

オペレーティングシステム



資料 第 **1** 分冊(H27)

村田正幸 (murata@ist.osaka-u.ac.jp)
○松田秀雄(matsuda@ist.osaka-u.ac.jp)

講義の進め方について(1)

- 情報科学科 3年配当
- 「計算機アーキテクチャ」の受講を前提とする
- 教科書 近代科学社「コンピュータサイエンスで学ぶ オペレーティングシステム OS学」
- ほぼ教科書に沿って講義を進める
- 教科書にとりあげられていない話題については、別途プリントを配布する
 - デッドロックと資源割り当てなど

講義の進め方について(2)

- 松田担当分 7回 4/13~6/1
- 村田先生担当分 7回 6/8~7/27
(松田担当分の内容については別紙を読むこと)
(参考 8/1~8/2 情報科学研究科 大学院入試)
- レポートを、松田・村田担当分の中でそれぞれ課す予定
- 成績:レポート50%、試験50%

オペレーティングシステムはなぜ必要か？

- オペレーティングシステムがないと何が起る？
例えば、
 - コンピュータシステムで、プロセッサとユーザインタフェース(入出力機器)をつないで使うことができない
 - プロセッサ(クロックGHz) ナノ(10^{-9})秒単位で動作
 - ユーザインタフェース ミリ(10^{-3})秒~秒単位で動作
 - コンピュータシステムを効率よく使うことができない
 - 例 一度に一つのプログラムしか実行できない
 - 一度に一人のユーザしか実行できない
- 今はオペレーティングシステムについて知るチャンス？(理由は後で)

オペレーティングシステムについて 知る必要性(1)

- オペレーティングシステムを知らないと何が起るか？
 - コンピュータシステムの異常時に、ハードウェアの障害かソフトウェアの障害かを切り分けできない
 - 例 キー入力をして何も反応しなくなった



オペレーティングシステムについて 知る必要性(2)

- オペレーティングシステムを知らないときの対処
キーボードを引っこ抜いて、もう一度指してみる
電源を切って、もう一度電源を入れてみる
→ **プロのやり方ではない！**
- オペレーティングシステムを知っているときの対処
Unix系 他のコンピュータからリモートログインして、プロセスをkillする
Windows タスクマネージャを起動する
- 「情報のプロ」を目指すなら当然知っておくべき！

本講義の概要(松田担当分)

- 第1回 オペレーティングシステムの基礎概念
- 第2回 オペレーティングシステムの機能
- 第3回 オペレーティングシステムの構成と割り込み制御
- 第4回 プロセス管理の基礎概念
- 第5回 並行プロセス
- 第6回 プロセスの同期と相互排除
- 第7回 プロセス管理の実装

本講義の概要(村田先生担当分)

- 3. メモリ管理
 - 3. 1. メモリ管理技法
 - 3. 2. 仮想メモリ
 - 3. 3. ページ置換アルゴリズム
- 4. ファイルシステム
 - 4. 1. ファイルの管理と操作、ファイルアクセス方式
 - 4. 2. ファイル割り付けとスケジューリング
 - 4. 3. ファイルシステムの実装方法
 - 4. 4. UNIXにおける実際
- 5. 入出力制御
 - 5. 1. 入出力装置とその制御

1.1 OSの基本的な役割 ー 現代のOSと各世代のOS ー

- オペレーティングシステム: Operating Systemの頭文字を取って**OS**と書くことが多い
- OSの位置付け
 - ー OSはわかりにくい?
 - ー ソフトウェアの中では、一番、ユーザからなじみのない位置にある
 - ー 直感的な説明:「OSは、コンピュータシステムを構成するソフトウェアの中で、ユーザプログラムと言語処理プログラムを除いたもの」
 - ー これらを除いた後、いったい何が残っているのか?

コンピュータシステム

- ハードウェアとソフトウェアからなる(図1.1)
 - ー ハードウェアによる機能→**高速処理機能**
 - ー ソフトウェアによる機能→**問題適応機能**
- ハードウェアとソフトウェアで機能分担が必要
 - ー **コンピュータアーキテクチャ**
 - ハードウェアとソフトウェアとのインタフェースとして、両者の機能分担を決める
 - 実際には、マシン命令(マシン語)の機能レベルとして実現する

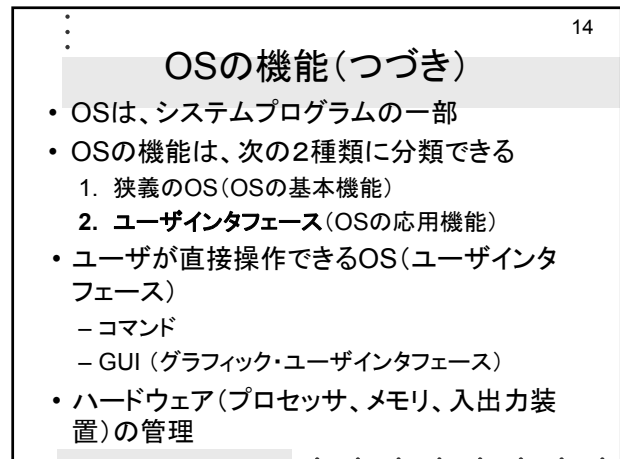
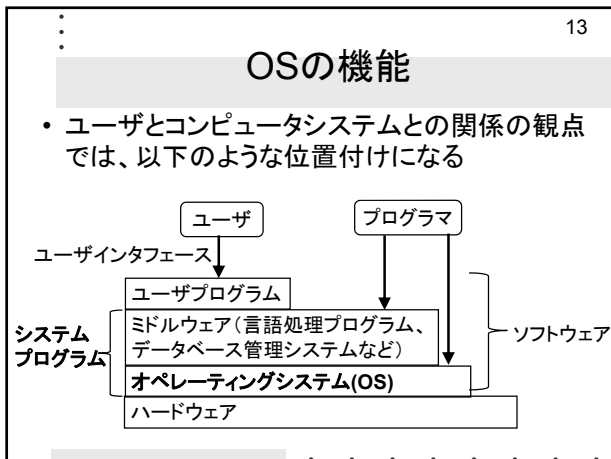
コンピュータシステムによる 情報処理過程

コンピュータシステムでの機能の実現

- **ハードウェアでの機能の実現**
 - マシン命令の機能を作成(既存のマシン命令の改善が主流)
- **ソフトウェアでの機能の実現**
 - 言語処理プログラム(コンパイラなど)によって、プログラムをマシン命令の列に翻訳または変換する(図1.2 コンピュータシステムによる情報処理過程)

基本ソフトウェア(= システムプログラム)と 応用ソフトウェア(= ユーザプログラム)

- **基本ソフトウェア**
 - ー コンピュータシステムに、原則として“唯一”登録されている
 - ー 基本プログラム、システムソフトウェア、システムプログラムともいう(以下、「**システムプログラム**」と呼ぶ)
- **応用ソフトウェア**
 - ー ユーザが後からインストールして使用する
 - ー 応用プログラム、ユーザソフトウェア、ユーザプログラムともいう(以下、「**ユーザプログラム**」と呼ぶ)



15

OSの発展史

- OSはコンピュータと共に発展
→コンピュータの世代に対応してOSにも世代がある

| 世代 | コンピュータ | OS |
|------|-----------|-----------------|
| 第1世代 | 真空管 | システムプログラムなし |
| 第2世代 | トランジスタ | 簡易システムプログラム |
| 第3世代 | IC(集積回路) | 本格的システムプログラム |
| 第4世代 | マイクロプロセッサ | 統合プログラミング環境 |
| 第5世代 | インターネット | 隠ぺいされたシステムプログラム |
| 第6世代 | ユビキタス | 透明なシステムプログラム |

16

OSの発展史(つづき)

- OSの発展の歴史は繰り返す？

| 世代 | 携帯電話 | OS |
|------|---------------|--------------------------------------|
| 第1世代 | 「携帯」する電話 | OSなし |
| 第2世代 | 簡易ネットワーク機能 | 簡易OS(i-モード, EZweb, ...) |
| 第3世代 | スマートフォン | 本格的OS(iOS, Android, ...) |
| 第4世代 | クラウドコンピューティング | インターネット上でサービスの提供(iCloud, Gmail, ...) |
| 第5世代 | ? | |
| 第6世代 | ? | |

17

第1世代(1940～1950)

- システムプログラムがない世代
- コンピュータシステムを、一人のユーザや単一のプログラムが占有して利用
- OSは不要
- ネットワーク機能がない、通話だけの携帯電話に相当

18

第2世代(1950～1960)

- 簡易システムプログラム世代
- 簡単な割り込み処理プログラムを、システムプログラムとして実装
- プログラミング言語の開発と使用(言語処理プログラムの登場)
- バッチ処理(複数人のユーザや複数個のユーザプログラムを一括して処理)が主流
- 簡易ネットワーク(i-mode)機能のレベル

第3世代(1960～1970)

- 本格的システムプログラム世代
- システムプログラムは、言語処理プログラム(実行前機能)とOS(実行時機能)に明確に役割を分担
- バッチ処理に加えて、TSS (Time Sharing System: コンピュータの処理時間を時分割して複数の端末装置(ユーザ)に割り当てる)処理が主流になる
- 本格的なOSを搭載したスマートフォン

第4世代(1970～1980)

- 統合プログラミング環境世代
- メディア(情報を伝達する媒体)の多種多様化(マルチメディア)に対応
- 複数のコンピュータシステムをLAN(ローカルエリアネットワーク)で結合して分散処理を可能とする
- メディアの処理方式やLANの通信方式には多数の異なる方式が存在
- スマートフォンでクラウド

第5世代(1980～1990)

- 隠ぺいされたシステムプログラム世代
- インターネットの出現により、世界中のコンピュータシステムが通信方式に関係なく常時接続して相互に通信
- インターネットを活用して、多様なマルチメディア情報をWebで発信
- GUIをOSの機能として提供

第6世代(1990～現代)

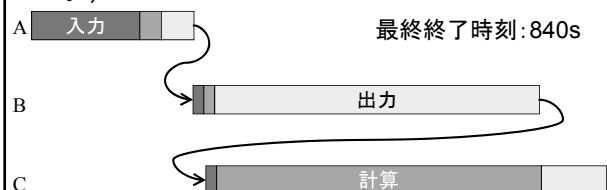
- 透明なシステムプログラム世代
- ユビキタス(いつでもどこでも)コンピューティングシステムの実現
- コンピュータシステムのハードウェア機能とソフトウェア機能がシームレス(継ぎ目なし)に一体化することで、人間どうしのコミュニケーションの道具になる

「OSなし」、「簡易OS」、「本格OS」²³ で何が違うか？

- いろいろな違いがあるが、ユーザプログラムの実行の違いを例に説明する
- 入力時間、計算時間、出力時間の違う3種類のプログラムを考える
 - プログラムA 入力100s, 計算 20s, 出力 30s
 - プログラムB 入力 10s, 計算 10s, 出力300s
 - プログラムC 入力 10s, 計算300s, 出力 60s

「OSなし」での実行

- 一度に一つのプログラムを実行できるだけ
- プログラムの実行の順番を決めることしかできない(下図でBとCは実際には重なっていない)

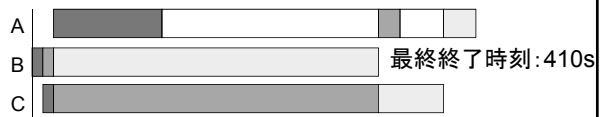
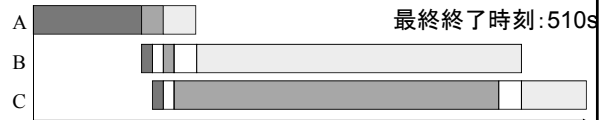


「簡易OS」による実行(1)

- **マルチプログラミング**(同時に複数のプログラムを実行できること)が可能になる
 - 同時に複数のプログラムやデータをメモリに置く(この機能を「スプーリング」という)
 - プログラムがあるハードウェア装置を使い始めると、使用が終わるまで別のプログラムから使えない(「**横取り(preemption)**ができない」という)
 - 入力、計算、出力は、順番は固定だが、処理の起動時刻は任意に調整できる

簡易OSによる実行(2)

- 各処理の起動時刻の決め方(「**スケジューリング**」という)によって終了時刻が大きく変わる



簡易OSの欠点

- スケジューリングによって、プログラムの終了時刻が大きく変わる
 - 実際には、あらかじめ各処理の時間を見積もることは困難
- いずれかのプログラムが非常に長時間装置を占有すると、他が待たされる
 - あるプログラムが無限ループを実行すると、他のプログラムは永遠に実行されない

本格OSによる実行(1)

- **マルチタスキング**:プロセッサの時間を分割して、一定時間間隔ごとに別々のプログラムを実行すること(**TSS**という)により、複数のプログラムの実行が可能になる
 - プログラムがあるハードウェア装置を使い始めても、使用の終了を待たずに、別のプログラムから使うこと(「**横取り(preemption)**」という)ができる
 - 無限ループを実行するプログラムがあっても、他のプログラムを実行できる

本格OSによる実行(2)

以下の条件を仮定する

- すべてのハードウェア装置は横取り可能
- TSSの分割単位(**タイムスライス**という)は10s(実際のOSではもっと短い)
- 最終終了時刻は、処理の起動順に影響されない

