オペレーティングシステム

Ver. 0.0.0

徳山工業高等専門学校
情報電子工学科

目次

第 1 章

オペレーティングシステムとは

オペレーティングシステム（Operating System : OS）は，Windows，macOS，Linux，FreeBSD，Android，iOS 等である．皆さんは，これらを使用した経験を持っているだろう．そして，これらが次のようなソフトウェアから構成されていることを何となく感じているのではないだろうか．

1. カーネル（OS の本体）
2. ライブラリ（プログラムが使用するサブルーチン，DLL）
3. ユーザインタフェース（GUI，CLI）
4. ユーティリティソフトウエア（ファイル操作，時計，シェル，システム管理 ...）
5. プログラム開発環境（エディタ，コンパイラ，アセンブラ，リンカ，インタプリタ）

広義では上に列挙した全て[*1]がオペレーティングシステムの一部である．逆に**狭義**では「カーネル」だけをオペレーティングシステムと考える．この講義では狭義のオペレーティングシステムの仕組みを勉強する．

1.1 オペレーティングシステムの役割

オペレーティングシステムの重要な役割は次に述べる二つである．

1.1.1 拡張マシンとしてのオペレーティングシステム

OS はハードウェアの機能を**抽象化**した便利な拡張マシンを提供する．次に抽象化と拡張マシンの例を示す．

例 1 二次記憶装置の抽象化（ファイルシステム）
ハードディスク，USB メモリ，CD-ROM 等の二次記憶装置は，どれもデータを記録する機能を持ったハードウェアである．しかし，それらの制御方法や記録されるデータの構造は全く異なる．オペレーティングシステムは，二次記憶装置をファイルの集合（ファイルシステム）として**抽象化**してユーザプログラム（アプリケーションプログラム）に提供する．

例 2 コンピュータそのものの抽象化（プロセス）
プロセスはプロセス専用の仮想 CPU と仮想メモリを持つ．システムコールを通じて入出力も可能である．プロセスは CPU，メモリ，入出力を持っているので 1 台のコンピュータと考えることもできる．
プロセスはコンピュータを**抽象化**したものだとも言える．（プロセス＝仮想コンピュータ）

例 3 拡張されたコンピュータ（システムコール）
オペレーティングシステムを備えたコンピュータ上では，アプリケーションプログラムがシステムコールを発行できる．システムコールを追加命令を考えると，オペレーティングシステムを備えたコンピュータは追加命令を実行可能な**拡張マシン**だと言える．（拡張マシンの命令＝機械語命令＋システムコール）

オペレーティングシステムが拡張マシンをアプリケーションプログラムに提供するイメージを図 1.1 に示す．ハードウェアの複雑で統一されていない凸凹のインタフェースは，オペレーティングシステムによってスッキリし

[*1] 上に挙げたソフトウェアの中で「プログラム開発環境」は，Linux や FreeBSD では OS に含まれているが，それ以外では別にインストールする必要があり OS の一部とは言い難くなっている．

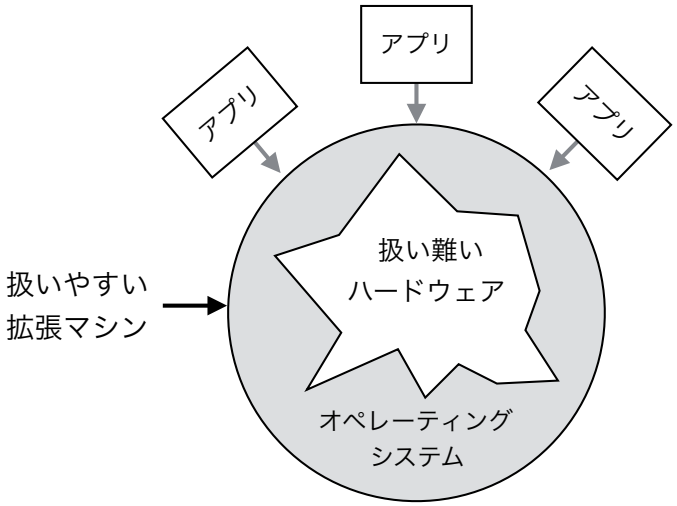


図 1.1　抽象化

図 1.2　コンピュータシステムの構成

た円弧のインタフェース（使いやすい**抽象化**されたインタフェース）に変換される．

オペレーティングシステムの円がハードウェアの外側にあるのは，オペレーティングシステムによって機能が拡張されたことを示す．ハードウェアを含む円全体が拡張マシンを表している．

1.1.2　ハードウェア管理プログラムとしてのオペレーティングシステム

オペレーティングシステムはハードウェア資源を管理・制御し，アプリケーションプログラムにシステムコール等のサービスを提供する．図 1.2 はカーネルの役割を説明している．

オペレーティングシステムは管理するハードウェア資源をアプリケーションプログラムに割当てる．複数のアプリケーションプログラムに割り付けるために資源は必要な数だけ**仮想化**して準備する必要がある．例えば，CPU は時間を区切って複数のプロセスが共有する（時分割多重による**仮想化**）．メモリはアドレスで区切って複数のプロセスが共有する（空間分割多重による**仮想化**）．

1.2　オペレーティングシステムの歴史

1.2.1　第 1 世代（1945〜1955，真空管の時代）

初期のコンピュータはコンソールパネルを通して操作する，巨大な TeC のようなものだった．OS は存在せずプログラマはまさに TeC と同様なプログラミングとデバッグを行っていた．

しかし，当時のコンピュータは TeC と異なり大変高価な装置であった．その高価なコンピュータを一人のプログラマが長時間にわたって独占使用することになる．プログラマがバグの原因を考えている間，とても高価なコン

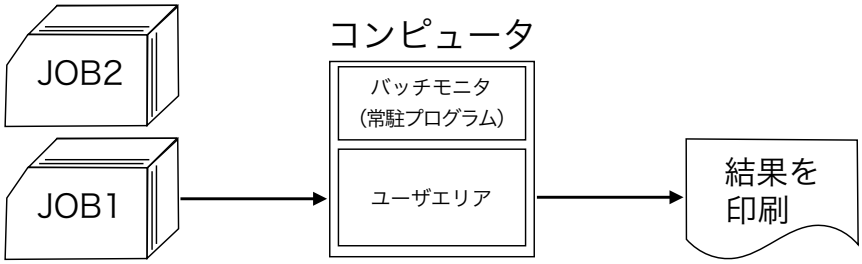


図 1.3　バッチ処理

図 1.4　紙カード

ピュータが遊んでしまい勿体ないものであった．

1.2.2　**第２世代（1955〜1965，トランジスタの時代）**

トランジスタ回路で製作されるようになり，**メインフレーム**と呼ばれる大型コンピュータが大企業，政府機関や大学等で実用的に使用されるようになった．メインフレームは数百万ドルと高価だったので，ハードウェアを遊ばせること無く使用することが優先課題であった．そこで人手を介すること無く自動的に次々と連続して処理を行う「コンピュータの自動運転」が行われるようになった．この処理方式のことはバッチ処理と呼ばれた．図 1.3 にバッチ処理の概要を示す．

プログラマは図 1.4 のような紙カードにプログラムやデータを一行ずつ打込む．100 行のプログラムは 100 枚の紙カードを使用して記録する．このようにして出来た紙カードの束が一つの処理を単位（ジョブ）になる．コンピュータでは**バッチモニタ**と呼ばれる常駐プログラムが実行される．バッチモニタは紙カードからジョブを読み込み実行させる．ジョブが終了するとバッチモニタに制御が戻り次のジョブが自動的に実行される．バッチモニタが発展してやがて OS になる．

この方式でうまく処理できるように次のような発明があった．

1. JOB 制御言語（JCL : Job Control Language）
   バッチモニタを制御するコマンド言語を JOB 制御言語（JCL）と呼ぶ．JCL コマンドはジョブ途中の紙カードに記載する．図 1.5 に JCL を含むジョブの構成を示す．これは，FORTRAN 言語で記述したプログラムを実行し，後半にあるデータを処理するジョブの例になっている．
2. 実行モード

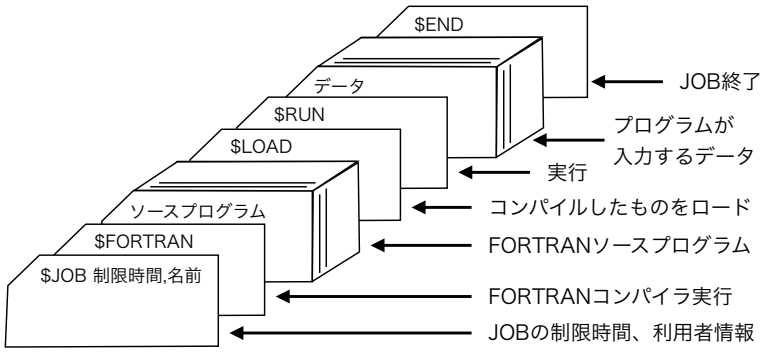


図 1.5　ジョブの構成

ユーザプログラム（ジョブ）のバグでバッチモニタが破壊されないようにするために，ユーザプログラム実行中なのかバッチモニタ実行中なのか区別する必要がある．どちらを実行中なのかを示すハードウェアのフラグを導入し，ユーザモードとカールモード（スーパバイザモードとも呼ぶ）を区別するようになった．ユーザモードではハードウェアへのアクセスや，実行できる機械語命令に制限がある．

3. システムコール
   ユーザプログラムが直接に入出力装置等にアクセスすることは，バッチ処理を継続できなくする恐れがあるので許されない．例えばユーザプログラムがハードウェアのモードを切り換えてしまうと，以降のジョブが正常に実行されなくなる．そこで，ユーザプログラムはバッチモニタに依頼（システムコール）して入出力を行う必要がある．
   プログラムが終了する時はカーネルモードに切り換えてバッチモニタに戻る必要がある．カーネルモードに切り替える機械語命令をユーザプログラムが実行可能だと，実行モードが無意味になるので許可すべきではない．システムコールを使用してプログラムを終了する．
4. 記憶保護
   ユーザプログラムのバグでバッチモニタが破壊されないように，ユーザモードで実行中は主記憶のバッチモニタ領域に書込みができないようにする．

1.2.3　第３世代（1966〜1980，IC とマルチプログラミングの時代）

1960 年代のコンピュータは IC（Integrated Circuit）を用いて作られるようになり価格性能比が随分改善された．第 3 世代と呼ばれる当時のオペレーティングシステムの中には，現代のオペレーティングシステムの先祖であったり，強い影響を与えたものがある．図 1.13 に第 3 世代から現代に至るまでの系統図を示す．

IBM が開発した System/360（図 1.6）は高価な大型のものから，安価な小型のものまでで同じオペレーティングシステムが使用でき，同じユーザプログラムを実行できる**シリーズ化**を行い商業的に大成功をおさめた [27]．System/360 はそれ以前のものとは異なり科学技術計算にも事務処理にも使用できる．System/360 のオペレーティングシステムは，1966 年にデビューした OS/360 である．図 1.13 に示すように，OS/360 の子孫である z/OS が現代でも使用されている [1]．

OS/360 を含む第 3 世代のオペレーティングシステムが実現した重要な新しい機能を紹介する．

- 仮想記憶
  主記憶を仮想化し実際より大きい主記憶があるように見せる．実際の主記憶より大きいプログラムが実行可能になる．
- マルチプログラミング
  図 1.7 のように，複数のプログラム（ジョブ）を主記憶にロードしておき，その中で実行可能なものを選んで実行する．入出力待ち等で実行できなくなったら他のプログラムを実行する．高価な CPU が入出力待ちで停止する可能性を低くすることができた．
- タイムシェアリング

ウィキメディア / Bundesarchiv, B 145 Bild-F038812-0014 / Schaack, Lothar / CC-BY-SA 3.0 de

図 1.6　フォルクスワーゲンで使われている System/360

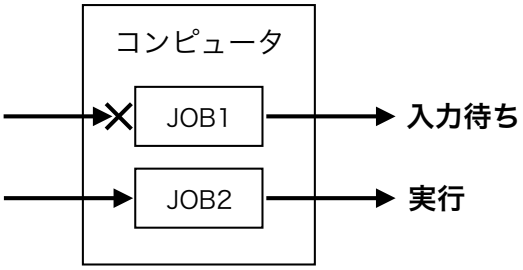


図 1.7　マルチプログラミングシステム

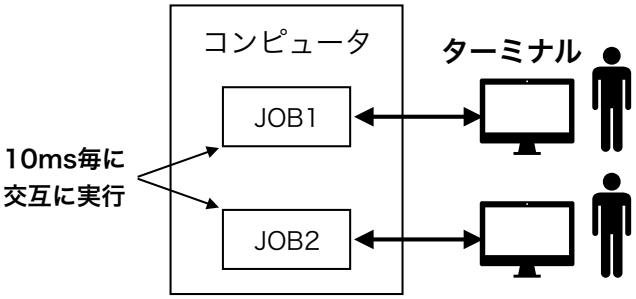


図 1.8　タイムシェアリングシステム

マルチプログラミングの一種である．図 1.8 のように，複数のターミナルをコンピュータに接続し複数のユーザが同時にコンピュータを使用できるようにする．短時間（例えば 10ms）で処理するジョブを次々に切り換えることで，ユーザは自分がコンピュータを独占しているように感じることができる．なお，ターミナルは図 1.9 のような，キーボードと表示装置だけを備えた安価な装置である．

この時代のオペレーティングシステムやコンピュータシステム，そして，それらの開発プロジェクトの中で，その後のオペレーティングシステムに多くの影響を与えた有名なものを紹介する．

写真：http://commons.wikimedia.org/wiki/File:Televideo925Terminal.jpg （パブリックドメイン）

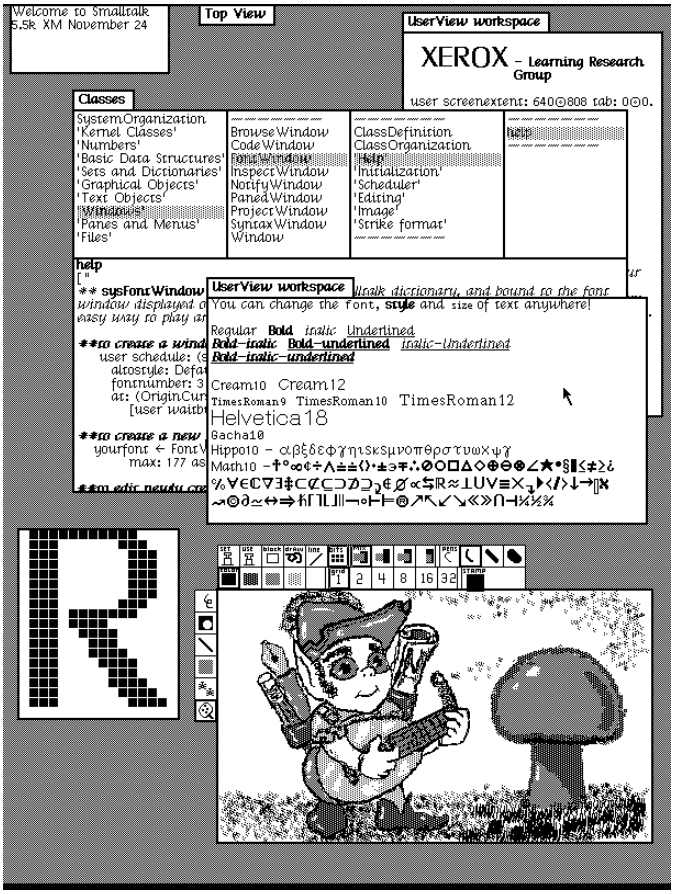
図 1.9　ターミナル

- OS/360
  世界初の本格的な商用オペレーティングシステムである．メインフレームの主流 OS となり子孫は現在でも使用されている [1].
- MULTICS（MULTiplexed Information and Computing Service）プロジェクト [27]
  MIT，ベル研究所，General Electric が共同で始めた巨大で強力なコンピュータシステムを構築するプロジェクトである．強力な一台のコンピュータで都市一つ分のコンピュータサービスを提供する構想だった．完成までに長い期間を要し（その間にベルと GE が脱落し），商業的には失敗であったがその後のオペレーティングシステムに影響を与える多くのアイデアが出てきた．
- UNIX（ユニックス）
  MULTICS プロジェクトから抜けたベル研の Ken Thompson らにより開発された [5]．図 1.13 に示すように，現代のオペレーティングシステムの多くが UNIX を起原にしている．子孫ではないものも UNIX の影響を強く受けている．LINUX は UNIX 互換のオペレーティングシステムを作ろうとして開発が始まった [19]．Android の中身は LINUX である [20]．z/OS は UNIX 互換環境を備えている [4]．Windows にも UNIX 互換環境（POSIX サブシステム）を利用可能なものがある [22].
- DynaBook（ダイナブック：OS だけでなくコンピュータ全体を指す）[31]
  アラン・ケイが 1972 年に著した「A Personal Computer for Children of All Ages」[29, 30] に登場する理想のパーソナルコンピュータである．アラン・ケイがゼロックスのパロアルト研究所に在籍中の 1970 年代に開発した Alto 上の「暫定ダイナブック環境」（図 1.10）は既に GUI やマウスを使用していた．スティーブ・ジョブスが Alto を見たことが LISA 開発きっかけになったと言われている [16].

1.2.4　第４世代（1980〜現代，PC の時代）

1970 年代に単一の LSI に CPU 全体を集積したマイクロプロセッサが登場した．1970 年代中旬にはマイクロプロセッサを用いて個人向けのコンピュータであるパーソナルコンピュータ（当時はマイクロコンピュータと呼んでいた）を作ることが可能になった．それに伴いパーソナルコンピュータ用のオペレーティングシステムが登場した．

1. 8bit マイクロコンピュータの時代
   1977 年に Digtal Reserch 社が CP/M（Control Program for Microcomputer）と呼ばれる 8bit マイクロコンピュータ用の簡単なオペレーティングシステムを開発し成功した．しかしこのオペレーティングは 16bit パーソナルコンピュータの時代には早々に消え去ってしまった [28].
2. 16bit パーソナルコンピュータの時代
   IBM が 1981 年に 16bit パーソナルコンピュータ IBM PC[23] （図 1.11）を発売した．IBM PC は現在の



ウィキメディア / SUMIM.ST /

Alto や NoteTaker で動作したアラン・ケイ達の暫定 Dynabook 環境 (Smalltalk-76、同-78 の頃) / CC-BY-SA 4.0

図 1.10　Alto（Alto エミュレータ）のスクリーンショット

Windows PC の先祖である．IBM PC の子孫は改良や拡張を続けながら現在まで高いシェアを維持し続けている．IBM PC のオペレーティングシステムとして開発されたのが，Microsoft 社の MS-DOS（MicroSoft Disk Operating System）[21] である．バージョン 2 からは UNIX のような階層ディレクトリやパイプ，リダイレクト等の機能を持っている．図 1.13 に示すように，MS-DOS は Windows に置き換わり Windows ME までバージョンアップが繰り返された．

Apple 社は 1984 年に Macintosh（図 1.12）を発売した．Machintosh の OS である MacOS は LISA を経て DynaBook[29, 30] の影響を受けていると言われている [28]．図 1.13 に示すように，当初の MacOS は MacOS 9[15] まで改良が続けられた．

3. 32bit パーソナルコンピュータの時代

1990 年頃には 32bit のマイクロプロセッサがパーソナルコンピュータにも使用されるようになった．32bit のマイクロプロセッサは実行モードを備え，またメモリ管理ユニットも利用可能であった．つまり，カーネルモードとユーザモードを使い分けたり仮想記憶を利用する本格的な第 3 世代のオペレーティングシステムを実行できる環境がパーソナルコンピュータにも整った．

そこで，従来ワークステーションやミニコンで使用されていた UNIX を安価なパーソナルコンピュータ（特に IBM PC 互換機）で動くようにする人たちが現れ，オープンソースソフトウェアとして LINUX や FreeBSD 等の開発が始まった．また，もともとパーソナルコンピュータ用の Windows や Mac OS も 32bit マイクロ

ウィキメディア / Bundesarchiv, B 145 Bild-F077948-0006 / Engelbert Reineke / CC-BY-SA 3.0 de

図 1.11　IBM PC

ウィキメディア / w:User:Grm wnr / File:Macintosh 128k transparency.png /GFDL

図 1.12　初代 Macintosh

プロセッサの機能を使いこなす本格的なオペレーティングシステムに生まれ変わった．

- LINUX
  1991 年に開発が始まった LINUX は UNIX 互換のオペレーティングシステムをパーソナルコンピュータ（IBM PC 互換機）用に独自に作成したものである [19]．LINUX は改良され続け，現在ではパーソナルコンピュータだけでなく，スーパーコンピュータ「京」のオペレーティングシステム [32] から，スマートフォンのオペレーティングシステムである Android[20]，テレビ等の組込みシステムのオペレーティングシステムまで，広く使われるようになっている．
- BSD 系の UNIX
  386BSD[11] は BSD UNIX を Intel 80386 CPU を搭載したパーソナルコンピュータ（IBM PC 互換機）で動作するようにしたものである．386BSD は FreeBSD 等に受継がれるが UNIX のライセンス問題が発生する [5]．ライセンス問題が片付き安心して使用できるようになった 4.4BSD-Lite Release 2[5] をベースに FreeBSD，NetBSD，OpenBSD 等の多くの BSD 系 PC-UNIX が開発された．

その後，FreeBSD は MacOS X に取り込まれている．また，FreeBSD に ZFS が移植された [33] のでファイルサーバ用に特化した FreeNAS[13] にも使用されている．なお，徳山工業高等専門学校・情報電子工学科のパソコン室では 1993 年 10 月に 386BSD の利用を開始して以来，2014 年 3 月まで FreeBSD を学生用 PC やサーバのオペレーティングシステムとして使用してきた [25]．

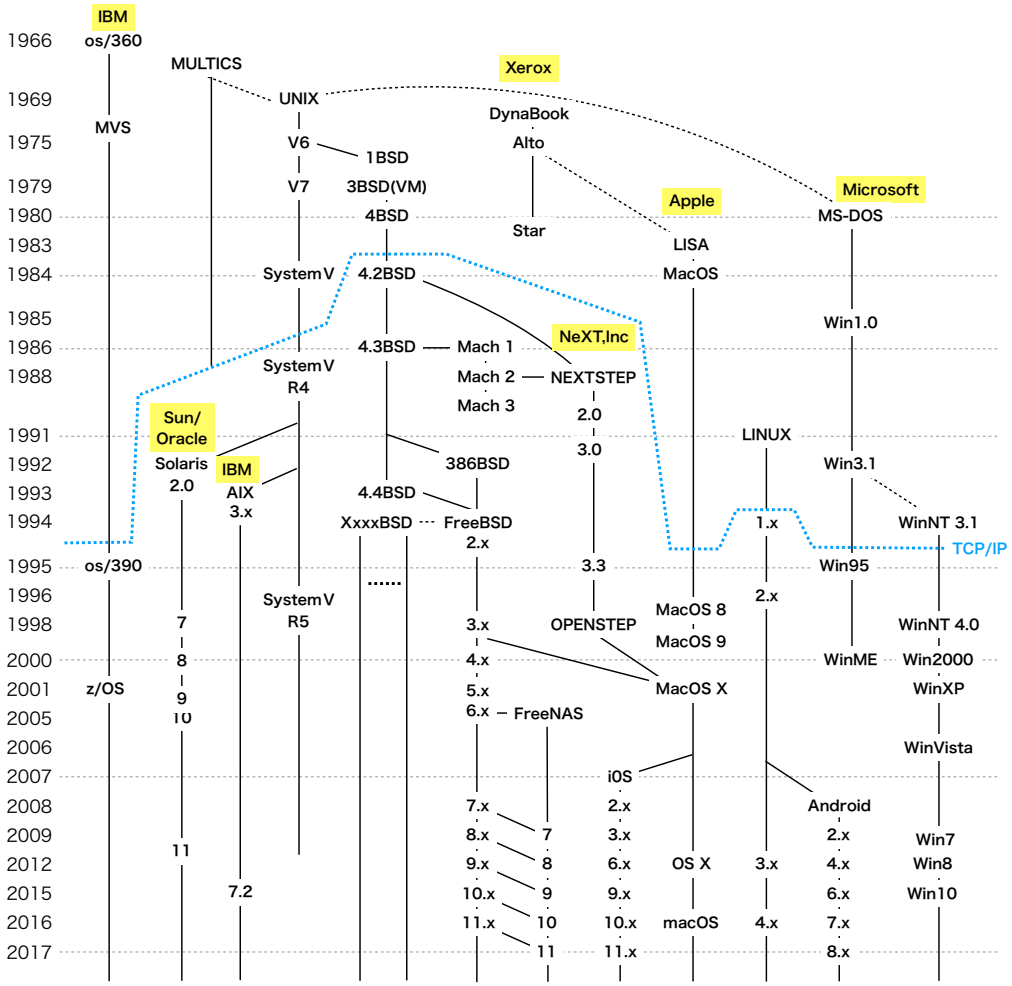
- System V 系の UNIX
  System V の流れを汲む Solaris[6] は，RISC マイクロプロセッサ SPARC を搭載するサーバやワークステーションでも，パーソナルコンピュータ（IBM PC 互換）でも使用できる．
- 従来のパーソナルコンピュータ用オペレーティングシステム
  従来の Windows や Mac OS は CPU の実行モード等を使用していなかったので，アプリケーションプログラムのバグによりシステム全体が停止するようなトラブルを防ぐことができなかった．そこで，32bit マイクロプロセッサの使用を前提に新しく作り直された．
  新しく作り直された 32bit の Windows NT 系列の製品は，徐々に従来の Windows を置換えた．(図 1.13 参照)．現在（2017 年 10 月）の最新版は Windows 10 である．
  MacOS は，2001 年に UNIX の流れを汲み安定して動作する OPENSTEP ベースの MacOS X[17] に置き換わった（図 1.13 参照)．その後，名称が OS X， macOS と変更されたがこれらは MacOS X の改良版である．現在（2017 年 10 月）の最新版は macOS 10.13 High Sierra である．iPhone の iOS は MacOS X をタッチパネル用に再構成したものである [18]．

1.2.5　インターネット世代

現在のオペレーティングシステムは TCP/IP 機構が組込まれインターネットに接続することができる．今ではパーソナルコンピュータやスマートフォンの使用をインターネット抜きに考えることができない．オペレーティングシステムにとってインターネットに接続できることは重要なことである．

TCP/IP を実装した 4.2BSD が 1984 年に公開された [10]．以来，4.2BSD の子孫はインターネットに対応している．1988 年に公開された System V R4 は BSD 起原の TCP/IP の実装を含んでいた [24]．これの子孫もインターネットに対応している．LINUX も 1.0 の頃には TCP/IP の実装を含んでいた [26]．Windows は Windows 95 から TCP/IP を標準装備している [22]．MacOS は MacOS 8 が発表されるまでにはインターネット対応がされていた [15]．メインフレームの世界でも OS/390 はインターネットに対応した [3]．

このようにして 1990 年代の後半には多くのオペレーティングシステムがインターネット対応を完了させた．インターネット対応を完了させたオペレーティングシステムを「インターネット世代のオペレーティングシステム」と言うことができる．



系統図は [1, 2, 3, 4, 5, 6, 7, 8, 10, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22] の内容を総合して作成した.

図 1.13　オペレーティングシステムの系統図

第 2 章

前提知識

以下では，本書で想定しているコンピュータのハードウェアやソフトウェアの構成について解説する．

2.1 コンピュータのハードウェア構成

本書は，コンピュータのハードウェア構成が図 2.1 のようになっていることを前提にしている．

1. CPU（Central Processing Unit）
   CPU はコンピュータの頭脳である．図は CPU が二つの構成になっているが，実際は一つの場合ももっと多い場合もある．
2. メモリ（主記憶装置）
   プログラムやデータを記憶し，CPU がプログラム実行に直接使用する記憶装置である．
3. タイマー
   タイマーは指定された時刻になったら CPU に割り込みを発生する．一定間隔で繰り返し割り込みを発生する場合もある．
4. グラフィックアダプタ
   ディスプレイを接続するためのアダプタである．表示内容を記憶するメモリを独自に持つ場合と，主記憶装置を使用する場合がある．最近のパーソナルコンピュータでは，グラフィックアダプタに GPU(Graphics Processing Unit) が組込まれている．

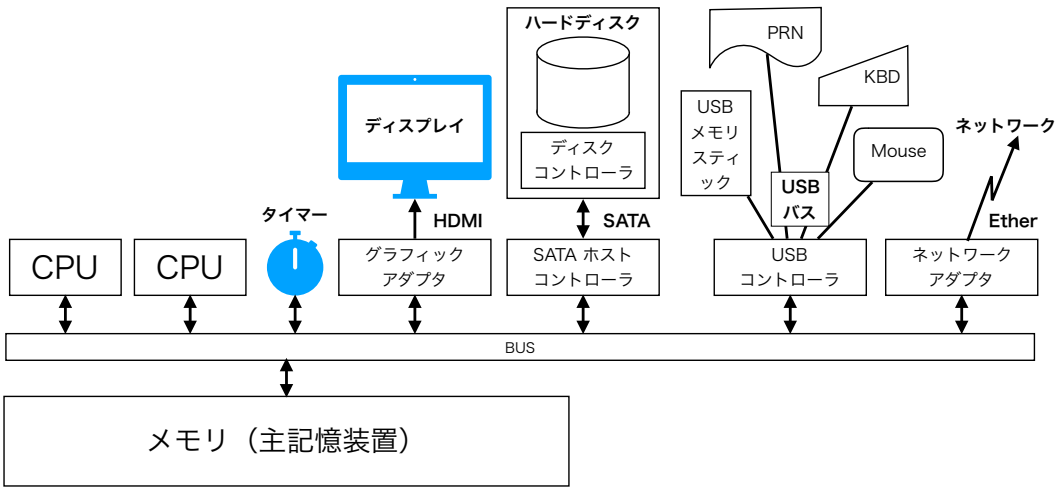


図 2.1　ハードウエア構成

5. SATA ホストコントローラ
   SATA（Serial Advanced Technology Attachment）は，パーソナルコンピュータと二次記憶装置（ハードディスクや CD-ROM）を接続するためのインタフェース規格である．SATA ホストコントローラは次のような動作をする．
   (a) CPU がコマンドを SATA ホストコントローラに書き込む．コマンドは，「読み・書き」，「セクタアドレス」，「セクタ数」，「メモリアドレス」を含んだものである．
   (b) SATA ホストコントローラは，ディスクコントローラと通信しハードディスクにコマンドを渡す．
   (c) ハードディスクの読み・書きが可能になったら，ホストコントローラはハードディスクとメモリの間でデータ転送を行う．このような CPU を介さないデータ転送のことを，**DMA（Direct Memory Access）**と呼ぶ．
   (d) SATA ホストコントローラは CPU に割り込み信号を送り，データの転送が完了したことを知らせる．（完了割り込み）
   CPU は，SATA ホストコントローラにコマンドを送ってから割り込みが発生するまでの間，他の仕事をすることができる．ハードディスクにの操作（I/O 操作）と CPU の計算は並列実行される．
6. USB コントローラ
   USB（Universal Serial Bus）は，パーソナルコンピュータと周辺装置を手軽に接続できるインタフェースである．USB メモリスティックやプリンタ，キーボード，マウス等，多くの周辺装置が USB を通して接続できる．USB コントローラも SATA ホストコントローラのように DMA 機能を備えている．
7. ネットワークアダプタ
   パーソナルコンピュータのネットワークアダプタは，GbE（Gigabit Ethernet）規格のものが普及している．これも SATA ホストコントローラのように DMA 機能を備えている．
8. BUS（バス）
   パーソナルコンピュータのハードウェアを構成する装置の間でデータをやり取りするための配線である．CPU だけでなく DMA を使用するコントローラやアダプタが大量のデータ転送を行うので，バスのデータ転送能力がパーソナルコンピュータの性能向上のボトルネックになる．
   そのため後で説明するように，実際の物理的な接続は図 2.1 とはかなり異なった構成になっている．しかし，オペレーティングシステムが意識しなければならない論理的な接続は図 2.1 のようになっていると考えてよい．

2.2　CPU の構成

本書では、CPU は図 2.2 のような部品で構成されると考える．図 2.1 に示したように，CPU は BUS を通して他の装置と接続される．CPU は，他の装置から割り込みを受け付けることができる．

1. PSW（Program Status Word）
   PSW は，PC（Program Counter）と Flags（フラグ）から構成されるものとする[*1]．PC は CPU が実行中のプログラムの命令アドレスを保持するカウンタである．Flags は計算の結果によって変化するフラグの他に，割り込み許可／不許可を表現するビット，実行モード（ユーザモード／カーネルモード）を表現するビット等が含まれる．
2. CPU レジスタ
   計算に使用する CPU の汎用レジスタのことである．TeC では G0，G1，G2，SP のこと，情報処理技術者試験の COMET では GR0，GR1，GR2，GR3，GR4 のことである．

PSW と CPU レジスタはソフトウェアから見える[*2]ので，CPU を仮想化する際に保存しなければならない記憶装置である．

[*1] 教科書によっては，フラグだけを PSW と呼ぶ場合もある．
[*2] 機械語プログラムから見える記憶装置である．

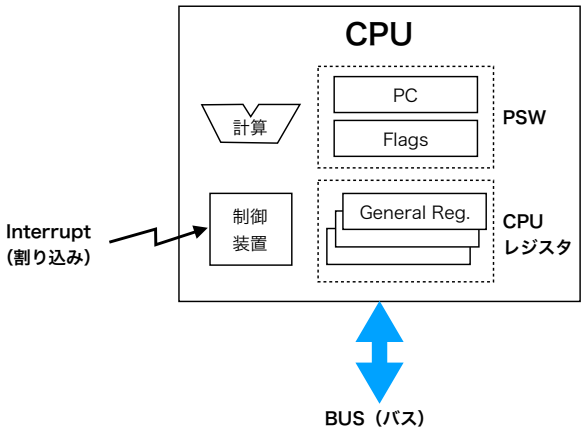


図 2.2　CPU の構成

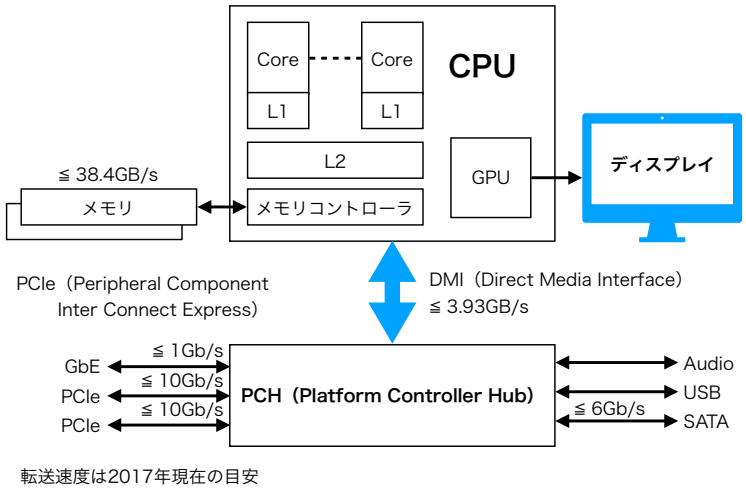


図 2.3　デスクトップ PC の構成

2.3　最近のコンピュータの実際の構成

Intel 社の CPU を使用したデスクトップ・パーソナルコンピュータとサーバコンピュータの構成を説明する．バスがボトルネックにならないように，メモリを CPU に直接接続してある．

2.3.1　デスクトップ・パーソナルコンピュータ

図 2.3 は Intel 社の CPU を使用した近年のデスクトップ・パーソナルコンピュータの構成を表している．Intel 社の用語では，これまで「CPU」と読んでいたものが「Core（コア）」と呼ばれる．「CPU」は複数のコアを含んだ LSI のことを指している．デスクトップ用の CPU には一つから 4 つのコアが集積されている．

コアに隣接している L1 はレベル 1 キャッシュ（Level 1 cache）を表している．L2 は複数のコアにシェアされるレベル 2 キャッシュ（Level 2 cache）を表している．メモリとのデータ転送量が多いコアと GPU が CPU と同じ LSI に集積され，I/O 装置のコントローラやアダプタは PCH に集積されている．CPU と PCH は DMI と呼ばれる専用のインタフェースを用いて接続される．

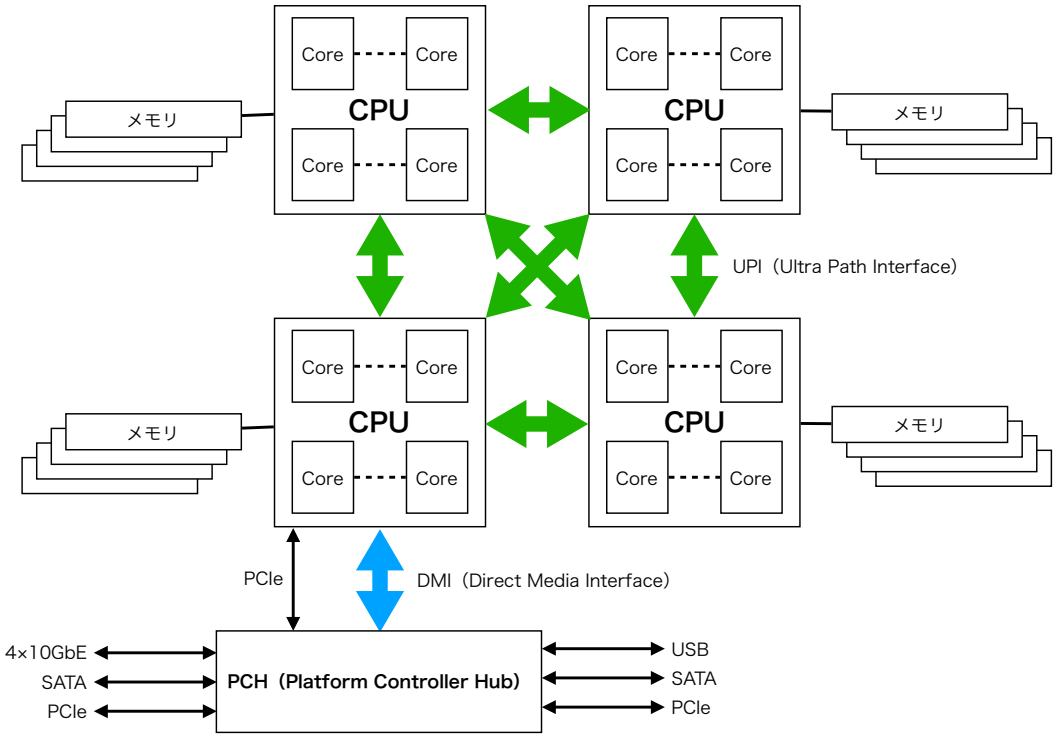


図 2.4　サーバ PC の構成

2.3.2　サーバコンピュータ

より強力な処理能力が必要なサーバ用コンピュータでは，図 2.4 のように多くのコアを内蔵する CPU を複数個使用する．現在（2017 年秋）最新の Intel Xeon Processor Scalable Family の場合，CPU 同士は UPI と呼ばれる高速な専用インタフェースで接続される．最大の構成は，28 コアの CPU を 8 個使用し合計 224 コアのものである．PCH もサーバ用のものでは，より多くのストレージやネットワークを接続できる．

2.4　オペレーティングシステムの構造

図 2.5 にオペレーティングシステムの構造を示す．オペレーティングシステムのカーネルは図 2.5 中央部分のソフトウェアである．ユーザプロセスはユーザモードで，カーネルはカーネルモードで実行される．

2.4.1　カーネルの構成

図 2.5 に示すように，カーネルは以下のようなモジュールから構成される．

1. 割り込みハンドラ
   割込みが発生した時に自動的に実行される割込み処理ルーチンである．割込みが発生した原因を判断し，必要なモジュールを呼出す．例えば，タイマーからの割込みならタイマーのデバイスドライバを呼出す．
2. ディスパッチャ
   カーネルの処理が終了した時，実行可能なプロセスの中から一つを選んで実行を再開させる．
3. コア
   割込みハンドラとディスパッチャを含むコアは，資源の仮想化を行うために必ずカーネルモードで実行される必要がある部分である．
4. サービスモジュール
   サービスモジュールは，ハードウェアを抽象化した便利なコンピュータをユーザ・プロセスに提供するためのプログラムである．

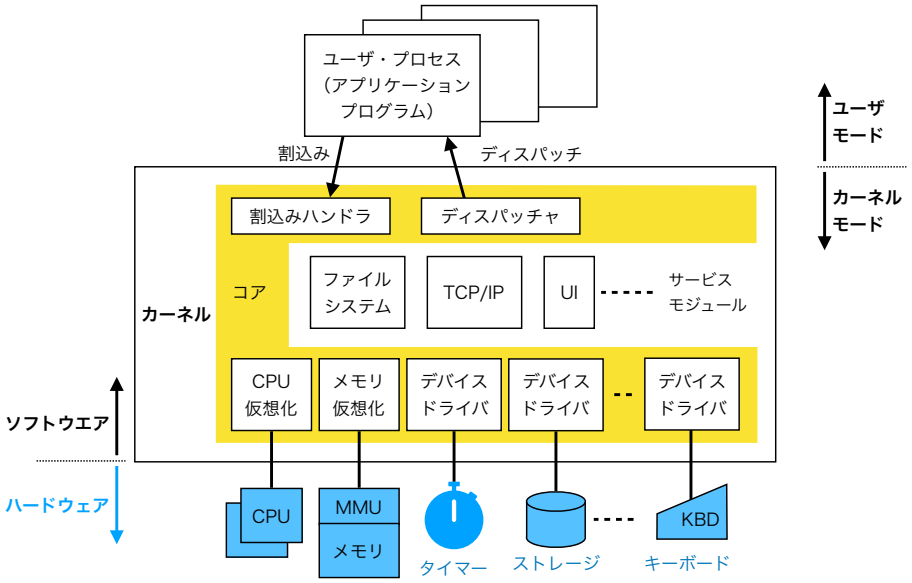


図 2.5　オペレーティングシステムの構造

2.4.2　カーネルの動作概要

通常，コンピュータはユーザ・プロセスを実行し目的の仕事をしている．何かイベントが発生すると割込みにより CPU に通知される．CPU はカーネルモードに切り替わり割込みハンドラに制御を移す．CPU がユーザ・プロセスの実行からカーネルの実行に移行するのは，**割込みが発生した時だけ**である．

割込み原因

カーネルへ実行を移すには割込みを発生する以外に方法がない．割込みが発生する原因には以下のようなものがある．システムコール以外はユーザ・プロセスが意図しない間に発生する．

1. I/O 完了・タイマー
   ホストコントローラやネットワークアダプタ，タイマーのようなハードウェアが，コマンドの実行完了等を CPU に知らせるために発生する．
2. システムコール
   ユーザ・プロセスは，割込みを発生する特殊な機械語命令である SVC（Supervisor Call）命令を用いてシステムコールを発行する[*3]．カーネルは SVC 命令実行時の CPU レジスタの値などによりシステムコールの種類やパラメータを知ることができる．
3. 保護違反
   ユーザ・プロセスがユーザ・モードでは実行が許可されない命令を実行したり，アクセスが許可されないメモリ領域をアクセスした場合に発生する．
4. ソフトウェアのエラー
   ユーザ・プロセス実行中に計算でオーバーフローが発生したような時に発生する．
5. ハードウェアのエラー
   ハードウェアの故障や電源の異常を検知した時に発生する．

割込み発生時のカーネルの動作

割込みが発生するとカーネル・モードに切り換わり割込みハンドラに制御が移る．その後，カーネル内では以下のような手順で処理がされる．

[*3] CPU によっては TRAP 命令，INT 命令と呼ばれることもある．

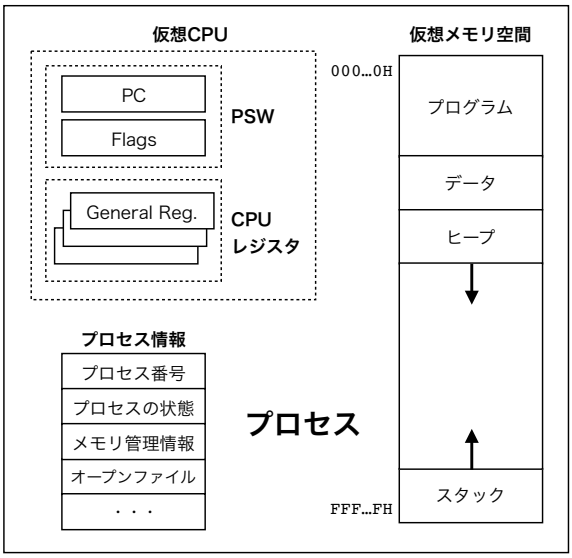


図 2.6　プロセスの構造

1. 割込みハンドラは後でプロセスの実行を再開できるように，プロセスの CPU の状態（**コンテキスト**）を保存する．
2. 割込みハンドラは割込み原因を調べ，原因に応じたカーネル内のサービスモジュールやデバイスドライバに制御を渡す．例えばファイル操作のシステムコールならファイルシステムへ制御を渡す．
3. サービスモジュールやデバイスドライバの処理が終了したらディスパッチャに制御が渡される．ディスパッチャは実行可能なプロセスの一つを選び，コンテキストを復旧しプロセスの実行を再開させる．

2.4.3　プロセスの構造

図 2.5 のユーザ・プロセス部分を詳しく描いたものを図 2.6 に示す．プロセスを構成する各部を以下で説明する．

1. 仮想 CPU
   CPU を仮想化しプロセス毎に CPU が存在するように見せることで，マルチプログラミングを可能にする．プロセスが CPU を使用する時間を区切り，次々に切替える時分割多重により CPU の仮想化は達成される．他のプロセスが CPU を使用している間に，プロセスのコンテキストを保存する領域を仮想 CPU と呼ぶことにする．ハードウェアの実 CPU に対応して PSW と CPU レジスタの保存先が必要である．前の節で説明したように，プロセスからカーネルに制御が移る時にプロセスのコンテキストを保存する．プロセス実行時にはコンテキストが実 CPU にロードされる．
2. 仮想メモリ空間
   メモリを仮想化しプロセス毎に専用のメモリ空間が存在するように見せかける．実現方法は第？章の「メモリ管理」で詳しく学ぶ．仮想メモリ空間は次の部分から構成される．
   (a) プログラム
   機械語プログラムがここに配置される．C 言語では関数の実行文（式文，if 文，for 文，while 文など）が翻訳された機械語が該当する．
   (b) データ
   プログラムの変数部分がここに配置される．C 言語ではグローバル変数が該当する．
   (c) ヒープ
   プログラム実行時に動的に拡大される領域である．C 言語では `malloc()` 関数はヒープに新しい領域を確保する．`malloc()` 関数が使用される度にヒープ領域は後ろに向かって拡大していく．
   (d) スタック

プログラム実行時にメモリ空間の最後から前に向かって伸びて行く領域である．サブルーチン・コール時に戻りアドレスを保存したり，C 言語のローカル変数や関数引数を置いたりするために使用される．

3. プロセス情報
   名前にあたる「プロセス番号」，実行中／実行可能／スリープのどの状態なのか表す「プロセスの状態」，使用しているメモリの大きさ等を表す「メモリ管理情報」，CPU を使用した時間を表す「CPU 時間」等の情報のことである[*4]．
   その他に，プロセスが現在オープンしているファイルに関する情報や，親プロセス，子プロセス，シグナルハンドラの登録状況，プロセスの優先度など，様々な情報がここに記録される．

2.5 カーネルの構成方式

カーネルが動作不良を起こすと，実行中の全てのユーザ・プロセスを巻き込んでシステムが停止したりするので，カーネルには非常に高い信頼性が要求される．しかし，カーネルは非常に大きなプログラムになりがちであり[*5]．高い信頼性を確保するにはカーネルの構成方法に工夫が必要である．

2.5.1 単層カーネル（モノリシック・カーネル）

最も一般的な構成方法である．図 2.5 のカーネルは単層カーネルの例になっている．カーネル内の全てのモジュールがリンクされ，一つのプログラムになる．カーネル内でモジュールの呼出しは CALL 機械語命令を用いて行うので効率が良い．

しかし，モジュール同士が密にリンクされているので，モジュール間で情報の隠蔽がし難くバグが入りやすい．また，全てのモジュールがカーネル・モード実行されるので，一つのモジュールのバグが致命的な結果を引き起こす．Linux や FreeBSD は，この方式のカーネルを持つ．

2.5.2 マイクロカーネル（micro-kernel）

図 2.5 の「コア」からデバイスドライバを取り除き[*6]，カーネル（マイクロカーネル）とし構成する方式である．図 2.7 にマイクロカーネル方式の概要を示す．カーネル・モードで実行されるのはマイクロカーネルだけである．

サービスモジュールはカーネルから独立したサーバ・プロセスとし，権限の低いユーザ・モードで実行される．ユーザ・プロセスは，マイクロカーネルが提供する IPC （プロセス間通信：Inter-Process Communication）を用いて，サーバ・プロセスにサービスを要求する．サーバ・プロセス同士，サーバ・プロセスとデバイスドライバ・プロセスも IPC を用いて通信する．

デバイスドライバは I/O ポートにアクセスするのでカーネル・モードで実行される必要があると考えられるが，I/O ポートへのアクセスをマイクロカーネルのシステムコールに置換えることで，デバイスドライバもユーザ・プロセスとして実装することが可能である．この場合は，デバイスドライバがアクセスしても良い I/O アドレスの範囲内かどうか，マイクロカーネルがチェックすることが可能である．

マイクロカーネル方式は，サービスモジュールやデバイスドライバが権限の低いプロセスとして実行されるので，これらのバグでシステム全体が停止する危険が低い。また，サービスモジュールやデバイスドライバ毎に独立したプログラムになりモジュール化が徹底しやすいので，巨大な単一プログラムであるモノリシックカーネルと比較してバグが発生しにくい．信頼性の高いオペレーティングシステムを構成するために有利である．しかし，IPC とプロセス切り換えのオーバヘッドが大きいため性能が低くなる．**多くの場合，信頼性と性能はトレードオフの関係にある．**

2.6 もう一つの仮想マシン

1.1 で述べたように，オペレーティングシステムは抽象化され便利な拡張マシン（仮想マシン）を，必要な数だけ提供する．ここで述べた仮想マシンは，単一ユーザ・プロセスの実行環境のことである．同じ「仮想マシン」と

---

*4 これらは UNIX の ps コマンドで表示することができる．

*5 Linux や Windows のカーネルのソースコードは 500 万行にもなる [34]．

*6 タイマーのデバイスドライバは CPU の仮想化に必要なので，マイクロカーネルに残す．

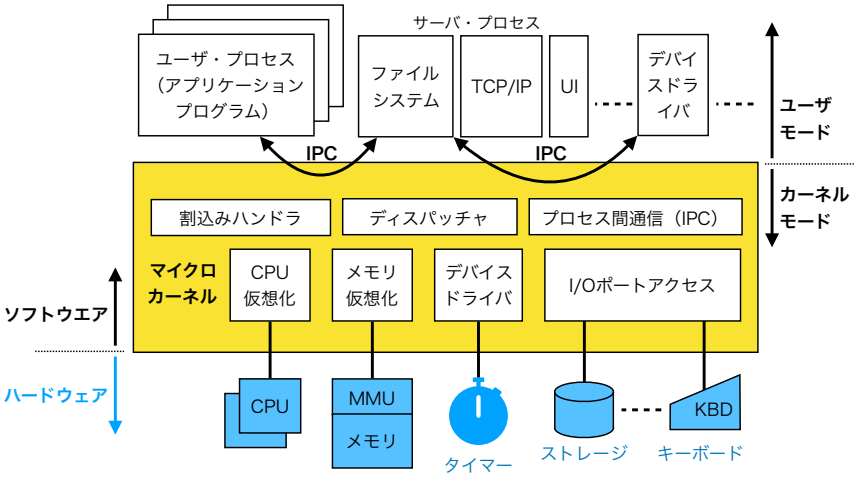


図 2.7　マイクロカーネル方式

言う用語が，オペレーティングシステムを実行することが可能な，よりハードウェアを忠実に再現した仮想マシンを指す場合もある．ここでは，一台のコンピュータ上で複数のオペレーティングシステムを実行可能なもう一つの仮想マシンについて紹介する．

2.6.1　Type 2 ハイパーバイザ

例えば，Mac を使用している人が Windows でしか動作しないアプリケーションを使用する場合を想像してしてみる[*7]．予め Mac のハードディスクに macOS とは別に Windows もインストールしておき，電源投入時に macOS と Windows を選んでブートする方法もあるが，オペレーティングシステムを切換える度にコンピュータを再起動するのは不便である．また，macOS のアプリケーションと Windows のアプリケーションを同時に実行したい場合もある．

そこで，図 2.8 に示すような「Type 2 ハイパーバイザ（Type 2 Hypervisor）」を用いた仮想化が用いられる．ハイパーバイザはホスト・オペレーティングシステムの一つのユーザプロセスとして実行され，コンピュータ一台の機能をエミュレーションする．ハイパーバイザがエミュレーションするコンピュータの中で，ゲスト・オペレーティングシステムが稼働する．エミュレーションはソフトウェアだけで完全に行うのではなく[*8]，ハードウェアの支援を受けて行うので高速に行うことができる[*9]．Type 2　ハイパーバイザとして有名は製品は，VMware Workstation，VMware Fusion，VirtualBox 等を挙げることができる．

2.6.2　Type 1 ハイパーバイザ

メインフレーム上で 1960 年代から使用されている方式である．現在では PC サーバの仮想化にも使用されている．Type 1 ハイパーバイザはホスト・オペレーティングシステム無しにハードウェア上で直接実行される．Type 1 ハイパーバイザとして有名な製品は，IBM z/VM，VMware vSphere，Xen，Hyper-V 等を挙げることができる．

サーバ向けの製品が主流であり，例えば VMware vSphere は実行中のゲストを他の物理サーバに移動する等，非常に高度な機能を持っている [36]．一台のサーバ上に効率よく多数の仮想マシンを動かすことができる．徳山高専情報電子工学科のパソコン室でも，2 台のサーバ上に 50 台の仮想デスクトップマシンを動かしていたことがある．

---

*7 徳山高専情報電子工学科のパソコン室の Mac では，Windows や Linux でしか動作しない Xilinx ISE WebPACK を使用するために，Type2 ハイパーバイザ（VirtualBox）上で Windows を動かしている．

*8 完全にソフトウェアで行う場合もある．

*9 詳しくは参考文献 [35]

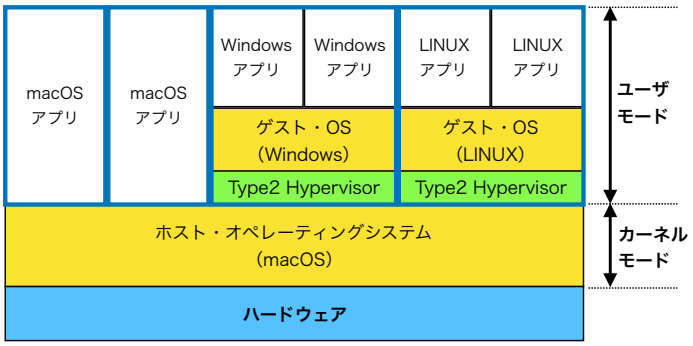


図 2.8　Type 2 ハイパーバイザ

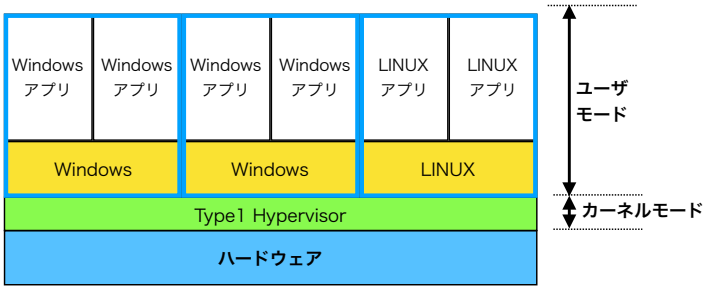


図 2.9　Type 1 ハイパーバイザ

2.6.3　仮想アプライアンス

ゲスト・オペレーティングシステムとアプリケーションまでインストールし，すぐに使用できる状態で配布される仮想マシンである．例えば，メールフィルタソフトをインストールした仮想マシンを入手しハイパーバイザで実行するだけですぐにメールフィルタリングが開始できる．

同じ手法で，すぐに使用できるパーソナルコンピュータ用のデスクトップ・オペレーティングシステムが配布されている場合もある．LINUX の一種である Ubuntu の場合，VirtulBox ですぐに実行できるディスクイメージがダウンロードできる [37]．この手法は，ソフトウェア流通の新しい方式になりつつある．

第 3 章

CPU の仮想化

これまでに述べたように，オペレーティングシステムはハードウェアを抽象化した，使いやすい拡張マシン（仮想マシン）を必要な数だけ提供する．数に限りがある必要な数だけあるように見せるために仮想化が行われる．CPU も仮想化し，各プロセスが自分専用の CPU を持っているように見せかける．

3.1 時分割多重

CPU を仮想化するためには時分割多重を用いる．ハードウェアである実 CPU の数は限られているので，時間を区切って実 CPU を使用するプロセスを次々に切換えていく．図 3.1 に CPU 仮想化の原理を示す．

図で実 CPU は図 2.2 のような構造をもつハードウェアである．プロセスの構造は図 2.6 に示した通りであり，仮想 CPU を含んでいる．実 CPU が短時間（例えば 10ms 毎）に次々と実行するプロセスを切換えていくことで，複数のプロセスが夫々に専用の CPU を持ち並行して実行されているように見せかける．

その際，切換える直前の CPU のコンテキストをプロセスの仮想 CPU 領域に保存する．次に新しく実行するプロセスの仮想 CPU 領域にからコンテキストを実 CPU に読込み，新しいプロセスの実行を再開する．実 CPU にコンテキストを読込んで実行を再開することを**ディスパッチ**と言う．ディスパッチを行うプログラムを**ディスパッチャ**と呼ぶ．図 2.5 にもディスパッチャは描かれていた．

3.2 プロセスの状態

プロセスはキーボド等の入出力装置からの入力を待つ状態になったり，時間が経過するのを待つ状態になったりする．**待ち**状態のプロセスには CPU を割当てる必要がない．このようにプロセスは幾つかの状態を持っている．プロセスの状態は UNIX では ps コマンドで確認できる．プロセスを模式的に示した図 2.6 では，「プロセス情報」

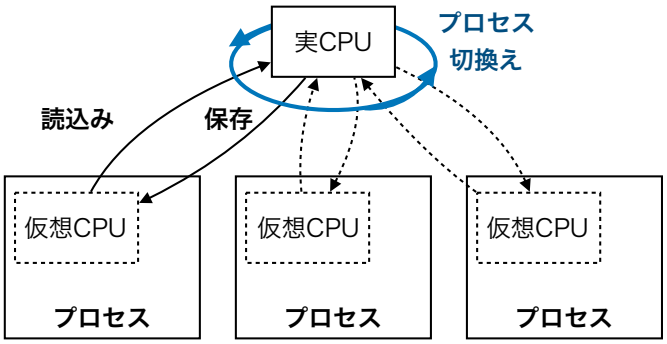


図 3.1　時分割多重による CPU の仮想化

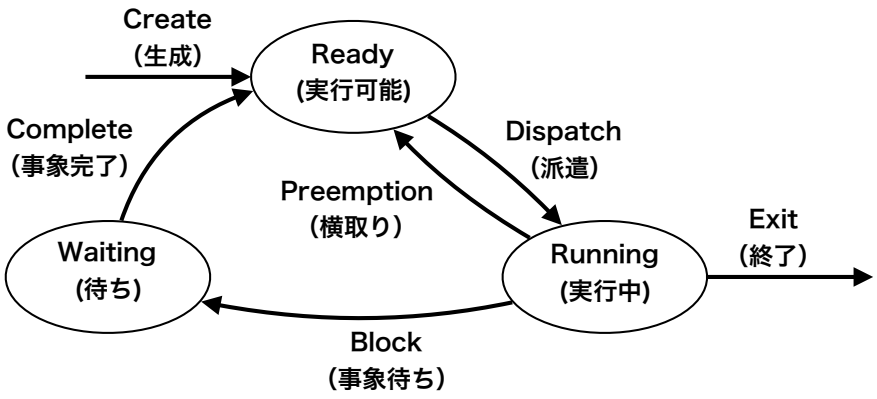


図 3.2　プロセスの状態遷移

の「プロセスの状態」のことである．次にプロセスの最も基本的な三つの状態を説明する．

- Ready（実行可能）
  CPU を割当てれば実行を開始できる状態のことである．プロセスは CPU が割当てられるのを待っている．
- Running（実行中）
  CPU が割当てられて実行中の状態のことである．CPU の数より多くのプロセスが同時に Running になることはできない．
- Wating（待ち）
  シグナルの到着や入出力の完了等の事象を待っている状態のことである．

図 3.2 にプロセスの状態遷移図を示す．図に示された六つの状態遷移の意味は以下の通りである．

1. Create（生成）
   新しいプロセスが生成されると Ready 状態になる．
2. Dispatch（派遣）
   Ready 状態のプロセスは CPU が割当てられ Running に遷移する．
3. Preemption（横取り）
   Running のプロセスは，決められた時間（クオンタムタイム）を使い切ったとき，より優先度の高いプロセスが Ready になったとき等に，CPU を取り上げられて Ready 状態に遷移する．
4. Block（事象待ち）
   Running のプロセスが，システムコールを発行して自ら Waiting 状態に遷移することがある．例えば入出力システムコール（`open()`，`read()`，`write()`，`close()` 等）や，シグナル待ちシステムコール（`pause()`，`wait()`，`sleep()` 等）を発行した場合である．また，他のプロセスからシグナルを受信したり，優先度の高いプロセスが実行可能になった場合も，Waiting 状態に遷移することがある．
5. Complete（事象完了）
   入出力の完了，シグナルの発生等によりシステムコールが完了すると Ready 状態に遷移する．
6. Exit（終了）
   プロセスが自らシステムコール（`exit()`）を用いて終了する場合，プロセスがシグナルを受ける等して終了させられる場合にこの遷移が起こる．

3.3　プロセスの切換え

プロセスは CPU が割当てられ状態が Running になって実行される．Running のプロセスは Waiting または Ready に状態遷移する際に，CPU を取り上げられる．

3.3.1 切換えの原因

プロセスの状態遷移が起こる原因を以下にまとめ直す．プロセスが以下の二つの原因により状態遷移を起こすことにより，実行を開始したり待ち状態になったりする．

1. イベント
   プロセスが自ら「システムコールを発行」することで Block 遷移をすることがある．入出力待ちのプロセスが「入出力完了」により Complete 遷移をすることがある．時間待ちのプロセスが「時刻になった」時に Complete 遷移をすることがある．他のプロセスからの「干渉[*1]を受け」Block 遷移や Complete 遷移をすることがある．
2. タイムスライシング
   Running のプロセスが連続して実行しても良い時間（クオンタムタイム）を使い切り，Preemption 遷移をする．Preemption 遷移した後，もしも優先度の高い他のプロセスが Ready ならそのプロセスに切替わる．優先度の高いプロセスに Ready のものが無ければ Dispatch 遷移し，再度，Running に状態になる．

[*1] 干渉には，より優先順位の高いプロセスが実行可能になった，別のプロセスからシグナル等を受取った等がある．

参考文献

[1] ウキペディア，OS/360，https://ja.wikipedia.org/wiki/OS/360（2017.10.03 閲覧）
[2] ウキペディア，MVS，https://ja.wikipedia.org/wiki/Multiple_Virtual_Storage（2017.10.03 閲覧）
[3] ウキペディア，OS/390，https://ja.wikipedia.org/wiki/OS/390（2017.10.03 閲覧）
[4] ウキペディア，z/OS，https://ja.wikipedia.org/wiki/Z/OS（2017.10.03 閲覧）
[5] ウキペディア，UNIX（「UNIX および UNIX 系システムの系統図」を含む），https://ja.wikipedia.org/wiki/UNIX（2017.10.03 閲覧）
[6] ウキペディア，Solaris，https://ja.wikipedia.org/wiki/Solaris（2017.10.03 閲覧）
[7] ウキペディア，AIX，https://ja.wikipedia.org/wiki/AIX（2017.10.03 閲覧）
[8] ウキペディア，Mach，https://ja.wikipedia.org/wiki/Mach（2017.10.03 閲覧）
[9] ウキペディア，BSD の子孫，https://ja.wikipedia.org/wiki/BSD%E3%81%AE%E5%AD%90%E5%AD%AB（2017.10.03 閲覧）
[10] ウキペディア，BSD，https://ja.wikipedia.org/wiki/BSD（2017.10.04 閲覧）
[11] ウキペディア，386BSD，https://ja.wikipedia.org/wiki/386BSD（2017.10.04 閲覧）
[12] ウキペディア，FreeBSD，https://ja.wikipedia.org/wiki/FreeBSD（2017.10.03 閲覧）
[13] ウキペディア，FreeNAS，https://ja.wikipedia.org/wiki/FreeNAS（2017.10.03 閲覧）
[14] ウキペディア，NEXTSTEP，https://ja.wikipedia.org/wiki/NEXTSTEP（2017.10.03 閲覧）
[15] ウキペディア，Classic Mac OS，https://ja.wikipedia.org/wiki/Classic_Mac_OS（2017.10.03 閲覧）
[16] ウキペディア，ダイナブック，https://ja.wikipedia.org/wiki/ダイナブック（2017.10.03 閲覧）
[17] ウキペディア，macOS，https://ja.wikipedia.org/wiki/MacOS（2017.10.03 閲覧）
[18] ウキペディア，iOS (アップル)，https://ja.wikipedia.org/wiki/IOS_(アップル)（2017.10.03 閲覧）
[19] ウキペディア，Linux，https://ja.wikipedia.org/wiki/Linux（2017.10.03 閲覧）
[20] ウキペディア，Andriod，https://ja.wikipedia.org/wiki/Android（2017.10.03 閲覧）
[21] ウキペディア，MS-DOS，https://ja.wikipedia.org/wiki/MS-DOS（2017.10.03 閲覧）
[22] ウキペディア，Microsoft Windows（「Windows ファミリー系統図」含む），https://ja.wikipedia.org/wiki/Microsoft_Windows（2017.10.03 閲覧）
[23] ウキペディア，IBM PC，https://ja.wikipedia.org/wiki/IBM_PC（2017.10.04 閲覧）
[24] ウキペディア，UNIX System V，https://ja.wikipedia.org/wiki/UNIX_System_V（2017.10.04 閲覧）
[25] 重村哲至，情報電子工学科電算機室における PC-UNIX の歴史，http://www2.tokuyama.ac.jp/giga/Sigemura/Public/IeNet/history.html（2017.10.03 閲覧）
[26] Linux kernel release 1.0，https://www.kernel.org/pub/linux/kernel/v1.0/linux-1.0.tar.gz（2017.10.04）
[27] Andrew S. Tanenbaum, Herbert Bos：“The Third Generaton(1965–1980):ICs and Multiproguramming”，Modern Operating Systems（4th Edition），pp.9-14，Pearson Education,Inc（2014）.
[28] Andrew S. Tanenbaum, Herbert Bos：“The Fourth Generation(1980–Present):Personal Computers”，Modern Operating Systems（4th Edition），pp.15–19，Pearson Education,Inc（2014）.
[29] Alan C. Kay：“A Personal Computer for Children of All Ages”，Proceeding ACM ’72 Proceedings of

the ACM annual conference - Volume 1 Article No 1 （1972）.

[30] アラン・ケイ：すべての年齢の「子供たち」のためのパーソナルコンピュータ，阿部和広，小学生からはじめるわくわくプログラミング，pp.130–141，日経 BP 社（2013）.

[31] アラン・ケイ：Dynabook とは何か？「すべての年齢の「子供たち」のためのパーソナルコンピュータ」の後日談，阿部和広，小学生からはじめるわくわくプログラミング，pp.142–149，日経 BP 社（2013）.

[32] 師尾　潤 他：スーパーコンピュータ「京」のオペレーティングシステム，http://img.jp.fujitsu.com/downloads/jp/jmag/vol63-3/paper07.pdf （2017.10.03 閲覧），富士通（2012）.

[33] Marshall Kirk McKusick, George V. Neville-Neil, Robert N. M. Watson：The Zettabyte Filesystem, The Design and Implementation of the FreeBSD Operating System Second Edition，Pearson Education,Inc（2015）.

[34] Andrew S. Tanenbaum, Herbert Bos：“INTRODUCTION”, Modern Operating Systems（4th Edition），pp.1-3，Pearson Education,Inc（2014）.

[35] Andrew S. Tanenbaum, Herbert Bos：“VIRTUALIZATION AND THE CLOUD”，Modern Operating Systems（4th Edition），pp.471-516，Pearson Education,Inc（2014）.

[36] ヴイエムウェア株式会社：“VMware 徹底入門 第 3 版”，廣済堂（2012）.

[37] 仮想ハードディスクイメージのダウンロード，https://www.ubuntulinux.jp/download/ja-remix-vhd（2017.10.19 閲覧），Ubuntu Japanese Team（2012）.

オペレーティングシステム Ver. 0.0.0

発行年月　２０１７年１０月　Ver.0.0.0
発　　行　独立行政法人国立高等専門学校機構
徳山工業高等専門学校
情報電子工学科　重村哲至
〒745-8585 山口県周南市学園台
sigemura@tokuyama.ac.jp