ハードリアルタイムスケジューリング理論入門

14th Nov. 1997

高田 広章
ITRON専門委員会／東京大学 理学部

hiro@is.s.u-tokyo.ac.jp
http://tron.um.u-tokyo.ac.jp/~hiro/

リアルタイムシステムとは？

- 処理の正しさが処理結果を出す時刻に依存するシステム

↓

処理に対する時間制約

従来のリアルタイム構築方法

- 経験に基づいたアドホックな方法で設計
  → テストによって時間制約を満たすことを確かめる
- システム設計時に強い制約を課す

↓

- システムの保守性が悪い
- 作ってみるまで時間制約を満たすかがわからない

↓

システマティックな開発手法の必要性

ハードリアルタイムシステム

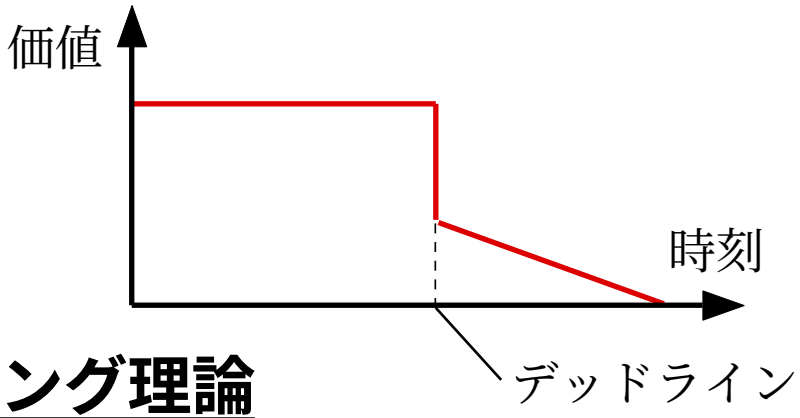
✕ ソフトリアルタイムシステム

定義1: 時間制約を満たせなかった場合に重大な事態
人命にかかわる, 大きな経済的損失

定義2: デッドラインにおいて価値関数が不連続
➡ firm real-time と呼ぶ場合も

⬇

時間制約を満たすことを
保証する設計をしたい



ハードリアルタイムスケジューリング理論

- 1980年代後半から米国を中心に研究が進む
- 代表的な成果　*rate monotonic analysis* (RMA)

システムに対する負荷

- すべての処理が厳密に時間制約を満たして実行できることを保証するためには、システムに対する最大負荷が決まっていなければならない

↓

- 最大負荷を決定/想定してシステムを設計

$$\text{最大負荷} = \sum_{\text{各タスク}} \text{1回の最大実行時間} \times \text{最大実行頻度}$$

Transient Overload

- 想定したシステムの最大負荷を一時的に越えた状況

→ 次善の策を講じるべき

Admission Control

- システムに対する最大負荷が全く予測できない場合

リアルタイムスケジューリングの種類/分類

Cyclic Executive

- システム設計時に強い制約を課すことでリアルタイム性を確保
- 処理を短いタイムスライスに分割し、それらを大きい処理周期内のタイムフレームに当てはめていく

  ➡ システムの改造に弱い

Preemptive Scheduling

- 実行中のタスクからプロセッサを横取りして先に実行

優先度ベーススケジューリング

- 優先度が高いタスクから順に実行

静的優先度割付と動的優先度割付

Liu and Layland の結果

- リアルタイムスケジューリング理論の古典的な結果

タスクセットに対する仮定

1. 周期タスク (periodic task) のみを考える
2. 各タスクの最大実行時間はあらかじめわかっている
3. 各タスクは周期の始めに実行可能に
4. 各タスクのデッドラインは周期の終わり
5. タスク間に同期・通信がない
6. タスクは自ら実行を中断することはない
7. タスク切替等のオーバヘッドは考えない
8. プロセッサが 1つのケースのみ考える

   ➡ プロセッサに対する負荷が変動しないタスクセット

- タスクセットのスケジュール可能性を議論

Rate Monotonic Scheduling (RMS)

周期の短いタスクから順に高い優先度を割り付ける

- 最適な静的優先度割付
- タスクセットがスケジュール可能になる十分条件

$$\sum_{j=1}^{n} \frac{C_j}{T_j} \leq n\,(2^{1/n} - 1)$$

n	右辺
2	0.83
3	0.78
∞	0.69

n : タスク数

T_j : 優先度が j 番目のタスクの周期

C_j : 優先度が j 番目のタスクの最大実行時間

➡ 悲観的な上限値　*平均 0.88 まで大丈夫という結果も*

Earliest Deadline First Scheduling (EDFS)

デッドラインの近いタスクから順に高い優先度を割り付ける

- 最適な動的優先度割付
- タスクセットがスケジュール可能になる必要十分条件

$$\sum_{j=1}^{n} \frac{C_j}{T_j} \leq 1$$

➡ *プロセッサの能力を 100% 使える*

n : タスク数

T_j : 優先度が j 番目のタスクの周期

C_j : 優先度が j 番目のタスクの最大実行時間

- Least Laxity First Scheduling も最適

RMS と EDFS の比較

- EDFS の方がプロセッサを有効に利用できる
  - ← RMS で優先度を静的に割り付けていることから来る本質的な限界
    - *cf. 周期の長いタスクの方がデッドラインが近い場合*
- RMS では transient overload 時の振舞が予測できる
  - → 優先度の低いタスクから順にデッドラインをミスする
- RMS の方が優先度のビット長が短くてよい
  - RMS: 8bit あれば十分 → 多くのRTOSと相性が良い
  - EDFS: 32bit 程度ほしい
- RMS では優先度の高いタスクに対して jitter が小さい
- RMS の方が前提条件を緩める研究が進んでいる

→ *一概にどちらが良いとは言えない*

RMSをベースに前提条件を緩める

1. 非周期タスクを考える

- 最小起動間隔を周期とする周期タスクと考えてよい
- 非周期タスクをなるべく早く実行する方法もある

4. タスクのデッドラインが周期の終わりに一致しない

- rate monotonic に代えて deadline monotonic を使う

デッドラインまでの時間が短いタスクから順に高い優先度を割り付ける

ただし、スケジュール可能性の判定条件は変わる

5. タスク間に同期・通信がある

7. タスク切替のオーバヘッドを考える

- タスク切替 2回分を各タスクの最大実行時間に加える

優先度逆転 (Priority Inversion)

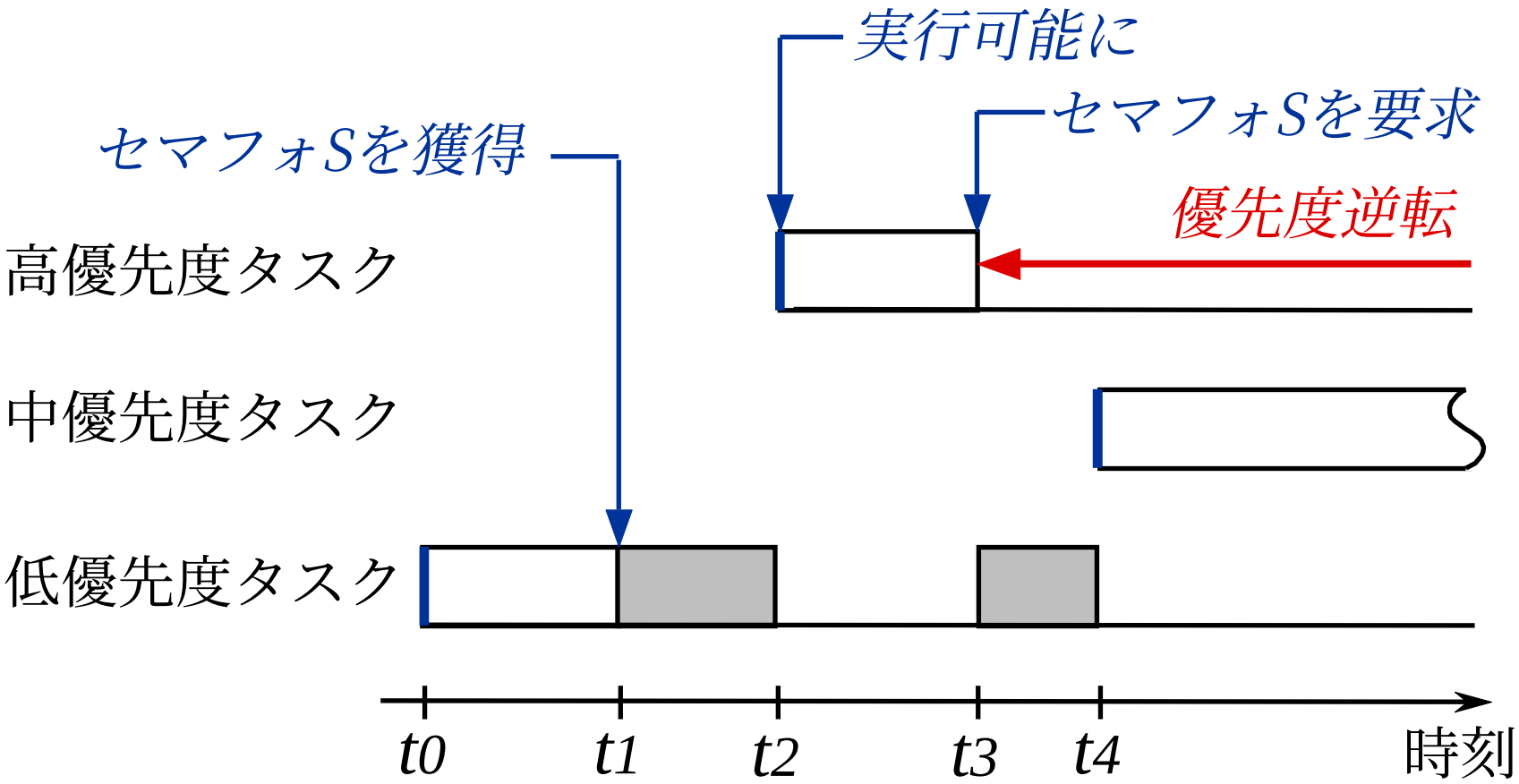
優先度の高い処理が優先度の低い処理に待たされること

Uncontrolled Priority Inversion

- 優先度逆転が無意味に長時間続く現象
- 処理が時間制約を満たせなくなる大きな原因の1つ

有力な解決策

- クリティカルセクション中のタスク切替の制限
  Kernelized Monitor, Highest Locker Protocol
- 優先度継承
  Basic Priority Inheritance Protocol
  Priority Ceiling Protocol
- クリティカルセクションのアボート



Uncontrolled Priority Inversion の例

QOS制御の必要性

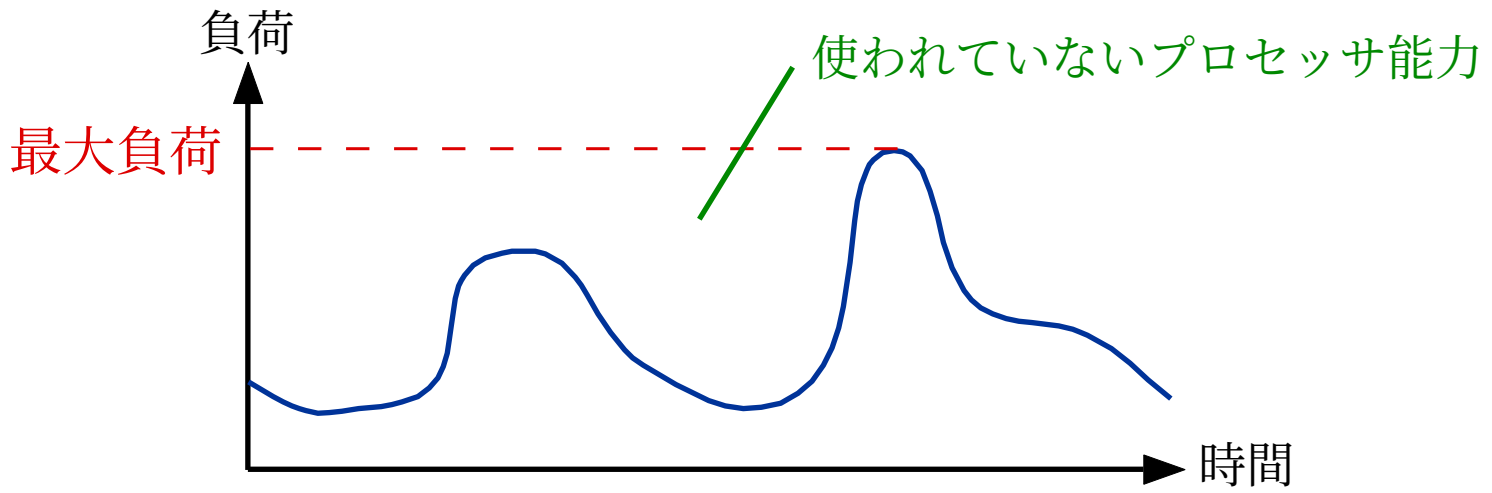
※ QOS (Quality of Service) ＝ サービスの品質

- Transient overload 時には、サービスの品質 (計算の精度, 制御の精度) を落とすことで負荷を下げる

➡ プロセッサの能力を有効に利用

⬆

システムに対する負荷は変動する



QOS制御の実現手法

- 静的な QOS制御

  過負荷状態の発生を未然に検出し、システムの動作モードを変更する

- 動的な QOS制御

  過負荷状態が発生したのを検出し、(できる限り重要性の低い) 一部の処理をさぼる

動的な QOS制御実現の手法の例

- 重要性の高い処理と低い処理を別のタスクにし、重要性の低い処理の優先度を下げる
- 1つのタスク内で重要性の高い処理ほど前に実行し、過負荷になればタスクを途中で打ち切る

QOSと実行時間のトレード手法

Milestone Method

- 処理が進むほど精度が上がるようにプログラムを作成
- 過負荷時は処理を途中で打ち切る

Sieve Method

- 実行することで前の結果の精度を上げるような処理を過負荷時にはスキップ
- 同じ処理が連続してスキップされない配慮が必要

Multi-version Method

- QOS は高いが実行時間が長いバージョンと、QOS は低いが実行時間が短いバージョンを用意

上のアプローチほどスケジュールが容易
下のアプローチほど柔軟

Overrun のチェックと Policing

※ Overrun ＝ 見積もった以上の計算時間を使うこと

- タスクの overrun を放置すると、システム全体の transient overload につながる

↓

タスクの overrun を早めに検出し、それ以上の計算時間を与えないというアプローチ ＝ policing

End-to-Endの QOS制御

- 分散システムでは、サーバから端末に至る各ノードで計算資源を確保する必要
- 一箇所ででも資源が不足すると、指定した品質ではサービスできない

→ 何も制御しないと他の資源の無駄使いにつながる

割込みハンドラとタスク

- 理論上は割込みハンドラもタスクとして扱えば良い
  - 割込みハンドラも優先度を持てる (静的割付のみ)
  - 割込みの周期が短い程、高い優先度を割り付ける

割込みハンドラとタスクの扱いの違い

- 割込みハンドラの優先度をタスクの優先度より低くすることはできない
- 割込みハンドラは待ち状態を持たない
  - ➡ タスク間同期の方法に制約
- 割込みハンドラの方が起動/終了オーバヘッドが小さい

注意点

- ! 割込みハンドラの policing が重要になるケースがある

まとめ

- ハードリアルタイムスケジューリング理論の基礎を紹介

他のリアルタイムスケジューリング理論

- Proportional Resource Share

  各タスクにあらかじめ定めた比率のプロセッサ時間を割り当てる ➡ *理想ケースとの差を押さえる*

- より動的なタスクスケジューリング

ITRONプロジェクトでの取り組み

- ITRONハードリアルタイムサポート研究会
- リアルタイムスケジューリング理論に基づいたアプリケーション構築ガイドラインの作成 (ソフトウェア部品を重視)
- それをサポートするためのカーネル仕様の検討