

オペレーティングシステム



資料 第2分冊(2021)

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

1.2 OSの機能 ー概要ー

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

OSの機能の説明として、まず、OSの実際的な役割から説明します。

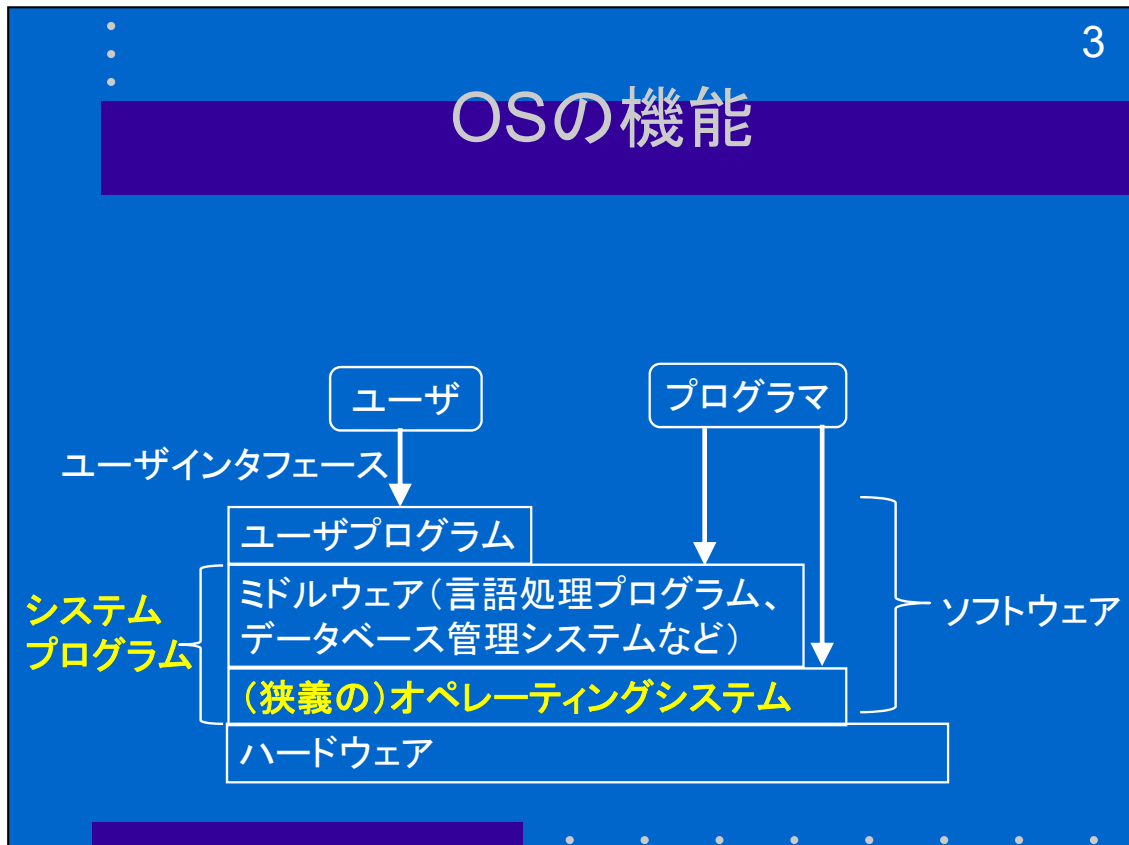
ユーザプログラムは、第1回の講義でも出てきましたが。一般ユーザがコンピュータシステム上で使用する「応用」プログラムを指します。

ここで、「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、すなわち、「システムプログラム」と区別するためです。

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ます。

さらに、広義のオペレーティングシステムとは、システムプログラム全体のことを指し、

狭義のオペレーティングシステムとは、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を指します。



この図は、第1回の講義でも出てきましたが、システムプログラムとユーザプログラムの関係を示したものです。

第1回の講義でオペレーティングシステムとして説明したものは、「狭義」のものを指します。

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する

→ **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

→ **ソフトウェアの実行時の保護**

次にOSの原理について説明します。

OSの本質的で具体的な役割は、次の3種類です。

一つ目は、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することです。

これは、ハードウェア機構の隠ぺいと呼ばれます。

2つ目は、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ことです。

これは、ハードウェア装置の管理と呼ばれます。

3つ目は、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ことです。

これは、ソフトウェアの実行時の保護と呼ばれます。

⋮

(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)

1つ目のハードウェア機構の隠ぺいについて説明します。

まず、「何」を隠すのか？ということですが、

ハードウェア全体を隠すのではないことに注意してください。

隠すのは、個々のハードウェアが持つ物理的な特性や相違点です。

例えば、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を隠すことになります。

隠すというのは、同じ種類のハードウェア装置を共通化するという意味があります。

次に「何」から隠すのか？について説明します。

これは、ユーザプログラム(ユーザを含む)に対して隠します。

「なぜ」隠すのか？というと、

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です。

つまり、移植性の確保のためです。

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は、実際には、OSがハードウェア機構を仮想化することによって実現します。

教科書の図1.5に説明の図があるので見てください。

ここで、仮想化とは、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を指します。

仮想化については、後で例を示します。



ハードウェア機構の隠ぺいと仮想化について、具体的な例で説明します。

図ではファイル装置の例をしめしています。

この図で示されているように、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、アクセス方法がまちまちです。

もし、OSが装置を隠ぺいしないと、ユーザプログラムでは、物理的な装置の違いがあるため、それぞれ異なる方法でアクセスすることになります。

OSが装置を隠ぺい、すなわち、物理的な装置の違いをユーザプログラムから見えなくして、共通化されたアクセス方法のみを見せることで、

ユーザプログラムには、ファイル装置としての統一したアクセス方法を提供することができます。

(B) ハードウェア装置の管理

- 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

2つ目のハードウェア装置の管理について説明します。

これは、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・制御することを指します。

ハードウェア装置の例には、プロセッサ、メモリ、ファイル装置、入出力装置、通信装置等があります。

ユーザプログラムの実行に先立って、どの時刻にどの装置が使われるか決定するのは困難であるため、

実行時に決定する必要があります。これが、OSの役割となります。

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ます。

(C) ソフトウェアの実行時の保護

- 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

3つ目のソフトウェアの実行時の保護について説明します。

まず、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必要があります。
つまり、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ということです。

また、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をアクセス権と呼びますが、アクセス権の管理や競合の調停を行います。
つまり、ソフトウェアへの不正アクセスの防止となります。

さらに、OSとユーザプログラムとの実行時の区別である実行モードを設定します。

なお、OSは特権モードでファイルにアクセスできます。

1.2.2 OSの機能

- 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

それでは、OSの機能を項目でまとめてみましょう。

ハードウェアの仮想化による管理では、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・制御します。

また、マシン命令セットによる支援により、次にあげている機能を実電子ます。

プロセス

- 「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、複数のプロセスを作ることができる

次にプロセスについて説明します。

「実行されるプログラム」のことをプロセスと言います。

つまり、プロセスとは、プログラム + 実行の状態 を指します。

実際には、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ます。

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のでしょうか？

それは、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のに対して、

プロセスは、実行されるたびに作成されるからです。

一つのプログラムに対して、複数のプロセスを作ることができる点に注意してください。

プロセス管理

- 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

プロセスの管理について説明します。

まず、プロセスの割り付けがあります。

これは、プロセスを、メモリの特定の領域に置く、つまりプロセスとメモリを対応付けることを指します。

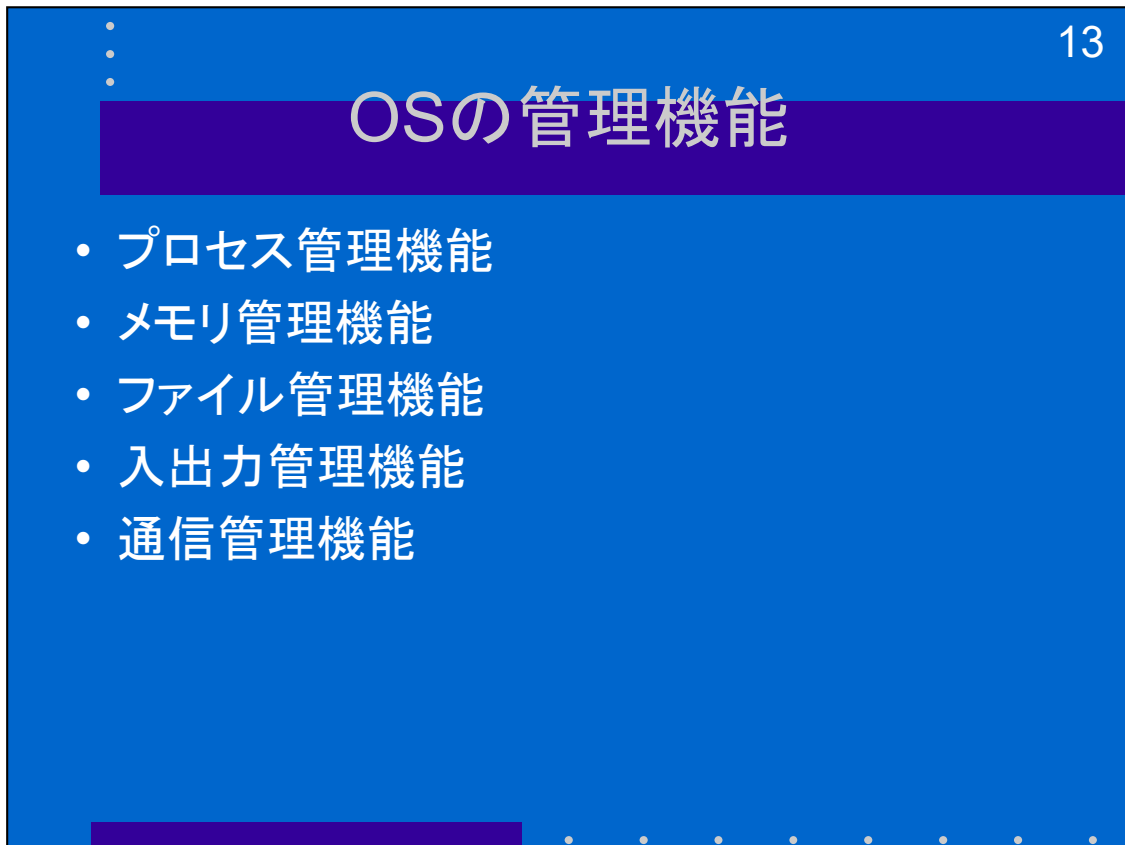
プロセスを「作る」ときに必要な処理となります。

次に、プロセッサ割り付けがあります。

これは、プロセスとプロセッサの時間を対応付けることを指します。

プロセスを「実行する」ときに必要な処理です。

詳細は2章で説明します。



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

OSの管理機能としては、ここにあげたような機能があります。
以下で、それぞれを説明します。

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

プロセスの管理機能とは次のようなものです。

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)

これについては、教科書の図1.6を参照してください。

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となります。

これらの詳細は2章で説明します。

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)

メインメモリの仮想化とは、

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することです。

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します。3章で詳細に説明します。

ファイル装置の仮想化とは、

ファイル装置の物理的な相違点を隠ぺいすることです。

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します。こちらも3章で詳細に説明します。

⋮

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

その他の管理機能には、入出力管理機能と通信管理機能があります。

入出力管理機能とは、
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ものです。
4章で詳細を説明します。

通信管理機能には、
ネットワーク通信などがあります。
こちらも、4章で詳細を説明します。

：(まとめ) OSによるハードウェア 17 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

ここまでの内容をいったんまとめておきます。

OSによるハードウェア装置の隠ぺいとは、
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ことです。
これは、OSの基本的な機能の一つです。

「隠ぺい」という用語は、「統一」、「共通化」、「仮想化」、「標準化」と置き換えることができます。

隠ぺいの対象には、
プロセッサ、メインメモリ、ファイル装置、入出力装置、通信装置などのハードウェア装置があります。

・(まとめ)ユーザプログラムに対するOS機能 18

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対するOSの機能についてまとめます。

これは、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ことを指します。ハードウェア装置は、隠ぺいして仮想化されます。

- ・ **プロセッサ**: ユーザプログラムを実行します
- ・ **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として使われます
- ・ **ファイル装置**: ユーザプログラムを長期的に格納する記憶装置となります。
- ・ **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装置となります。

19

： ユーザプログラムによる ハードウェア装置の利用

- ・ 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

ユーザプログラムによるハードウェア装置を利用する仕組みについて説明しておきます。

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ようになっています。

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」する必要があります(教科書の図1.8を参照)

ここでいう「通信」には、割り込みが使用される

- ・ ハードウェア装置から、OSを通じて、ユーザプログラムへの通信では、外部割り込み(ハードウェア割り込みともいう)を使用します。
- ・ ユーザプログラムから、OSを通じて、ハードウェア装置へ通信するときは、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)を使用します。

これらは、第3回の講義で説明します。

1.3 OSの構成

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

1.3節では、OSの構成について説明します。

まず、OSの基本構成についてです。

OSはモジュールで構成されています。

ここで、OSの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」と呼びます。

OSモジュールの要件は次の2つです。

- ・情報の隠ぺい
- ・方針と機構の分離

以下、これらについて説明します。

方針と機構の分離

- 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」

OSのモジュールの機能を理解するのに、「方針」と「機構」の分離について理解する必要があります。

同じような機能を実現する機能単位であっても、これら2つは別のモジュールとして分離する必要があります。

まず、方針(policy)は、ある機能を実現するときに生じる選択肢の決定を指します。

次に、機構(mechanism)では、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を決めます。

一言で説明すると、方針は「(機能を決めるための)アルゴリズム」であり、機構は「(機能を実現する)実装」となります。

なぜ方針と機構を分離するのか？

- 「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

そもそも、なぜ方針と機構を分離する必要があるのでしょうか？

まず、「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めていくものです。

具体的には、次のような運用方法に応じて決めていきます。

- ・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して決めるものです。

- ・機構では、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ことになります。

方針と機構は、それぞれ、独立に変更できることが望ましいため、別のモジュールにすることになります。

方針と機構の分離の例(1)

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

方針と機構を分離した例についていくつか説明します。

まず、プロセスのスケジューリングです。

方針にあたるものは、スケジューラです。

ここでは、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)の割付けを指示します。

つまり、どのプロセスから実行するかを、方針として決めることになります。

これに対して、機構にあたるのは、ディスパッチャです。

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します。

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して実装することになります。

方針と機構の分離の例(2)

メモリの割り付け

- 割り付け**制御部**
 - 記憶領域の割付けの**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

方針と機構の分離について別の例をあげます。

メモリの割り付けを、方針と機構で分離してみます。

方針にあたるのは、メモリの割り付け制御部です。

記憶領域の割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します。

機構にあたるのは、割り付け実行部です。

メモリ装置のハードウェアに依存して、上の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します。

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

次に、OSのモジュールについて、機能の点から説明します。

OSのモジュールは、大きく分けると、狭義のカーネルとシステムサーバに分けられます(教科書の図1.9を参照)

狭義のカーネル(kernel)とは、

名前の通りOSの核となる機能を実現するプログラムのことです。

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します。

システムサーバとは、

狭義のカーネル以外の個別のOS機能を提供するものです。

具体的には、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、通信管理はシステムサーバとして実現されます

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ります。

OSとカーネルは何が違うか？

- OSの役割が拡大するにつれて、OS本来の機能(「OSの管理機能」参照)に加えて、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、ユーザインタフェースなど)としての機能が加わるようになった
 - OSそのもののサイズが大きくなり、モジュールに分割して整理する必要が生じた
- OSの中核的なモジュール(OSの管理機能を実現するモジュール)を他と区別して議論する必要が生じたため、それを**カーネル**と呼ぶようになった

前のスライドでいきなり、カーネルという用語が出てきたので、少し補足します。

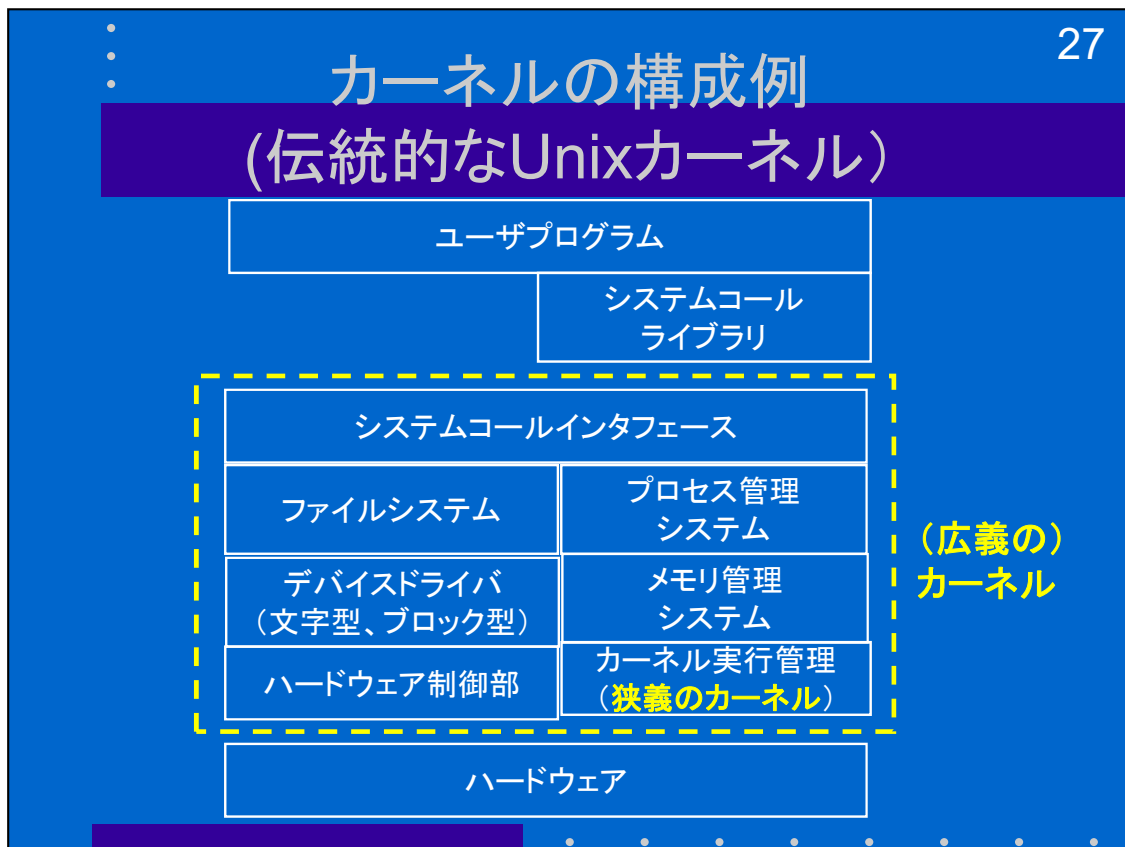
カーネルは、OSの中で、OS本来の機能(この授業の前の方で出てきた「OSの管理機能」のことです)を実現するソフトウェアです。

こう聞くと、OSとカーネルは何が違うのかという疑問が出るかと思います。

確かに初期のOSでは、OSとカーネルの区別はほぼなかったと思われますが、OSの役割が拡大していくにつれて、OSの管理機能以外に、ミドルウェアと呼ばれる、言語処理プログラム(コンパイラなど)、データベース管理システム、ユーザインタフェース(コマンドラインやグラフィックのユーザインタフェース)など多様なソフトウェア群が加わるようになりました。

これに応じて、OSそのもののサイズが大きくなり、OSをモジュールに分割して整理する必要が生じてきました。

このようなモジュールの中で、OSの管理機能を実現する、つまりOSの中核的なモジュールを、他のモジュールと区別して議論する必要が生じたため、この中核的なモジュールを「カーネル」と呼ぶようになりました。



カーネルの構成例を、伝統的なUnixカーネルで説明します。

Unixカーネルでは、この図のような構成をとり、狭義のカーネルはカーネル実行機能のモジュールを指し、

単にカーネルというと、それ以外のシステムサーバにあたるモジュールも含めた、広義のカーネルを指します。

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

カーネルを中心とした、OSモジュールの構成方式について説明します。
カーネルの代表的な構成法は次の2種類です。

1つ目は、単層カーネル方式です。

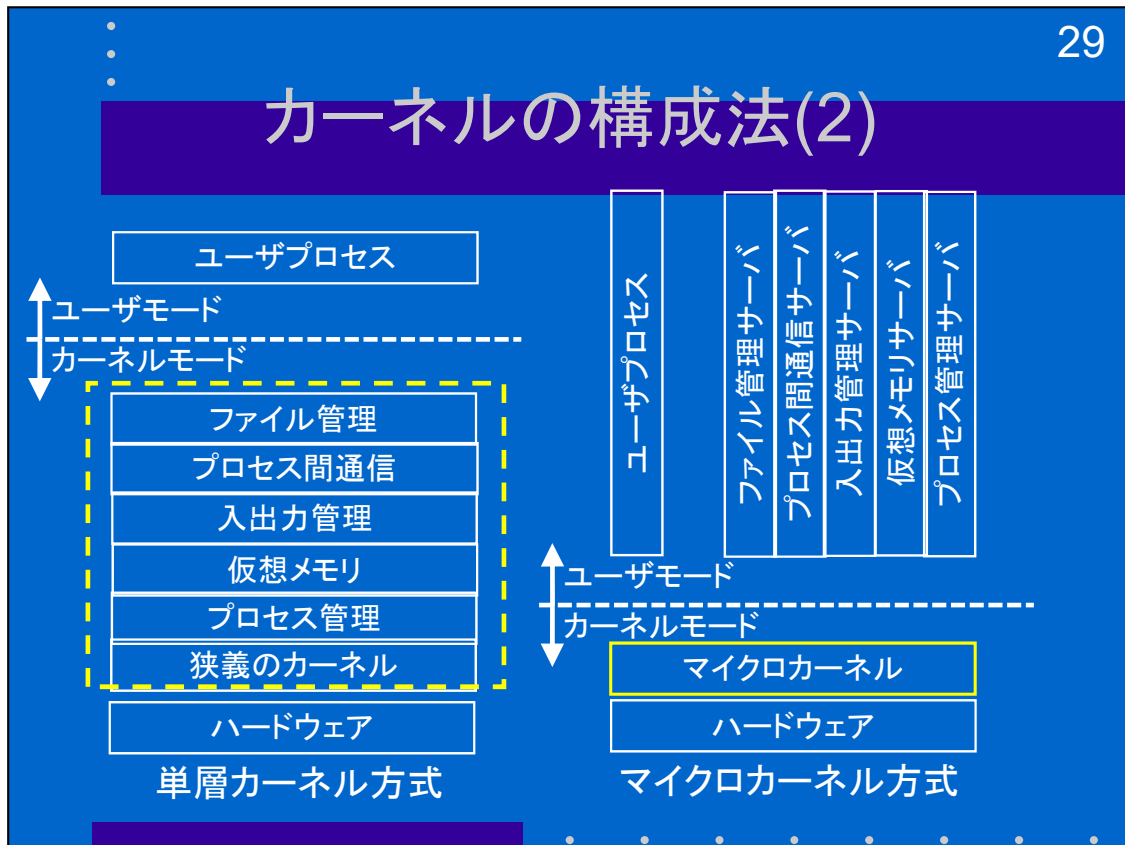
これは、モノリシックカーネルとも呼ばれます。

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します。

広義のカーネルを一つのプログラムにまとめたものとみなせます。

2つ目は、マイクロカーネル方式です。

カーネルを機能ごとに分割して、狭義のカーネルを除く、それぞれをシステムサーバとして実現する方式です。



前のスライドの2つの方式を図にして比較してみます。

単層カーネル方式では、広義のカーネル機能がすべて単一のプログラムにまとめられます。

ユーザプログラムから、カーネル機能が呼び出されたときは、常にカーネルモード、つまり特権モードで処理されます。

一方、マイクロカーネル方式では、狭義のカーネルの機能以外は、全部システムサーバとして構成されます。

なお、この方式で狭義のカーネル機能を実現するモジュールは「マイクロカーネル」と呼ばれます。

システムサーバは、ユーザモードで動作するため、ユーザプロセスからカーネル機能が呼び出されたときは、いったんマイクロカーネルに制御が移りカーネルモードになりますが、その後、要求された機能ごとに対応するシステムサーバが呼ばれます。

システムサーバはユーザモードで動作します。

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単一のアドレス空間で実行するプログラム(広義のカーネル)として実装
- 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)

個々の方式についてももう少し詳しくみていきましょう。

まずは、単層カーネル方式です。

この方式では、オペレーティングシステムの全機能を単一のアドレス空間で実行するプログラム(広義のカーネル)として実装しています。

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します。

システムコールのパラメータで、ユーザプロセスの要求を記述します。

また、システムコール処理後には、ユーザプロセスの次の命令に戻る(ユーザモードに遷移)ことになります。

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

カーネルモードとユーザモードについても詳しくみていきます。

カーネルモードは、特権モード、スーパーバイザモードとも呼ばれます。

カーネルモードでは、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になります。

ユーザモードは、保護モードとも呼ばれます。

ユーザモードでは、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ます。

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）

単層カーネル方式を、マイクロカーネル方式と比較したときの長所と短所をあげます。

まず、長所ですが、

- ・ 実行効率がよい（カーネル内は、同一プログラム内なので、手続き呼び出しと同じように呼び出せる）
 - ・ モジュールを追加するだけで新機能を追加できる
 - ・ OSの全体機能が小さければ、実装が比較的容易である
- などがあげられます。

逆に、短所としては、

- ・ 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する（機能ごとの厳密なモジュール設計が必要）
 - ・ 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）
- などがあげられます。

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)

次にマイクロカーネル方式について説明します。

この方式では、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されます。

マイクロカーネルの機能は次のようなものです。

- ・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

また、システムサーバプロセスは次のような機能を持っています。

- ・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には、メモリマネージャ、ファイルサーバ、ネームサーバなどがある。

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

マイクロカーネル方式の特徴としては、

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ことがあげられます。

具体的には、マイクロカーネル⇄システムサーバ、システムサーバ⇄システムサーバでプロセス間通信が行われます。

長所としては、

- ・ 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(基本的にマイクロカーネルだけメモリに常駐すればよい)
 - ・ 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
- があります。

逆に、欠点としては、

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ことがあげられます。

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

それぞれの方式の実装例をあげておきます。

単層カーネル方式のOS実装例には次があります。

- 初期のUnix
- 多くの商用Unix
- FreeBSD, Linux

マイクロカーネル方式のOS実装例には次があります。

- Windows NT系列のOS
- macOS

詳細は第7回で説明します。