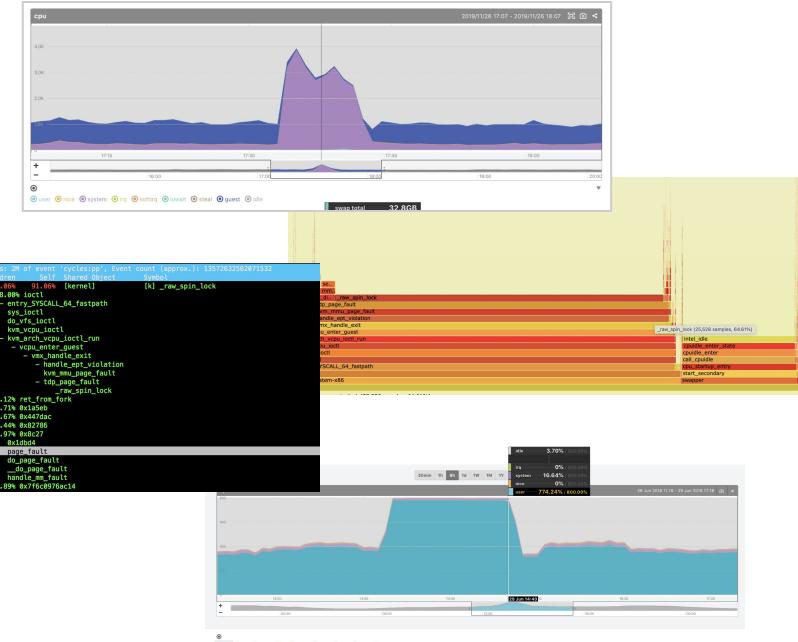
見送りスライド

時間がないのでボツにしました

トラブルシューティングからのカーネルコードへのエントリー

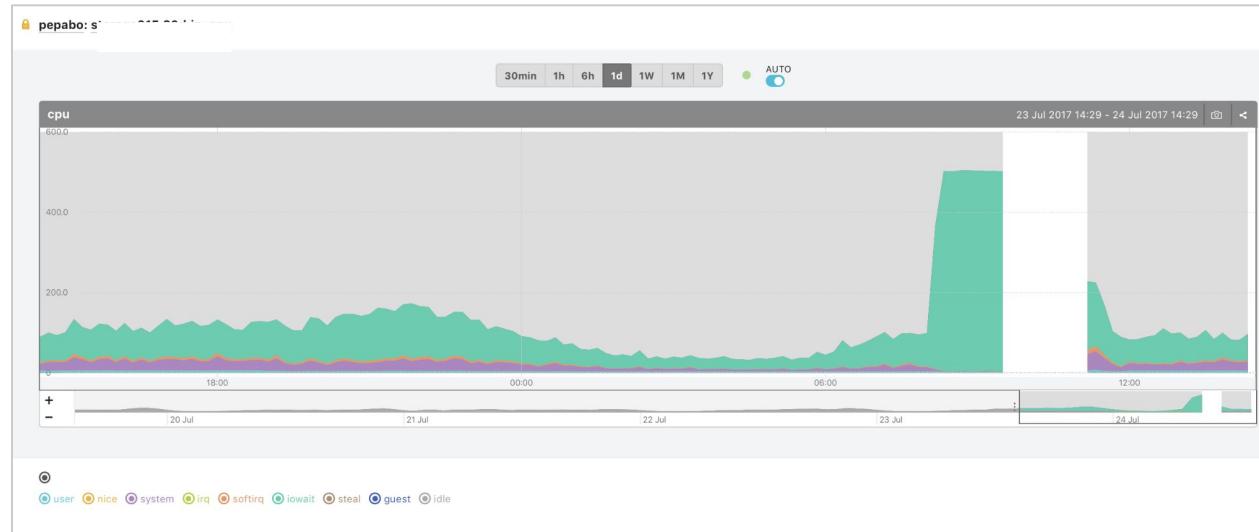
何を端緒にカーネルに潜っていくか？

- CPU のメトリクス
- dmesg のログ
- メモリのメトリクス
- I/O のメトリクス
- 可観測ツールの情報
- /proc 以下の情報
- ...



トラブルシューティング事例：%iowait CPU が刺さるケース

STATE D (TASK_UNINTERRUPTIBLE) なプロセスが出て %iowait が刺さるケース。よく遭遇する



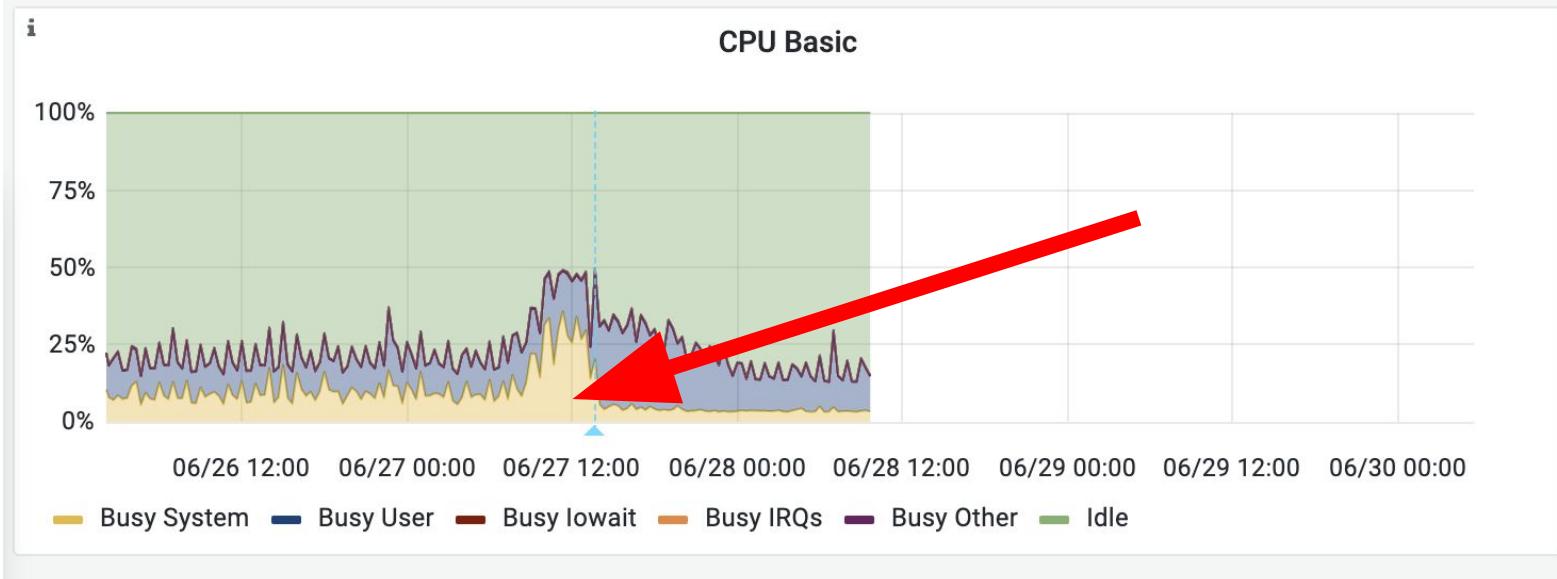
いろんなケースがあるが、XFS のバグで shrink_slab でデッドロックするケースを例に挙げた

<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=7a29ac474a47eb8cf212b45917683ae89d6fa13b>

トラブルシューティング : %system CPU が慢性的に高い

これもよく遭遇する

▼ Basic CPU / Mem Graph



トラブルシューティング : %system CPU が慢性的に高い

環境 (OpenStack)

- vCPU をたくさん生やした GitHub Enterprise Server (*) が VM で動いていた

症状

- 業務時間帯やバックアップを採取する時間に Web UI が重い
- %system CPU も慢性的に高い状態

注) “GitHub Enterprise Server は、エンタープライズ内のソフトウェア開発用のセルフホスティング プラットフォームです”(オンプレ のVM で動く GitHub)

トラブルシューティング : %system CPU が慢性的に高い

問題

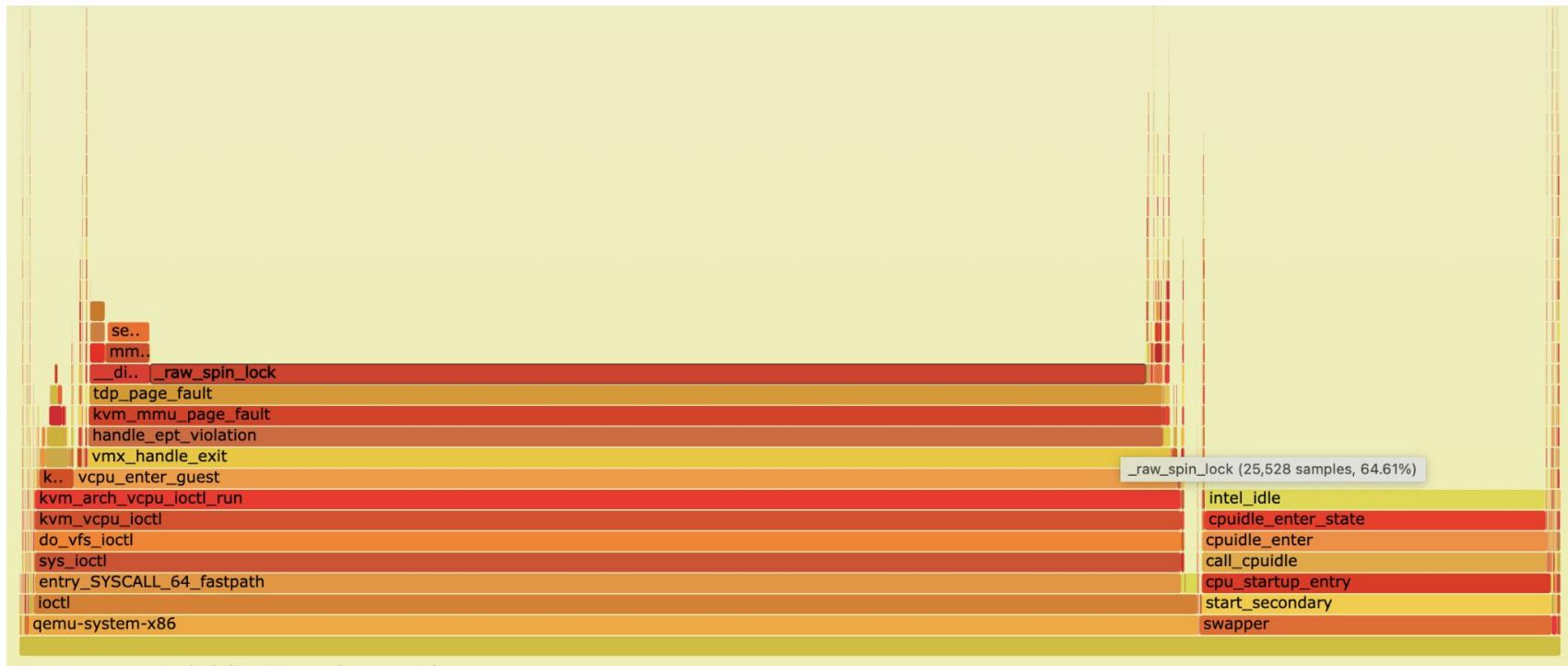
- perf で調べたら ホストの tdp_page_fault() でスピンロックが競合しまくっていた 🔥
- struct kvm の mmu_lock() 獲得部分

解決

- tdp_page_fault() 周りのパフォーマンス改善が入ったupstream の情報を見つけた
 - [Linux 5.15 KVM Defaults To The New x86 TDP MMU, Enables AMD SVM 5-Level Paging - Phoronix](#)
- ホストのカーネルアップデートで改善できた

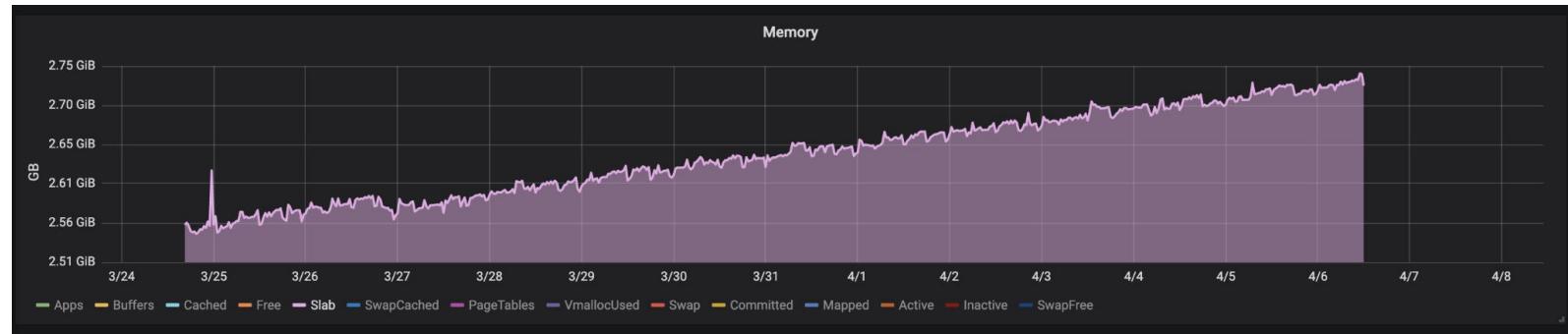
(注) もしかしたら、一つ前の事例も同様のボトルネックだったかもしれない？けど深追いできておらず

トラブルシューティング : %system CPU が慢性的に高い



トラブルシューティング : slab キャッシュのリーク

たまに遭遇する気がする : slab キャッシュがリークする



SUnreclaim な slab (kmalloc-256, Acpi-Operand) がリークしてたケース。

調べてみたけどよく分からず、カーネルのバージョンをあげたら再発しなくなった (Ubuntu Bionic)

トラブルシューティング : dmesg のログから

dmesg に出てるログを調べるケース。よく遭遇する

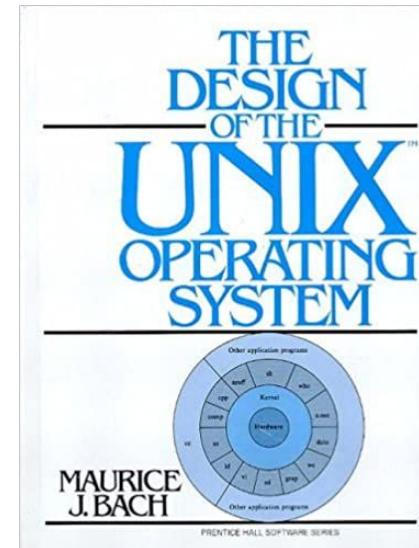
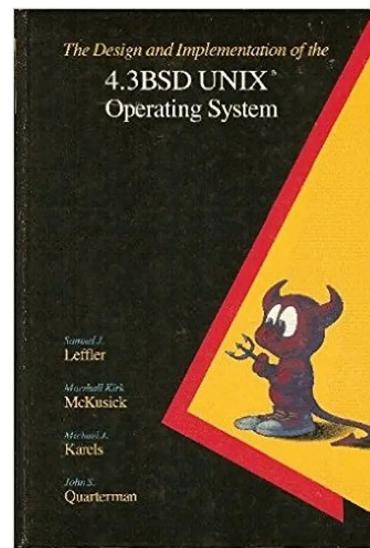
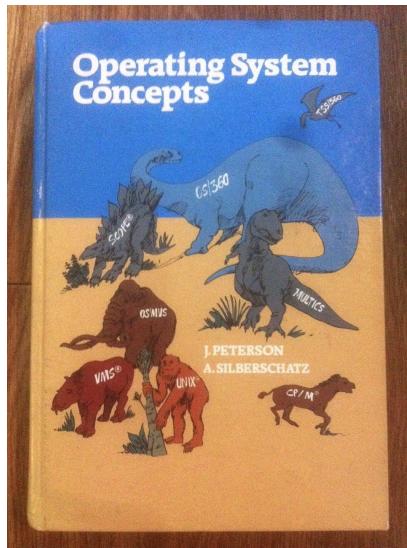
```
[4946925.516514] INFO: task ip:1747 blocked for more than 120 seconds.  
[4946925.516558]    Not tainted 4.4.0-137-generic #163-Ubuntu  
[4946925.516599] "echo 0 > /proc/sys/kernel/hung_task_timeout_secs" disables this message.  
[4946925.516666] ip      D ffff8819d1f17dc8  0 1747 1746 0x00000080  
[4946925.516671] ffff8819d1f17dc8 ffff882118dae840 ffff883ff0c27000 ffff8801c4602a00  
[4946925.516674] ffff8819d1f18000 ffffffff81efc764 ffff8801c4602a00 00000000ffffffff  
[4946925.516676] ffffffff81efc768 ffff8819d1f17de0 ffffffff818511b5 ffffffff81efc760  
[4946925.516679] Call Trace:  
[4946925.516689] [<fffffff818511b5>] schedule+0x35/0x80  
[4946925.516692] [<fffffff8185145e>] schedule_preempt_disabled+0xe/0x10  
[4946925.516695] [<fffffff81853099>] __mutex_lock_slowpath+0xb9/0x130  
[4946925.516698] [<fffffff8185312f>] mutex_lock+0x1f/0x30  
[4946925.516704] [<fffffff8173ce4e>] copy_net_ns+0x6e/0x120  
[4946925.516710] [<fffffff810a5b1b>] create_new_namespaces+0x11b/0x1d0  
[4946925.516713] [<fffffff810a5d5a>] unshare_nsproxy_namespaces+0x5a/0xb0  
[4946925.516718] [<fffffff81084461>] SyS_unshare+0x1f1/0x3a0  
[4946925.516721] [<fffffff8185538e>] entry_SYSCALL_64_fastpath+0x22/0xc1
```

... 同様のログがたくさん

トラブルシューティング : dmesg のログから

```
Jun 20 21:07:18 example kernel: [32668670.771896] BUG: unable to handle kernel paging request at 00000000b068c000
Jun 20 21:07:18 example kernel: [32668670.774197] IP: [<fffffffffc0241df7>] kvm_zap_rmapp+0x47/0x60 [kvm]
Jun 20 21:07:18 example kernel: [32668670.777209] Oops: 0000 [#1] SMP
Jun 20 21:07:18 example kernel: [32668671.052705] Call Trace:
Jun 20 21:07:18 example kernel: [32668671.058576] [<fffffffffc0241e1e>] kvm_unmap_rmapp+0xe/0x20 [kvm]
Jun 20 21:07:18 example kernel: [32668671.064464] [<fffffffffc023fe5d>] kvm_handle_hva_range+0x12d/0x190 [kvm]
Jun 20 21:07:18 example kernel: [32668671.070332] [<fffffffffc0249fc7>] kvm_unmap_hva_range+0x17/0x20 [kvm]
Jun 20 21:07:18 example kernel: [32668671.076150] [<fffffffffc0224633>] kvm_mmu_notifier_invalidate_range_start+0x53/0x90 [kvm]
Jun 20 21:07:18 example kernel: [32668671.087460] [<ffffffff811e5d35>] __mmu_notifier_invalidate_range_start+0x55/0x80
Jun 20 21:07:18 example kernel: [32668671.098756] [<ffffffff811cae3c>] change_protection_range+0x79c/0x830
Jun 20 21:07:18 example kernel: [32668671.104506] [<fffffffffc0223327>] ? kvm_vcpu_ioctl+0x137/0x620 [kvm]
Jun 20 21:07:18 example kernel: [32668671.110159] [<ffffffff811caee4>] change_protection+0x14/0x20
Jun 20 21:07:18 example kernel: [32668671.115703] [<ffffffff811e367c>] change_prot numa+0x1c/0x40
Jun 20 21:07:18 example kernel: [32668671.121112] [<ffffffff810b2efb>] task numa_work+0x1fb/0x2f0
Jun 20 21:07:18 example kernel: [32668671.126412] [<ffffffff8109f001>] task_work_run+0x81/0xa0
Jun 20 21:07:18 example kernel: [32668671.131573] [<ffffffff81003242>] exit_to_usermode_loop+0xc2/0xd0
Jun 20 21:07:18 example kernel: [32668671.136652] [<ffffffff81003c6e>] syscall_return_slowpath+0x4e/0x60
Jun 20 21:07:18 example kernel: [32668671.141683] [<ffffffff8183c7d0>] int_ret_from_sys_call+0x25/0x8f
Jun 20 21:07:18 example kernel: [32668671.161698] RIP [<fffffffffc0241df7>] kvm_zap_rmapp+0x47/0x60 [kvm]
```

古典にあたる



文献にあたる

