情報基礎シリーズ3

ソフトウェア

電子開発学園出版局

情報基礎シリーズ3

ソフトウェア

電子開発学園出版局

はじめに

コンピュータが開発されて以来、コンピュータは多くの分野で利用されてきました。現在では例えば、旅券を発行する、ご飯を炊く、ゲームをする、ニュースを知る、電話をするなど私たちの生活の中に根付いた分野で利用され、なくてはならない物として、また身近な物として存在しています。また現在のネットワーク社会はコンピュータなくして成り立ちません。

コンピュータはこれらの多くの分野で実現する様々な事柄を実現するために、“情報”を取り扱い、制御し、処理しています。コンピュータで取り扱い、処理する“情報”とはいったいどのようなもので、それを取り扱い、処理するためにどのような技術が使用されているのでしょうか。本書を含む「情報基礎シリーズ」では、このような知識、技術を“インフォメーションテクノロジ（IT）”と総称しております。

この“インフォメーションテクノロジ”を学ぶことは、様々な情報とそれを取り扱うコンピュータを学ぶことであり、それにより現代社会の基盤となるコンピュータについて理解を深めることとなり、そしてその社会についても知ることになります。

本書は、多くの分野から成立している“インフォメーションテクノロジ”のうち、コンピュータを成立させているソフトウェアについて、基礎から学ぶためのテキストであります。

ソフトウェアはワープロソフトや表計算ソフト、ゲーム、インターネットを見る為のブラウザといったものが挙げられます。例えば、ワープロソフトをパソコンに登録する（インストールする）と、パソコン本体にワープロソフトを動かすために必要なプログラムやデータが入り、パソコンでワープロソフトを動かすことができます。つまり、コンピュータを動かすために必要なプログラムやデータを総称したものがソフトウェアです。

本書によって、コンピュータとはどのようなものか、ソフトウェアとそのシステム等を学び、理解を深め、情報処理技術者試験の取得等に役立てていただければ幸いです。

編著者

目　　次

情報基礎シリーズ３

ソフトウェア

第１章　オペレーティングシステム

本章では、コンピュータの基本ソフトとなるオペレーティングシステム（ＯＳ：Operating System）についての説明を行う。

１．１　ＯＳの基礎

１．１．１　ＯＳの種類と特徴

（１）ソフトウェアとＯＳ

コンピュータの世界におけるソフトウェアは、一般的には次のように定義されている。

・プログラム（コンピュータに行わせる処理手順を表す一連の命令群）
・プログラムの使用方法を表した文書類（ドキュメント）
・業務の分析・設計技法・規則およびこれらの文書類（ドキュメント）
・コンピュータの運用技術

つまり、ソフトウェアとは知的な技術、技法、手段などの総称のことだが、狭くはプログラムだけを指すこともある。プログラムとは、ハードウェアに要求する動作をコンピュータが理解できる言語（プログラム言語）で記述したものである。ここではこの狭義のソフトウェアについて説明する。

そのソフトウェアは、機能的に**システムソフトウェア**と**アプリケーションソフトウェア**に分けることができる。

利用者はアプリケーションソフトウェアを使い自分の要求を実現するが、直接ハードウェアを操作しているのではなく、システムソフトウェアがアプリケーションソフトウェアとハードウェアの仲立ちを行い、利用者の要求を満たしている。

ソフトウェアの種類

＜システムソフトウェア＞

・基本ソフトウェア（オペレーティングシステム：ＯＳ）
・ミドルウェア

＜アプリケーションソフトウェア（応用ソフトウェア）＞

・共通応用ソフトウェア
・個別応用ソフトウェア

(a)　システムソフトウェア

システムソフトウェアは､ハードウェアの機能を効果的に活用できるようにするとともに、コンピュータ利用者の手助けをする共通的なソフトウェアのことである。

システムソフトウェアは、基本ソフトウェアとミドルウェアに分けることができる。

基本ソフトウェアは、**オペレーティングシステム**（**ＯＳ**：Operating System）ともいい、システムを制御するプログラムを中心に、応用ソフトウェアの作成に必要な機能やシステムの運用を支援する機能などを提供するプログラムから構成される。**ミドルウェア**は、基本ソフトウェアとアプリケーションソフトウェアとの仲介をするソフトウェアのことで、アプリケーションソフトウェアの作成を容易にする各種のソフトウェアなどが含まれている。

(b)　アプリケーションソフトウェア

アプリケーションソフトウェアとは、利用者が業務処理を行うためのプログラム（アプリケーションプログラム）のことである。特定の業務・業種にとらわれず、多くの業種にわたって共通的に使用されるソフトウェアや特定業種の特定業務に対応したソフトウェアがある。

アプリケーションソフト、アプリケーション、アプリなどと略して呼ばれることもある。

（２）ＯＳの目的

オペレーティングシステムの目的は、資源の有効利用を図り、生産性を向上させ、かつコンピュータシステムを使いやすいものにすることである。

また、コンピュータには様々な用途があり、利用者が操作を行うクライアント、クライアントから依頼のあった処理を行うサーバ、機器を制御するための専用コンピュータ、モバイルコンピュータなどがある。利用目的や用途によって、ＯＳも求められる性能に違いが存在するため、複数の種類のＯＳが存在する。だが、多くのＯＳは以下のような目標を設けている。

(a)　システム資源の有効活用

ハードウェア資源、ソフトウェア資源、人的資源などを有効活用し、生産性を向上させる。

(b)　処理能力の向上

スループットを向上させる。このためには、資源（主としてハードウェア資源）の空いている時間（**アイドルタイム**：idle time）が最小限になるように有効利用する必要がある。

(c)　ＲＡＳＩＳの向上

信頼性、可用性などＲＡＳＩＳを向上させる必要がある。

(d)　汎用性・拡張性の向上

コンピュータを利用するユーザ（利用者）の業務はさまざまで、利用する環境もそれぞれ異なるため、必要なコンポーネントやパッケージをインストールすることにより、ユーザの業務に適したＯＳを構築し、また新しい技術などにも対応する。

(e)　使いやすさの向上

システム運用の自動化ツール、ジョブのメニュー化、プログラム開発の支援ツールなどでＯＳが人手に頼っていた作業に対して支援することで、生産性を向上させる。

(f)　応答時間の短縮

利用者が求める結果を素早く手に入るようにするため、レスポンスタイムの短縮を行う。

（３）ＯＳの種類

前述したように、ＯＳはその利用目的に応じて様々な種類がある。代表的なＯＳの種別としては大きく分けて次のようなものがある。

・汎用機用ＯＳ
・オープンＯＳ
・リアルタイムＯＳ

他にもモバイル機器やゲーム機などで使用されているＯＳなども存在する。

(a)　汎用機用ＯＳ

メインフレームなど汎用機に用いられるＯＳで、汎用機のハードウェアのメーカがその汎用機、またはそのメーカの汎用機のシリーズ専用に作成したＯＳである。バッチ処理などの定型業務に使用されることが多いため、効率性の向上やマルチユーザに特化した機能を持っている。ＩＢM社のMＶＳなどが代表的な汎用機のＯＳである。また、現在ではオープンＯＳのＵＮＩＸなども汎用機で使用されている。

(b)　オープンＯＳ

オープンＯＳは、汎用機用のＯＳのように汎用機のメーカがその汎用機専用に作成したＯＳとは異なり、標準規格によって作られたコンピュータに使用できるＯＳである。一般的にはサーバ用やパソコン用（ＰＣ用ＯＳ）に使用されている。利用者は技術者以外の人を対象としていることが多いため、使いやすさが重視され、ＧＵＩなどの機能を持っていることが特徴である。代表的なＯＳとしてはＷｉｎｄｏｗｓOSや、ＵＮＩＸ、Ｌｉｎｕｘ、スマートフォンやタブレットコンピュータ向けのＡｎｄｒｏｉｄなどがある。

(c)　リアルタイムＯＳ

組み込みシステムで使用されているＯＳがリアルタイムＯＳである。組み込みシステムで使用されているため、機械制御がメインとなり処理能力や応答性能の向上に主眼が置かれている。代表的なＯＳとしてはｉＴＲＯＮなどがある。

１．１．２　ＯＳの構成と機能

（１）ＯＳの構成

ＯＳは、基本的に制御プログラムと処理プログラムによって構成されている。

(a)　制御プログラム

制御プログラムは、それ自体では生産的な仕事は行わずコンピュータシステム全体の動きを監視したり、次の仕事のための準備をしたりする。制御プログラムの中でジョブ管理、タスク管理（プロセス管理)、データ管理は制御プログラムの３大管理機能という。

制御プログラムの中でも最も中核となる部分を**カーネル**と呼ぶ。

（ア）カーネル

カーネルはリソースの管理を行い、ハードウェアに対しデバイスドライバによって制御をする。つまり、ソフトウェアとハードウェアの仲介を行うＯＳのもっとも基礎的な部分である。

カーネルには**マイクロカーネル**と**モノリシックカーネル**がある。

【マイクロカーネル】

マイクロカーネルはＯＳの最小限の機能だけを持つカーネルである。マイクロカーネルはカーネルのサイズが小さくなり、保守などが容易になるが、システムコールが多くなるため効率が悪くなる。

【モノリシックカーネル】

モノリシックカーネルはＯＳのほとんどの機能を持つカーネルで、マイクロカーネルに比べても効率がよく、実装が容易なのが利点である。だが、大きなサイズになってしまい保守が難しくなることと、ＯＳの機能間での関係が複雑になることが欠点である。

（イ）カーネルモードとユーザモード

プロセッサがプログラムを動作させる場合に､その動作に対する権限が設定されている。カーネルは非常に高い権限で、優先度も高く設定されており、ハードウェアに対する制御などが可能になっている。このカーネルが動作するモードを**カーネルモード**と呼ぶ。カーネルモードでは使用するメモリの場所（メモリ空間）もユーザモードとは違う場所を使うため、他のアプリケーションの影響を受けにくく、ＯＳ本来の機能を実行することができる。

アプリケーションプログラムがプロセッサを使用する場合のモードが**ユーザモード**にな

る。ユーザモードではそのアプリケーションに関する部分に対してのみ権限を持つ。アプリケーションプログラムがＯＳの機能を利用するとき、たとえばデータの入出力を行う場合、アプリケーションプログラムが入出力の**システムコール**（ＯＳの機能を呼び出すこと・ＳＶＣ）を行うとユーザモードからカーネルモードに切り替えられ入出力処理が行われる。

(b)　処理プログラム

処理プログラムは、制御プログラムの管理下でデータ処理に関する生産的な仕事を行うプログラムで、**サービスプログラム**と**言語プロセッサ**から構成される。

処理プログラムは、すべて制御プログラムの管理下で実行される。制御プログラムの機能によって、処理プログラムが単独で働くよりもシステム資源の有効利用が図られ、システム全体としての生産性が向上する。なお、制御プログラムの実行中は、処理プログラムは実行されない。この制御プログラムの動作時間をオーバヘッドタイムといい、オーバヘッドタイムは短い方が処理効率は上がる。

（２）ＯＳの機能

ＯＳにはさまざまな機能があるが、代表的な機能としては**ジョブ管理**、**タスク管理**、**データ管理**の三大管理機能がある。

それ以外にも、入出力管理や記憶管理、ネットワーク管理、ユーザ管理や運用・セキュリティ管理など多くの管理機能を持っている。それぞれの管理機能については後述する。

１．２　ＯＳの管理機能

１．２．１　ジョブとタスク

ＯＳが持つ機能の中で、最も重要な管理機能がジョブ管理、タスク管理、データ管理の３大機能である。まずこのうち最初の２つ、ジョブ管理、タスク管理を説明する。

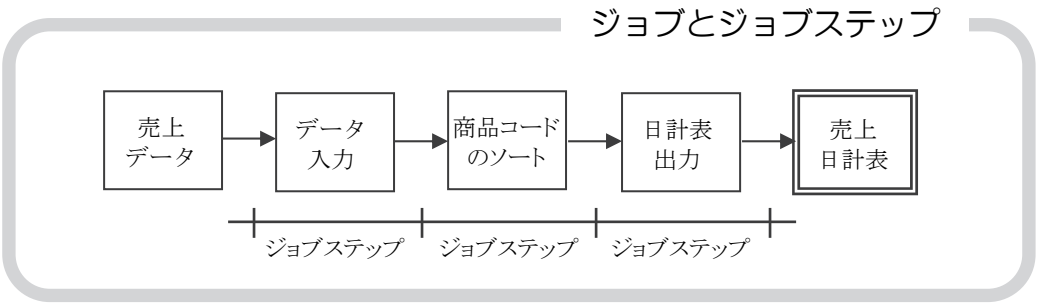
（１）ジョブ管理

人間の側からコンピュータに仕事を依頼する単位を**ジョブ**という。コンピュータ側から見た処理単位はタスク（プロセス）と呼ばれる。

コンピュータで処理しようとする仕事は、ジョブという単位でコンピュータに依頼される。ジョブ管理は、依頼されたジョブを効率よく処理するために欠かせないＯＳの機能である。主な働きは、ジョブの認識、資源の割当て、実行順序の制御等である。

(a)　ジョブとは

次の図はある商店での売上日計表を作成する流れを示している。この順番に作業を進行すれば、１つの仕事が達成される。この全体をジョブといい、ジョブの要素、例えばデータ入力とか並べ替え（ソート）に当たるのが**ジョブステップ**である。



（ア）ジョブ

コンピュータで実行する手順は、多くが“データ入力→ソート→結果出力”だが、手順のうちの１つが抜けても、また順序が変わっても実行はできない。

ジョブは、この様に、他の仕事と完全に区別でき、それ自体で１つのまとまった仕事と見なせるものである。

（イ）ジョブステップ

ジョブはいくつかの処理の組み合せからなり、これらジョブの構成要素をジョブステップという。コンピュータでプログラムを実行する手順は、“入力→処理→出力”が基本であり、その１つ１つがジョブステップである。

ジョブは１つ以上のジョブステップから構成され、先行ジョブの出力が後続ジョブの入力となる。各ジョブステップ間は通常は、データファイルを介して連絡をとり、順番に実行される。

このジョブステップが次のタスク管理によりいくつかのタスクになる。

（ウ）ジョブの指示

【ＪＣＬ】

ＪＣＬ（Job Control Language）とは、利用者が汎用コンピュータにジョブを行わせるために、使用するファイルの指定や実行、プログラムの指示などを、ジョブステップ単位に処理要素として指示するための言語である。利用者がコンピュータを使って何らかの処理をどのような資源を使い、どのような手順で何を行わせるかを事細かに通知する。ＪＣＬを記憶媒体に記録しコンピュータに読み込ませると、ＯＳがＪＣＬを解釈してジョブを実行する。

【コマンド】

ＬｉｎｕｘやＷｉｎｄｏｗｓなどのオープンＯＳではコンピュータに仕事をさせる場合は、コマンドで命令を与え仕事を行わせる。このコマンドを入力することで実行される仕事をジョブという。コマンドはキーボードから入力されると、コマンドインタプリタで解釈され即座に実行に移るが、一連の処理を行うコマンドをファイルに記述しておきスクリプトとして実行することも可能である。

通常、コマンドが入力されるとジョブの実行が開始される。このジョブをフォアグラウンドジョブといい、ジョブが完了するまで次のコマンド入力を受け付けなくなる。しかし、並行して複数のジョブを実行させたり、ジョブの実行中に他の作業を行いたい場合もある。こういった時はコマンド入力後、ジョブの実行を裏側で行いジョブが実行中であってもすぐに次のコマンド入力状態に戻すことが可能である。このように裏側で動いているジョブをバックグラウンドジョブという。

(b)　ジョブの処理

汎用コンピュータのＯＳでは以下のような順序でジョブが実行される。汎用コンピュータを使用するオペレータと汎用コンピュータの間では**タスクスケジューラ**が仲介することによって、ジョブを管理している。タスクスケジューラは、インプットリーダ、イニシエータ、アロケータ、ターミネータ、ライタなどのサブプログラムを持つ。

１）ジョブの入力

インプットリーダ（リーダ）によってジョブが読み込まれジョブが登録される。

２）ジョブの選択

実行待ち行列に登録されたジョブは、システムで定められた規則に従い、**ジョブスケジューラ**によって選択され、実行される。優先的に実行させたいジョブや、同期が必要で同時に実行することのできないジョブなどが存在するため、実行に最適なジョブを選ぶことが必要である。

３）ジョブの解読

選択されたジョブは**イニシエータ**によって解読され実行に移される。実際の実行はジョブステップごとにタスク管理によって行われる。

４）資源の割り当て

実行に先立ち**アロケータ**が周辺装置やファイルなど必要な資源を割り当てる。

５）資源の解放

ジョブの実行が完了すると**ターミネータ**がジョブの終了処理を行う。生成されたプロセスが消滅され、割り当てられていた資源が解放される。

６）ジョブの出力

ジョブの結果はプリントキューに溜められ出力の順番を待つ。出力順序の決定や出力指示は**ライタ**が行う。

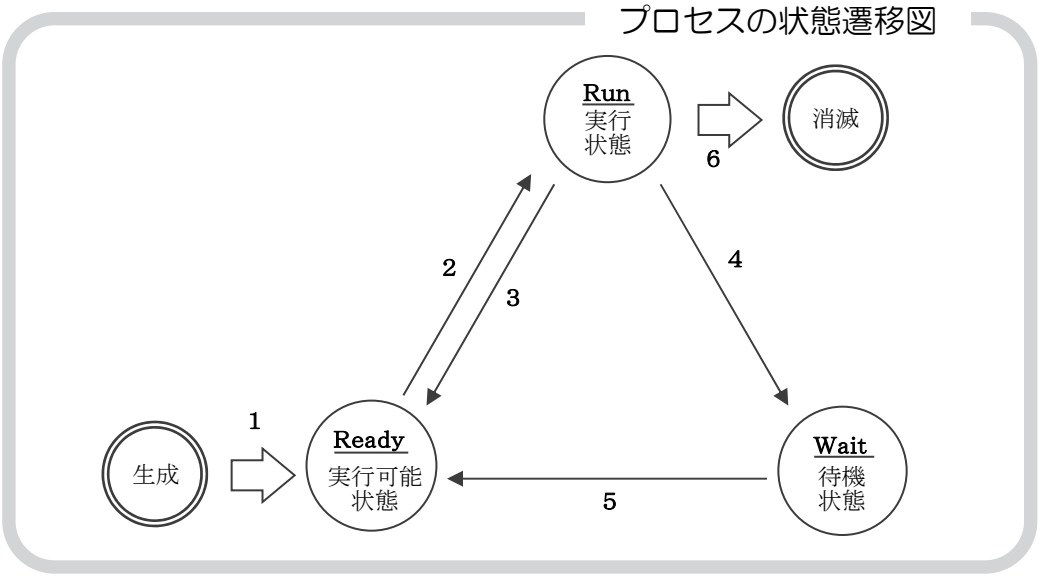
（２）タスク管理

コンピュータ側から見た処理単位を**タスク**または**プロセス**という。ジョブ管理により投入されたジョブは、ジョブステップとして順次実行される。このジョブステップごとに１つ以上のタスクが生成され、コンピュータはこのタスクを単位として実行している。

(a)　タスクの状態遷移

タスクは生成から消滅まで３つの状態を遷移する。

タスクが生成されると**実行可能状態**（Ｒｅａｄｙ）となり、プロセッサの使用権の割当てを待つ。プロセッサの時間割当てがなされたら**実行状態**（Ｒｕｎ）となる。実行状態のものが入出力動作を開始したら、**待機状態**（Ｗａｉｔ）となり、事象の終了を待ってＷａｉｔ状態からＲｅａｄｙ状態へと遷移する。プロセスが終了したときには、プロセスの消滅という現象となる。



１）生成→Ｒｅａｄｙ

ジョブステップが到着すると、ただちにタスクが生成され実行されるわけではなく、いったん実行可能状態となる。

２）Ｒｅａｄｙ→Ｒｕｎ

それまで実行状態にあったタスクの終了や待ち状態への遷移を機に、実行可能状態で待機しているタスクの中から、待ち行列の中から優先度の高いタスクにＣＰＵ時間が与えられる。

このようにプロセッサの使用権を割り当てることを**ディスパッチング**という。

３）Ｒｕｎ→Ｒｅａｄｙ

実行状態から実行可能状態への遷移は、あるタスクがプロセッサを割り当てられた実行時間（**タイムスライス**または**タイムクオンタム**という）を消費してＣＰＵ時間を使い切ったときや、より優先順位が高いタスクが生成されたときに起きる。

４）Ｒｕｎ→Ｗａｉｔ

実行状態から待機状態への遷移は、タスクが入出力操作を開始するときに起きる。そのタスクは入出力動作の完了まで待機状態になる。

５）Ｗａｉｔ→Ｒｅａｄｙ

待機状態から実行可能状態への遷移は､入出力動作によって待機状態に入ったタスクが、その完了とともに実行可能状態に移るときに起きる。

６）Ｒｕｎ→消滅

タスクはその実行する作業が完了した時点で、消滅する。

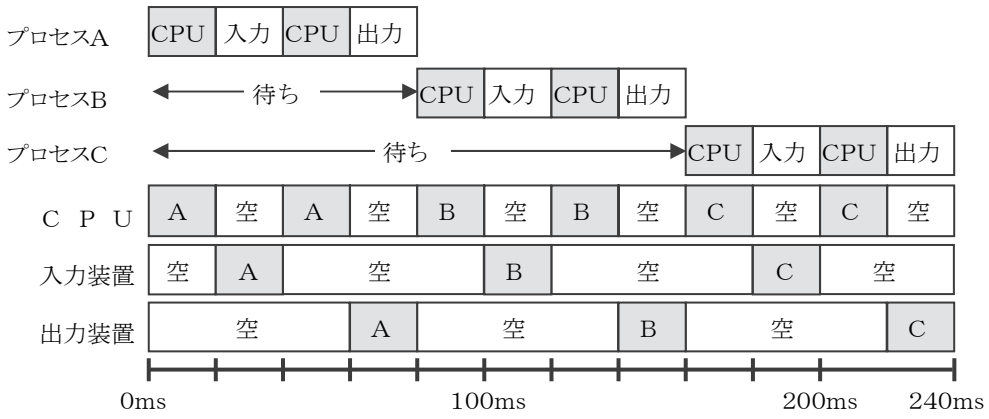
(b)　マルチプログラミング（マルチタスク）

マルチプログラミングは、複数のプログラム（タスク）を同時に起動し、プロセッサの使用権を順番に割当てることで見かけ上、複数のプログラムを同時並行的に実行させる方式のことである。**マルチタスク**または**マルチプロセス**ともいう。これが実行できるＯＳは**マルチタスクＯＳ**と呼ばれる。

シングルプログラミングとマルチプログラミング

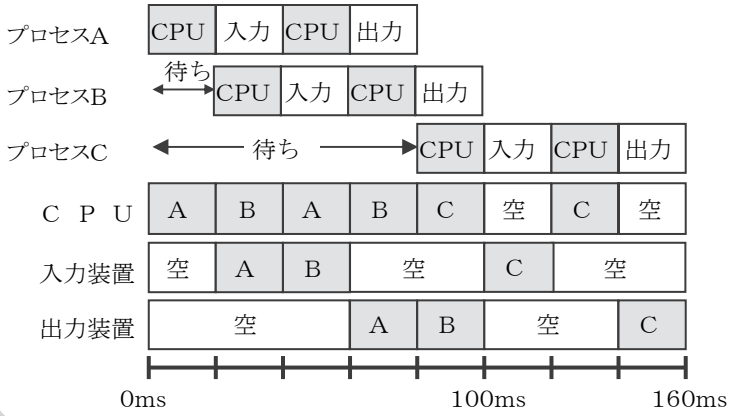
＜シングルプログラミングの場合＞

１本のプログラムの終了を待って他のプログラムの処理を開始する



＜マルチプログラミングの場合＞

これら３本のプロセスは、次のように入出力の動作によってプロセッサ処理が切り替わる



(c)　タスクの割り当て

（ア）ディスパッチング

複数個ある実行可能状態のタスクの中と、実行状態のタスクを切り替える処理によりマルチプログラミングが可能になるが、この切替え処理をするもの（プログラム）が**ディスパッチャ**である。ディスパッチャが切替えを行うことを**ディスパッチング**と呼ぶ。ディスパッチャは基本的には次の時点で起動して、タスクの切替えを行う。

・新たにタスクが生成されたとき
・実行中のタスクが待機となったとき
・タイムスライスの終了時
・優先度変更時
・入出力命令に対する処理が完了したとき

【ディスパッチャの動作】

タスクを実行可能にディスパッチングするには、事象による割込みとインタバルタイマによる**割込み**がある。割込み動作は次のとおりである。

割込み動作では、前記のタスクの切替えのタイミングで、現在処理を行っているタスクを退避させ、新たなタスクを実行させる。新たなタスクが完了、または割り込みが発生した場合には対比させていたタスクを元に戻し処理を続行する。

（イ）タスクの切替え

事象による割込みとインタバルタイマによる割込みは、それぞれ**イベントドリブン方式**と**タイムスライシング方式**と呼ばれる。

【イベントドリブン方式】

この方式は何らかの事象（イベント）の発生により、タスクを切り替える方式である。事象とはシステムにおける状態の変化のことで、入出力の完了、入出力要求の発生、ジョブの到着、ジョブの終了などである。この方式は、あるタスクが事象の発生によりＣＰＵ時間を使う必要がなくなったときなどに、別の実行可能なタスクへＣＰＵ時間を与えることである。それによりスループットの向上を図ろうとするもので、マルチプログラミングの基本概念である。

【タイムスライシング方式】

タスクの中にはＣＰＵバウンドなものと入出力バウンドなものの２種類がある。ＣＰＵバウンドはプロセッサのパワーを大きく消費し、プロセッサ処理の比率が入出力に比べて大きいもので、入出力バウンドはその逆である。

ＣＰＵバウンドなタスクによるＣＰＵ時間の独占を禁止するために、時間をスライスして一定時間ずつ割り当てる。各タスクにプロセッサを割り当てる一定時間を**タイムクォンタム**または**タイムスライス**という。タイムスライスを検知するためのハードウェア機構にインタバルタイマが使用されており、タイマ割込みによってタイムクオンタムの消費を認識する。この割込みをＯＳが察知して、実行中のタスクを実行可能状態にし、実行可能状態の中からタスクを１つ選んで実行状態にする。

なお、オペレーティングシステムがプロセッサ使用時間や優先度によってプロセッサに割り当てるタスクを強制的に切り替えることをプリエンプションと言い、このようなマルチタスクを**プリエンプティブマルチタスク**と呼ぶ。一方、**ノンプリエンプティブマルチタスク**では、タスクの切替えをアプリケーションに委ねることになり、プログラムの性質によっては効率が低下しがちである。それを避けようとすればアプリケーションを作るプログラマの負担を増やすことになるが、プリエンプティブマルチタスクではハードウェアタイマによってタイムスライスを作り出すので、アプリケーションに依存しない切替えが可能になる。

（ウ）スケジューリング

ディスパッチャに対し、次にＣＰＵ時間を割り当てる（実行状態にする）タスクを決定するものが**スケジューラ**（タスクスケジューラ）である。スケジューラが順番を決定することを**スケジューリング**と呼ぶ。

スケジューラによるタスクの順番の決定には、次のようなアルゴリズムを使用する。

【ＦＩＦＯ方式】

ＦＩＦＯ（First In First Out）方式はタスクの実行可能待ち行列に到着した順番に割り当てる方式である。

【ラウンドロビン方式】

ＣＰＵ時間を各タスクへ均等に分配する方式である。到着順にタイムスライスを考慮して割り当てる。

【プライオリティ方式】

タスクの優先度をその特性によってあらかじめ設定しておき、優先度の高いタスクから割り当てる方式である。

【最短ジョブ優先方式】

あらかじめジョブの実行時間が判明している場合に、処理時間の短いタスクから優先的に割り当てる方式である。

【多段フィードバック方式】

ジョブの実行時間を前もって知ることができない場合の方式で、すべてのタスクに対して当初は優先度を高くし、短いタイムスライスを平等に与え、それでも終了しないタスクは順次、優先度を下げタイムスライスを長くしていく方式である。

【ダイナミックディスパッチング方式】

タスク単位に入出力の事象までの処理時間を計測して、ＣＰＵバウンドなタスクに対しては優先度を低く抑える方式である。これによりシステム全体として、スループットの向上を望めることが経験的に実証されている。

(d)　スレッド

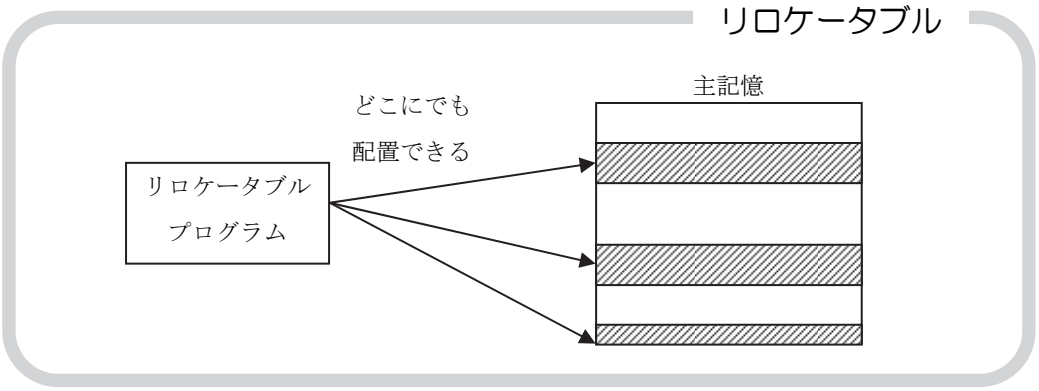
スレッドとは軽量プロセスとも呼ばれ、プロセスの中に作られる小型のプロセスのことである。マルチスレッドのプログラムでは１つのプロセスの中に複数のスレッドがあり、スレッドを切り替えながら並列的に処理される。スレッドは軽量プロセスという名のとおり、プロセスに比べプロセッサへの負担が小さく高速に処理される。スレッドの生成、スレッド間の同期・通信、実行の切替えは、普通のプロセスと比較して、１０倍から１００倍程高速に行われる。プロセスは異なったアドレス空間で実行されるが、スレッドは親プロセスのアドレス空間を共有して利用できる。

（３）プログラムの属性

タスクとして投入されるプログラムには属性（**プログラム属性**）がある。プログラム属性とはプログラムが備えている性質のことで、プログラムの使用、プログラムの共有、プログラムのロードに関する属性がある。

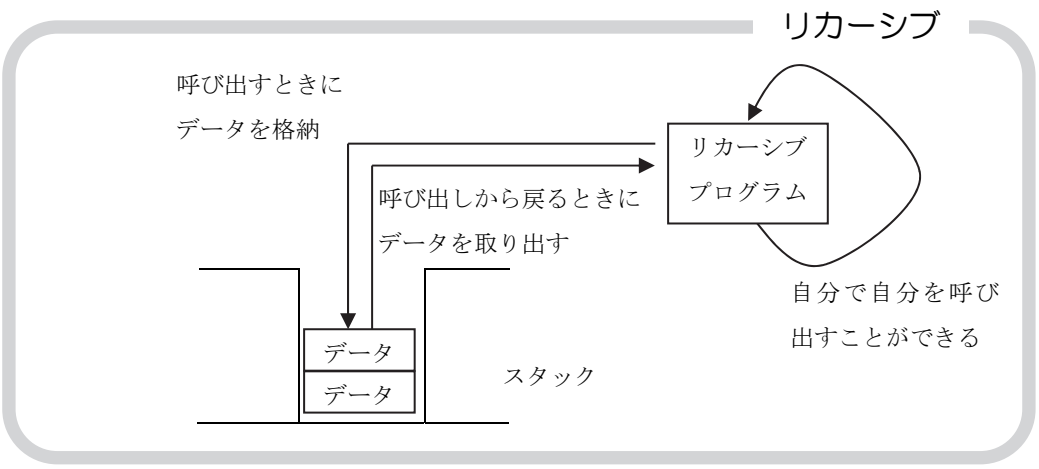
(a)　リロケータブル

プログラムは実行に先立って補助記憶装置からメモリにロードされる。この時、メモリ上のどのアドレスにでもロードできるプログラムの属性を**リロケータブル**（**再配置可能**）という。この属性を持ったプログラムはプログラムを構成している機械語命令のアドレス部が絶対番地ではなく相対番地になっている。プログラムがロードされた番地をベースアドレスレジスタに保持しており、実行時にアドレス修飾を行うため、どこに配置されても問題なく実行できる。



(b)　リカーシブ

自分自身を呼び出すことの出来る形式のプログラムを**リカーシブ**（**再帰呼び出し**）プログラムといい、通常はサブルーチンや関数の形をしている。自分自身を呼び出す**リカーシブコール**を使うといい､クイックソートがリカーシブコールを使うプログラムで有名な例である。他にも基数変換、リスト処理、将棋やチェスのプログラムなど、広く利用されている。



(c)　リユーザブル

通常、プログラムは補助記憶装置に記憶されており、実行時にメモリにロードしてから実行される。

プログラムの実行が終了した後、他のプロセスが同じプログラムを実行したい場合、そのプログラムがまだメモリ内に残っていれば、再度ロードすることなく実行できれば効率よく処理できる。このように再使用できる性質を**リユーザブル**（**再使用可能**）という。

(d)　リエントラント

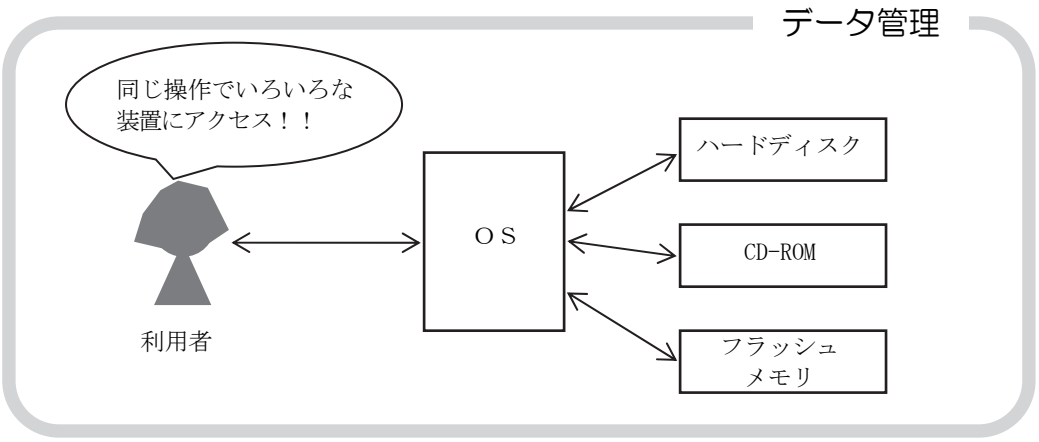
あるプロセスが実行中のプログラムを、他のプロセスが同時並行して実行できる場合、そのプログラムの属性を**リエントラント**（**再入可能**）という。同時に複数のプロセスで１つのプログラムを実行するためにはプロセスごとにデータ部分が必要である。そして、リエントラントプログラムが実行するデータ操作の命令は各プロセスが自分のデータ部分に対して実行してもらう必要がある。各プロセスは各自のデータ部分のベースアドレスを保持しているので、同じデータ操作命令の実行であってもアドレス修飾することによって各自のデータ部分に対してデータ操作が行われる。

１．２．２　データと記憶

ＯＳの３大機能の残りの１つであるデータ管理、そしてそれと関係が深い入出力管理と記憶管理を説明する。

（１）データ管理

データ管理とは、コンピュータで扱うデータの統一的な管理とそれに伴うハードウェア装置の制御を行うことである。コンピュータが取り扱う記憶装置は、ハードディスク、ＣＤ－ＲＯＭやＤＶＤ－ＲＯＭ、フラッシュメモリなど多岐にわたり、それぞれが保有するデータへのアクセス方式は全く異なる。だが、利用者やアプリケーションソフトはその差異を全く感じずに、同じ操作で別々の記憶装置を使うことができる。これがデータ管理の役割である。



（２）入出力管理

複数バイトからなるデータを１単位のブロックとして入出力を行なうデバイスをブロックデバイスという。ブロックデバイスの入出力はバッファを介して行う。具体的には、ハードディスクやＣＤ－ＲＯＭ、テープなどがブロックデバイスである。これに対し、１文字単位またはピクセル単位に入出力を行うプリンタやディスプレイ、キーボードなどをキャラクタデバイスという。

入出力管理は、データ管理の指示に従って、アプリケーションプログラムから入出力命令を受け取り、要求のあったデータを入出力用の**バッファ**に格納する。その後、対象装置の使用状況により待合せを行った後、複数ある入出力の経路のうち空いている経路を発見して入出力を行う。また、その入出力の実行を監督しているときに障害が発生した場合は、再試行を何度か繰り返し、それに失敗したら異常終了させるという仕事もする。

また入出力管理は、「４．４．２」で説明したような、**ＤＭＡ制御方式**、**チャネル制御方式**を用いて入出力機器と主記憶装置とのデータのやりとりを制御し、**入出力割込み**を使用してタスク・プロセスに対して割込みを行う。

(a)　ブロック化とバッファ

入出力管理は、出力要求のあったデータを、バッファに１レコードずつ格納し、バッファがいっぱいになった時点でバッファへの格納をいったん停止する。これをブロッキングという。

なお、ブロッキング後、入出力管理は出力装置を指定してチャネルに出力命令を出す。逆に、入力要求があった場合は、入力装置からバッファに転送されたブロックを１レコードずつ実行中のプログラムに引き渡す。1 レコードずつに分解することをデブロッキングという。

(b)　スプール処理

プログラムからプリンタやモデムなどの低速の周辺装置へ直接データを出力すると周辺装置を占有してしまい、そのプログラムが終了するまで他のプログラムはその装置を使用できなくなる。

プリンタやモデムはプロセッサやハードディスクの速度に比べかなり低速なのでデータを直接プリンタやモデムに送るのではなく、一時的にハードディスクなど高速の補助記憶装置にファイルとして出力（**スプールファイル**）し処理を進めるようにする。

各プログラムは低速な周辺装置の空きを待つ必要が無く、効率よく処理を進めることができる。そして、ハードディスクに出力された処理結果は出力キューに登録され出力の順番を待つ。こうすることによって、プログラムの出力待ち時間が少なくなり、周辺装置の稼働率も上がり処理効率が高くなる。

このような処理を**スプール処理**（ＳＰＯＯＬ：Simultaneous Peripheral Operation On-Line、周辺機器の同時オンライン動作）という。スプール処理を実施することを**スプーリング**と呼ぶ。同時に複数のプログラムが実行されているコンピュータシステムやプリンタを共有して使用しているネットワーク環境ではプリンタのスプール処理が行われている。各プログラムからの出力要求はプリントキューとして登録され、出力の順番を待つ。

（３）実記憶管理

記憶管理機能のうち、メインメモリに対する管理を行うのが**実記憶管理**機能である。（４）で説明する仮想記憶に対してメインメモリのことを実記憶と呼ぶ。メインメモリの管理として必要な機能は次の４項目である。

・メインメモリの各記憶場所がプログラムに割り付けられているかいないかを把握する
・どのプロセスにメインメモリのどの領域をいつどれだけ与えるのかを決定する
・割付けを管理するための情報を更新する
・プロセスの終了を契機に記憶領域の解放を行う

メインメモリの割り付けとは、プログラムの命令を取り出せるようにするためプログラムを補助記憶装置からメインメモリに読み込み（**ロード**）することである。一番簡単な方式としては、メインメモリに現在実行するタスクのプログラムをロードすることである。これにより、少なくとも必ずメインメモリ上にロードしておく必要のあるＯＳのプログラムと、タスクのプログラムの２つがロードされた状態になる。

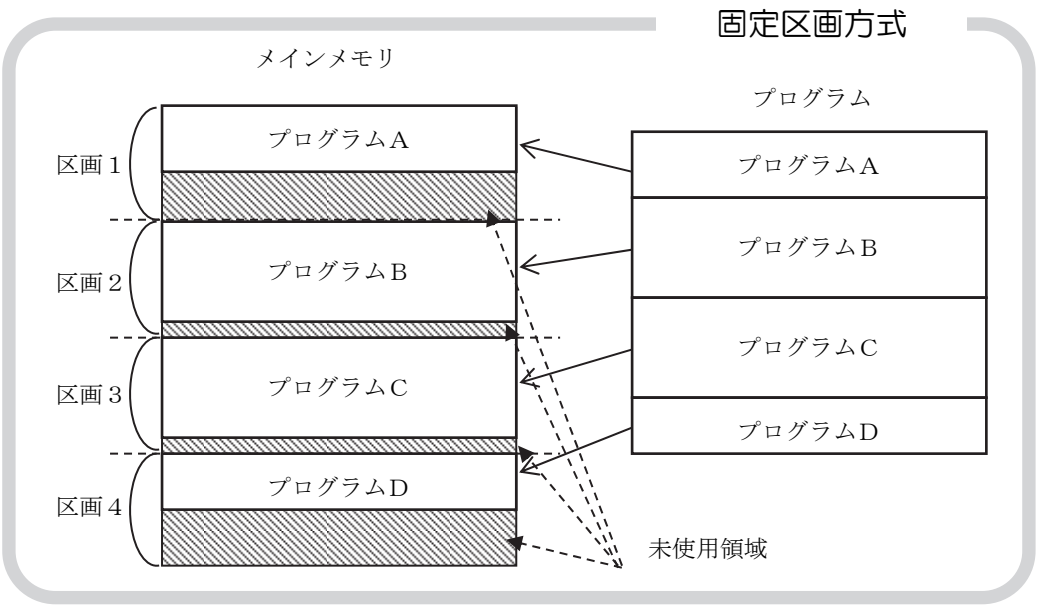
だが、メインメモリのサイズは有限であるためタスクで必要なプログラムがすべて同時にロードできるとは限らない。また、現在のＯＳはマルチタスクが可能であるので、複数のタスクのプログラムが同時にメインメモリに配置されることになり、容量の問題が重要になる。さらにどのように複数のプログラムをメモリに配置するかという管理の問題が出てくる。

これらの問題を解決するためにさまざまな手法が考え出されている。

(a)　固定区画方式

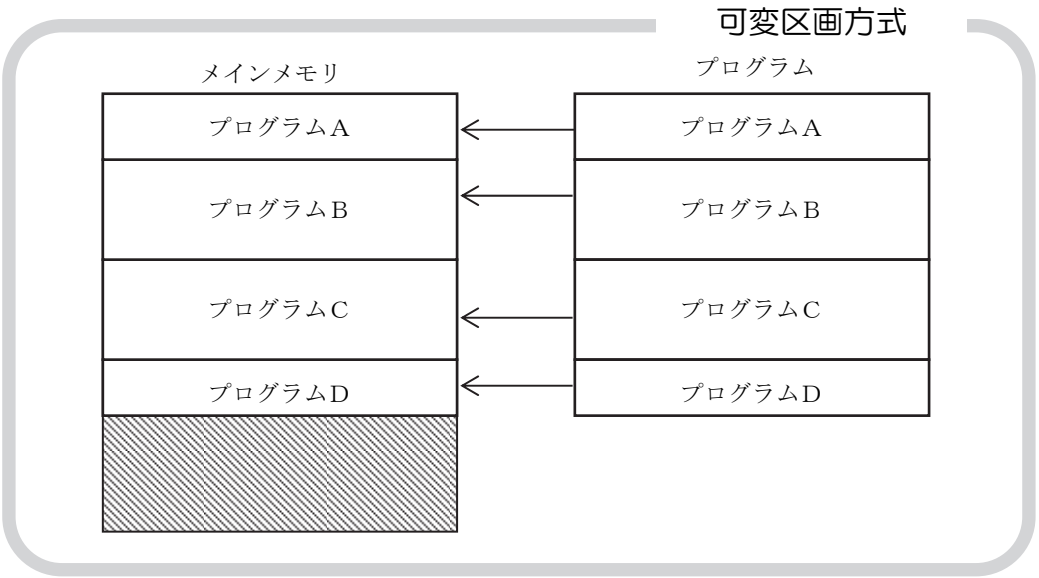
固定区画方式は、メインメモリをある一定のサイズの区画に分割して、各区画にそれぞれプログラムを１つずつ格納する方式である。これにより、同時に複数のプログラムがメモリ上に配置されることになり、マルチプログラミングが可能になった。

固定区画方式では、分割した区画の数までしかプログラムがロードできない。また、区画の大きさより大きいプログラムはロードできない、また逆にプログラムが区画のサイズより小さい場合は区画の未使用の領域は使用できないためメインメモリの利用効率が低下する、などの問題がある。



(b)　可変区画方式

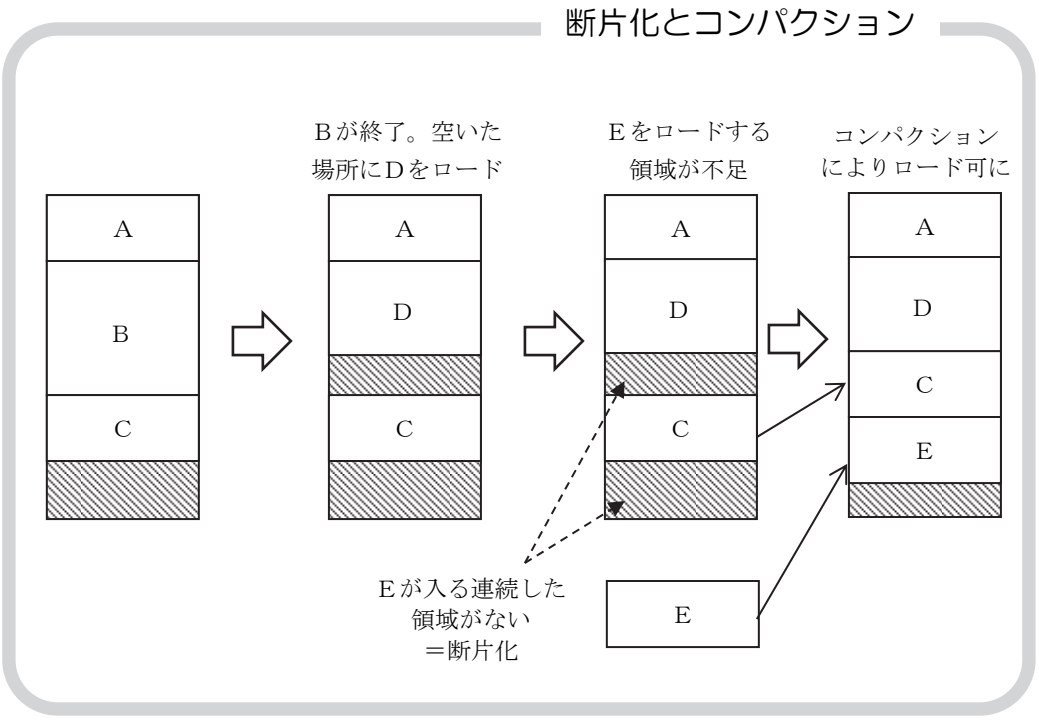
可変区画方式は、プログラムをロードする際にプログラムの大きさに合わせて区画を決定する方式である。固定区画方式よりもメインメモリを効率的に使用できる。



だが、プログラムのロードとプログラムの終了に伴いメインメモリの解放を繰り返していくと、こま切れの未使用領域が発生し、**断片化**（**フラグメンテーション**）が発生する。フラグメンテーションが発生すると、メインメモリの利用効率が下がってしまう。

（ア）断片化とコンパクション

断片化が発生すると、こま切れの未使用領域が発生し、新たなプログラムがロードできない事態が発生する。そのため、プログラムをメインメモリ上で再配置し、こま切れの未使用領域をまとめることを行う。これを**コンパクション**と呼ぶ。



（イ）ディスク断片化

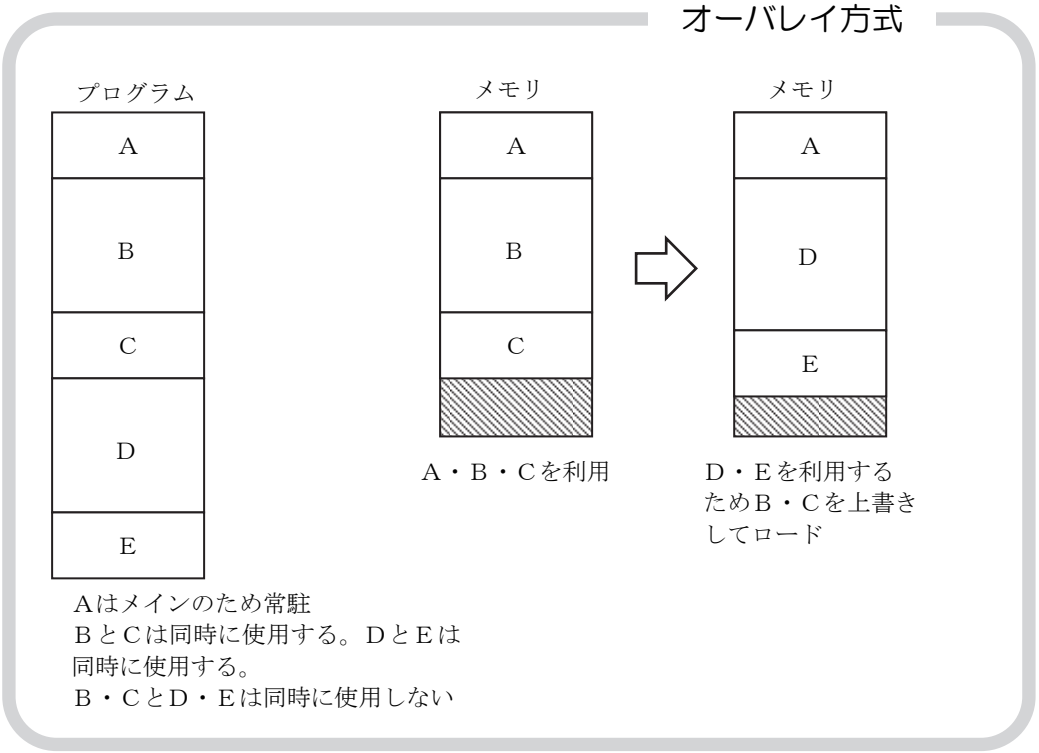
メインメモリだけでなく、補助記憶装置でも断片化は発生する。補助記憶装置を長時間使用していると、ファイルの保存と削除が繰り返され連続した領域が確保できなくなる。このような場合ファイルを分割して保存することになるが、それにより読み込み時間の増大などが発生するようになる。

そこで補助記憶装置では連続した領域を確保できるようにし、分割して保存していたファイルをまとめる作業を行う。これを**デフラグ**（デフラグメンテーション）と呼ぶ。

(c)　オーバレイ方式

メインメモリの領域が、プログラム全体をロードするために必要なサイズを持たない場合に**オーバレイ方式**を使用することがある。

オーバレイ方式ではプログラムを小さく分割する。この分割した単位はセグメントまたはモジュールと呼ばれる。このセグメントのうち、同時に使用する必要がないものはロードせず、その時に必要とされるものだけをロードする。新たにセグメントが必要になった場合は、使用していないセグメントに上書きしてロードを行う。



(d)　スワッピング方式

スワッピング方式では、メインメモリの容量が不足している場合に、優先度の低いプログラムを現在の状態を含め補助記憶装置に保存して、退避させておく方式である。退避させたプログラムが必要になった時は、補助記憶装置からロードする。優先度の高いプログラムが割り込みする場合などに使用する。

メインメモリから補助記憶装置へ退避させることを**ロールアウト**または**スワップアウト**、補助記憶装置からメインメモリへロードすることを**ロールイン**または**スワップイン**と呼ぶ。

(e)　メモリの利用

（ア）メモリの領域の種別

既に説明したとおり、プログラムはコンピュータのメインメモリ上に展開されてから実行を開始する。プログラムが使用するメモリ領域には大まかに次の３種類がある。

【スタティック領域】

プログラム中の静的データ領域に対して割り当てられるメモリ領域である。プログラムの開始から終了までメモリ割り当てが変化しないため、動的に割り当てられるスタック領域やヒープ領域に対して、静的（スタティック）領域と呼ばれている。

【ヒープ領域】

現在のマルチタスクＯＳでは、プログラム開始時点でのメモリ割当て以外に、空きがあれば必要に応じてメモリを割り当てる仕組みが用意されている。この仕組みを利用し、初めは少なめにメモリ領域を割り当てておき、実行途中でメモリ領域が不足する都度、追加のメモリ割当てを行うようになっている（最初から最大限の割当てを行うことは無駄が大きいため)。このように実行中に、新たに割当てすることが可能なメモリ領域をヒープ領域という。

【スタック領域】

サブプログラムの呼出しなどで、戻り番地の保存に使われる。その他、実行中に定義された自動変数もこのスタック領域に割り当てられる。スタックにデータを出し入れすることで、再帰呼出しを可能にし、同じメモリ領域を異なる変数で再利用できる利点も持っている。

（イ）メモリリーク

メインメモリにロードされたプログラムは、その実行が終了し使用しなくなった時点でメインメモリ上から消し、その部分を未使用領域に戻す。これをメモリの解放と呼ぶ。

だが、なんらかの原因でメモリが解放されず、使われていないプログラムがメインメモリ上に残ったままになることがある。このようなプログラムがメインメモリ上に積み重なると、本来ならば使用できるメインメモリの領域が解放されず、利用可能なメモリの領域が減ってしまう。これによりプログラムやＯＳの動作が不安定になったり、場合によっては停止することがある。これを**メモリリーク**と呼ぶ。

このような使用されていないプログラムをメモリから解放するためのプログラムは、**ガーベジコレクション**と呼ばれ、プログラミング言語によっては、これが用意されている場

合がある。

（４）仮想記憶管理

仮想記憶とは、補助記憶装置を利用して実際のメインメモリよりもはるかに大きな容量を使用可能とした機能で、メモリを利用するプログラムからみれば、広大な容量を持つメモリ（仮想メモリ）を利用できることになり大きなデータや複数のプログラムの同時実行などが行えるようになる。

プログラム自体は一般的には補助記憶装置（これを仮想記憶装置と呼ぶ）に格納しておき、必要な部分だけをメインメモリにロードする形で仮想記憶は動作し、プログラムは、広大な仮想記憶のアドレス（**仮想アドレス**）を利用してプログラムを実行できる。仮想記憶管理により実際のメインメモリのアドレス（実アドレス）へそれを変換して動作する。この変換は**動的アドレス変換**（ＤＡＴ：Dynamic Address Translation）と呼ばれる。

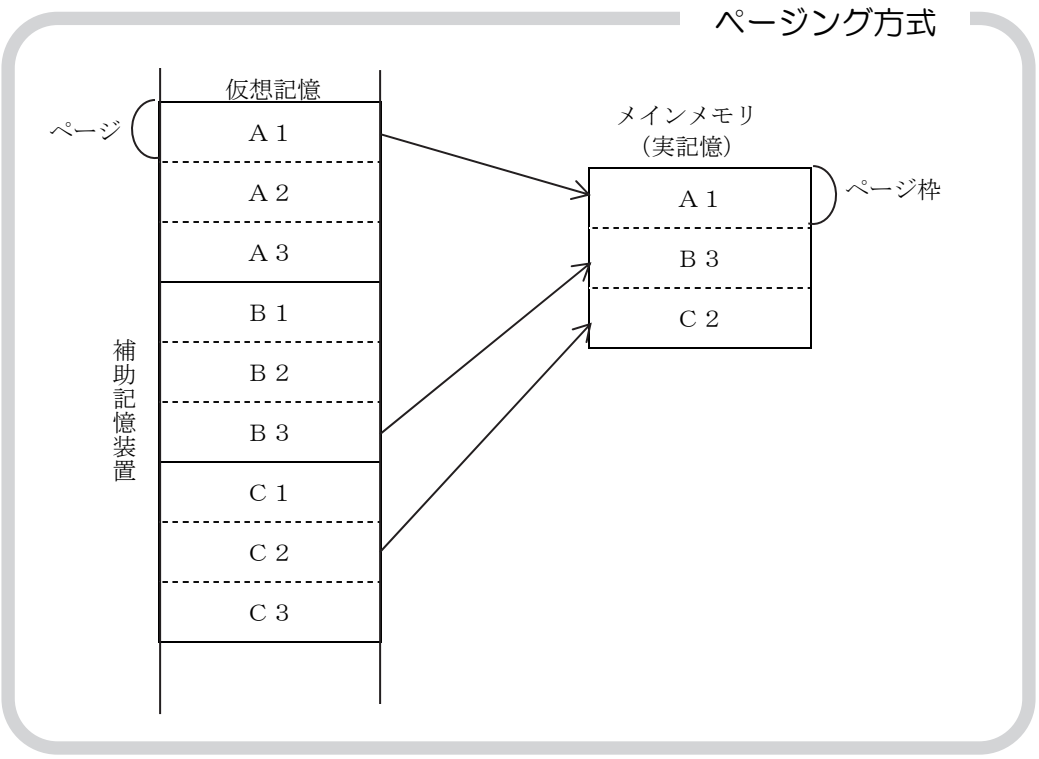
仮想記憶を管理する方式にはページング方式とセグメント方式がある。

(a)　ページング方式

ページング方式とは、仮想記憶システムの考え方の一つで、実記憶をある一定の大きさに区切って区画管理をしておき、その区画と同じ大きさに分割されたプログラムを配置していく方式である。この区切られたプログラムをページと呼ぶ。

プログラムは仮想記憶装置に格納されている。一方のメインメモリもページと同じ大きさの**ページフレーム**（ページ枠）に分割されている。

プログラムのページは必要になると仮想記憶装置からページ単位で取り出され、メインメモリ上に配置される。



プログラムを実行するためには、実行するプログラムのページが実メモリ上に存在しなければいけない。もちろん、実メモリの容量には限界がありすべてのページが実メモリ上には存在しない。そのため、仮想記憶と実記憶でページのやりとりをする必要がある。これを**ページリプレースメント**と呼ぶ。

ページリプレースメントでは、実行するために必要なプログラムのページが実メモリ上に存在しないことを、**ページフォルト**と呼ぶ。ページフォルトが発生した場合、必要なページを仮想記憶から実メモリにロードする必要がある。これを**ページイン**と呼ぶ。

ページインが発生する際に、実メモリ上のメモリ枠に空きがあればよいのだが、ない場合は現在必要としていないページを仮想記憶に戻し、ページ枠を空けなければいけない。これを**ページアウト**と呼ぶ。

(b)　セグメント方式

ページング方式は、ページという固定長の単位でロードしていた。セグメント方式（セグメンテーション方式）は、セグメントという可変長で論理的なまとまりを作り、このセグメント単位でロードするため、ページング方式に比べメモリの有効活用ができる。また、共通に利用するセグメントとそうでないセグメントを分けて配置できる。それ以外の点についてはページング方式と大きな違いはない。

セグメント方式では、ページイン、ページアウトはロールイン、ロールアウトと呼ばれる場合がある。

また、セグメント方式ではセグメントは可変長であるため、セグメントの長さがそれぞれ異なり、管理が複雑になるうえ、メインメモリの領域を連続してとらなければいけないという欠点がある。そこで、セグメントをさらにページに分割して行う**セグメントページ方式**もある。

(c)　ページリプレースメント

（ア）ページ置き換えアルゴリズム

仮想記憶方式では、ページインとページアウトというページリプレースメントが行われている。この時、ページアウトさせるページを選ぶ方法を、**ページ書き換えアルゴリズム**（**ページングアルゴリズム**）という。ページ書き換えアルゴリズムの主なアルゴリズムを説明する。

【ＦＩＦＯアルゴリズム】

ＦＩＦＯ（First In First Out）はメインメモリに入った順番にページアウトする方法である。つまり、ページインしてからの時間が最も長いページをページアウトするという形になる。

【ＬＲＵアルゴリズム】

ＬＲＵ（Least Recently Used）は最も有名なアルゴリズムで、ページアウトを行う時点でそれ以前に使われない時間が一番長いページをページアウトの対象に選ぶ方法である。一定の時間内に使われていないということは、それ以後も使われることがないであろうという考えのもとに成り立っている。

この方法を厳密に実現すると、命令の実行のつど、ページの使用状況を監視しておく必要があるので、実際には擬似的なＬＲＵ方式が利用されている。この方法はある一定時間ごとに、ページの参照ビット（このページが参照されたかどうか）を検査して、オフのものをページアウト対象にするという形で運用している。

【ＬＦＵアルゴリズム】

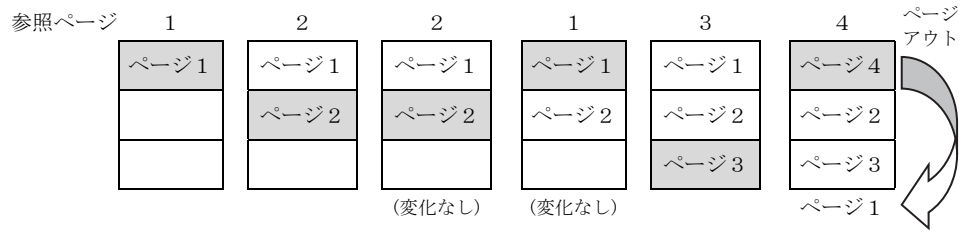
ＬＦＵ（Least Frequently Used）は、ページアウトを行う時点でそれ以前に参照された頻度（回数）が最も少ないページをページアウトの対象に選ぶ方法である。ＬＲＵアルゴリズムが「最後に参照されてからの経過時間」を基準にページアウト対象を選ぶのに対し、ＬＦＵアルゴリズムは「参照頻度（回数）」を基準とする点が特徴である。

ＦＩＦＯとＬＲＵ、ＬＦＵアルゴリズム

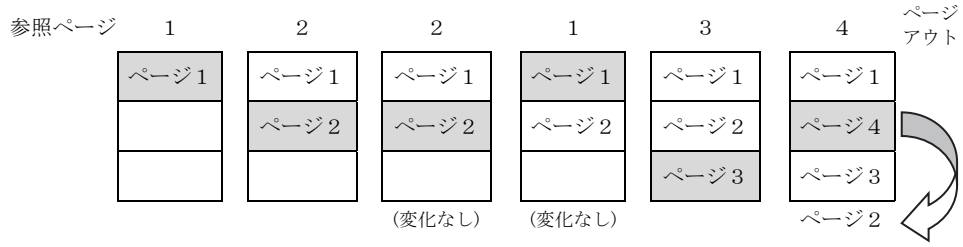
ページ参照順

ページ１→ページ２→ページ２→ページ１→ページ３→ページ４の順に参照するものとする。

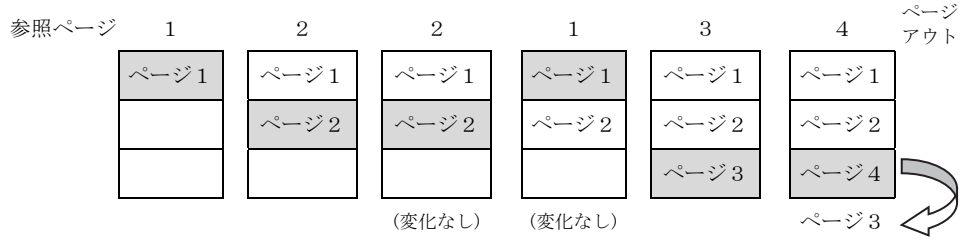
＜ＦＩＦＯの場合＞



＜ＬＲＵの場合＞



＜ＬＦＵの場合＞



（メインメモリのページ枠：３）

【ＯＰＴアルゴリズム】

この方法では近い将来使われるであろう時間的距離が、一番遠いものをページアウトする。この置換えアルゴリズムがＯＰＴ（Optimum）アルゴリズムで、実現されると最良のものとなり、ページフォルトが少なくてすむが、実現されていない。

【プライオリティアルゴリズム】

運用者があらかじめページに対して、メインメモリ内にとどまらせておきたい度合を優先度として設定しておく方式である。最初に設定したプライオリティはある一定時間参照されないと１ポイント下げるという手法を採っている。

【ランダムアルゴリズム】

ネーミングのとおり置き換えられるページに規則性はない。どのページを将来的に参照されるのかは明確ではないので、それと同じようにどれを置き換えたらよいのかもわからない、結局どれをページアウトしてもよいという考え方で作られたアルゴリズムである。

（イ）スラッシング

ページリプレースメントが頻繁に行われると、その処理にＣＰＵ時間を費やされてしまう。これにより動かしたいプログラムが動かなくなる現象を**スラッシング**という。

ページフォルトの回数を単位時間当たりで出したものがページング率である。メインメモリの容量が少ないと、ページフォルトの発生が多くなり、ページフォルトが多くなるとページインとページアウトによる処理が多くなり負荷が上がってしまい、最終的にはスラッシングは発生する。よって、スラッシングはページング率から判断することができるということになる。

実記憶が大きいうちはページング率が小さい値にとどまっているが、実記憶が小さくなるにしたがいページング率が急上昇する。