組込みOSの現状と未来

早稲田大学理工学術院
コンピュータネットワーク工学科
中島　達夫

1

発表概要

- 組込みOSの現在と問題点
  - 特にLinuxを中心に
- 次世代組込みシステムのためのOS
- まとめ

2

組込みシステムの時代

- 情報サービスや電子制御が当たり前になってきた。
  - 車載システムの革新の80%が電子制御に依存している。
  - 携帯電話やデジタルテレビなど情報サービスが増大している
  - センサーやアクチュエータの制御は電子制御があたりまえ
  - 組込みシステムがネットワーク接続により機能を統合できるようになっていく
- 微細化の進展によるさらなるソフトウエア化が進んでいくと思われる
  - 表示技術の進化、小型モータ、センサーのユビキタス化

3

組込みシステム

- 情報系組込みシステム
  - 携帯電話、デジタルテレビ、カーナビ
  - メディア処理、インターネットアクセス、ユーザエクスペリエンス
- 制御系組込みシステム
  - ロボット、車載機器
  - センサーとアクチュエータ、実時間処理、セーフクリティカル
- 組込みシステムの本質は制御系処理と制御系処理の融合が重要であることである。

4

なぜOSが必要か？

- ソフトウエア再利用の基盤としてのOS
  - 標準化によるソフトウエア資産の再利用
  - インフラのための共通言語
  - 共通ソフトウエアの利用
    - TCP/IP, ファイルシステム, デバイスドライバ
- ツールとしてのOS
  - 高レベル抽象化の提供によるプログラミングの容易化
    - スレッド、イベント、ケーパビリティ
  - アイソレーション、信頼性、セキュリティ、実時間性等の非機能要件の支援

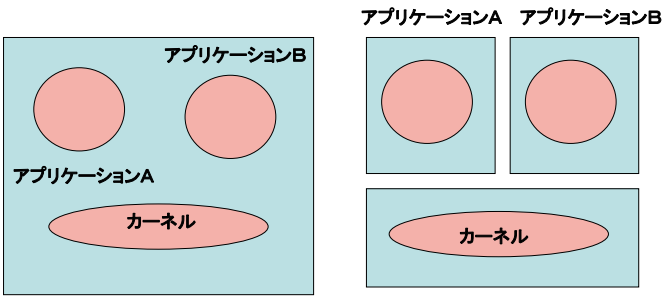
5

組込みOS

- 比較的小型の組み込みシステム向け
  - ITRON
  - VxWorks
  - pSOS
  - OSEK
  - Toppers
  - TinyOS
- 比較的複雑な組込みシステム向け
  - Linux
  - Windows CE
  - Symbian
  - Real-Time Mach

6

組込みOSの構造



a) μITRON　　b) Linux

7

複雑な組込みシステムとOSの関係

- アイソレーションがシステムの高信頼化の基本
  - メモリ隔離を利用することにより、あるアプリケーションのメモリエラーが別なアプリケーションには伝播しない。
  - アプリケーションのメモリエラーがカーネルには伝播しない。
  - Linuxでは、カーネルとデバイスドライバは同一のメモリ空間内に存在する。
- 特に、オープンプラットフォーム化していく次世代の組込みシステムでは信頼性とセキュリティの保証が極めて重要となっていく

8

- **台湾FIC、プログラム可能なLinux携帯電話を発表**
  - TrolltechがLinuxベースの携帯電話「Qtopia Greenphone」を開発者コミュニティに向けて発表してからわずか数カ月後、プログラム可能でオープンなLinuxを採用した携帯電話の2機種目が登場した。
  - オランダのアムステルダムで現地時間11月7日から8日にかけて開催されたカンファレンス「Open Source in Mobile」の中で、台湾のハードウェアメーカーFirst International Computer（FIC）のSean Moss-Pultz氏は、「OpenMoko」というLinuxベースの環境を実行するスマートフォン「Neo1973」を紹介した。

9

ZDNet JAPANより 2006/11/10

組込みLinux（1）

- 組込みシステムの情報系処理の増大
- インターネットへの接続が当たり前になっている
  - メモリ保護、リアルタイムスケジューラ
  - セキュリティ機能（マルチユーザ、SELinux）
  - 様々なインターネットサービス
  - ウインドウシステムなどのミドルウエア

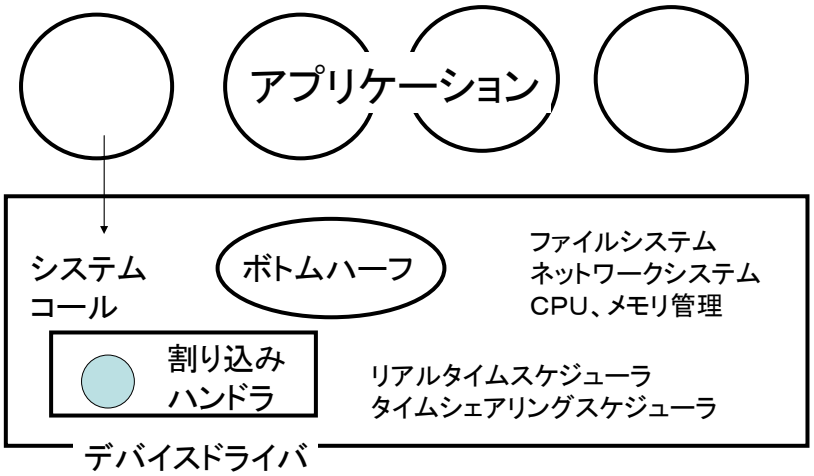
10

組込みLinux（2）

- 多くのLinuxを採用する組込みシステムでは、制御系処理用のCPUを利用している。
  - 制御系処理のためμITRONが広く使われている
    - 実時間サポートと既存のμITRONの資産を利用するため
  - 情報系処理と制御系処理の間の通信はアドホックに扱われている

11

Linuxカーネルの概要



12

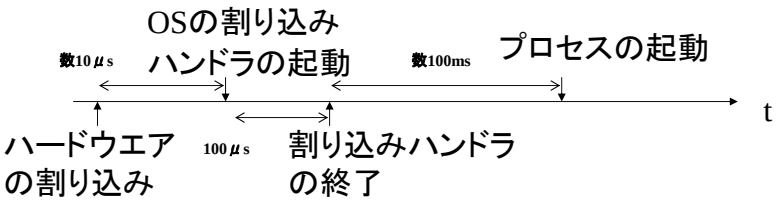
組込みシステムにおける問題点

- 応答性の改善
- ブートタイムの改善
- リソース管理
- OS APIのインパクト

13

応答性が低下する原因

- 低下する要因



14

応答性の改善（1）

- 応答性の改善
  - マルチメディア処理、入出力管理、通信処理
  - 組込みシステムはスイッチをONにしたらすぐに利用可能とする必要があるスケジューリングレイテンシ
  - 割り込み禁止区間
    - 100μs程度
  - 割り込みハンドラ
    - 100μ程度
  - カーネルプリエンプション
    - システムコールの実行時間
    - 数100ms程度
  - ボトムハーフ
    - ネットワーク処理、タイマー処理
    - システムコールの実行時間を増加する原因でもある.

15

応答性の改善(2)

- 基本的な問題点
  - システムコールの実行中はプリエンプションすることが出来ない
  - システムコールの実行時間が長いと応答性が低下する可能性がある
    - 現状のLinuxは100ms以上応答性が低下することがある。

16

応答性の改善(3)

- カーネルをプリエンプタブルにする
  - 割り込みハンドラを抜けるときにより優先度の高いプロセスが存在するときはコンテキストスイッチが起きるようにする
  - SMPロックを取得中はSMPロックが解放されるまでコンテキストスイッチを遅らせる
  - 応答時間がSMPロックを取得している時間にまで短くなる
  - SMPロックを極力短くする
  - 割り込み不可の部分を出来る限り短くする

17

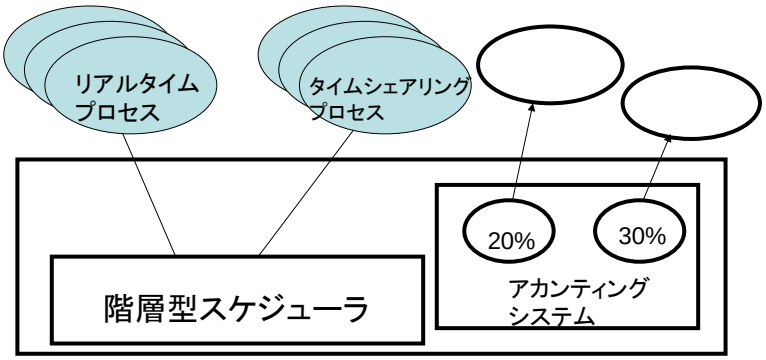
ブートタイムの改善

- 組込みシステムでは電源をONにして瞬時に立ち上がる必要がある。
  - CPUクロックのキャリブレーション
  - 存在しないデバイスのプルーブの省略
  - 出来る限り不必要な初期化を省略
  - 初期化時に必要でない処理をブート後におこなうようにする

18

CPUリソース管理

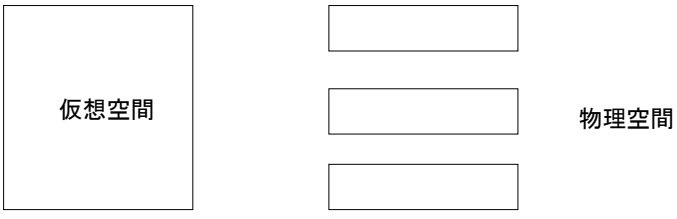
- 各プロセスが使用するリソース量を制限する
  - CPU利用率、メモリ、ファイルブロック、ネットワークキャパシティ



19

メモリリソース管理

- 現在のLinuxは遅延評価を利用してプロセス生成をおこなっているため、物理メモリが足りなくなると任意のプロセスを停止したり、システム全体がクラッシュする可能性がある。
- オープンプラットフォームを実現するためには、信頼性が低いプロセスが物理メモリを必要以上に使用することを避けることが重要である。
- 各プロセスが使用する物理メモリを制御するための仕組みが重要である。



20

OS APIのインパクト（1）

- プロセスのクラッシュを検出できない
  - TCPやpipe
    - コネクションの障害とプロセスの障害を区別できない。エラーが返ったので相手プロセスがクラッシュしたとは限らない。
  - プロセスのクラッシュを正確に検出できるOS APIがないとシステム全体の信頼性を向上することは不可能である
- 障害時の振る舞いが予測可能でない
- オーバロード時の振る舞いが予測可能でない

21

OS APIのインパクト（2）

- セキュリティ
  - すべてのリソースにユニフォームなラベルをつけることを可能とする必要がある
  - 各リソースの制御をケーパビリティを介しておこなえるようにする
  - 名前でアクセスする場合は、ポリシに応じたアクセス制御をおこなう必要がある。
- リアルタイム
  - 明示的に優先度を指定するAPIが必要
    - プロセス間で制御を移す場合は、一貫性の取れた優先度をマッピングする必要がある。
    - 実行を中断する処理を実行する場合も優先度に従う必要がある
  - システム全体の優先度の一貫性を維持する必要がある

22

組込みOSの選択

- すべての要求を満足する組込みOSは存在しない
  - セキュリティ、信頼性、リアルタイムなど保証のレベルが上がる毎に実装が複雑になる。
  - 出来るだけ不必要な機能は提供しない。
  - 例えば、ITRONとLinuxのどちらが優れているかという議論は無意味である。
- 複数のOSを組み合わせる方法が重要
  - 各機能毎に適したOSを利用することが可能となる。
  - マルチコアプロセッサになると動的にどのコアを利用するかを選択できるようになる

23

次世代の組込みシステム

- 特殊化された組込みシステム
  - 比較的小規模で、特殊な目的に少量必要
- 超高機能組込みシステム（強力な情報系）
  - 情報家電を代表に高機能を実現するため、大量のコードが必要
- セーフティクリティカルシステム（強力な制御系）
  - 安全性が最も重要なファクター
- これらが複雑に入り組んでいく

24

より複雑さを増す組込みシステム

- 高機能化
  - コミュニケーション、センシング、ストレージ
- ネットワーク化
  - あらゆるものがつながり、機能を複合化する
- 組込みシステム
  - 制御機器、家電機器、携帯電話、自動車…
  - 建築、土木、オフィス、パブリックスペース
- 異なる分野が異なるプラットフォームを用いるのはコスト増加をもたらす
- ハードウエア、ソフトウエアの共通化が重要

25

ネットワーク化のインパクト

- 情報処理の日常物への組込み
  - 大量の情報を適時に利用可能となる
  - コミュニケーションの支援
  - 生活環境を安全、快適でストレスを少なくする
- Webサービスとの統合
  - Web2.0のインパクト
  - 情報やサービスを個人が発信するインパクト
- センサーの利用
  - 実世界のモデルを利用可能となるインパクト

26

次世代高機能組込みシステム

- 次世代の情報家電では情報系・制御系の両方での機能が拡張していく
  - 機能の向上により人間により適応するサービスが提供可能となる
    - よりストレスが少ないサービス
  - ユーザエクスペリエンスを向上する多様なインタラクションデバイス
  - センサーやアクチュエータの利用の拡大
    - 例えば、家庭はロボット・自動車化していく

27

次世代組込みシステム

- 問題点
  - 信頼性とセキュリティの向上
    - アイソレーションの支援が不十分
  - 情報系と制御系統合
    - 現在は統合がアドホック
  - プラットフォームアーキテクチャの欠如
    - 現在は、ハードウエアのみ進んでいる
  - 診断機能
    - 現在は、共通に扱えるフレームワークがない
  - I/Oマネジメント
    - 現在は、従来のRTOSを利用するなどアドホック
  - マルチコア管理
    - SMP以外の構成をうまく扱えない

28

高度なアイソレーションの支援

- 信頼性やセキュリティの向上
  - メモリ保護
  - リソースアイソレーション(QoS Crosstalk)
    - CPU, I/Oデバイス, メモリ
- 独立したモジュール化が問題解決の基本
  - Linuxが提供するプロセスという抽象化をベースとするアイソレーションでは柔軟性が低い
  - Linuxカーネル内はアイソレーションのサポートがない

29

情報系処理と制御系処理の融合

- 情報系処理と制御系処理間の通信方式
  - 通信方式を共通化することによりお互いを独立に扱えるようにする
  - お互いが独立に動作するため非同期の通信をベースとする必要がある
- 機能と実行CPUの動的マッピング
  - ソフトウエア機能とハードウエアを独立に扱えるようにすることにより柔軟にシステムのファミリーを作ることが出来るようになる

30

様々な要求

- APIセマンティクスのギャップ
  - APIの抽象レベルが高くなればアプリケーションの開発が容易になる
  - しかし、抽象レベルが高くなるとミスマッチの可能性も高くなる
- 実装方式のギャップ
  - 高信頼、高セキュリティ、実時間性と性能は常にトレードオフの関係にある。
  - つまり、最適な実装法はアプリケーションに応じて異なり、1つのOSでは不十分となる。

31

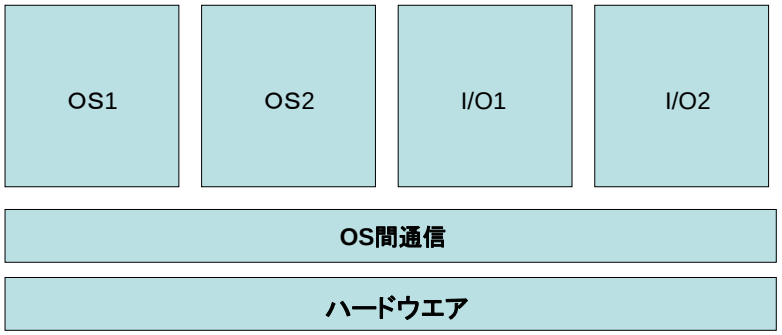
問題解決の手法

- 仮想化の利用
  - 低レベルのモジュールに対してもアイソレーションを利用可能とする
    - I/Oデバイスドライバ、セーフティクリティカルな制御系処理
  - 物理リソースを直接扱い、アイソレーションすることが可能となる
    - CPU, メモリ
- フレームワークの提供
  - 大域物理リソース管理
  - 診断フレームワーク
  - I/Oデバイス管理

32

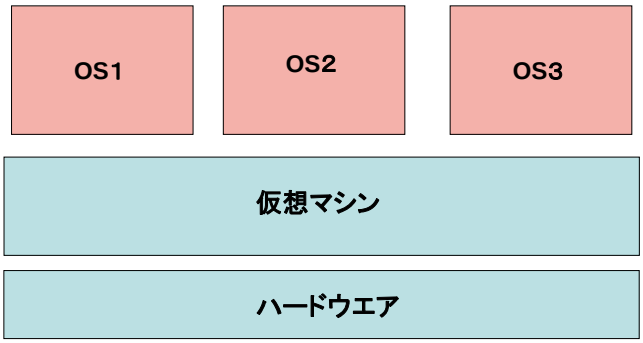
プラットフォームアーキテクチャ

- 既存のOSを拡張することにより様々な要求を満足するようにする



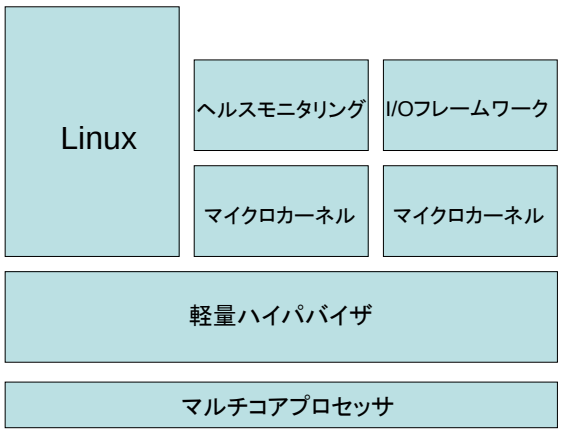
33

仮想化の利用



34

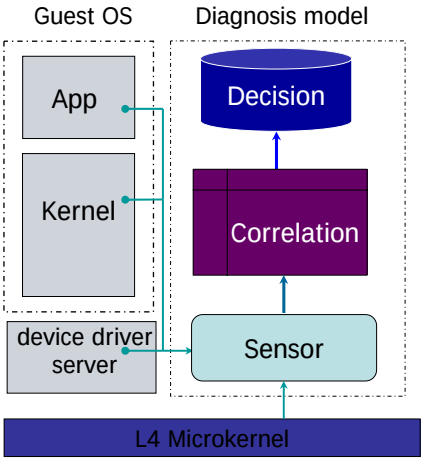
システム構造



35

ヘルスモニタリングフレームワーク

- OSの信頼性の向上
  - システムインテグリティ
  - 異常検出
  - システムプロファイリング
- μカーネル上の診断モジュール
  - Linux等のゲストOSとは独立
  - フォールト隔離可能
  - 汎用診断フレームワーク
- Linuxカーネルの完全性管理
  - セキュリティの向上



Micro-kernel diagnosis model

36

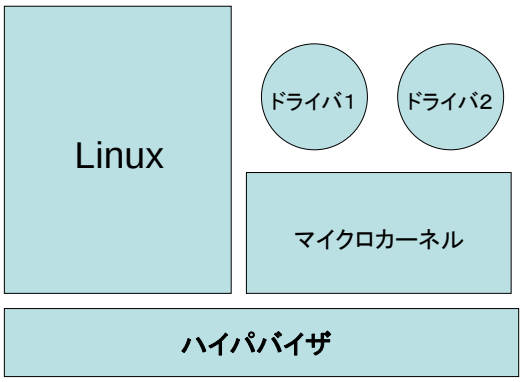
OS間通信機能

- 制御系と情報系処理間の通信に主に使う
- デバイスドライバへのアクセス

- 高レベル通信API
  - 大きなデータを効率よく転送するために共有メモリを利用する
  - 非同期通知、サブスクリプション等のイベントベースの通信をベースにする
  - マシン内とマシン外で同じ抽象化を利用する
- 低レベル通信API
  - マイクロカーネルが提供するIPCを用いる
  - 各コア間の通信はプロセッサ割り込みを用いる
  - 共有メモリ、高速通信ネットワーク

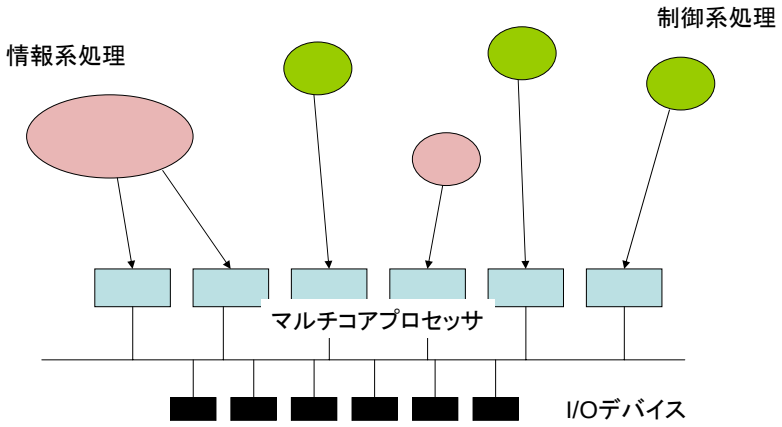
37

デバイスドライバの分離



38

システム構成



39

将来課題(1)

- ハードウエアの一時的エラー率の増加
  - 半導体の微細化
  - 使用電圧の低下
  - インストラクション部のエラー
    - 命令違反
    - 無限ループ
  - データ部のエラー
    - 情報の変更
    - 間違ったポインタのアクセス

40

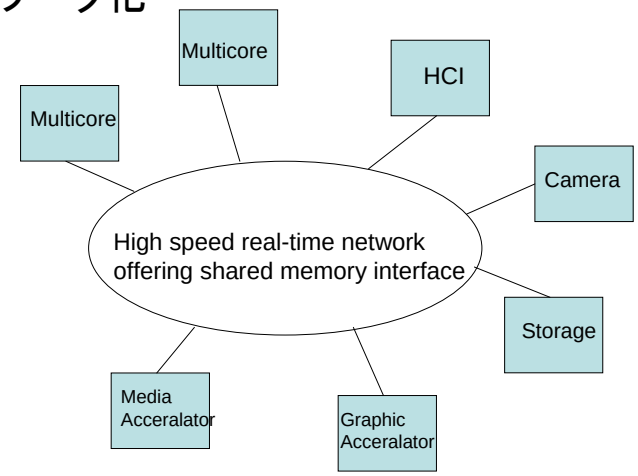
将来課題（2）

- パワーマネジメントとマルチコア
  - 各コア毎にパワーをON/OFF可能とする
  - OFFとなるプロセッサのレディーキューにプロセスが存在する場合は、別なプロセッサに移動する必要がある。
  - スタンドバイ時の処理には最小のコアで処理の実行をおこなう。

41

将来課題（3）

- ネットワーク化



42

オープンソースとしてのLinux

- 様々な理由から選択肢がないのでLinuxとなった。
- Linuxは多くのコミュニティの成果である。
  - コミュニティの大多数はサーバ系の開発者である
  - Linuxはよりサーバに適したOSへと進化している
  - 数年後も組込みシステムに適しているのか？
  - 例　リソース管理フレームワーク
    - Class-based Kernel Resource Management(CKRM)
    - 仮想化と統計的リソース割り当て
  - その他　障害時、オーバロード時の振る舞い
- 組込みコミュニティからの貢献がないとLinuxは組込みシステムで使えなくなってしまうかもしれない
- オープンなOSとはただで使えるOSではなく、使う人の協力が最も重要である。

43

まとめ

- 組込みシステムの重要性は益々増加していく。
- ネットワーク化や高性能化、高機能化等様々な新しい問題を解決していく必要がある。
- 次世代のソフトウエアプラットフォームは以下の3つの問題を考慮する必要がある
  - 信頼性とセキュリティの向上
  - 情報系処理と制御系処理の融合
  - 既存のOS上の資産の有効利用

44