

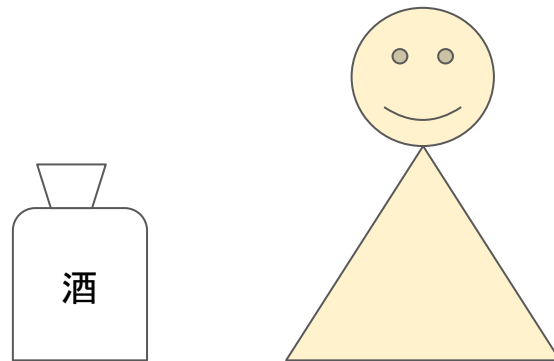
利きプロセススケジューラ

Nov. 9th, 2024
Satoru Takeuchi
X: satoru_takeuchi

利き酒

利き酒(ききざけ、唎き酒、聞き酒とも)とは、酒の品質を判定すること[1]。

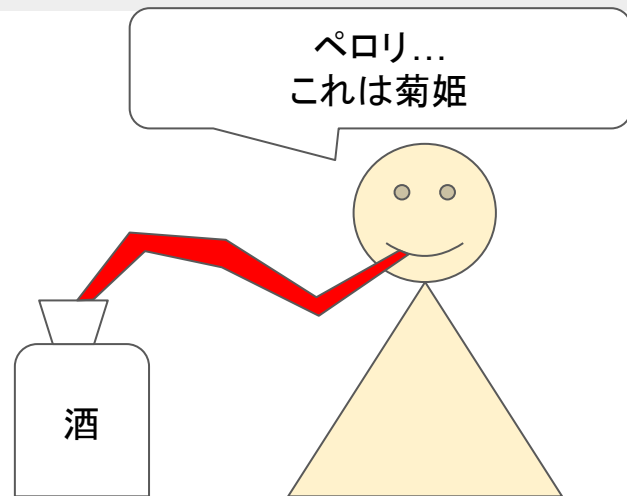
引用元:
「利き酒」(2024年7月22日 (月) 07:14 UTCの版)『ウィキペディア日本語版』。



利き酒

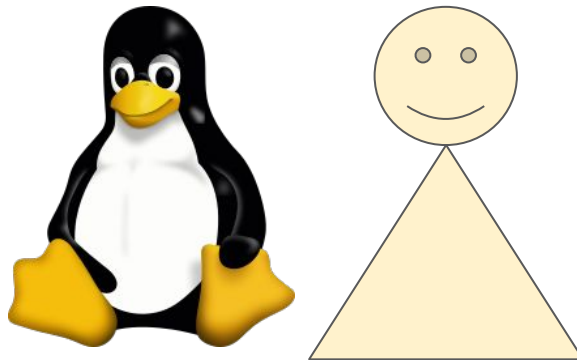
利き酒(ききざけ、唎き酒、聞き酒とも)とは、酒の品質を判定すること[1]。

引用元:
「利き酒」(2024年7月22日 (月) 07:14 UTCの版)『ウィキペディア日本語版』。



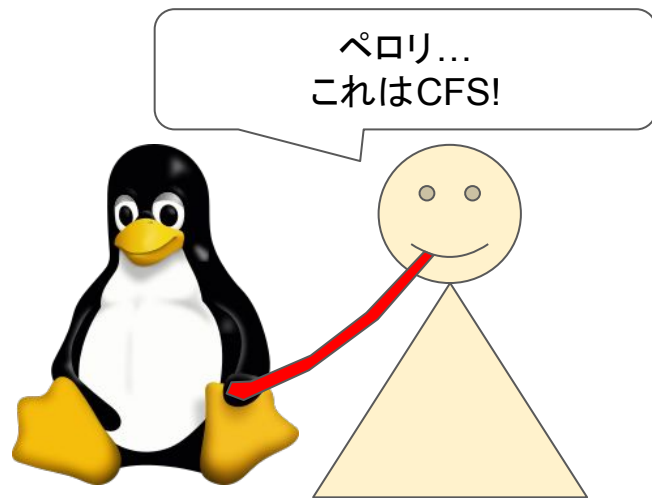
利きプロセススケジューラ

カーネルソースのバージョン見ず、ソースも読まず、プロセスの挙動のみからプロセススケジューラが何かを判定すること[要出典][独自研究][誰によって?]



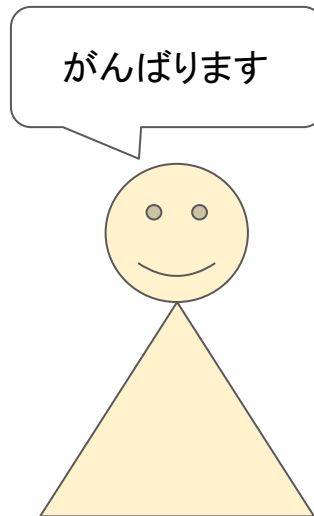
利きプロセススケジューラ

カーネルソースのバージョン見ず、ソースも読まず、プロセスの挙動のみからプロセススケジューラが何かを判定すること[要出典][独自研究][誰によって?]



今日のお題

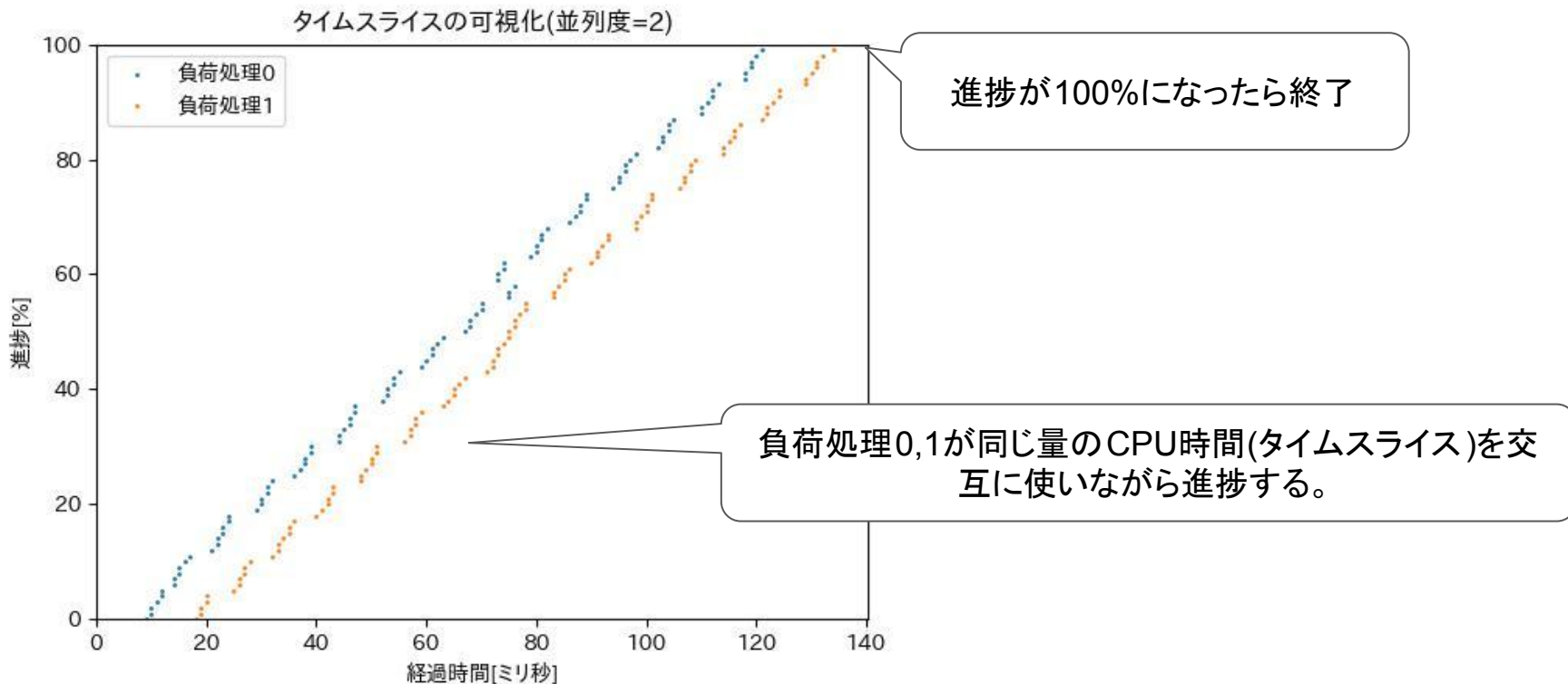
- 3つの環境A,B,Cのプロセススケジューラが何かを判定する
- プロセススケジューラは以下のうちのいずれか
 - O(1)スケジューラ: Linux v2.6.0~v2.6.22
 - CFS(Completely Fair Scheduler): Linux v2.6.23~v6.5
 - EEVDF(Earliest Eligible Virtual Deadline First): Linux v6.6~
- 実験プログラムを動かした結果得られるグラフから判定



実験プログラム

- 使い方
 - `./sched.py <並列度>`
- やること
 - 1. CPU時間を所定量使った後に終了する負荷処理を **<並列度>**の数だけ起動する
 - 全て1つのコア上で動作させる
 - 2. すべての負荷処理の終了を待つ
 - 3. 負荷処理の開始時からの経過時間と進捗をあらわすグラフを描く
 - x軸は経過時間[ms]、y軸は進捗[%]
- ソース
 - <https://github.com/satoru-takeuchi/sched-tasting/blob/main/sched.py>

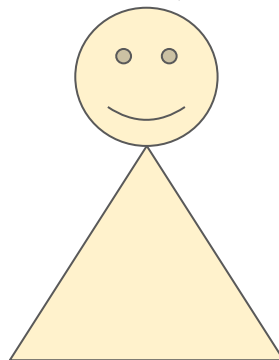
グラフの見かた



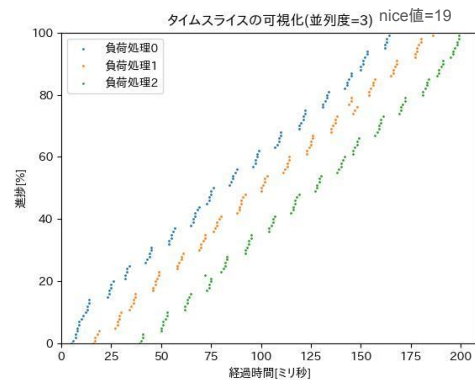
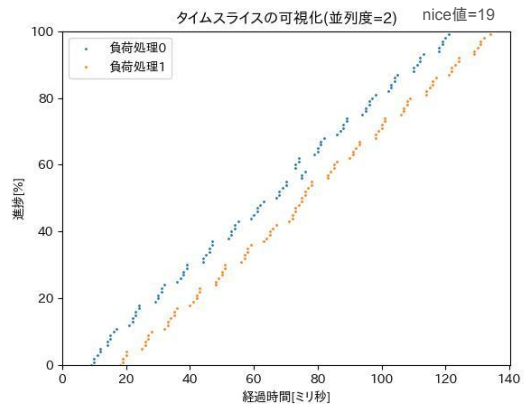
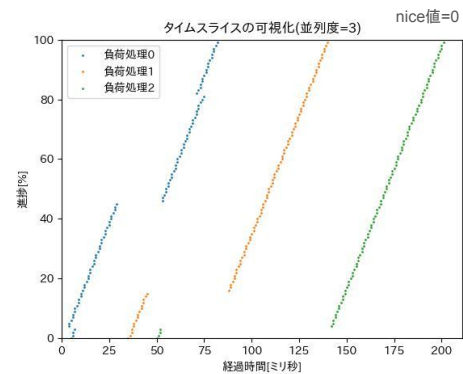
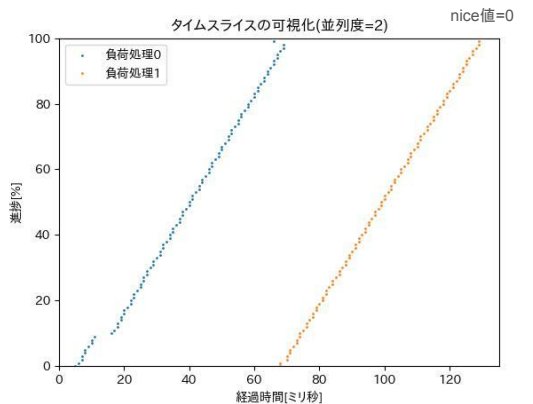
利きプロセススケジューラ、開始!

- 環境A,B,C上で./sched.pyを動かし、得られたグラフを見る
- パラメタ
 - 並列度: 2,3
 - nice値: 0(デフォルト値), 19(最高値。優先度でいうと最低)
 - niceコマンドを使って指定
- グラフの特徴からプロセススケジューラが何かを判定する

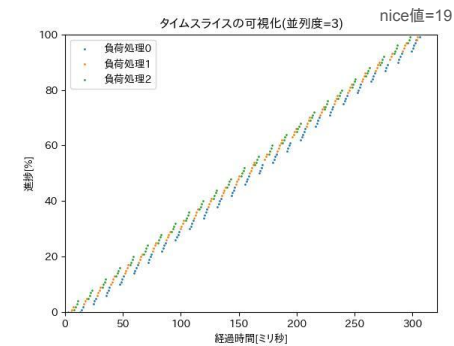
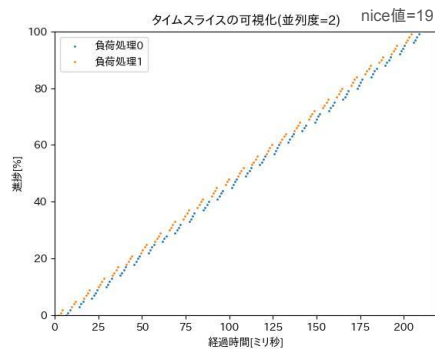
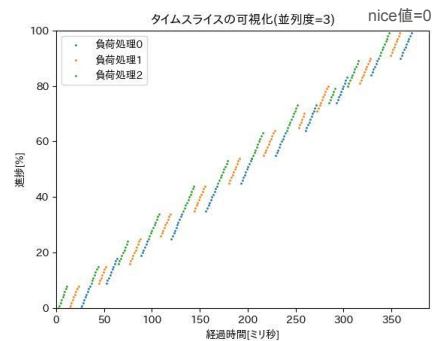
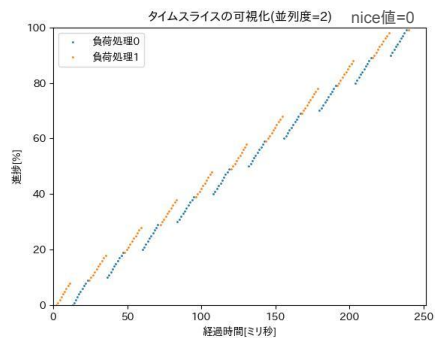
みんなもやってみよう!



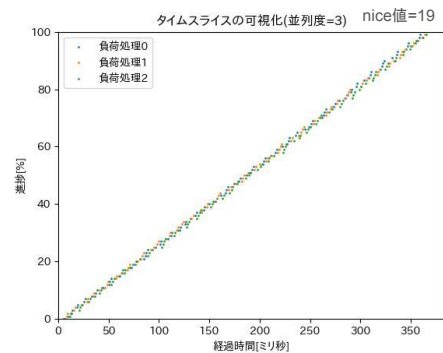
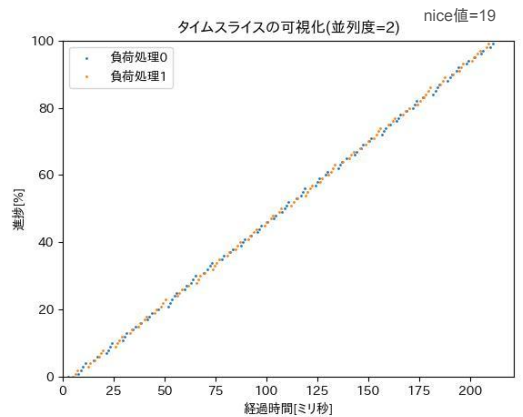
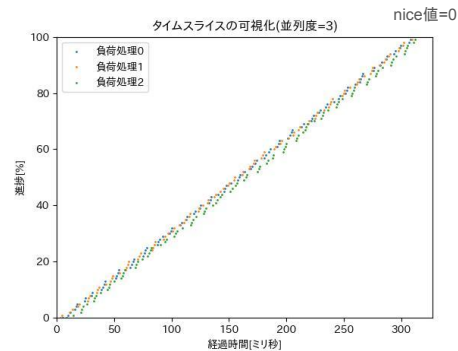
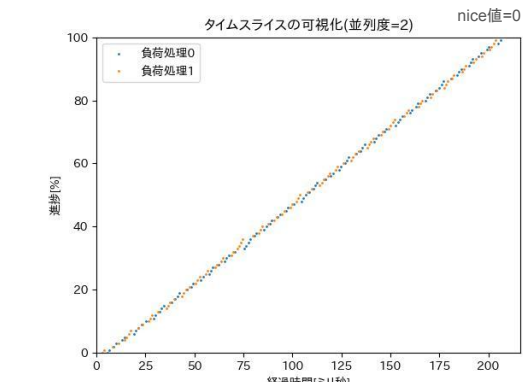
環境Aで得たグラフ



環境Bで得たグラフ

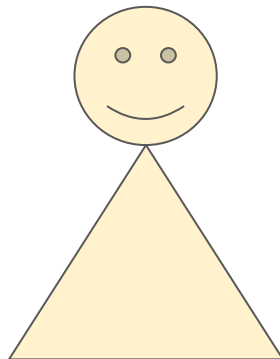


環境Cで得たグラフ

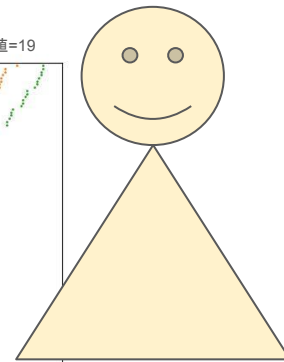
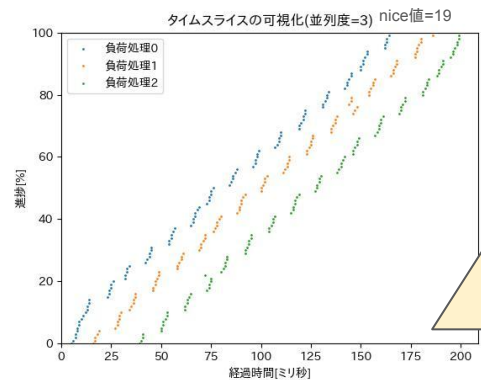
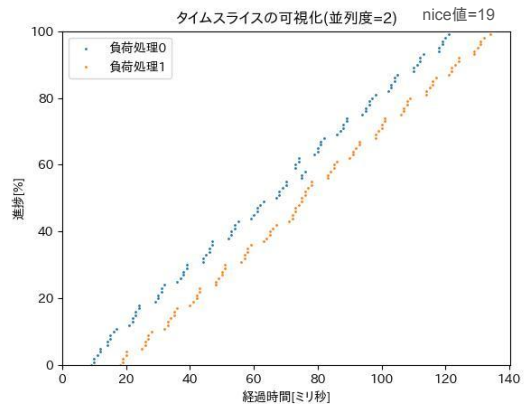
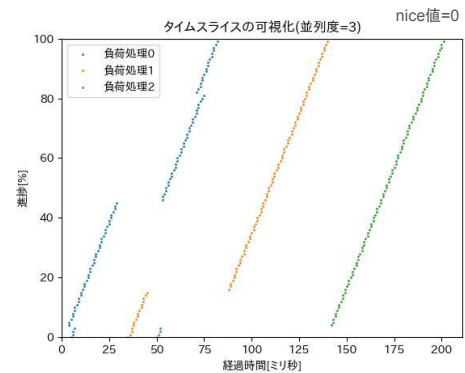
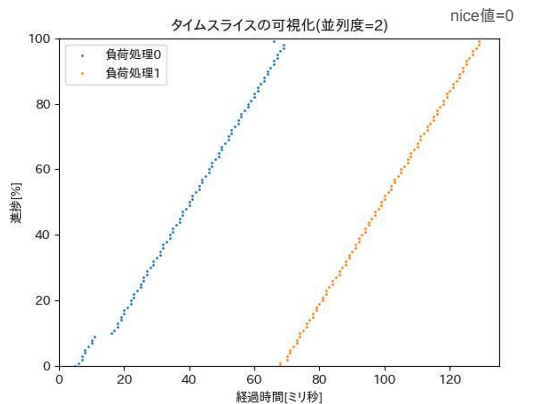


答え合わせ

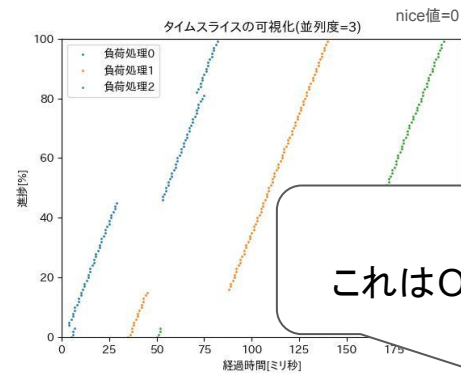
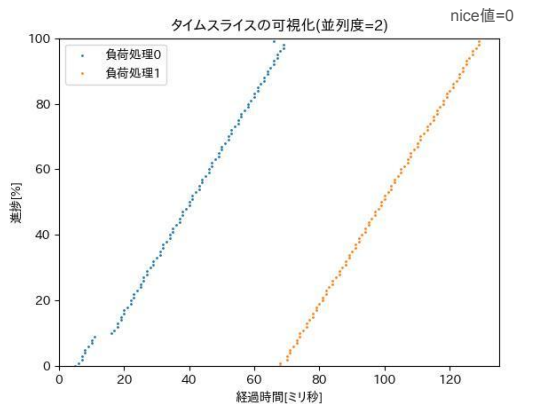
わかったかな？



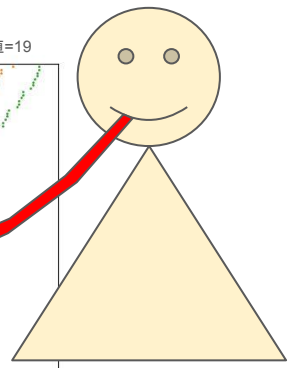
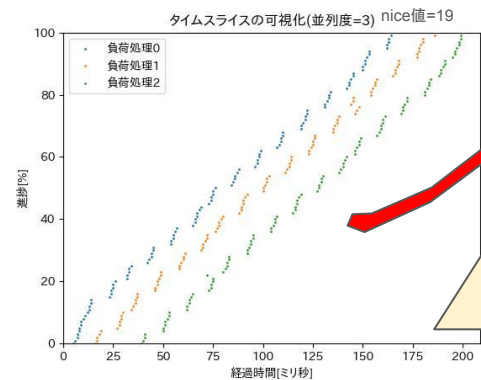
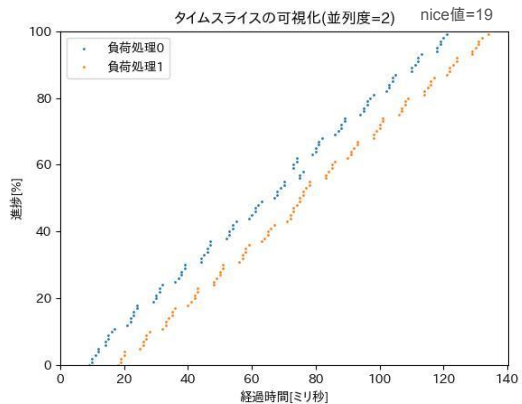
環境Aは...



環境Aは...



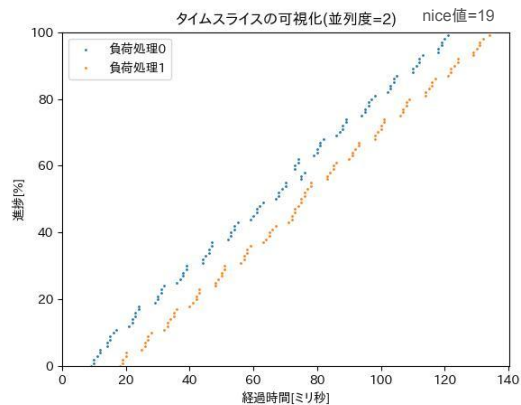
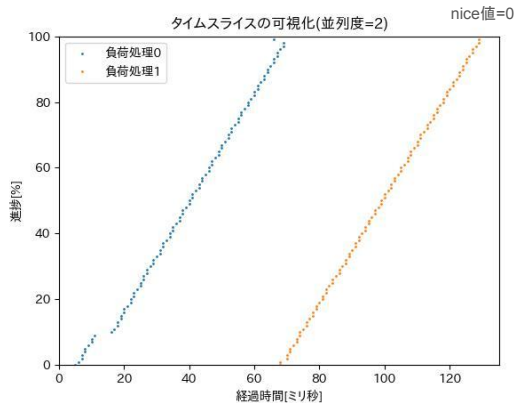
ペロリ...
これはO(1)スケジューラ!



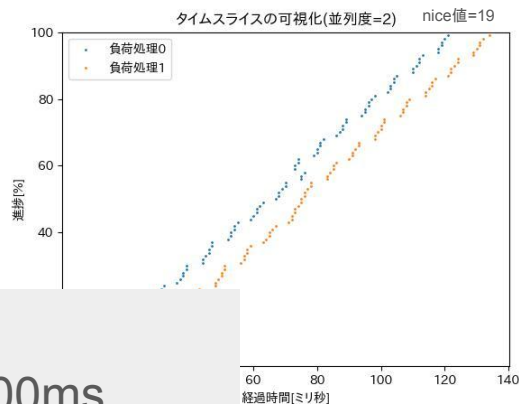
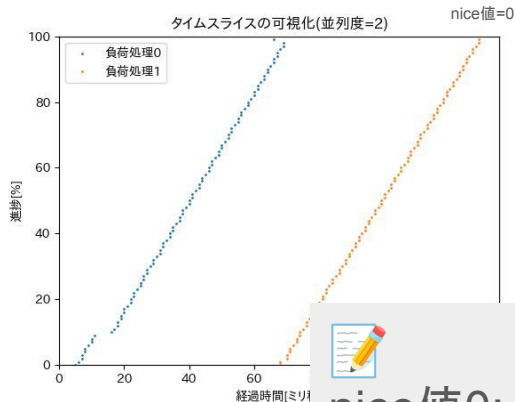
根拠

- $O(1)$ スケジューラの特徴
 - nice値が変わるとタイムスライスが変わる
 - 実行可能プロセス数を変えてもタイムスライスは不変
- 2つの特徴を兼ね備えるのは $O(1)$ スケジューラのみ

nice値が大きくなるとタイムスライスが短くなっている

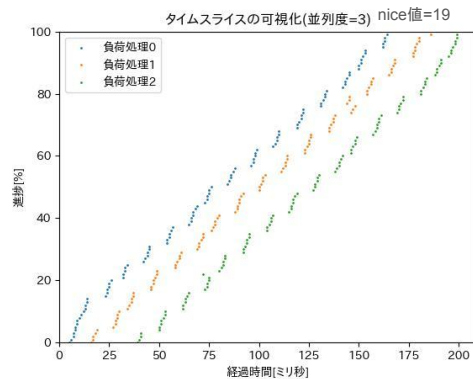
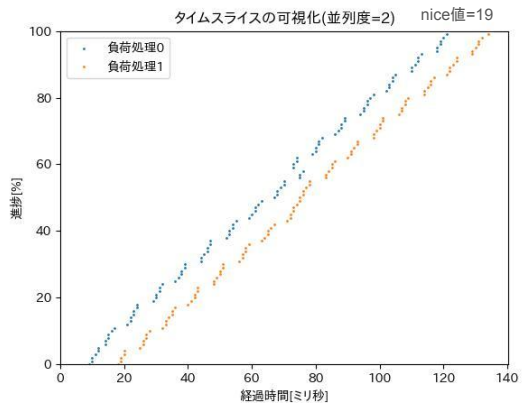


nice値が大きくなるとタイムスライスが短くなっている

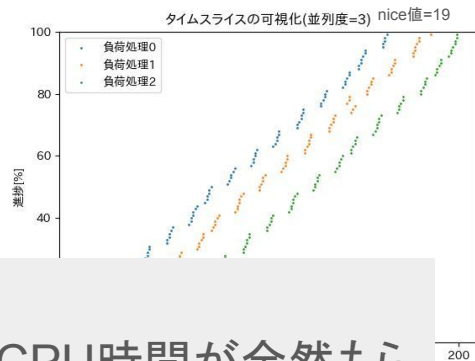
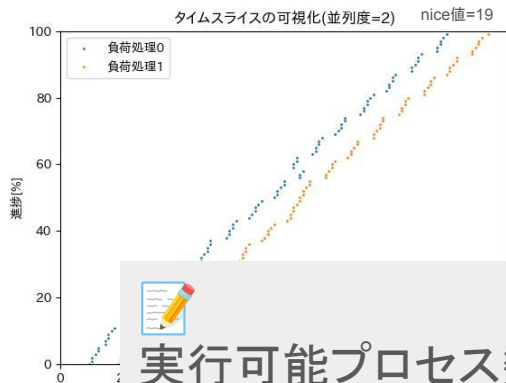


nice値0: タイムスライスは100ms
nice値19: タイムスライスは5ms

並列数が増えてもタイムスライスが変わらない

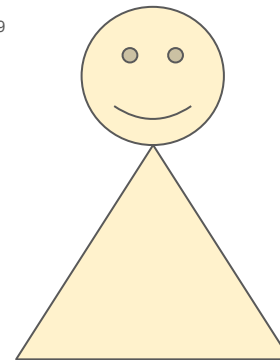
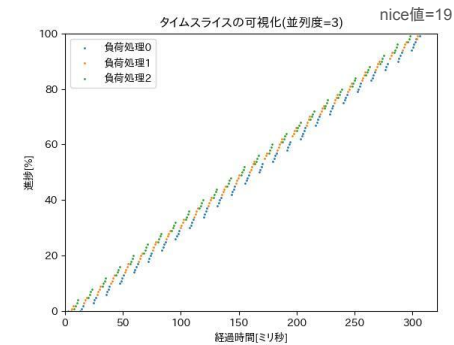
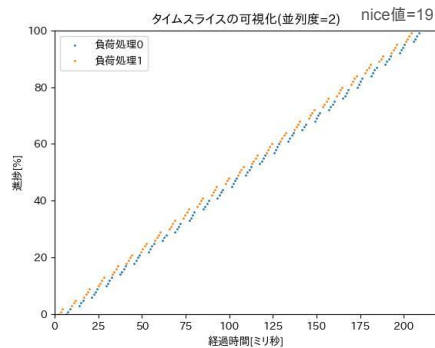
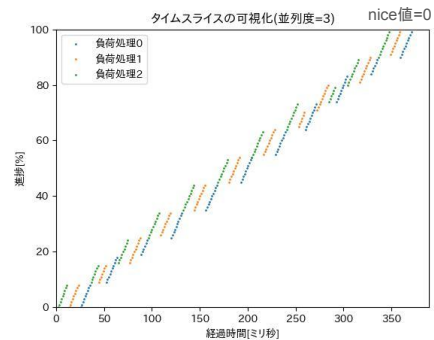
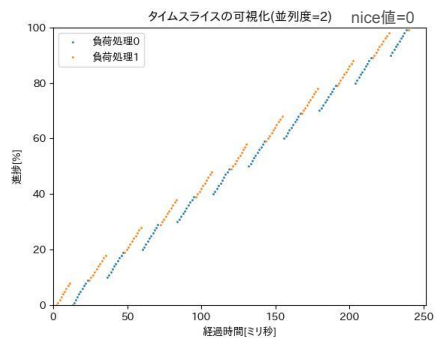


並列数が増えてもタイムスライスが変わらない

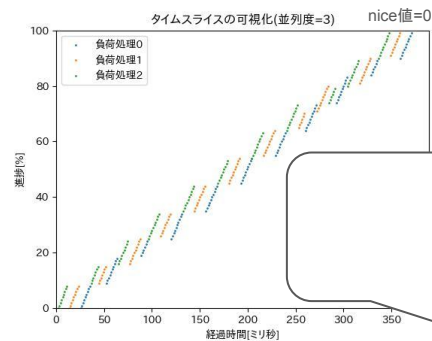
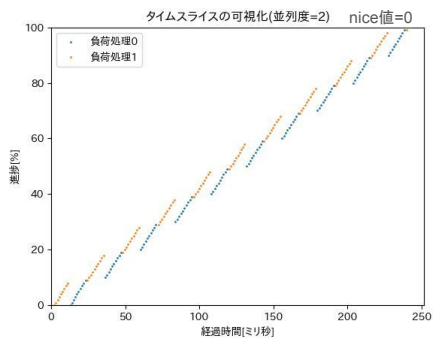


実行可能プロセス数が増えるとCPU時間が全然もらえなくなりがちという問題がある

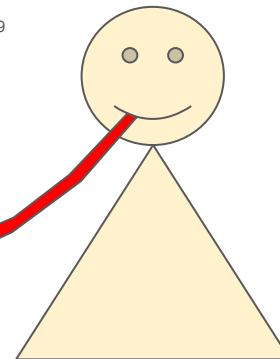
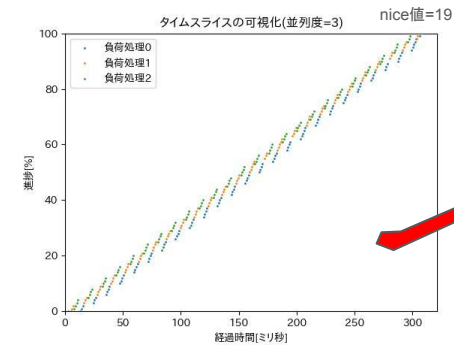
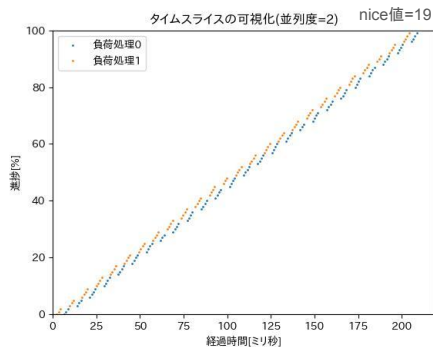
環境Bは...



環境Bは...



ぺろり...
これはCFS!

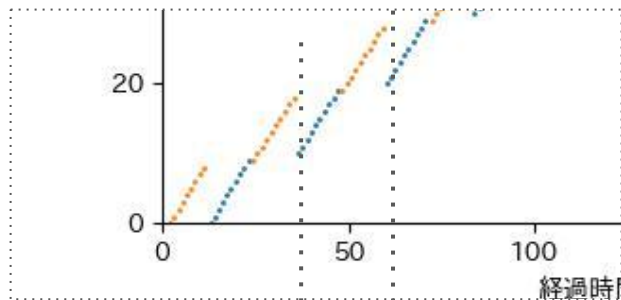


根拠

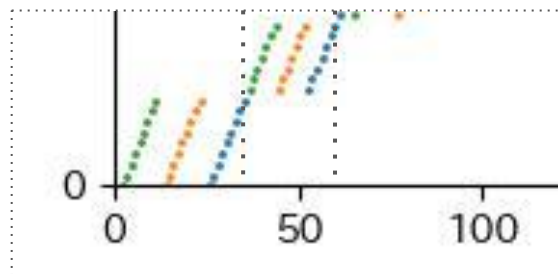
- CFSは実行可能プロセス数が多くなるとタイムスライスが短くなる
 - 全プロセスの実行が一周する期間を指す「レイテンシターゲット」という概念がある
 - レイテンシターゲットは一定値に保たれる
 - タイムスライスは” $\frac{\text{レイテンシターゲット}}{\text{実行可能なプロセス数}}$ ”になる
- この特徴を持つのはCFSだけ

レイテンシターゲットが一定

並列度2, nice値0

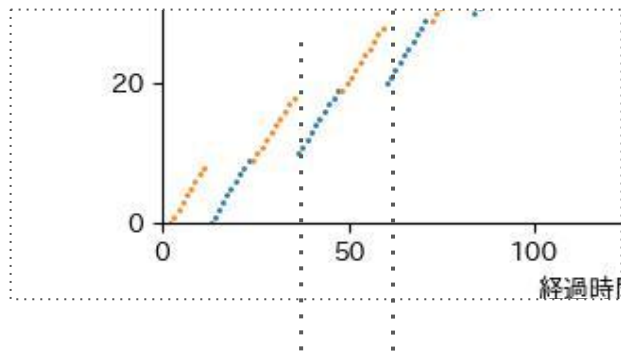


並列度3, nice値0



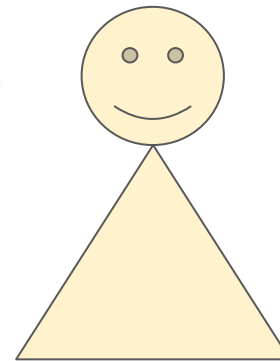
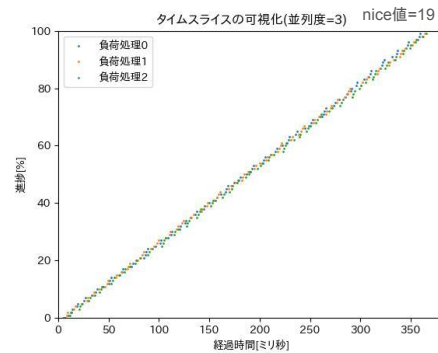
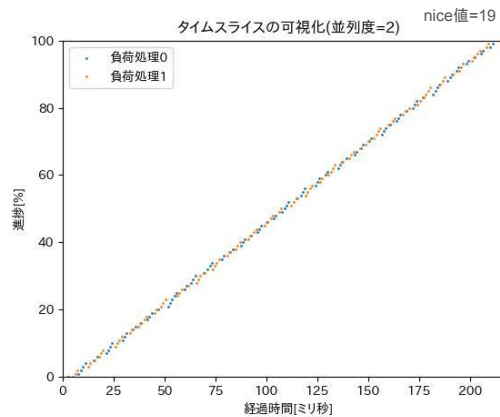
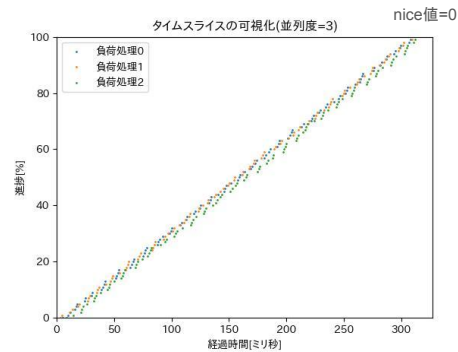
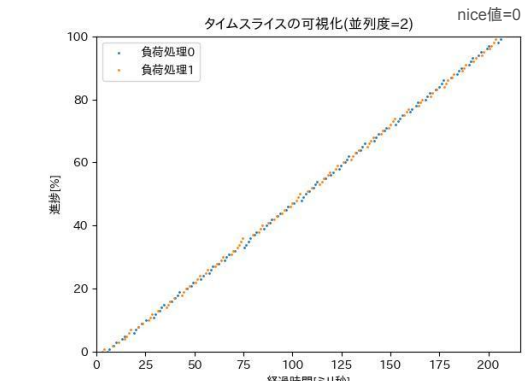
レイテンシターゲットが一定

並列度2, nice値0

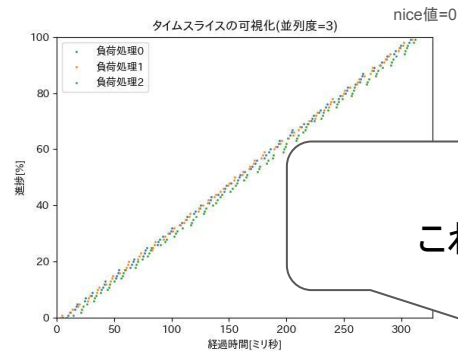
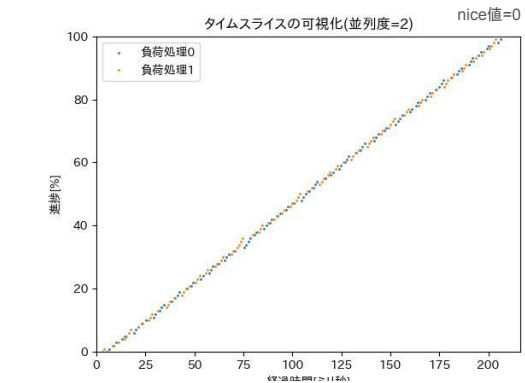


- レイテンシターゲットは” $\langle \text{定数} \rangle * (1 + \log_2(\langle \text{コア数} \rangle))$ ”
 - 環境Bは定数が6[ms]でコア数8なので $6 * 4 = 24$ [ms]
- 実行可能プロセス数が増えすぎるとタイムスライスは無限に小さくならず、最低保証値がある。この場合はレイテンシターゲットが長くなる

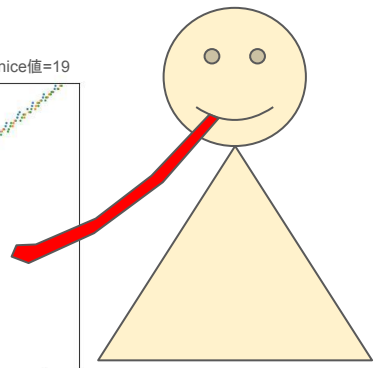
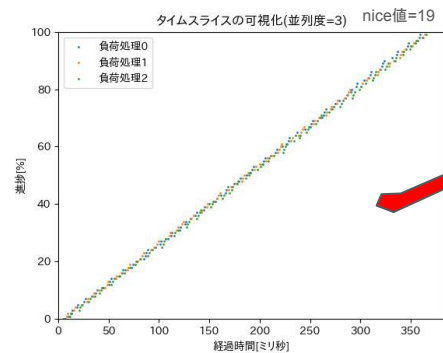
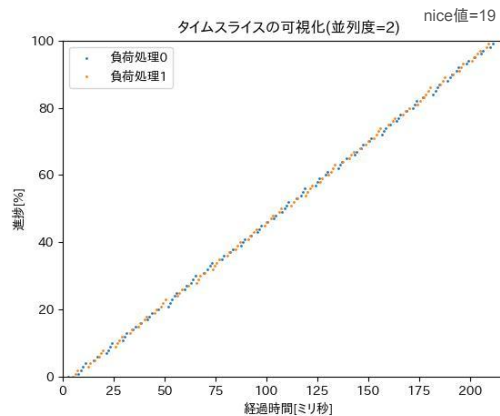
環境Cは...



環境Cは...



ぺろり...
これはEEVDF!

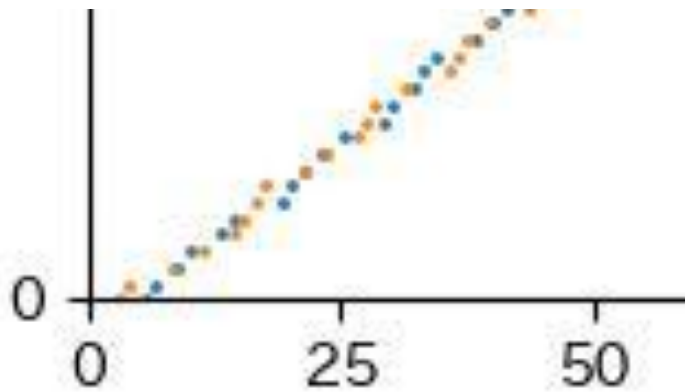


根拠

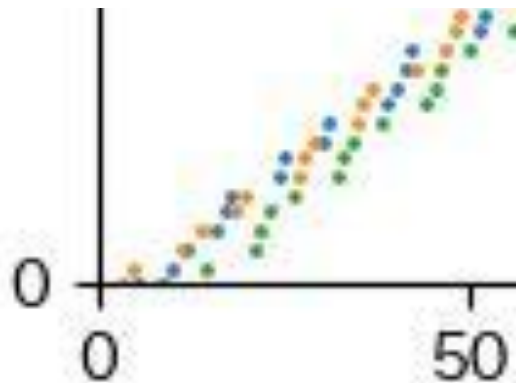
- EEVDFの特徴
 - 実行可能プロセス数が増えてもタイムスライス是不変
 - nice値を大きくしてもタイムスライス是不変
- 2つの特徴を兼ね備えるのはEEVDFのみ

実行可能プロセス数が増えてもタイムスライスは不変

並列度2, nice値0

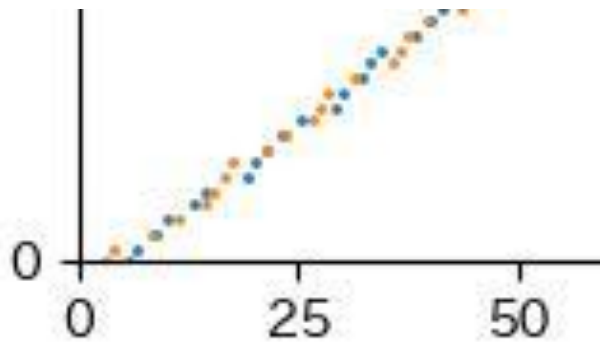


並列度3, nice値0

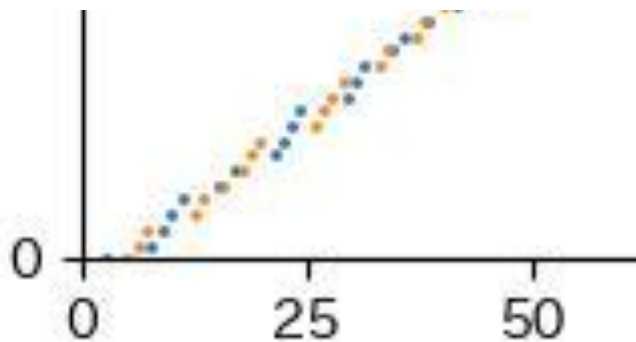


nice値を大きくしてもタイムスライス是不変

並列度2, nice値0

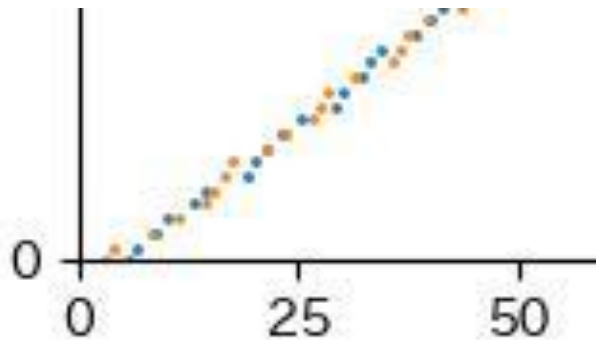


並列度2, nice値19



nice値を大きくしてもタイムスライスは変わらない

並列度2, nice値0



並列度2, nice値19






- タイムスライスは”<定数>*(1+log₂(<コア数>))”
 - 環境Cは定数が0.75[ms]で8コアなので0.75*4=3ms

まとめ

- 実行可能プロセスが数個程度なら以下のことが言える

	実行可能プロセスが増えるとタイムスライスが...	nice値が大きくなるとタイムスライスが...
O(1)スケジューラ	変わらない	短くなる
CFS	短くなる	短くなる
EEVDF	変わらない	変わらない

参考情報

- 各環境
 - 環境A: Ubuntu 7.04, linux v2.6.20 ->  O(1)スケジューラ
 - 環境B: Ubuntu 24.04, linux v5.15 ->  CFS
 - 環境C: Ubuntu 24.04, linux v6.8 ->  EEVDF
- ./sched.pyのソース、全グラフと元データのありか
 - <https://github.com/satoru-takeuchi/sched-tasting>

終わり

プロセススケジューラおいしい

