［3］プロセス

定義 1.3（プロセス，タスク）
プロセッサで実行するプログラム，すなわち，動的にできるマシン命令列やそれらが使用するデータの集まりを**プロセス** (process) あるいは**タスク** (task) という．

注意　本書では，「マルチタスキング」（70ページの2.1.1項［2］を参照）以外のすべてで「プロセス」を用いる．

注意　「生成元となるプログラムが“ユーザプログラム”であるプロセス」を“ユーザプロセス”という．本書では，“ユーザプロセス”を単に“プロセス”という．

“プロセス”は，“実行前（静的）”にメインメモリに割り付け，“実行時（動的）”にメインメモリで保持しつつ，プロセッサで使用する．また，“プロセス”は実行制御対象における“論理的な単位”となる．

プロセッサでプロセスを実行するためには，プロセスをメインメモリにあらかじめ置き（**プロセス割り付け**という，実際については88ページの2.1.3項で詳述），それから，プロセッサとプロセスの対応付け（**プロセッサ割り付け**という）を行うことが必要となる．**プロセス割り付け**と**プロセッサ割り付け**とはOSの主要な機能であり，これらを併せて**プロセス管理**という．プロセスとプロセッサとの関係については，2.1.1項［1］（69ページ）で述べる．

［4］OSの管理機能（概要）

(A) プロセス管理機能（概要）

コンピュータシステムにおける主要なハードウェア装置（内部装置，本体）である**プロセッサ**に対するプロセス管理機能としては，“時間的管理”が主となる．具体的には，図1.6に示すように，「プロセッサの利用ないしは実行時間を数〜数十ミリ（14ページの脚注[*1]を参照）秒ごとに分割し（「分割した時間」を“時間スライス (time slice)”あるいは“クォンタム (quantum)”という），それらを順次（これも分割した）プロセスへ割り付けて実行する（**TSS** (Time Sharing System) という）」プロセッサ割り付け機能を管理する．

共用あるいは共有して動作する．OS（プログラム）はハードウェア装置上でのユーザプログラムの実行形態や共用形態を管理・制御する．

「ハードウェアの隠ぺいや仮想化」はOSの本質的な機能であり，その隠ぺいあるいは仮想化の方法や程度はOSごとに相異なる．したがって，ユーザプログラムの大半はある特定のOSの管理・制御下でしか動作しない．言い換えると，OS機能が同一であれば，そのOSによるハードウェアの隠ぺいあるいは仮想化方式も同一であるので，ハードウェア装置そのものやそれらで実現するハードウェア機構（“ハードウェア環境”という）が異なっていても，ユーザプログラムの大半は動作する．

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（31ページの参考1.5を参照）が相異なるハードウェア環境で動作することを「**オブジェクト互換性**（がある）」という．

（オブジェクトプログラム形式の）ユーザプログラムのオブジェクト互換性は同一あるいは同種のOSの下で実現するのが普通であり，この場合，それらのユーザプログラムは「**OS レベルで互換**（である）」という．「ユーザプログラムからハードウェアを隠ぺいする」OS機能は「OSレベルで互換なユーザプログラムの実現」あるいは「ユーザプログラムに対するオブジェクト互換性の実現」といえる．

身近な実例として，たとえば，WindowsとUNIXとMac OSはメーカも異なる異種のOSであるので，「それぞれのOSの下で動作するユーザプログラムは他のOSの下では動作しない，すなわち，互換性がない（非互換である）」のが普通である．また，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の場合，同種のOSでも版（バージョン(version)あるいはリビジョン(revision)）が異なるだけで動作しない，すなわち，互換性がない（非互換である）ことがある．

［2］ OS の起動

身近なパソコン用OSを例にとって，OSの“起動”（**ブート**(boot)という）過程を見ておこう．図1.12に，OSの“起動（ブート）”におけるハードウェア

なわち，ハードウェア装置からOSへの業務依頼を，統一して実現する仕組みが“割り込み”である．（「割り込みの必要性」については，55ページの［3］および62ページの1.4.2項［6］で詳述）

［2］マシン命令の実行と割り込み

コンピュータシステムでは，「ハードウェア（特に，プロセッサ）はコンパイラ（5ページの1.1.1項［5］を参照）などの言語処理プログラムが実行前すなわち静的にソースプログラムから生成したマシン命令列（オブジェクトプログラム，31ページの参考1.5を参照）を逐次実行（後の注意を参照）する」のが原則である．オブジェクトプログラムはマシン命令（列）の実行順序（“実行フロー(flow)”という）をあらかじめ規定している．したがって，オブジェクトプログラムを解析すれば，マシン命令（列）の実行フローの概略は分かる．

注意　逐次実行の原則は“並び順”である．“並び順”は実際には「メインメモリにおける格納順」である．「マシン命令の実行順序を並び順実行とする」原則によって，マシン命令中に次に実行する命令（“次命令”という）のアドレスを明示することが不要となる．
並び順実行ではなく特定アドレスのマシン命令を実行する（“分岐”あるいは“ジャンプ(jump)”という）ときには，マシン命令中にその特定アドレスを次命令アドレスとして明示する分岐命令（“ジャンプ命令”ともいう）を使用する．しかし，分岐命令も実行前すなわち静的に生成するオブジェクトプログラムに埋め込むので，実行するマシン命令（列）や実行フローはあらかじめ決まっている．「条件による分岐」（“条件分岐”という）の有無，すなわち，「どのマシン命令を実行するか」については，実行時すなわち動的に決まるので，実行前には不明である．しかし，「分岐する場合の実行フロー」および「分岐しない場合の実行フロー」については，どちらも実行前に判明している．

言い換えると，ユーザは，コンピュータシステムに実行させたいあるいはコンピュータシステムで実行することのすべての場合をあらかじめ予測してソースプログラムとして記述（“プログラミング(programming)”である）し，さらには，オブジェクトプログラムとしてコンピュータシステムに搭載しなければならない．しかし，コンピュータシステムでのプログラムの実行時すなわち動的には，「予測できない事態」すなわち「不測の事態」がしばしば発生する．「不測の事態」は実行時に動的に発生するので，実行前にあらかじめ対処することができない．したがって，「不測の事態」や「不測の事象」への対処や処理の

「方針と機構の分離によるプロセススイッチ機能の設計」において，プロセススイッチ機能は『“ プロセススケジューリング方針 ” にしたがって動作する “ プロセスディスパッチ機構 ”』となる．「プロセススイッチ機能の設計」では，『“ 方針 ” としての “ プロセススケジューリング ” 機能の設計」と『“ 機構 ” としての “ プロセスディスパッチ ” 機能の設計」とを独立させて別々に行う．

［3］プロセススイッチとプロセスコンテクストスイッチ

プロセッサ（時間）は唯一であるので，どの時点でも，唯一プロセッサ上でOSまたはユーザプロセスのどちらかが実行されている．したがって，2.1.2項［4］（79ページ）で述べたように，“ 割り込み ” をきっかけとするユーザプロセス間の**プロセススイッチ**は，実際には，

1. **割り込み処理**：「ユーザプロセス→OS（カーネル）」スイッチ（切り替え）；
2. **プロセススケジューリング**（狭義）：スイッチ先ユーザプロセスの選定；
3. **プロセスディスパッチ**：「OS（カーネル）→ユーザプロセス」スイッチ（切り替え）；

のプロセス管理手順を，この順で，完遂する．1および3の「ユーザプロセス⇔OS（カーネル）」スイッチを “ 狭義のプロセススイッチ ” という．

一方，2.1.3項［2］（90ページ）や2.1.3項［4］（94ページ）で述べたように，“ プロセススイッチ ” は，具体的には，「“PCB” というソフトウェアプロセスコンテクスト，および，“PSW” というハードウェアプロセスコンテクストの切り替え（どちらもあるいは併せて**プロセスコンテクストスイッチ**という）」である．したがって，実際には，“ プロセススイッチ機構 ” は “ プロセスコンテクストスイッチ機構 ” すなわち「PCBおよびPSWの切り替え機構」である．

この “ プロセススイッチ ” の実際的な手順を『“ プロセスコンテクストスイッチ ” 機構の動作」の観点でまとめると，次のようになる．（図2.13を参照）

(A) プロセススイッチ手順1の “ 割り込み処理 ” における**プロセッサ内PSWのメインメモリ内PCBへの退避**：
スイッチ元のすなわち現在実行中（状態）のユーザプロセスのPSW（プロセッサ内）を旧プロセスコンテクストとしての当該ユーザプロセスのPCB（メインメモリ内）に退避する．（64ページの1.4.3項［1］で述べている割

り込み処理手順における手順6の「ソフトウェア状態の退避」）

(B) プロセススイッチ手順3の“ プロセスディスパッチ ”における**メインメモリ内PCBのプロセッサ内PSWへの回復**：
スイッチ先のユーザプロセスの新プロセスコンテクストとしてのPCB（メインメモリ内）からスイッチ先の当該ユーザプロセスのPSW（プロセッサ内）を回復する．（64ページの1.4.3項［1］で述べている割り込み処理手順における手順8の「ソフトウェア状態の回復」）

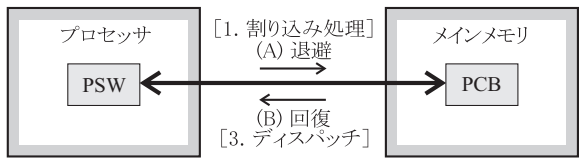

図 2.13　プロセスコンテクストスイッチ機構の動作

PSW全体も一種のプロセッサ状態である．また，プロセスコンテクストであるPCBはメインメモリ上に置く．したがって，実際の「PCBのスイッチすなわち退避と回復」は「スイッチ前後の（旧新）PCBを実体とする“ PSW（プロセッサ内）⇔PCB（メインメモリ上）間転送 ”」となる．

この(A)(B)を併せた“ プロセスコンテクストスイッチ ”において，(A)での退避先のPCB（旧プロセスコンテクスト）と(B)での回復元のPCB（新プロセスコンテクスト）とが，(a) 同じであれば，中断しているユーザプロセスの“再開”;(b) 異なれば，相異なるユーザプロセス間での“ プロセススイッチ ”;となる．(a)(b)のどちらになるかは，(1) プロセススイッチ手順1を引き起こした割り込み要因が何であるか；(2) プロセススイッチ手順2でのプロセススケジューリングの結果；によって決まる．（73ページの2.1.2項および103ページの2.1.5項を参照）

［4］ プロセスの生成と消去

OSが行う実際的なプロセスの**生成**手順は次の通りである．

1. **プロセス領域の割り付け**：メインメモリ上に，プロセスの実体の格納用領域である“ プロセス領域 ” を確保する．
2. **PCB の割り付け**：メインメモリ上に，“PCB” を確保する．
3. **PCB の設定**：“PCB” に情報や内容を設定する．
4. **PCB リストの更新**：実行可能キューとして構成してあるPCBﾘｽﾄ〔90ページの2.1.3項［2］を参照）の最後尾に “PCB” を挿入する．
5. **プロセス状態の設定**：プロセス（状態）管理として，当該プロセス状態を“ 実行可能 ” にする．

したがって，生成したプロセスは，当初，“ 実行可能 ” 状態である．

これら1〜5の“ プロセス生成 ”は，実際には，2.1.2項［4］（79ページ）で概要をまとめたプロセス管理手順1の割り込み処理と手順2のプロセススケジューリングの間で，OS（カーネル）が行う．

一方，OSが行う実際的なプロセスの**消去**手順は次の通りである．正常終了する場合と異常終了（“ 中途消滅 ” といえる）する場合とで手順がやや異なる．“ 実行中 ”, “ 実行可能 ”, “ 待ち ” のどの状態のプロセスも消去（消滅，削除）できる．

［正常終了］

1. **プロセス領域の解放**：確保し使用している“ プロセス領域 ” を空き（未使用）領域としてOSに返却する．
2. **PCB の解放**および**PCB リストの更新**：実行可能キューあるいは待ちキューとして構成してあるPCBリストから “PCB” を削除する．
3. **プロセス状態の設定**：プロセス（状態）管理として，当該プロセス（状態）を“ 消去（削除）” する．

［異常終了（中途消滅）］

1. **PCB およびプロセス領域の更新**：“PCB” や “ プロセス領域 ” からハードウェア資源に関する情報を削除する．
2. **プロセス領域の解放**：確保し使用している“ プロセス領域 ” を空き（未使用）領域としてOSに返却する．

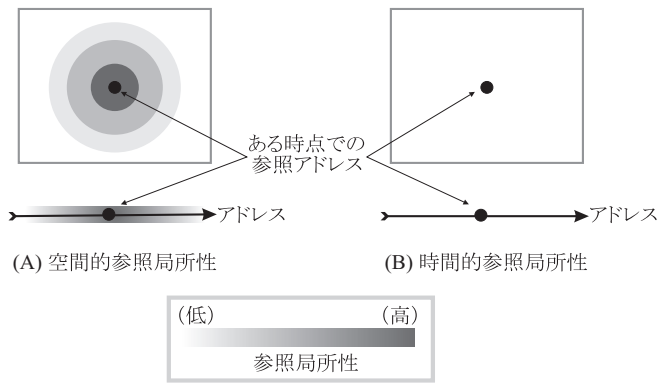

図 3.6　参照局所性

ほとんどの命令やデータは「空間的参照局所性と時間的参照局所性とを併示する」性質をもつ．また，実行するプログラムや適用する問題ごとに“参照局所性”の傾向や性質は相違するのが普通である．

コンピュータシステムは標準的にメモリ階層を備えているので，「OSによるメモリ管理機能」の設計では，「メモリ階層の管理における“参照局所性”の活用」が要点となる．すなわち，OSは，隣接するメモリ階層の特性を利用すれば，参照局所性が高いプログラムやプロセスを，当該メモリ階層において空間的または時間的に効率良く管理し処理できるようになり，結果として，当該メモリ階層そのものの機能を改善できる．

「OSのメモリ管理機能において，“メモリ階層”と“参照局所性”を活用する」代表例が本節（特に，3.2.2〜3.2.4項）で詳述する“仮想メモリ”である．

［4］ メモリ階層によるメインメモリ性能の改善 —代表例：仮想メモリ—

“仮想メモリ”は「メインメモリとファイル装置の隣接する2種類のメモリ階層を利用してメインメモリ性能を“空間的”に改善する」ことを目標する．すなわち，“仮想メモリ”は「プロセッサから見えるメインメモリの“アドレス空間”（後の定義3.3を参照）」という空間的制限を撤廃する．

ク（**セグメント** (segment) という）でマッピングする．

セグメントは，“ プログラム（プロセス）” や “ データブロック ” という「論理的な意味」を考慮した論理的な単位で構成するので，そのサイズは “ 可変 ” である．3.1.2 項［2］（141ページ）で述べた “ 可変長領域割り付け ” の実例である．（161ページの［4］で詳述）

(C) **ページセグメンテーション** (paged segmentation)：(A) と (B) を融合したマッピングである．（163ページの［5］で詳述）

［3］ページング

図 3.14 に示すように，**ページング**では，仮想アドレス空間も実アドレス空間も “ 一定 ” かつ “ 固定 ” 長の**ページ**サイズで分割する．そして，ページ単位で，仮想アドレス空間上のページ（“ 仮想ページ ” または “ 論理ページ ” という）と実アドレス空間上のページ（“ 実ページ ” または “ 物理ページ ” という）とをマッピングする．したがって，仮想アドレス空間には仮想ページごとに “ 仮想ページ番号 ” を，実アドレス空間には実ページごとに “ 実ページ番号 ” を，それぞれ振る．「仮想ページと実ページの “ マッピング ” 」は “ ページテーブル ” と呼ぶ**アドレス変換テーブル**に記述しておく．「仮想ページから実ページへの “ アドレス変換 ” 」は「ページテーブルを（普通に）引く」操作となる．

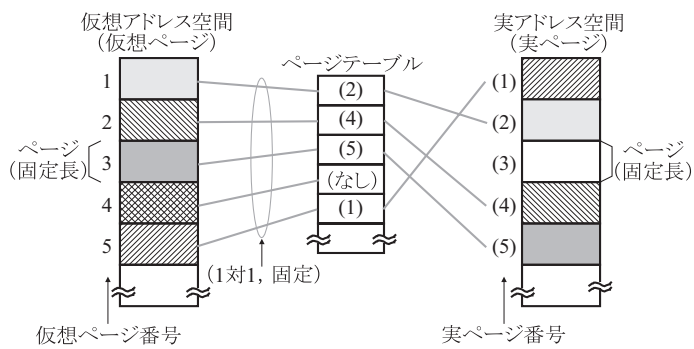

図 3.14　ページングによるマッピング（例）

生しない.

(3) セグメントサイズは“可変”かつ“自由”なので，メインメモリ（実メモリ）やファイル装置（バックアップメモリ）での領域管理，および，実メモリと仮想メモリとのマッピングの管理は非常に複雑である．また，メインメモリ（実メモリ）での外部フラグメンテーション（142ページの参考3.4を参照）が発生しやすいので，それが過ぎると，メインメモリの利用効率は格段に悪くなる．

(4) メインメモリサイズより大きな仮想アドレス空間をセグメントとしてマッピングできないので，その場合に，長所の(1)(2)を喪失してしまう．

参考 3.11（ライブラリ）　共用できる形式で保持しているあるいは格納してあるオブジェクトプログラム（31ページの参考1.5を参照）（群）を**ライブラリ** (library) という．

---

［5］ページセグメンテーション

図3.16に示すように，**ページセグメンテーション**では，

1. 全体では，**セグメント**単位（**セグメンテーション**）；
2. 各セグメント内では，**ページ**単位（**ページング**）；

の2種類のマッピングをこの順で適用する．

“ページセグメンテーション”では，仮想アドレス空間も実アドレス空間も一定かつ固定長のページサイズで分割する．

1. まず，“**セグメント**単位”で，仮想セグメントと実セグメントとをマッピングする．このとき，セグメントはひとまとまりであるので，その全体サイズは可変かつ自由（ただし，ページ単位で）である．仮想セグメントと実セグメントのマッピングは“セグメントテーブル”に記述してある．
2. 次に，“**ページ**単位”で，仮想セグメントと実セグメントのそれぞれのセグメント内のページどうし（仮想ページと実ページ）を　　マッピングする．仮想ページと実ページのマッピングは各実セグメントごとに備

図 3.22　逐次アクセスファイル

- “ 読み出し ” と “ 書き込み ” に限ると，ファイル操作は単純；

である．

一方，短所は，

- “ 読み出し ” と “ 書き込み ” 以外の，たとえば “ 生成（追加）” や “（一部分の）削除 ” などの，ファイル操作は面倒；
- ファイル途中でのブロックの挿入や削除が困難；

である．

(B) **直接アクセスファイル**（図3.23を参照）

「番号付けしたファイルブロックの集まり」という “ ファイル構造 ”（論理構造）である．直接アクセスファイルに対しては，ブロック番号（**相対ブロック番号**，後の定義3.11 を参照）を指定すれば，任意のファイルブロックに “ 直接アクセス ”（“ ランダムアクセス ”，前の［1］で述べた (B) のファイルアクセス方式）できる．**ランダムアクセスファイル** (random access file) ともいう．

定義 3.11（相対ブロック番号）
ファイルアクセス方式（ファイル構造を含む）における「当該ブロックの『先頭ブロックからの相対的な位置』(“ オフセット (offset) ” という)」を**相対ブロック番号**という．ファイルブロックへのアクセス時に，相対ブロック番号によって当該ブロックを指定する．

“ ハードディスクドライブ装置 ” を代表例とするファイル装置では，ファイル装置上のどのファイルブロックにも直接アクセスできる．この特徴によって，これらのファイル装置を**直接アクセスファイル装置**あるいは**ランダムアクセ**

3. **リスト**：ファイル情報を一覧表示する．“ ディレクトリ ”そのものが“ リスト（一覧表）”であるので，「“ ディレクトリ ”をそのまま，あるいは，要求によっては並び替えて，出力する」だけで十分である．

［3］ ディレクトリの構造 —代表例：木構造ディレクトリ—

“ ディレクトリの構造 ”すなわち“ ディレクトリの構成方式 ”について，UNIX や Windows などの現代の代表的な OS が採用している**木構造ディレクトリ**（“ 木構造 ” については，後の参考 3.13 を参照）を例にとって，説明する．

“ 木構造ディレクトリ ” は次のように構成する．（図 3.25を参照）

1. ディレクトリ全体を 1 個の “ 木構造 ” とする．
2. その木の唯一ルート（根）とすべてのノード（節）を “ディレクトリ” とする．その唯一ルートが**ルートディレクトリ** (root directory) である．
3. 各ディレクトリは「それが管理する部分木の “ ルート ディレクトリ ”」である．
4. あるディレクトリ，すなわち，その部分木のルート，に属する “ ファイル ” は，その部分木の葉として，置く．

木構造ディレクトリによるファイルへのアクセスは “ 論理的な**パス**（path; 経路）名 ” で行う．

> **定義 3.14（パス名）**
> あるディレクトリ（始点，木構造でのルートかノード）からアクセス対象（終点）であるファイル（木構造での葉）や別のディレクトリ（木構造でのノード）に至る論理的なパス（path; 経路）上にある “ ディレクトリ名 ” を，あらかじめ規定してある規則にしたがって，順次連結して作る文字列を**パス名**という．

「木構造ディレクトリ（唯一ルートをもつ 1 個の木である）」における「パス名の付け方」には，始点のディレクトリが “ ルート ” であるか “ ノード ” であるかによって，次の 2 種類に大別できる．（図 3.25を参照）

(a) **絶対パス名**：木全体の唯一ルートである “ ルートディレクトリ ”（図 3.25で