オペレーティングシステム(OS)

柴山　潔

4.　OSの構成

- OSの基本構成
- OSの実際

OSのモジュール構成

［定義1.4］ **モジュール(module)**

- あるひとまとまりの機能を実現するためのデータとそれを操作するプログラム（狭義）（手続き(procedure)）群

➢ **OS**はモジュールの集まり ＝ **OS**モジュール

OSモジュールの要件 (1)

(A) **情報の隠ぺい**

- 他モジュールには，自モジュールのインタフェースや仕様だけを公開
- モジュール内部の構成方式や実現方式などの中身は隠ぺい

➢ 個々のモジュールの独立性が高

OSモジュールの要件 (2)

(B) 方針と機構の分離

- 方針（ポリシ(policy)）を実現する機能とその方針に基づいて実際の処理を行う機能（＝機構，メカニズム(mechanism)）を別々の独立したモジュールに

➢ モジュールの変更/拡張が容易に

（例）プロセススイッチ(process switch)

[方針] プロセススケジューラ(process scheduler)：実行順を決める戦略（＝スケジューリングアルゴリズム）にしたがってプロセスを選択

[機構] プロセスディスパッチャ(process dispatcher)：プロセスをプロセッサに割り付け（OSとハードウェア機構で機能分担）

OSモジュールの要件 (3)

(C) 階層化

- ユーザ（プログラム）⇔ハードウェアにおける種々のサービス機能を機能レベルごとに分離・分割&階層分け

↓

- モジュール間通信を同一階層内や隣接階層間だけに局所化・限定可

OSモジュールの要件（まとめ）

- **OS**を個々のモジュールに分割する際の基準（要件）としての
  - (A) 情報の隠ぺい
  - (B) 方針と機構の分離
  - (C) 階層化

  ↓ を満たせば，

OSモジュール個々の機能の改良&仕様の改訂が他の**OS**モジュール/ユーザプログラムに与える影響 → 少/無に！

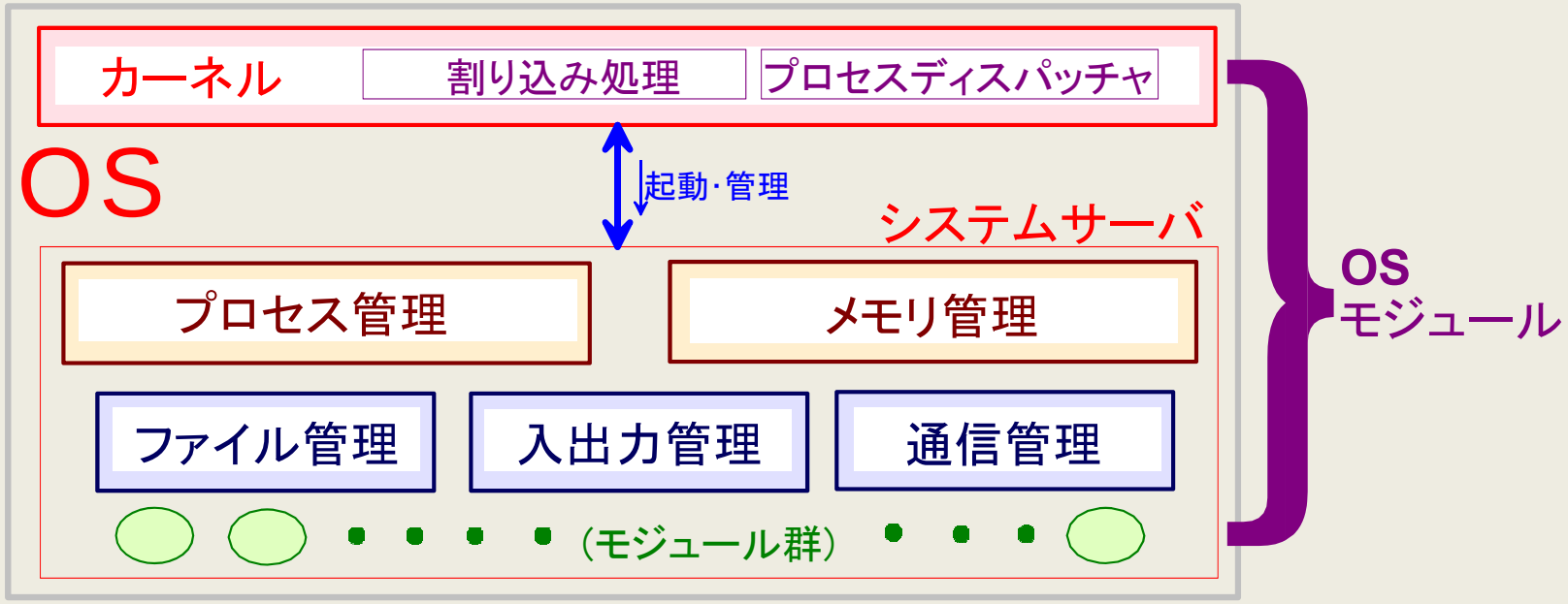
OSモジュールの機能

(A)カーネル（狭義） (kernel)：OSの基本モジュール＝プロセス管理の中核

- 割り込み処理 ＋ プロセスディスパッチャ（＝ システムサービス&プロセスの実行を管理）

(B)システムサーバ(system server)：機能/操作/対象ごとの個別のOS機能（＝ システムサービス）を提供するモジュールのそれぞれ

- カーネルが起動･管理



OSカーネルの構成方式

(A) 単層カーネル（モノリシックカーネル (monolithic kernel)）

- OS機能すべて（＝狭義のカーネル＋システムサーバ）を単一のカーネルで実現
  - 古典的/従来型/レガシ（legacy; 遺物）
    - （パソコンOSでの例）初期のWindows (95, 98, Me), OS/2, Linux (UNIX)

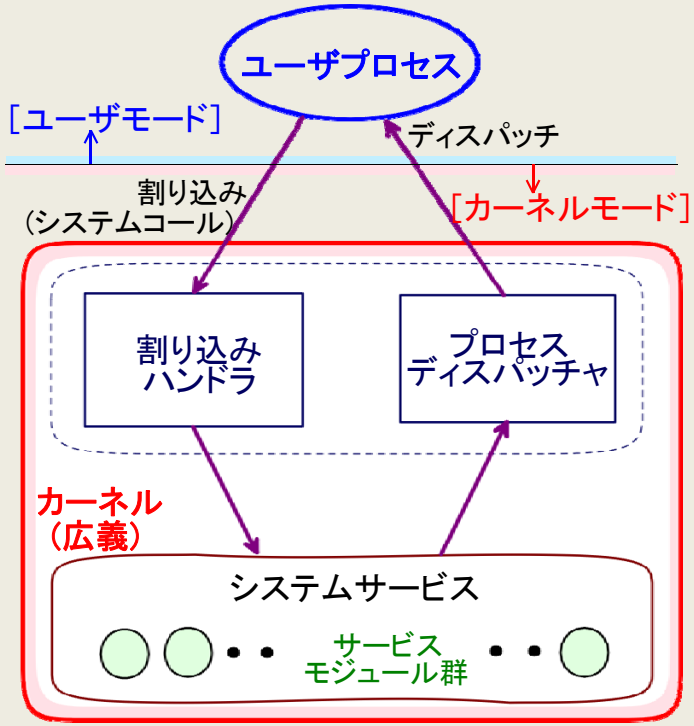
(B) マイクロカーネル (micro-kernel)

- 狭義のカーネルの一部としてのプロセスに対する基本操作だけをマイクロカーネルとして実現 → マイクロカーネルがシステムサーバの提供するシステムサービスを仲介
  - 最新, 現代的
    - （パソコンOSでの例）現代のWindows (2000/XP/Vista/8/10), Mac OS X

➢（OSの発展史では）単層カーネルからマイクロカーネルへ

単層カーネル (1)

- **カーネルモード**(＝ システムモード, OSプログラムの実行)と**ユーザモード**(ユーザプロセス/プログラムの実行)を明確に区別
  - ユーザプログラム(による不正アクセス)からOS(カーネル)を保護



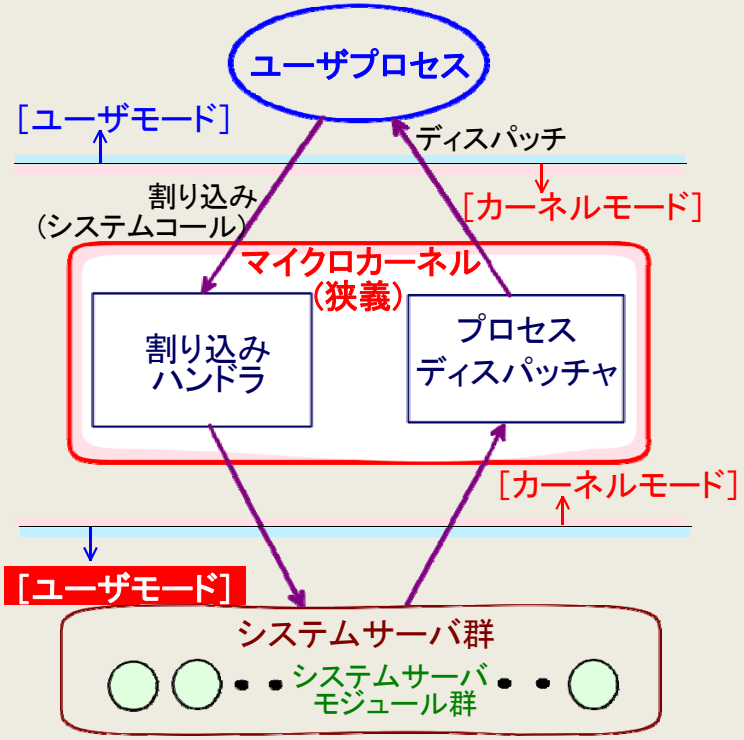
単層カーネル (2)

［特徴］

- ○ 実行効率が良
- ○ 新規機能はモジュールの追加 → 簡単
- △ 実装は比較的容易 ← 全体機能が小なら
- × OSのメモリ空間の占有面積（＝ 空間サイズ）大 → メインメモリにOSが常駐
- × 一部分の機能変更が全体に波及 → 保守が難 → 新規機能の追加をモジュールで実現 すれば回避可
- × システムサービスを含むOSの大部分をカーネルモードで実行 → カーネル自身の保護強度が弱，OSが（カーネル内部で）ハングアップ（hang-up; ＝ 無応答状態）し易い

マイクロカーネル(micro-kernel) (1)

- マイクロカーネル：割り込み処理，プロセスディスパッチ，プロセス間通信などのOSの基本操作だけ
- システムサーバ群：各種システムサービス用モジュール群，**ユーザモード**で実行



マイクロカーネル(micro-kernel) (2)

[特徴]

- ○ モジュール構成を活用，モジュール化が機能を明確に → 機能の変更/拡張が容易
- ○ OS機能の一部（＝ システムサービス）はユーザモード → カーネルの保護強度が大，安定性が高
- △ マルチプロセッサ対応が比較的容易
- × プロセス間通信が多いとそれがオーバヘッド → このオーバヘッドを小さくすれば，変更/拡張が容易な点で有利に

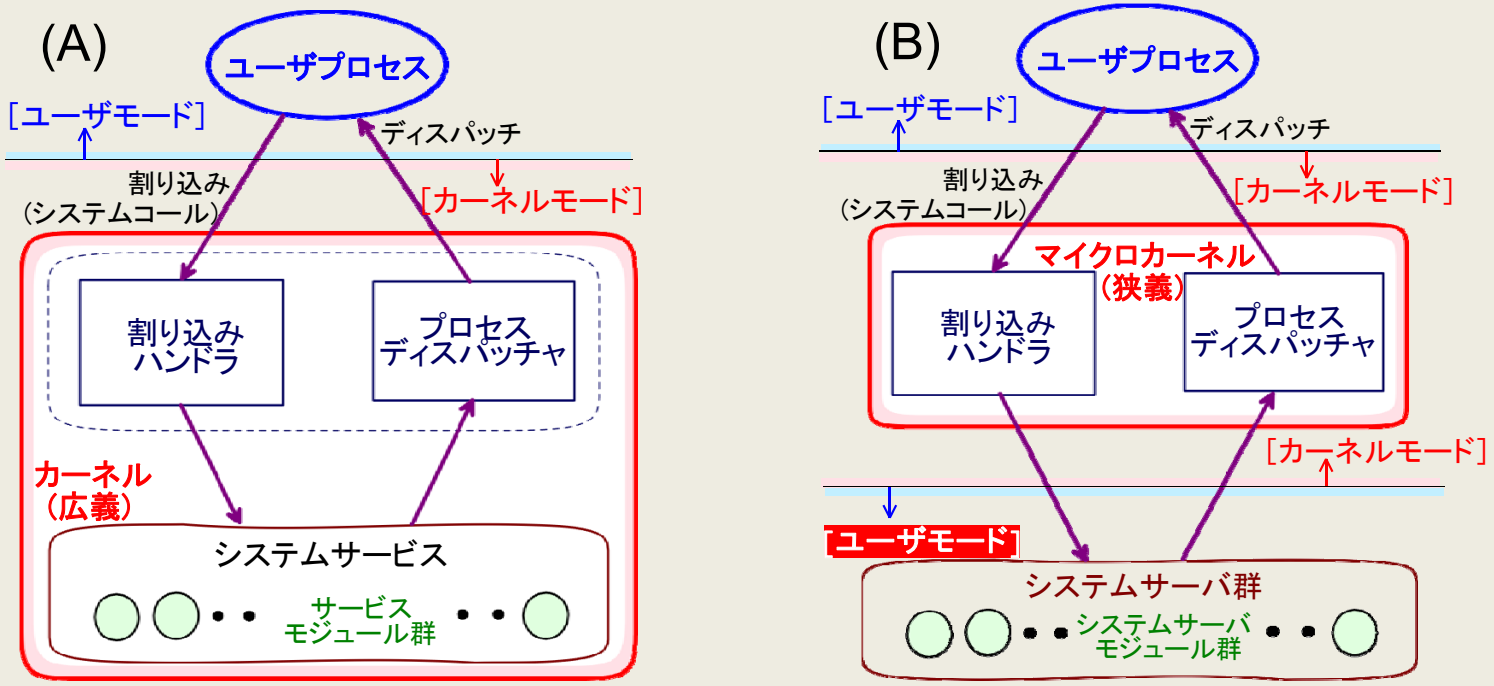
OSカーネルの構成方式（図説）（再掲）

(A) 単層カーネル（モノリシックカーネル(monolithic kernel)）

- OS機能すべてを単一のカーネルで実現

(B) マイクロカーネル(micro-kernel)

- プロセスに対する基本操作だけをマイクロカーネルとして実現



（参考1.7）

OSのオーバヘッド(overhead)

（本来の意味）

- ユーザプログラムではなくOSなどのシステムプログラムがプロセッサやメインメモリなどのハードウェア機構や装置を使用している時間あるいはその割合

➢ オーバヘッドが増大 → コンピュータの性能（＝ ユーザプログラムを処理する速度）が低下

↓

（実質的な意味）

- ユーザ（プログラム）が使用できる時間以外の（ユーザにとっての）むだな時間

システム(OS)サービス例

1. プロセス管理
   （例） プロセススイッチ；プロセススケジューリング
2. メモリ管理
   （例） メモリ領域の割り付け/解放；メモリ保護（アクセス制御，アクセス権管理）；ゴミ集め（ガーベジコレクション）
3. ファイル管理
   （例） 名前管理；ディレクトリ管理；ファイル割り付け；ファイル保護（アクセス制御，アクセス権管理）；ファイルのオープン/クローズ
4. 入出力管理
   （例） デバイスドライバ；入出力制御
5. 通信管理
   （例） 通信プロトコル処理；ネットワーク管理；通信制御

OSの実際 (1)

- **OS**自身もプログラム（＝ **OS**プログラム，システムプログラム）すなわちソフトウェア（＝ システムソフトウェア，基本ソフトウェア）
  - **OS**機能とハードウェア機構とがトレードオフ → 機能分担で解決
- **OS**も（コンパイラによってコンパイル&生成した）マシン命令列&データで構成
  - **OS**自身が**OS**プログラムとして管理
- **OS**は，コンピュータ（システム）の起動時に，(1) ファイル装置からメインメモリへ読み出し；(2) メインメモリに割り付け；(3) メインメモリで保持

  ＝ ブート(boot)

OSの実際 (2)

- **OS**の大半はメインメモリに常駐（＝ 常時メインメモリ上に存在）
- **OS**はハードウェア装置上でのユーザプログラムの実行/共用形態を管理･制御
  - **OS**以外のプログラム（ソフトウェア）は，ユーザプログラムとして，ハードウェ ア（プロセッサ/メモリ/入出力装置）を共用･共有して動作

OSの実際 (3)

- ユーザプログラムの大半は特定のOS(の管理・制御下)でしか動かない

  ← OSによる「ハードウェアの隠ぺい/仮想化機能」の方法や程度は相異

  ＝ OS(機能)が同一であれば，ハードウェア装置や機構(＝ハードウェア環境)が異なっていても，ユーザプログラム(オブジェクトプログラム形式)の大半は動く

  ＝ オブジェクト互換性

  ← OSによる「ハードウェアの隠ぺい/仮想化機能」の方法や程度は同じ

[定義1.5] オブジェクト互換性

- あるオブジェクトプログラムが相異なるハードウェア環境で動作すること
  - ◆「オブジェクト互換性がある」とも言う

OSの実際 (4)

- ユーザプログラムの **オブジェクト互換性** は同一/同種のOSの下で実現するのが普通
  - それらのユーザプログラムは「OSレベルで互換(である)」と言う

> 「ユーザプログラムからハードウェアを隠ぺいする」OS機能
> ＝「OSレベルで互換なユーザプログラムの実現」
> ＝「ユーザプログラムに対する **オブジェクト互換性** の実現」

■ 互換性がない(非互換)の例

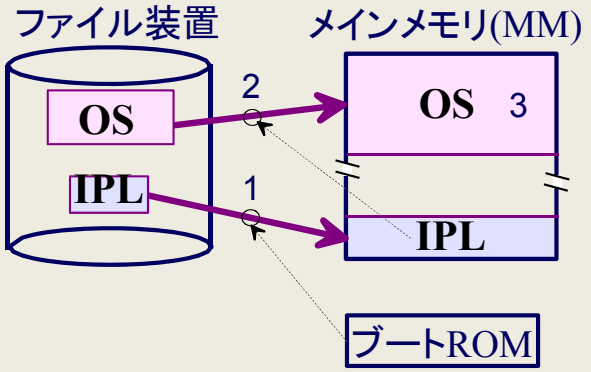
- WindowsとUNIXとMacOS ← 異種OS
- ハードウェア(装置，機構，環境)への依存度が高いユーザプログラム
  ← 同種OS but, 異なる版(バージョン(version), リビジョン(revision))

（上級）

OSの起動（ブート(boot)）

1. 電源オンによって，ブートROM（Read Only Memory; 不揮発性の読み出し専用メモリ）にあるブートストラップローダ(bootstrap loader)によって，ファイル装置からIPL（Initial Program Loader; 初期プログラムローダ，OSの一部）をメインメモリ(MM)に読み出し（＝ロード(load)）
2. ブートストラップローダからIPLへ制御が移り，IPLがファイル装置からOSプログラム全部をMMに読み出し（ロード）
3. OSを初期化

（例）ハードウェア装置の認識；ファイル装置上でのファイル位置の識別（＝ファイルのオープン(open)）；MM上でのOS領域（空間）の確保；



- リブート（reboot; 再起動）
- ハードリセット(hard-reset)（＝コールドブート）→ 1 へ
- ソフトリセット(soft-reset)（＝ウォームブート）→ 2 へ

（例）パソコンのBIOS (Basic I/O System)：基本入出力操作 ＋ ブートストラップローダ

（上級）

OSの停止（終了，シャットダウン(shutdown)）

1. OSが実行中のユーザプロセス（ユーザプログラム）すべての停止処理
   （例）使用ファイルのクローズ(close)
2. 電源オフ

（上級）

OS機能の隠ぺい（概要）

➢ **OSの原理** ＝ ユーザ（プログラム）からハードウェア機構を隠ぺい

↓（拡張）

- **OS機能そのものも**（ハードウェア機構と併せて）隠ぺい

（上級）

仮想マシン（1）

［定義1.2］ **仮想化** （再掲）

- ソフトウェアによって，実際の/物理的なハードウェア機構/機能を模倣（＝シミュレーション(simulation)）し，仮想的/論理的機構/機能に見せかけること

- **実コンピュータ**：実際の，あるいは，ハードウェアとソフトウェアで構成する物理的に実在するコンピュータ
  ＝ 実マシン/ホストコンピュータ(host computer)/ホストマシン

- **仮想マシン** (virtual machine)：仮想の，あるいは，ソフトウェアだけで構成する論理的なコンピュータ
  ＝ 仮想コンピュータ/ゲストコンピュータ(guest computer)/ゲストマシン

➢ エミュレーション(emulation)：実コンピュータによる“仮想マシン”のシミュレーション

（上級）

仮想マシン (2)

［主目的］

- 通常ならばハードウェアで構築する機能の一部をソフトウェアで構築することによって，ハードウェアでは実現できない問題適応能力や柔軟性を得る

（例）

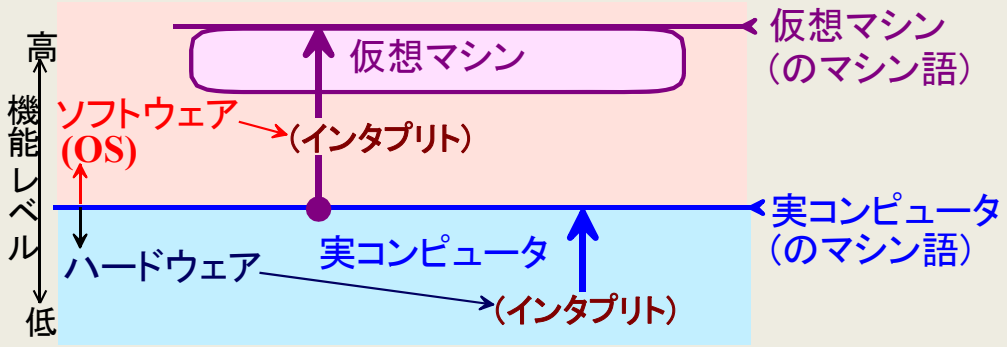
- あるハードウェア上で稼働させるOSの試作や開発時に，仮想マシンを活用すれば，当該ハードウェアをあらかじめ構築する手間や必要がなくなる
- ➢ 特に，コンピュータのハードウェアの開発と並行して/同時に，そのコンピュータ（＝ターゲットコンピュータ(target computer)）上で稼働させるOSを開発する場合
  → ターゲットコンピュータはまだない
  → 仮想マシンの構築と使用が不可欠

（上級）

仮想マシン (3)

- 仮想マシンによる実行
  ＝ 実コンピュータのソフトウェアによる「仮想マシンのマシン語」のインタプリト

（参考）
- インタプリト(interpret)：プログラミング言語の処理（＝解釈）と，解釈の結果であるマシン語による実行とを，連続して同時に行う言語処理
  - インタプリタ(interpreter) ＝ インタプリトする言語処理プログラム
- 一般的なコンピュータ（実コンピュータ）による実行
  ＝ そのコンピュータのハードウェアによる「実コンピュータのマシン語」のインタプリト



（上級）

仮想マシン (4)

- 仮想マシン（ソフトウェア）のマシン語の機能レベルは実コンピュータ（ハードウェア）のマシン語の機能レベルよりも高く設定
  - プログラミング言語とマシン語（＝仮想マシンのマシン語）との機能レベル差が狭まり，機能レベル差を埋めるシステムプログラム（＝コンパイラやOS）機能が簡素&簡略に

↑（トレードオフ）↓

- 「ハードウェア機構をソフトウェア機能で構築」という仮想マシン/仮想化の原理がハードウェアの特長である高速処理能力や高速性を喪失
  - 仮想マシンの管理/切り替え/稼働そのものがオーバヘッドに

（上級）

仮想マシン (5)

- 「仮想マシン/仮想化におけるオーバヘッドの顕在化」を回避するアーキテクチャ的（＝ハードウェアとソフトウェアの機能分担方式を適切に定める/具体化する）手法
  (a) 仮想化の実現に必要な一部機能をハードウェア化（＝ハードウェア機構として構築）
  (b) 仮想マシンのすべて/大半をファームウェア（firmware; ＝マイクロプログラム, 参考1.8）でエミュレーション（＝仮想マシンのインタプリタをファームウェアで作成･実行）

↓

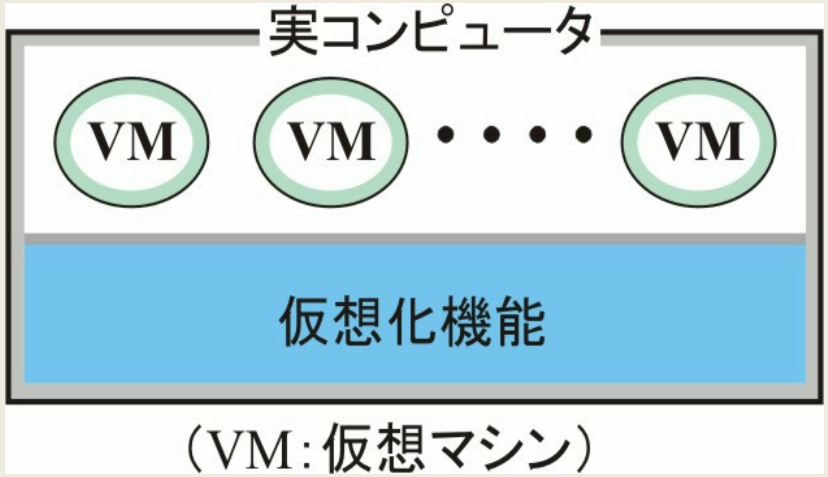
➢ 「問題適応能力や柔軟性の獲得」という仮想マシン/仮想化の特長を活用可

（上級）

実コンピュータと仮想マシン (1)

- 1台の実コンピュータ上で複数台の仮想マシンをエミュレーションする（＝同時に稼働させる）構成
  - ハードウェアの高速性とソフトウェアの問題適応性のそれぞれの特長を活用した機能分担によって，コンピュータシステムにおける究極の“仮想化”を実現
  - 1台の実コンピュータ上で複数台の仮想マシンを稼働 ＝ 複数台の仮想マシンが1台の実コンピュータを共有/共用

（図1.13）



（上級）

実コンピュータと仮想マシン (2)

↑ ↓

- 1台の仮想マシン上で複数台の実コンピュータを稼働
- 複数台の実コンピュータが1台の仮想マシンを共有

（図1.13において，実コンピュータと仮想マシンとを入れ替え）

（例）

- グリッドコンピューティング(grid computing)，クラスタコンピューティング (cluster computing)
  - ネットワークで相互接続した多数の（実）コンピュータシステム（＝個々はPC/WSが主，全体でグリッドとかクラスタ）を1台の仮想コンピュータシステム（＝並列コンピュータシステム）と見なし，その上で大規模処理や大規模計算を実行

（上級）

（参考1.8）

ファームウェア(firmware)

- ハードウェア機構の制御専用のソフトウェア（プログラム）
  ＝ マイクロプログラム(microprogram)

➢ ソフトウェア（マシン語プログラム）よりもハードウェアにより近い機能によってハードウェア機構を制御

■ マイクロプログラム制御：ファームウェアでハードウェアを制御する方式

- 「（マイクロ）プログラム」という点で，一種の“ソフトウェア”
  → ハードウェアよりも修正や更新が格段に容易
  - ➢ 配線論理制御（＝ハードウェア＝電子回路や論理回路だけで制御機構を構成）と比較すると，高速性では劣，問題適応性では優
- 商用のマイクロプログラム制御コンピュータでは，製造時に，ファームウェアを専用ROM（Read-Only Memory; 読み出し専用メモリ）に格納
  → ファームウェアはハードウェアの一種
  - ➢ 普通のソフトウェア（マシン語プログラム）よりも高速処理（実行）可

（上級）

OSと仮想マシン (1)

■ 実コンピュータ（＝ホストコンピュータ）上で複数の仮想マシン（＝ゲストマシン）が稼働するコンピュータシステムに搭載し機能するOS ［(A)(B) 2種類］

(A) **ホストOS** (host OS)

● 実コンピュータ（＝ホストコンピュータ）上で稼働し，その実コンピュータを管理･制御するOS

＝ 実コンピュータ（の）OS

- 1台の実コンピュータ上に1個だけ搭載･機能
- 実際の仮想化機能の実現では，「ホストOSの存在や役割が明確ではない」方式（＝ホストOSなし･単一共有仮想マシン）もある

（上級）

OSと仮想マシン (2)

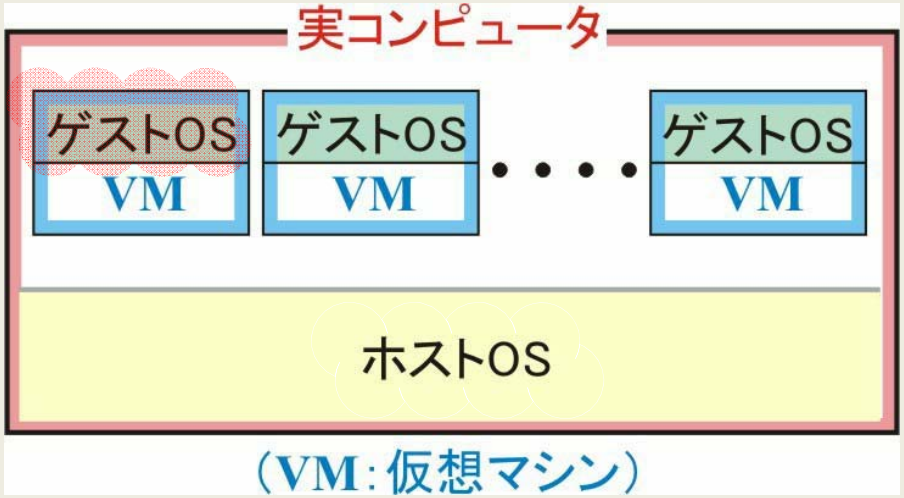
■ 実コンピュータ（＝ホストコンピュータ）上で複数の仮想マシン（＝ゲストマシン）が稼働するコンピュータシステムに搭載し機能するOS ［(A)(B) 2種類］

(B) ゲストOS (guest OS)

● 実コンピュータ（＝ホストコンピュータ）上で稼働する各仮想マシン（＝ゲストマシン）の個々を管理･制御するOS

＝ 仮想マシン（の）OS

➢ 各仮想マシン上に1個ずつ搭載し機能させるので，1台の実コンピュータ上には複数個



（上級）

マルチOSコンピュータ(multi-OS computer)

- 仮想マシンのそれぞれに「異種のゲストOS」を搭載し同時稼働させるコンピュータシステム

■ マルチOSコンピュータ上の仮想マシン
　＝ ソフトウェアシミュレーションによって実現する論理的なコンピュータ
  - 実コンピュータ上で（＝実コンピュータのソフトウェア機能によって）構成&エミュレーション

■ 仮想化ソフトウェア：各ゲストOSが対象とする各仮想マシンを稼働させて（＝エミュレーションして）＆ それらの動作全般を管理･制御する機能（＝仮想化機能）を実現するソフトウェア

（上級）

OS機能の隠ぺいとは？

➢ マルチOSコンピュータでは，

● 実ハードウェアだけではなく，ホストOSも，ゲストOSから隠ぺい

↑

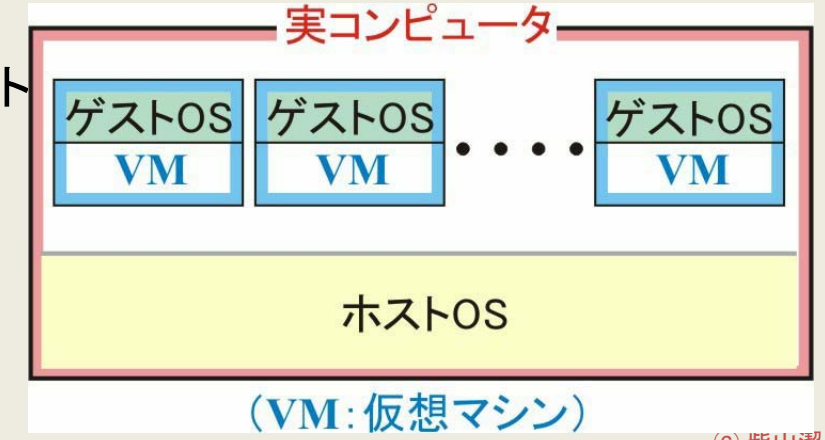
「ホストOS（ソフトウェア）による実コンピュータ（ハードウェア）の隠ぺい」

＋

「ゲストOSによる仮想マシン（実は，ソフトウェア）の隠ぺい」

➢ ゲストOSには仮想マシンを見せかけることによって，ゲストOSから ホストOS（と実ハードウェア）を隠ぺい

＝ 仮想マシンによる（ホスト）OS機能の隠ぺい

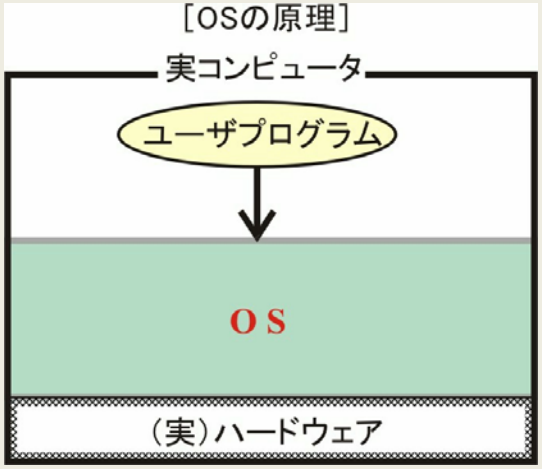


（上級）

OSの原理とOS機能の隠ぺい

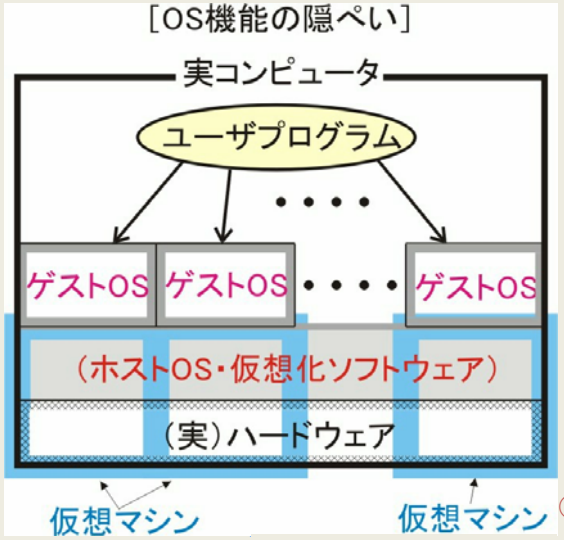
■ OSの原理

- コンピュータシステムのハードウェア機構上で稼働するOSによって種々のユーザプログラムからハードウェアを隠ぺい



■ 仮想マシンによるOS機能の隠ぺい

- 実ハードウェア機構だけではなく，その実ハードウェアを管理・制御するホストOSも，各仮想マシンを管理・制御する複数個&異種のゲストOSによって隠ぺい



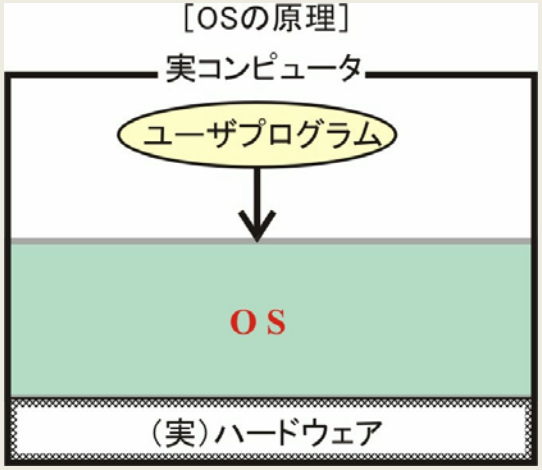
（上級）

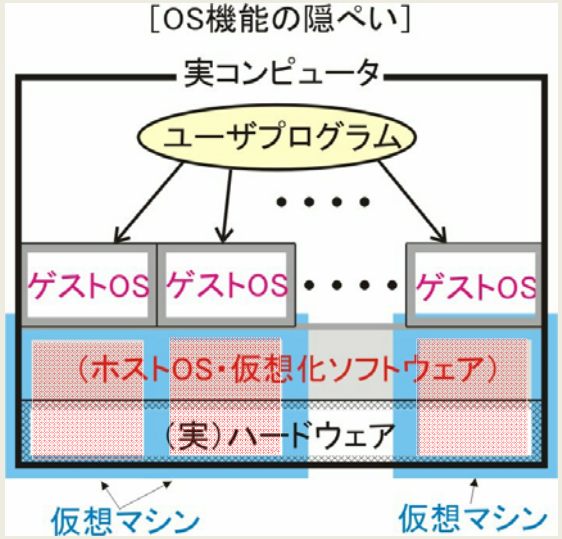
OS機能の隠ぺい —隠ぺいする機能レベルの観点—

- 「OSによる（実）ハードウェアの（ユーザプログラムからの）隠ぺい（左図）」から「ゲストOSによる仮想マシンの（ユーザプログラムからの）隠ぺい（右図）」へと，隠ぺいする機能レベルがより高へ

■ 仮想マシン：「ホストOSと仮想化ソフトウェアによる（実）ハードウェアの隠ぺい」で構成

  - OS機能の隠ぺいは，（実）ハードウェアからホストOS（ソフトウェア）へと，仮想化する（＝仮想化の対象とする）機能レベルをより高へ





（上級）

OS機能の隠ぺいの目的

■ **OS機能の隠ぺい**

＝ 1台の実コンピュータ上で，複数個&異種の仮想コンピュータシステム（＝ハードウェアとOSを一体化して仮想化するコンピュータシステム全体）が稼働

[主目的]

- 仮想化する機能レベルを（実）ハードウェアの機能レベルよりもより高いホストOSの機能レベルに設定
  - 種々のゲストOS（＝ゲストOSの多様性）に対処 → ゲストOSそのものの可搬性（＝機能を多種多様な環境に簡単に搭載/移行/移植可能，参考1.9）を高に

（上級）

OS機能の隠ぺいの具体的な効果 (1)

➢ 1台の実コンピュータシステム上で複数個&異種の仮想コンピュータシステムが稼働

↓

- 1台の実コンピュータシステム上で，多種多数のユーザプログラムの動作環境やプログラミング環境（＝プログラムの開発環境）を構築可能
  ← 異種OSの同時稼働
- 実コンピュータシステムの管理コストは低減可で，利用効率は向上
  ← 種々OSが対象とする実コンピュータシステム（ホストOSを含む）を1種類に絞れる

（上級）

OS機能の隠ぺいの具体的な効果 (2)

- 実コンピュータシステム全体の総合的な信頼度（＝耐故障性）は高に
  - ← 複数台の仮想コンピュータシステムが多重で稼働する冗長性を活用すると，故障している（仮想）コンピュータシステムのソフトウェア的な切り替えや切り離しが簡単に
- ソフトウェアの寿命（＝存続期間，ライフタイム(lifetime)）が延び，ハードウェアとソフトウェアのそれぞれの寿命のずれや不整合を修整可能に
  - ← 現在は稼働していない/稼働できない古い版の（＝レガシ）**OS**やその**OS**の管理下で稼働するユーザプログラムを現代の実コンピュータシステム上で稼働あるいは動作させることが可能

（上級）

OS機能の隠ぺいの実現 —仮想マシンの構成方式—

- ホストOS機能の隠ぺいの実現
  (1) 仮想化ソフトウェア機能で仮想マシンを構成
  (2) ゲストOSには仮想マシンを見せかける

↓

仮想マシン ＝ 仮想化ソフトウェア

↓

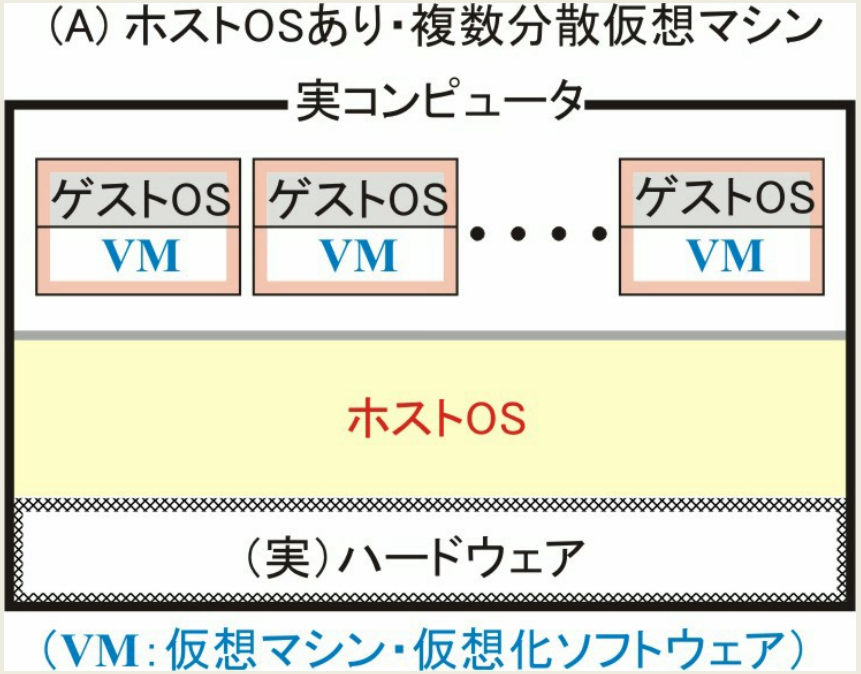
OS機能の隠ぺいを実現 ＝ 仮想マシンを構成

（上級）

仮想マシンの構成方式（図説）(1)
—ホストOSと仮想マシンとの関係による分類—

(A) **ホストOSあり・複数分散仮想マシン**

- 異種複数の仮想マシンをホストOSと連携するが独立した機能として構成
  - ホストOSは一般的なOS → 仮想化ソフトウェアの追加インストールだけ で各仮想マシンを実現可
  - 異種複数のそれぞれ独立した仮想マシンをホストOS上で実現し稼働さ せねばならない → コストは，(B)に 比べると，大

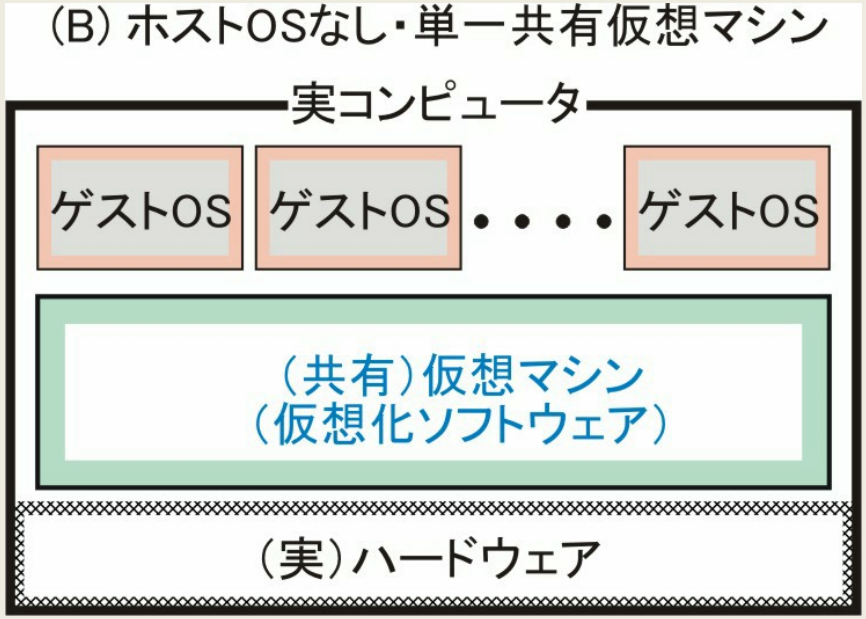


（上級）

仮想マシンの構成方式（図説）(2)
—ホストOSと仮想マシンとの関係による分類—

(B) ホストOSなし・単一共有仮想マシン

- 単一の仮想マシンを実現する仮想化ソフトウェアをホストOSの代わりにあらかじめ組み込んで構成
  - ホストOSそのものが仮想化ソフトウェア機能を備える仮想マシン → 既設の一般的なOSを置換する必要
  - 共有する仮想マシンを1台だけ直接（実）ハードウェア上で実現し稼働させればよい → コストは，(A) に比べると，小



（上級）

OS機能のファームウェア化

- OS機能の一部をソフトウェアではなくファームウェア（＝マイクロプログラム，参考1.8）によって実現
  - OS機能をアーキテクチャ的に（＝ハードウェアとソフトウェアの機能分担方式を適切に設定）改善/強化

■ ファームウェア化の対象とする一般的機能
- 高速処理が必要な/オーバヘッドが顕在化しやすいソフトウェア機能
- 可変性や問題適応能力が必要なハードウェア機能

（上級）

OS機能のファームウェア化の具体的な目的(例)

(1) 種々の仮想化/仮想マシンおよび言語処理プログラムにおけるオーバヘッドの発生を防止

(2) ハードウェア機能のテストや保守における柔軟性を確保

(3) プロセス管理におけるスケジューリングアルゴリズム，仮想メモリにおけるページ置換アルゴリズム，各種の通信プロトコルなどの，最適解を確定するのが難しい機能を可変に

(4) 命令実行例外を代表とする多種多様な割り込み要因に対して，柔軟&高速に対処