

2

オペレーティングシステム

資料 第**2**分冊(H30)

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

2

1.2 OSの機能 — 概要 —

1.2.1 OSの実際的な役割

- **ユーザプログラム**: 一般ユーザがコンピュータシステム上で使用する「応用」プログラム
 - 「応用」がついているのは、コンピュータシステム側が提供するプログラム(「システムプログラム」と区別するため
- コンピュータシステム上で動作するプログラムは、**システムプログラム**と**ユーザプログラム**のいずれかに分類される
 - **広義のオペレーティングシステム**=システムプログラム全体
 - **狭義のオペレーティングシステム**=システムプログラムから言語処理プログラムを除いたもの

3

OSの機能

```

graph TD
    User[ユーザ] -- ユーザインタフェース --> UserProgram[ユーザプログラム]
    Program[プログラマ] --> UserProgram
    Program --> OS["(狭義の)オペレーティングシステム"]
    UserProgram --- OS
    OS --- OS2["ミドルウェア(言語処理プログラム、データベース管理システムなど)"]
    OS2 --- OS3["システムプログラム"]
    OS3 --- Software[ソフトウェア]
    Software --- Hardware[ハードウェア]
  
```

4

OSの原理

OSの本質的で具体的な役割は、次の3種類

- (A) ユーザプログラムから、実際の、あるいは物理的なハードウェア機構を隠ぺい
 - **ハードウェア機構の隠ぺい**
- (B) ユーザプログラムが、共用あるいは共有するハードウェア資源を、実行時において管理する
 - **ハードウェア装置の管理**
- (C) 壊れやすいソフトウェア(特に、ユーザプログラム)を、その実行時に動的に保護する
 - **プロセス管理**

5

(A) ハードウェア機構の隠ぺい

- 「何」を隠すのか?
 - ハードウェア全体を隠すのではない
 - 個々のハードウェアが持つ**物理的な特性や相違点**を隠す
 - 例: プロセッサ、メモリ、ファイル装置、入出力装置、通信装置等で、特定の装置(機種・製品)間の**物理的な相違点**
 - 同じ種類のハードウェア装置を**共通化**する
- 「何」から隠すのか?
 - **ユーザプログラム**(ユーザを含む)に対して隠す
- 「なぜ」隠すのか?
 - ユーザプログラムが特定のハードウェア装置に依存して開発されるのを避けるため(**移植性の確保**)

6

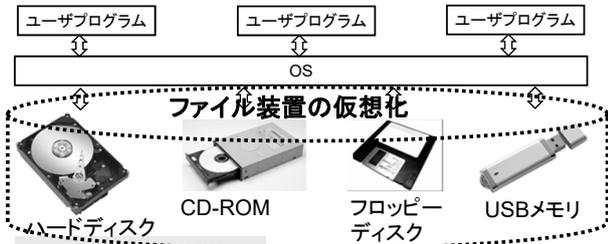
ハードウェア機構の仮想化

- ハードウェア機構の隠ぺいは、OSがハードウェア機構を**仮想化**することによって実現する(図1.5参照)
 - 仮想化: 物理的なハードウェア機構を、ソフトウェアによるシミュレーションで論理的な機構に見せかけること(後で例を示す)

ハードウェア機構の 隠ぺいと仮想化の例

7

- 同じ機能のハードウェア装置でも、物理的には多様な機器があり、アクセス方法がちまちま
→ 装置を隠ぺいして統一したアクセス方法を提供
➢ ファイル装置の例



(B) ハードウェア装置の管理

8

- ユーザプログラムの実行時に使用するハードウェア装置をOSが管理・制御する
 - ハードウェア装置の例: プロセッサ、メモリ、ファイル装置、入出力装置、通信装置等
 - 実行前にどの時刻にどの装置が使われるか決定するのは困難
 - 実行時に決定する必要がある: OSの役割
 - OSが、プログラムの実行時に装置の共用や共有が生み出す競合を解決し、各装置を効率的に並行して利用させる

(C) ソフトウェアの実行時の保護

9

- ソフトウェアがファイルをアクセスする権利を管理する
 - ファイルの読み書き等のアクセスを制限しないといけない場合がある
- OSがソフトウェアに対してアクセスしたり使用する権利(アクセス権)の管理や競合の調停を行う
 - ソフトウェアへの不正アクセスの防止
- OSとユーザプログラムとの実行時の区別(実行モードの設定)
 - OSは特権モードでファイルにアクセスできる

1.2.2 OSの機能

10

- ハードウェアの仮想化による管理
 - ユーザプログラム間で共用するハードウェア装置を仮想化によって統一的に管理・制御する
- マシン命令セットによる支援
 - プロセッサのマシン命令セットから適切なマシン命令を組み合わせて実行することで機能を実現
 - プロセッサおよびメモリについての時間管理
 - メモリについての空間管理機能
 - 入出力処理機能
 - 割り込みや例外処理機能
 - 実行制御機能

プロセス

11

- 「実行されるプログラム」のことをプロセスという
プロセス = プログラム + 実行の状態
 - プログラムの実行コード(マシン命令列)と、実行時のデータの集まりから構成される
- なぜ、「プログラム」と「プロセス」を分けて考えないといけないか?
 - プログラムは、実行するための命令列や(あらかじめ決められた)データ → 実行によって変化しない
 - プロセスは、実行されるたびに作成される
 - 一つのプログラムに対して、複数個のプロセスを作ることができる

プロセス管理

12

- プロセスの割り付け
 - プロセスを、メモリの特定の領域に置く(プロセスとメモリを対応付ける)
 - プロセスを「作る」ときに必要な処理
- プロセッサ割り付け
 - プロセスとプロセッサの時間を対応付ける
 - プロセスを「実行する」ときに必要な処理
- 詳細は2章で説明する

13

OSの管理機能

- プロセス管理機能
- メモリ管理機能
- ファイル管理機能
- 入出力管理機能
- 通信管理機能

14

プロセス管理機能

- **プロセッサの時間を管理して、プロセスへ割り付けて実行する**
- プロセッサの時間を一定間隔(**タイムスライス**または**クォンタム**)に分割して管理する(TSS: Time Sharing System) (図1.6参照)
- **メインメモリを、時間的かつ空間的に管理する機能も必要**
- 詳細は2章で説明する

15

メモリ管理機能 ファイル管理機能

メインメモリの仮想化

- **メインメモリの容量や動作速度など物理的な相違点を隠ぺい**
- **仮想メモリ技術として、メインメモリとファイル装置の連携を支援**(3章で詳細に説明)

ファイル装置の仮想化

- **ファイル装置の物理的な相違点を隠ぺい**
- **論理的なファイルを、物理的なファイル装置にマッピングする**(3章で詳細に述べる)

16

その他の管理機能

- **入出力管理機能**
 - プロセッサと入出力装置の間の動作速度の違いを吸収する
 - 4章で詳細を説明
- **通信管理機能**
 - ネットワーク通信
 - 4章で詳細を説明

17

:(まとめ) OSによるハードウェア装置の隠ぺい

- OS以外のソフトウェア(特に、ユーザプログラム)から、**ハードウェア装置の物理的な相違点を隠ぺいする**→OSの基本的な機能の一つ
- 「隠ぺい」は、「統一」、「共通化」、「仮想化」、「標準化」と置き換えることができる
- 隠ぺいの対象
 - プロセッサ、メインメモリ、ファイル装置、入出力装置、通信装置

18

:(まとめ) ユーザプログラムに対するOS機能

以下のハードウェア装置をユーザプログラムに対して利用できるようにする(隠ぺいして仮想化)

- **プロセッサ**: ユーザプログラムを実行
- **メインメモリ**: ユーザプログラムがプロセッサで実行されている間、それを保持する領域
- **ファイル装置**: ユーザプログラムを長期的に格納する
- **入出力装置・通信装置**: (OSも含めて) ユーザプログラムが利用する

ユーザプログラムによる ハードウェア装置の利用

19

- ハードウェア装置は、ユーザプログラムからはOS経由でしか利用できない
- ユーザプログラムがハードウェア装置を利用するときはOS経由で「通信」する(図1.8参照)
- 通信には、**割り込み**が使用される(1.4節参照)(詳細は第3回で説明)
 - ハードウェア装置→OS→ユーザプログラム
 - 外部割り込み**(ハードウェア割り込みともいう)を使用
 - ユーザプログラム→OS→ハードウェア装置
 - システムコール(**内部割り込み**、ソフトウェア割り込みともいう)を使用

1.3 OSの構成

20

1.3.1 OSの基本構成

- OSのモジュール構成
 - ある特定の機能を実現するためのプログラムおよびデータをひとまとまりにしたものを、「**モジュール**」とよぶ
- OSモジュールの要件
 - 情報の隠ぺい
 - 方針と機構の分離

方針と機構の分離

21

- 次の2つの機能単位を別のモジュールとして分離する
 - 方針(policy)**: 機能を実現するときに生じる選択肢の決定
 - 機構(mechanism)**: 決定した方針に基づいて、実際の処理をどのように実現するか
- 方針は「(機能を決めるための)アルゴリズム」、機構は「(機能を実現する)実装」

なぜ方針と機構を分離するのか？

22

- 「**方針**」は、**コンピュータシステムの利用目的や運用方法に依存して決まる**
 - 実行効率(スループット)重視**: 実行するプログラム全体の終了時刻を最小化(最短)にしたい
 - 応答性能重視**: 個々のプログラムの実行指示から結果が返ってくるまでの時間を最小化したい
 - 「**機構**」は、**ハードウェア構成に依存**
 - 個々のプロセッサ、メモリ、ディスク、通信装置、ネットワーク装置ごとに異なる
- それぞれ、独立に変更できることが望ましい

方針と機構の分離の例(1)

23

プロセスのスケジューリング

- スケジューラ**
 - スケジューリングアルゴリズムに基づいて、プロセスを選択し、プロセッサ(CPU)の割り付けを指示
 - どのプロセスから実行するかを、**方針**として決める
- ディスパッチャ**
 - スケジューラからの指示により、選択されたプロセスに実際にプロセッサを割り付ける処理を実行する
 - プロセッサを割り付けるのに必要な処理は、プロセッサの種類や数などのハードウェアに依存

方針と機構の分離の例(2)

24

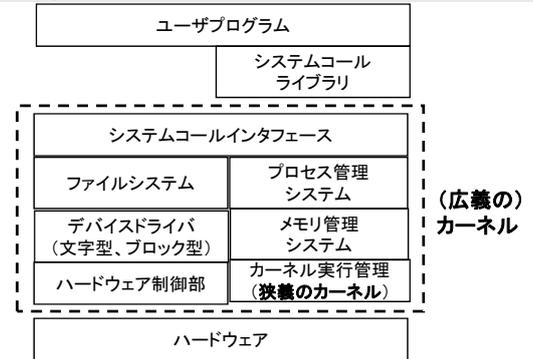
メモリの割り付け

- 割り付け**制御部**
 - 記憶領域の割り付けの**方針**に対応した割り付け技法(詳細は、教科書の3章参照)により、次に割り付けるべき領域を決定する
- 割り付け**実行部**
 - 割付制御部により選択された領域を、プロセスの使用領域として割り付ける処理を実行する

OSモジュールの機能

- 狭義のカーネルとシステムサーバに分かれる(図1.9)
- 狭義のカーネル(kernel)
 - 名前の通りOSの核となる機能を実現するプログラム
 - 割り込み処理とプロセスディスパッチャを基本機能として実現
- システムサーバ
 - 個別のOS機能を提供する
 - プロセス管理の一部(カーネル部分以外)、メモリ管理、ファイル管理、入出力管理、通信管理をシステムサーバとして実現
- 狭義のカーネルとシステムサーバを合わせて、(広義の)カーネルと呼ぶこともある

カーネルの構成例 (伝統的なUnixカーネル)

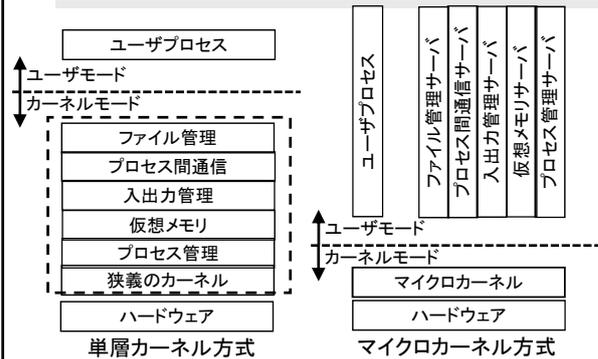


OSカーネルの構成方式(1)

代表的な構成法は次の2種類

- 単層カーネル方式
 - モノリシックカーネルとも呼ぶ
 - カーネルの機能を階層的につないで、単一のプログラムとして構成
- マイクロカーネル方式
 - カーネルを機能ごとに分割して、それぞれをシステムサーバとして実現

カーネルの構成法(2)



単層カーネル方式

- オペレーティングシステムの全機能を一つのプログラム(広義のカーネル)に取りこむ
- ユーザプロセスからのシステムコールでカーネルモードに遷移
- システムコールのパラメータ
 - ユーザプロセスの要求を記述
- システムコール処理後
 - ユーザプロセスの次の命令に戻る(ユーザモードに遷移)

カーネルモードとユーザモード

- カーネルモード
 - 特権モード、スーパーバイザモードともいう
 - すべての操作、すなわち、演算命令などだけではなく、任意の命令、例えば入出力操作や、任意のメモリ空間へのアクセスが可能
- ユーザモード
 - 保護モードともいう
 - 入出力などは禁じられ、メモリ空間へのアクセスも許された空間のみに制限される

単層カーネル方式の特徴

長所

- 実行効率がよい
- モジュールを追加するだけで新機能を追加できる
- 全体機能が小さければ、実装が比較的容易

短所

- **メモリを占有する領域が大きい**(カーネルはメモリに常駐するのが普通)
- 一部の機能変更が全体に影響(機能ごとのモジュール化が必要)
- OSの機能全体をカーネルモードで実行するので、**保護強度が弱い**(障害が起こったときの影響が大きい)

マイクロカーネル方式

- 必要最低限の機能のみの**マイクロカーネル**と、それ以外の付加的な機能を提供する**システムサーバプロセス**で構成
- マイクロカーネルの機能
 - メモリをアクセスするアドレス空間の管理機能
 - プロセッサ割り付け実行機能(プロセスのスケジューラやタイマー管理を含めることが多い)
 - プロセス間通信機能
- システムサーバプロセス
 - ユーザプロセスと同じレベル(ユーザモード)で動作する
 - 実現される機能の例(メモリマネージャ、ファイルサーバ、ネームサーバなど)

マイクロカーネル方式の特徴

- カーネルを機能ごとにプロセスに分割し、プロセス間通信で相互に連絡する
 - マイクロカーネル⇄システムサーバ
 - システムサーバ⇄システムサーバ
- 長所
 - 単層カーネル方式に比べて**メモリを占有する領域が小さい**
 - モジュール化が明確になるため、機能の拡張が容易になる
 - OSの機能の多くを**ユーザモード**で実行できる(**障害の影響を抑えられる**)
 - マルチプロセッサへの対応が比較的容易
- 欠点
 - プロセス間通信が多発するため、OSの処理で**通信処理のオーバーヘッドが生じる**

カーネルの構成方式のOS実装例

- 単層カーネル方式のOS実装例
 - 初期のUnix
 - 多くの商用Unix
 - FreeBSD, Linux
- マイクロカーネル方式のOS実装例
 - Windows NT系列のOS
 - macOS
- 詳細は第7回で説明