

ビューティフルコード (プ会 2008.4)

久野 靖*

2008.4.16

1 はじめに

□ 自分が学生だったころは「美しくコードを書く」ことを解説した本って読んで当たり前だったと思うんだけど…

- 「プログラム書法」「ソフトウェア作法」をはじめ、沢山…
- 今は必ずしも広く知られていないらしい!?
- なんちゃってプログラム書き人口が急増したため!?
- それってまずいんじゃない!?

□ 最近、そういうことを考えるきっかけが2つほど…

□ 話題 1: 「ドリトル本」付録「よいプログラムとは」

- 今回、兼宗さんがドリトル V2 の解説本 (?) を編纂。
- (ドリトル V2 の方がまだリリースできなくて止まっています。この話はあとで本人から)
- その中に「いいプログラムの書き方」の章 (付録) をというリクエストが兼宗さんからあった (なるほど!)
- そんなの先人が沢山書いてるでしょと思ったけど、実は知らない人が多いらしい→恐れ多いけど書いてみましょうか

□ 今回は→ (A)「昔の本にはこんなのがありました」(B)「こんなことを書いたけどどうでしょう」という話

□ 話題 2: 「ビューティフルコード」(翻訳)(4月23日発売!)

- 原 著: Oram, Wilson, eds., Beautiful Code: Leading Programmers Explain How They Think, O'Reilly, 2007. (Amazon で原著を買うと 4796 円。和訳は 3990 円)
- 昨年出てすぐ翻訳依頼→7月から作業→2月に脱稿 (大変でした)
- 内容: 33 人の著者が「自分が考える美しいコード」についてのエッセイ。すべての章においてコードの一部を掲載 (原著の方針)
- 今回は→「色々な人が挙げる『美しい』は様々ですね」という話

2 手元にある「よいプログラム」本

□ 自分の指導教官 (木村 泉先生) が翻訳をしていたことも結構大きい

- これ以外にもこういう関係の本があれば買っていた気がするけど…
- bit (共立の雑誌、廃刊になってしまった) にも色々な記事があったけど…

2.1 「プログラミングの心理学」

□ ワインバーグ, 木村, 角田, 久野, 白濱訳, プログラミングの心理学, 毎日コミュニケーションズ, 2005.

- 古典的名著だけれど今は手に入らないようだ (古本ならある)
- 人間的側面全般を広く扱っている
- プログラミングのプロセス、言語などについても載っている

2.2 「プログラム書法」

□ カーニハン, プロローガー, 木村訳, プログラム書法 第2版, 共立, 1982. (第1版は1976)

□ まさに「よいプログラムを書くための知恵」がぎっしり詰まっている。言語が古いが今でも十分読める。だから現在でも入手可能。

□ さまざまな「規則」が書かれている。いくつか挙げると…

- 「わかりやすく書こう --- うまさすぎるプログラムはいけない。」(冒頭にある規則)
- 「プログラムを上から下へ読めるようにしよう」
- 「同じ表現の繰り返しは共通関数への呼び出しに変えよう」
- 「だめなプログラムを修正するのはやめて、全部書き直そう」
- 「まずわかりやすい擬似言語で書いて、それから目的の言語に翻訳しよう」
- 「速くする前に、まず正しくしよう」

*筑波大学大学院経営システム科学専攻

2.3 「プログラミング 250 の心得」

□ George Ledin Jr. & Victor Ledin, 篠原 訳, プログラミング 250 の心得, 多賀出版, 1981.

□ これも規則とその説明の山だが、「書法」より実務指向なのが面白い。聞いたこともない出版社で知合いの誰も持っていない。珍しい本。

- 「ユーザのニーズにプログラムを適合させよ」
- 「ユーザの作成した入力データを表示せよ」
- 「ループを重視して、これを単位として取り扱え」
- 「同じ重みを持つ命令の頭を揃えよ」 ← 字下げをしろという意味
- 「ジョブ向きの正しい言語を使え」
- 「ライブラリ関数を使え」

2.4 「ソフトウェア作法」

□ カーニハン, プローガー, 木村 訳, ソフトウェア作法, 共立, 1981. この本は Ratfor というヘンな言語で例題が書かれているのに、とても人気があり、今でも大きな書店に行けば並んでいる (!!)

- この本は「日本で初めてコンピュータ原稿を使って制作された本」であり、それが「木村研の日本語処理システムの成果」。そのため、制作のお手伝いを久野ほかがずっとやっていた (磁気テープ抱えて漢字プリンタ出力に行ったり先生の赤入れに対応するカタカナファイル原稿のエディタ編集をしたり…)
- 内容は「書法」より上のレベルで、各種題材 (圧縮、文字列置換、整列、パターンマッチ、マクロ展開、文書整形、プリプロセサ) について設計してみせながら蘊蓄を語る。
- コードが全部掲載されていて、なぜこう書いてあるかという話も沢山書いてある。

2.5 「プログラミング作法」

□ カーニハン, パイク, 福崎 訳, プログラミング作法, アスキー, 2000. これは言語が C と C++ と Java だから現役。普通に買える。

□ 「書法」「作法」の集大成な感じで、第 1 章「スタイル」からはじまり「アルゴリズムとデータ構造」「設計と実装」「インタフェース」「デバッグ」「テスト」「性能」「移植性」「記法」とうまくテーマが選ばれている

- 「グローバルにはわかりやすい名前を、ローカルには短い名前を」(書法)
- 「多分岐の判定には else-if を使え」(書法)

- 「実装の詳細を隠蔽しよう」「ユーザに内緒で何かをするな」(インタフェース)
- 「エラーの検出は低いレベルで、その処理は高いレベルで」(インタフェース)
- 「打つ前に読め」「自分のコードを他人に説明してみよう」(デバッグ)
- 「回帰テストを自動化しよう」「テストの網羅範囲を測定しよう」(テスト)
- 「より優れたアルゴリズムやデータ構造を利用しよう」「関係ない部分を最適化するな」(性能)
- 「システム依存のコードは別個のファイルに」「システム依存部分はインタフェースの裏に隠蔽しよう」(移植性)

3 「よいプログラムとは」を書く…

□ 前提: 高校の先生とかに初めてこういう(「よいプログラム」についての)話を読んでもらい→「なるほど」と思い実践してもらおう(ほんとかな?)

- さまざまな言語とか技術的に難しい話とかはできない(教育プログラミング研で喋ったらいろいろ厳しい意見とダメ出しをもらってしまって…)
- 言語はドリトル(ドリトルでよいプログラムの書き方って…?)
- →一般的な指針とお話 + 「書き方改良の具体例」
- 構成: 目標→「プログラムの」指針→改良例→「作り方の」指針→評価規準→学び方(カリキュラム構成)→まとめ

3.1 「プログラムを作るときの目標」

□ まず「何を目標とするべきか」を提示したいと思ったので。具体的には次の3つ。

□ 「プログラムがコンピュータ上で動くこと」

- 最初は「書法」にならって「分かりやすく」を書いていたのだけれど、それはプロの言うことで、最初はず「動く」から始めないとだめらしい…

□ 「プログラムが『正しい』(やってほしい)動作をしてくれること」

- これも言わずもがなだが、Ledin には書いてあった。コンパイルエラーが取れて動いたらそれでいいと思っている人もいるから?

□ 「プログラムが、人間が読めるように(分かりやすく)書かれていること」これが「書法」の冒頭なのでまず言いたかったことだけれど、そういうわけで3番目になってしまった。

3.2 「よいプログラムの指針」

□ 前記 3 目標を達成するようなプログラムの書き方の指針を示し、理由を簡単に解説する

- 「素直に分かりやすく書こう」
- 「分かりやすい名前をつけよう」
- 「むずかしいところはコメントをつけよう」
- 「字下げや文字の配置にも気を配ろう」
- 「構造を整理しよう」
- 「ステップに分けて書こう」
- 「同じことは 1 箇所にとめよう」
- 「無理でない範囲で短く簡潔に書こう」

□ 多すぎても読んでもらえないので、そこそこの数になるように大切と思うことから選んだ

□ ところが「素直に分かりやすく書こう」が理解してもらえない! 「どういのが素直じゃないのか」(悪い例)を質問されるが、悪い例とはトリッキーなので納得してもらえない

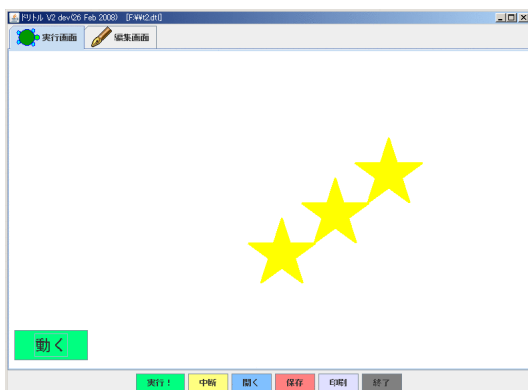
- 前の版: 「奇数か偶数か調べるのに $x \& 1$ はやめて剰余で」
- 今の版: 数値変換に「 $x-0$ 」整数変換に「 $x|0$ 」(JavaScript)
- どっちにせよまいち…

□ 「無理でない範囲で短く簡潔に書こう」も最初は「短くしようと頑張りすぎない」だった(それが我々の常識ですよね…)

- ところが、短くするのはいいことだという注文がついた
- 子供がコードを書くと、動いたものをコピペしてどんどん長く増やしてしまう→ループの代わりに展開されたものが延々と続く
- それをループで整理することは目指させて欲しいという意見でこのように。

3.3 「例題: プログラムを添削する」

□ 次のような感じで星を 3 つ動かすプログラム



□ 添削前

```
カメ太=タートル!作る。
「カメ太!100 歩く 144 右回り」!5 繰り返す。
x=カメ太!図形を作る(黄色)塗る -380 0 移動する。
「カメ太!100 歩く 144 右回り」!5 繰り返す。
y=カメ太!図形を作る(黄色)塗る -300 60 移動する。
「カメ太!100 歩く 144 右回り」!5 繰り返す。
z=カメ太!図形を作る(黄色)塗る -220 120 移動する。
カメ太!消える。
ボタン!『動く』作る -380 -180 位置。
ボタン:動作=「
時計=タイマー!作る 1 時間 0.1 間隔。
時計!「x!5 10 移動する。y!5 10 移動する。z!5 10
移動する」実行。
時計! 3 時間 0.1 間隔。
時計!「x!10 -5 移動する。y!10 -5 移動する。z!10
-5 移動する」実行。
時計! 2 時間 0.1 間隔。
時計!「z! 20 10 移動する」実行。時計!「y! 20 10
移動する」実行。時計!「x! 20 10 移動する」実行。
」。
```

- どの段階で何をしているのかが分かりにくい。
- 変数名が内容を表していない。
- 何回も同じ動作を記述して星を作っている。
- 何回も同じ動作を記述して複数の星を動かしている。
- いい加減な位置で行替えてあるので読みにくい。
- メソッドの範囲が字下げしていないので分かりにくい。

□ 添削後

```
// 星を 3 つ用意する
カメ太=タートル!作る。
「カメ太!100 歩く 144 右回り」!5 繰り返す。
星 1=カメ太!図形を作る(黄色)塗る -380 0 移動する。
星 2=星 1!作る 80 60 移動する。
星 3=星 2!作る 80 60 移動する。

// 星を配列に入れて扱えるようにしておく
星列=配列!(星 1)(星 2)(星 3) 作る。
カメ太!消える。

// ボタンを用意する
ボタン!『動く』作る -380 -180 位置。
ボタン:動作=「
時計=タイマー!作る 1 時間 0.1 間隔。
// 星を 3 つ斜め上に動かす
時計!「星列!「|x|x!5 10 移動する」それぞれ実行」実行。
時計! 3 時間 0.1 間隔。
// 星を 3 つ斜め下に動かす
時計!「星列!「|x|x!10 -5 移動する」それぞれ実行」実行。

// 星を個別に動かす
時計!「星 3! 20 10 移動する」実行。
時計!「星 2! 20 10 移動する」実行。
時計!「星 1! 20 10 移動する」実行。
」。
```

- どの段階で何をしているのかが分かりにくい。→コメントをつけて、プログラムのそれぞれの場所が何をしているか説明するようにした。

- 変数名が内容を表していない。→3つの星を表す変数を「星1」「星2」「星3」とした。
- 何回も同じ動作を記述して星を作っている。→最初の星を作ったら、あと2つの星はそれを「作る」でコピーして作るようにした。
- 何回も同じ動作を記述して複数の星を動かしている。→3つの星を「星列」という配列に入れておき、3つを一緒に動かす時は配列の「それぞれ実行」を使うようにした。
- いい加減な位置で行替えしてあるので読みにくい。→行が長くて行替えするときは命令の切れ目などきりのいいところで行替えするようにしている。
- メソッドの範囲が字下げしていないので分かりにくい。→メソッドの内側は字下げして範囲が分かるようにした。

3.4 「うまくプログラムを作るための指針」

□ こちらは「プロセス」の指針ということで(こういうこともたぶんあまり知られていないかも)

- 計画を立てよう ←「いきなり打ち始めるな!」
- 少しずつ動作を確認しながら作ろう ←一気に作って破滅な人が多い
- 系統的にテストしよう ←言うのは簡単だけど…
- 先に進む前に整理しよう ←リファクタリング?
- 間違いが分からなくなったら ←切り分け、他人に説明、print デバグ

3.5 「プログラムを評価する基準」

□ 教育現場では「点数つけ」が不可欠だからというので入れたけど…普通のソフト開発ではないことなので苦労した

- 「プログラムが『課題』の条件を守っていること」←注文ではなく課題がいいそうで。
- 「プログラムが『ちゃんと動く』こと」←エラーがあつて動かなかつたり途中で死んだりするものも提出されてくるから…
- 「プログラムの性能」←課題の充足に+αすること…使いやすさ、みばえ、たのしさ、オリジナル性、工夫、(速さ←速さを見やすさと引替えにしないこと)
- 「プログラムの美しさ」←何が美しいかという定義は困難なのでお茶を濁させていただいて…

3.6 「プログラミング学習で学んで欲しいこと」

□ 我々が「一般の人でもこういうことを知っておいてくれたら我々はずっと楽なんだけど/日本はずっとよくなるんだけど」と思うこと

- 「決まった手順に沿った厳密な実行」
- 「高速だが融通の効かないコンピュータ」
- 「考えることの大切さ」
- 「人間は間違える」
- 「変数と値の書き換え」「制御構造(繰り返し、枝分かれ)」「アルゴリズムの考え方」
- 「構造と設計の考え方」
- 「『注文』を満たすことの大切さ」
- 「製品と実験用模型の違い」←難しいか?
- 「プログラムの活躍している場所」

3.7 「プログラミングの学び方」

□ 自分が「この順番で学ぶのがいいと思う」おすすめのカリキュラム進行も書いてみた。

- 「『決まった書き方』があることを学ぶ
- 「決まった順序で実行されることを学ぶ」
- 「同じことをいく通りにも書けることを学ぶ」
- 「自分が考えたことを書き表すことを学ぶ」
- 「少しずつ確認しながら組み立てることを学ぶ」
- 「計画を立てて組み合わせて行くことの重要性を学ぶ」
- 「コンピュータとは結局何なのかを学ぶ」←おまけ

3.8 まとめると…

□ 結局自分が言いたいことは「わかりやすく書け」だけれど…

□ それだけでは分かってもらえないから色々回り道したというか…

4 「ビューティフルコード」翻訳

□ この本の特徴(冒頭で挙げた「美しいプログラム」本との違い)→33人の著者がバラバラに言いたいことを言っている

- 見事にバラバラで、意表を衝いて面白いものが沢山ある
- カーニハンの(1章)とかは、ある意味予想通りだなーという感じ
- ウォーレン Jr. の(10章)とかは、Hacker's Delight 中のネタの再利用

- ベントレーの(3章)は quicksort だが意表を衝いて「整列しないで計算量を測る」でコードを動かさないので確かに新しいけど…
- 久野個人が面白いのはやっぱりプログラムを動かす話で、しかも読んだことがないようなもの(贅沢)(まあ全部訳したわけだし…)

□ 結局、各章は読む人によって面白いものもワケワカなものもバラバラだろうと思う

- だからある意味、この本の面白いところはその「バラバラさ」
- 手慣れた著者はさらっと書いて終わるが、著者によっては「これが大切なんだ!」とか力が入っている→その力が入っているポイントが本当に章ごとにまったく違うのが面白い→読んで頂くのが一番
- とはいえ、せつかくの機会なので、いくつか久野個人が面白かった章について紹介させてください。

4.1 32章「働くコード」

□ まさに「よいコードの書き方」の話題なので…

- この章もセイワルド(著者の1人)の「整った(プリティ)コードの7か条」がネタとなっている(が、それほど有名でもないと思うので…)
- 『『本のように』であること」「似たものは似て見えるようにすること」「字下げを克服すること」「コードをもつれさせないこと」「ごちゃごちゃにしないこと」「既存のスタイルにとけこむこと」
- しかしちよつと異論もあつたりする…訳者が訳注で喧嘩を売るといふわけにはいかないのでここで。

□ 「本のようにであること」→幅狭く書けということ(じゃあ縦に伸びるのはいいのか?)

×

```
if( bf->end == bf->Lines() && lf1->end == lf1->Lines() &&
    lf2->end == lf2->Lines() ) return( DD_EOF );
```

○

```
if( bf->end == bf->Lines() &&
    lf1->end == lf1->Lines() &&
    lf2->end == lf2->Lines() )
    return( DD_EOF );
```

□ 自分では行を長くしてでも行数を減らす方なのでどうも…

□ 賛同できることももちろんある。

- 「入れ子を字下するのは当然だけど、それでコードが読みやすくなるわけではない(字下げを深くしないのがベスト)」
- 「カラフルな表示でなくても読めるコードであるべき(gdb とかが表示するときには色なんかついてないから)」

- しかしこの章、もっと色々語ってくれるのかと思ったらあつという間に終わってしまう感じ…(「作法」「書法」に慣らされているから食い足りない)

4.2 24章「美しきかな、並列」

□ 自分の好きな並列/並行の話…Haskell STMの売り込み

- 内容については略(面白いけど、木戸さんもいるし…)
- 章の題名について。原著は「Beautiful Concurrency」。もともとは「並行はキタナイけど、Haskell STMならこんなに美しいよ」というつもりらしい→「並行性を美しく書く」が適切な訳?
- 自分としては「普通の並行性はキタナイ」はやめてほしいし、宣伝ぼくない方がいいでしょ→奥さんの案を頂いた。「並列って楽しいよね」という意味になるのも嬉しい。

4.3 30章「世界につながる手段がボタンだけだったら」

□ 掲載コードはめちやくしゃしょぼい(言い訳も書いてある)

□ しかし記述されているユーザインタフェースはすごく頑張っている。

- 「ホーキング博士のためのインタフェース」→指1本で編集や任意のコマンドの起動ができる(実際はホーキング博士は使い慣れたソフトから移行してないそうだけど)
- 階層メニュー。各メニューが一定時間間隔で循環ハイライト、ハイライトされたときにクリックすると選択できる(上に戻るには戻るといふ項目を選ぶ)
- これで効率良くするため徹底して学習する(入力した単語とか文例とか全部20個保存しておいてLRUで更新)
- そのため使う「1行挿入すると最も古い行が捨てられる」プログラムが掲載されている(!あと2つ掲載されているが似たりよったり)

4.4 6章「フレームワーク設計の挑戦」

□ FITって知ってます? FITってこういうものだったのか、という勉強になる。しかしFITの話題ではない。

- 普通のフレームワークは中身を隠蔽してユーザが実装に依存しないように気をつける→常識
- しかしFITではずっと「あけっぱなし」で「いくらでもいじっていい」→カルチャーショック!
- このようなカルチャーの選択肢があり得ることと、それを可能にしている要因が説明されている→なるほどね～

4.5 33章『本』のためにプログラムを書く

□ 最後の章だし期待したのだけどなんか肩すかしというか…

- 問題: 「3点 (px,py) (qx,qy) (rx,ry) が一直線上にあるか否か?」
- 3点のうち2点から直線の式 $Y=AX+B$ を求め、残る1点がこれを満たすか調べると… 直線がY軸に平行だと傾きAを求めるとき困るので…(以下延々とさまざまな案が出ては否定されて進んで行く)
- あなたならどうしますか?

□ 自分ではこういう問題は (ICPCなどで) やらされるので:

- 「2つの直線の傾きが等しい」

$$\frac{(qy-py)}{(qx-px)} = \frac{(ry-py)}{(rx-px)}$$

- このままでは0割りになるから分母を掛けて

$$(qy-py)*(rx-px) == (ry-py)*(qx-px)$$

- こうすれば大丈夫、という訳注を書いて、最後の方まで読んで行ったら実はこれが「究極の解」として書かれているのであわてて削除した
- さらにこの著者 (ブライアン・ヘイズ) は大した意味もなくコードをすべてLispで書いていて、Lispコードの読み方をわざわざ説明している… その例題は再帰GCD(もちろん他のコードには再帰なんて出てこない)。どうなんでしょうね? 対象読者が易しめ?

5 22章「スプーン一杯の汚水で」

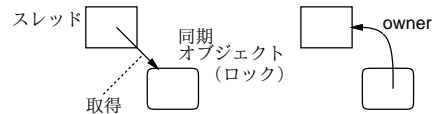
□ 題名の由来は…

樽いっぱい汚水にスプーン一杯のワインを注ぐと、
樽いっぱい汚水になる。
樽いっぱいワインにスプーン一杯の汚水を注ぐと、
樽いっぱい汚水になる。
(ショーベンハウエルのエントロピーの法則)

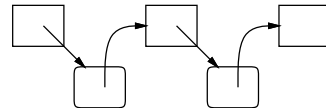
- そのココロは… ソフトは1箇所でもバグがあったら台無し (ソフトの種類によるけど、OSの同期プリミティブだと…)
- Solaris 8 (Sunの商用Unix) のリリース直前に発現したバグ→OSがパニック (緊急停止) で止まる→必死で直したら→OSがいきなりハングする→真っ青、というお話
- 自分には最高に面白かったがこれが楽しめる読者はどれくらいいるのか… (今回は結構いそうなので取り上げます)

5.1 スレッド/ロック/優先度継承

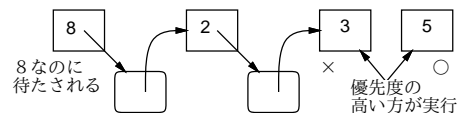
□ スレッドは同期/排他制御のため同期オブジェクト (ロック) を獲得



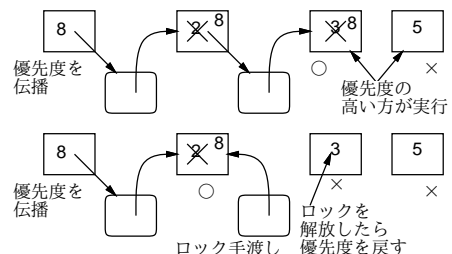
□ 取ろうとしたロックを他人が取っていると待つ必要がある→連鎖



□ スレッドには優先度がある→高いものが先に実行→ロックで待っている高優先度のスレッドが待たされてしまう (優先度逆転)

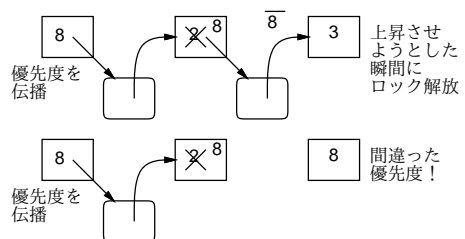


□ 待ちに入るスレッドの優先度がロックを持っているスレッドの優先度より高い場合、その高い優先度を「一時的に (ロックを持っている間)」伝播

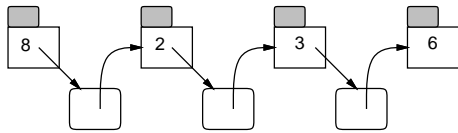


5.2 連鎖の整合性管理

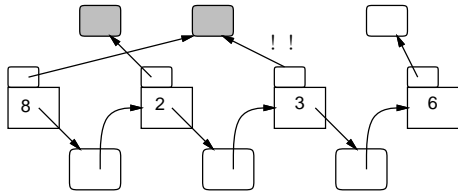
□ 複数CPUによる本物の並列実行→優先度を上昇させようとした瞬間にスレッドがロックを解放すると→間違った優先度→厳密な整合性の管理が不可欠!



□ スレッド自体にロックを掛けて整合性を管理。ではブロッキングチェーン全体に渡ってロックを掛けるかという…

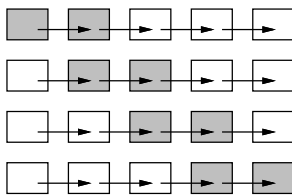


□ それは駄目。スレッドロックは実際は「ロック構造へのポインタ」であり、ロック構造はいくつかのものは共有されている→2つ目をロックしようとするとうとデッドロックに



- なせこうなっているかという、ロックを使う度にメモリ管理では遅くて駄目なので、ロック+キューの配列を用意し、ハッシュによりその配列のどのエントリを使うかを決定する (ハッシュが衝突した場合は同じエントリになる)

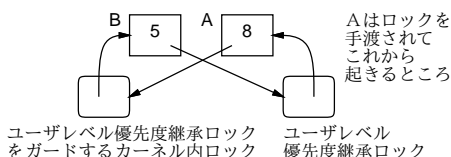
□ 実際にはブロック連鎖の「巻き戻り」は右端 (先端側) でしか起こらないので「常に2つロックを取っている状態」で進んで行けば左側 (根元側) のロックは解放してしまっ



- その2つのロックが同じな場合はチェックしてすぐ対処可能
- さらに AB/BA デッドロックを防ぐため、取る時は「アドレスの若い順に」取る (このため、原著には書いてないが3つずつ押えて行く?)

5.3 問題

□ 前記の点はカーネルの優先度継承。しかし今日ではユーザレベルでも複数スレッド間同期を行うため、ユーザレベルスレッド間でも優先度継承を行う必要→リリース予定の Solaris 8 に組み込む (1999.10) →バグが見つかり検死ダンプを読んでいた (1999.12)



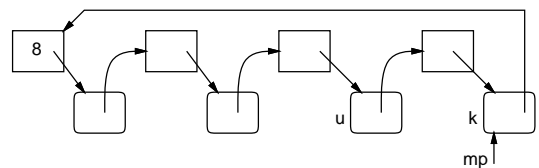
- ユーザレベル優先度継承を (カーネルレベルと同じようにして) やろうとしたときに、優先度継承ロックをガードするカーネルレベルのロックを介したループができていて、それを検出してパニック

□ 通常、カーネルはユーザレベルのデータ構造は「信用しない」のだが、優先度継承処理については厳密に状態を追跡する必要→ここに問題の根源

- カーネル内の偽デッドロック→パニックで停止
- ユーザレベルでの偽デッドロック→エラーで返る→エラーコードを見ないアプリでは取れていないロックを取れたと思って進行→データを壊してしまう
- カーネル内ロックを使わないで済ませようと努力したがこれは無理そうだと分かる
- 問題の状態を検出して何とかするしかない…

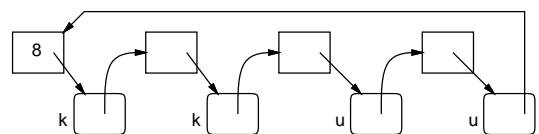
5.4 解決策

□ ケース 1: 優先度継承を処理して、自分に戻って来てしまったが、自分が持っているのが最初に処理を始めたカーネルロック (変数 mp) である場合



- ぐるっと1周して優先度継承処理を終えたので、これで止めてよい。処理が終わったら自分が mp を解放するのでサイクルではなくなる。

□ ケース 2: 優先度継承を処理して、カーネルロックからユーザレベルロックに渡った場合

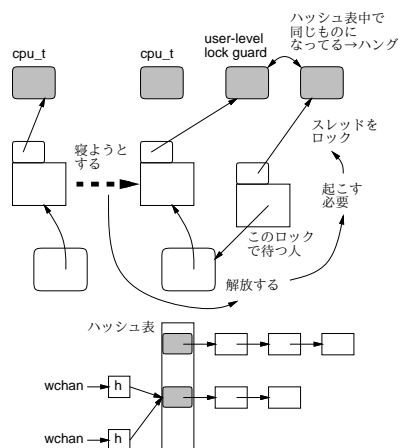


- カーネルからユーザに渡る (カーネルコードがユーザコードを待つ) というのはユーザレベルロックの優先度継承処理でしかあり得ない (これ以前には一切あり得なかった) 状況。これは現在誰か他の人が必死で優先度を調整しつつある過渡的な状況→しばらく待ってやり直せばなくなっているはず

5.5 問題 B

□ しかし、これを直して負荷テストを行うと…こんどはカーネルがいきなりハングするという症状が…

- `turnstile_block()` → `mutex_vector_exit()` → `turnstile_block()` という系列の中。
- つまり、ユーザレベル優先度継承ロックの「獲得時」と「解放時」それぞれについて、カーネルロックの獲得と解放が必要だった（データ構造をガードするため）、その前者において…
- 優先度継承を処理する→カーネルロック解放のため、データ構造をすべて切り替える→ロックを解放して `swtch()`
- しかしその間にそのロックで待ちに入ったスレッドがあると、それを起こす必要があるため、そのスレッドロックを獲得。加えて、継承した優先度を放棄するためカレントスレッドのロックも獲得。
- カレントスレッドのロックは既に待とうとしているユーザレベルロックをガードする（ハッシュ表中の）ロックになっている。



- カーネルロックとユーザレベルロックのハッシュ値が同じだと、同じロックを2回取ろうとしてデッドロック（実際に起こったこと）
- 同じでなかったとしても、アドレス順によって AB/BA デッドロックの可能性

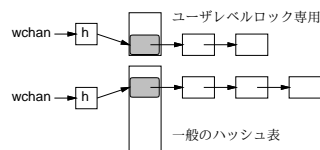
□ 基本的な問題は「ほぼ待ち状態に入った後でカーネルロックを解放する」という非常に困難な問題

- 待ち状態に変更する前に優先度を戻せないか? ∴ それでは優先度逆転の穴が発生
- `turnstile_block()` → `mutex_vector_exit()` → `turnstile_block()` という系列を検出できないか? ∴ 複数スレッドによる AB/BA デッドロックに対応できない
- うーん…

5.6 解決策 B

□ 要するに、ユーザレベルロックのエントリと他のロックのエントリがたまたま同じハッシュ値になるからいけない

→ ユーザレベルロックのエントリが入るハッシュ表だけ分離して低いアドレスに置く（これでアドレス順の獲得になり AB/BA デッドロックも防げる）



- あまりにも特定問題向けの解決策ではないかという疑念はあったが、これしかなかった。数年たってもここからバグが出ず、これより良い解法も考案されていない→これこそ「ビューティフルコード」!!

6 まとめ

□ 「ビューティフルコード」のまとめ

- 22章のような手に汗握る話は大好きだけど、しかしこれを楽しめるのには相当な経験が必要なんじゃないだろうか…
- まあこれだけ難しいのもあるというところが本書の幅広さなのかも。
- ずっと易しい章とか、ソフトウェア工学っぽい章なんかもあるので、これはこれでいいのかなと思う。
- 皆様もぜひご一読を。

□ 本日全体のまとめ

- 「美しいコード」の話に何を求めるかは、人によって全然違っている。
- 「美しいコード」の話は楽しい。

注：この資料は 2008 年 4 月 16 日に開催された「プログラミング・情報教育研究会」の資料です。本資料に言及されている「ドリトル本」は 2007 年 9 月にイーテキストから「ドリトルで学ぶプログラミング-グラフィックス、音楽、ネットワーク、ロボット制御」として刊行されましたが、本資料で言及している「よいプログラムを書く」の部分は兼宗先生が大幅改訂して、ここで述べられているものとは別内容になっています。

注：本資料の内容はコピー、抜粋、要約を含め、自由に利用されて構いません。ただし出典を明示してください。