■7 群（コンピュータ -ソフトウェア）－3 編（オペレーティングシステム）

2 章 ハードウェアとの接点

（執筆者：吉澤康文）［2013 年 2 月 受領］

■概要■

OS はハードウェア，プログラマ，そしてコンピュータ管理者との接点をもっている．この章ではハードウェアと協調して柔軟な機能をプログラマに提供する機構を説明する．まずハードウェアの割込みを整理して説明する. 次に, コンピュータの 3 資源が演算機構の CPU, プログラムやデータを格納する主メモリ，外部記憶である入出力装置を OS がどのような接点をもっているかについて説明する．

【本章の構成】

ハードウェアとの接点として最も重要なのは割込みである．まず，コンピュータの割込み機能と OS の接点となる機能を説明する（2-1 節）．主メモリでは，情報管理の基本的機能である保護機構をセキュリティの観点から説明する（2-2 節）．最後に，多種多様な外部記憶装置のいくつかを紹介し，その概要を説明する．各装置はそれぞれ機能を多種備えているが，仮に全プログラマが各々の装置をそれぞれ取り扱うプログラム作成をしなければならないなら，本来そのプログラマが解決すべきプログラム部分より大きいかもしれない．OS は全プログラマに共通な装置対応のコントロールプログラムを提供することでプログラマの生産性を向上させる目的をもつことを説明する（2-3 節）．

■7群 － 3編 － 2章

2-1 割込み機能

（執筆者：吉澤康文）［2013年2月 受領］

2-1-1 割込みが必要な理由

計算機の生産性（処理能力）を上げるためには，コンピュータ資源を遊ばせることなく使うことである．資源，つまり，演算装置（CPU），主記憶，ファイルは各プロセスに割り当てられている．したがって，資源を遊ばせないということは，プロセスを遊ばせないことと同じである．すなわち，プロセスの資源要求と割り当て，ならびに動作状況を OS は常に把握しておかねばならない．

プロセスがプログラムを実行している過程で割り付けられていない他のプロセスの記憶領域に書込みを行ったり，定義されていない命令コードを実行しようとしたりすることはプログラムにバグ（誤り）があることを意味している．このような場合，OS はプロセスの異常処理を行い，場合によってはプロセスを中断し，保有している資源を取り上げ，ほかのプロセスに割り当てるなどの処置を行う．

このようなプログラムの誤動作を検出するのも重要なハードウェア機構の割込みである．プログラムの誤動作を検出したときは，プロセスを中断し，CPU などの資源を待っているほかのプロセスを起動（スケジュール）することが可能となり，CPU を休みなく使うことができる．つまり，計算機の生産性が向上することになる．

2-1-2 割込み種別と OS の処理概要

割込み機構を 6 種類に分類し，OS が行う処理の概要を説明する．

(1) ソフトウェアの誤り検出

上記 2-1-1 項に示したとおり，ソフトウェアのバグが原因となる割込みが大半である．代表的なものとしては，実装されていない記憶領域をアクセスするアドレッシング誤り，割り付けられていないアドレス空間を参照する記憶保護などである．また，演算ではゼロで割算を行う場合などが代表的である．

プロセスによっては，異常処理ルーチンをプログラミングしてあり，異常が発生したときに制御を受け取ることを OS に宣言している場合がある．そのような場合には，制御をプロセスの指定したアドレスに渡す．フェールセーフなシステム構築をするには，このような異常状態が発生したときのことを予期した設計が必要である．したがって，OS に事前に宣言がない場合には，OS はプロセスを異常修了させる．

ソフトウェアのバグではないが仮想記憶におけるページフォールトがここに分類されているコンピュータもある．

(2) ハードウェアの誤り検出

主メモリから命令をフェッチ（Fetch）し，データへのアクセス時にパリティエラーなどが生じる場合などに通知される割込みである．コンピュータによっては瞬時の電源異常が生じたことを通知するものもある．このような通知はコンピュータの保守に重要な情報であるの

で，OS はログ（Log）として記録する．

(3) 入出力装置の修了報告

一般的に入出力装置は CPU に比べると低速である．したがって，ファイルへの入出力を行うプロセスがその完了まで CPU を占有していたのでは CPU を有効利用できない．そこで，入出力装置を CPU と独立に動作させ，入出力が完了した時点で CPU に完了割込みとして通知する機構が有効である．この詳細は 2-1-3 項で述べる．

(4) タイマ

コンピュータの応用によっては時間の設定や時刻の指定が必要になる場合がある．例えば，端末からの入力が 2 分以内に完了しないときは処理を打ち切るというようなことである．このような監視時計（Watch Dog Timer）は対話処理や機器の監視などに頻繁に利用される．そのためにタイマをセットし，指定した時刻を過ぎると割込みが入る機能が用意されている．OS はタイマを利用するプロセスにその時点で制御を渡す機能を提供する．

(5) システムコール

OS の機能をプログラムの中から要求するために特定の命令が用意されている．詳しくは 2-1-5 項で述べる．この種の命令が実行されると，一般的に，CPU のモードを切り替えて権限の強いモードになり，OS に制御が渡る．このとき，OS には多くの機能が用意されているので，機能コードをパラメータとするのが一般的である．

(6) 外部信号割込み

コンピュータは外部の測定機，操作ボタンなどと組み合わせることで自動制御を行い，人間との交信が可能になる．このような外界からのイベント情報を割込みとして受け取った直後に制御が OS に渡る．OS は外部のイベント情報を収集し，該当のプロセスに取得した情報を所定のメモリに格納した後に制御を渡す．

2-1-3 OS の制御機構

オペレーティングシステムに制御が移動するのは割込みが発生した結果である．**図 2･1** のように，割込み種別に応じて「割込み処理アドレスベクタテーブルへのポインタ」を示す制御レジスタをもつコンピュータがある．このようなコンピュータでは，通常，OS の初期設定の過程で割込み処理の各モジュールアドレスが設定されているので，初期設定完了後は割込み発生時に自動的にハードウェアが該当する OS のモジュールに制御を渡してくる．

2-1-4 多重プログラミングを容易にする割込み機構

プログラムの実行では単に CPU を使用するだけではなく，ファイル入出力を伴うことが多い．一般的に，入出力のなかには機械的な動作を含むものもあるため，**図 2･2** に示すように，逐次的に処理すると貴重な CPU 資源が遊んでしまう．

このため，入出力装置を使用中の処理は一時的に待ち状態にしておき，入出力装置のサービスが完了した時点で割込み信号が CPU に対して発生するようにしておけば，入出力要求後

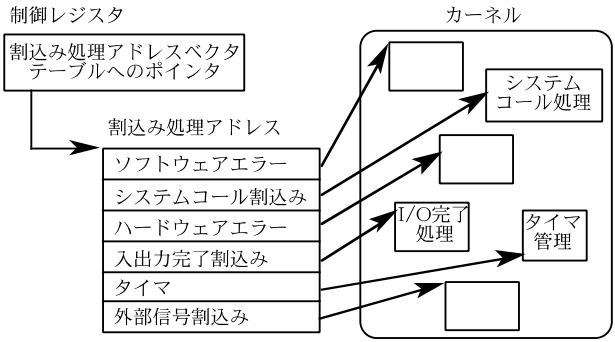

図 2・1　割込み発生により動作するカーネルの仕組み

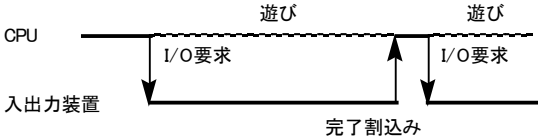

図 2・2　入出力を伴うプログラムの実行過程

の処理を直ちに続行可能となる. 割込みは装置の使用完了のような事象を CPU へ通知する機能であり，複数のプロセスを同時実行させる環境では CPU の割当てに有効な情報源となる.

そこで，複数のプロセスを実行可能な状況にしておき，あるプロセスが入出力要求を出した直後に別のプロセスに CPU を割り当てる．この操作で CPU を遊ばせることなく使用できる.

このように，1 台のコンピュータに複数のプロセスを同時に実行可能にする機能を多重プログラミング（Multiprogramming）と呼ぶ．仮に，コンピュータに複数の入出力装置が接続され，各タクスが別々の装置を使用する状況ならば，CPU を遊ばせることなく利用できる. **図 2･3** には二つのプロセスが二つの別々の装置を使用し，CPU の遊び時間を削減している例を示した.

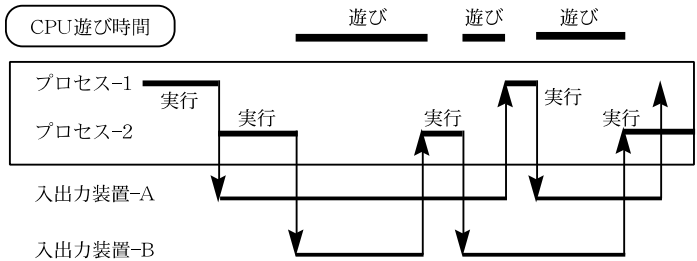

図 2・3　多重プログラミングによる CPU 利用率の向上

2-1-5　OS の機能呼出し

ソフトウェアの全くない裸のコンピュータを受け取り，ソフトウェアを開発する場面を想定してみる．このような環境では，作業の中心が入出力操作のプログラム作りになるであろう．入出力はキーボード，ディスプレイだけではなく，磁気ディスクなどもあり，そのすべてをプログラミングしなくてはならない．もちろん，ブートストラップのプログラムも必用となる．したがって，プログラマはハードウェアのすべての知識をもたねばならない．

先に述べたとおり，入出力操作はすべてのプログラマに共通の機能である．そこで，一つの最良な共通プログラムを開発し，プログラマ相互に利用することがプログラムの生産性向上につながる．繰り返し述べるが，この考えで発展したソフトウェアが OS である．

入出力装置は複数のプロセスから利用されるシステム資源である．同じように，タイマ，メモリなどもコンピュータシステム全体にかかわる資源である．そこで，このようなシステムワイドな資源に対してすべてのプロセスが独立に操作できる仕組みでは混乱が生じてしまう．そのためにシステムワイドな資源に対する操作は，CPU の特権状態（Privileged State）でしか実行できない仕組みになっている．

そこで，このような仕組みのコンピュータでは，システムワイドな資源操作を OS が実行する方が合理的である．それに，一般のプログラマには煩わしいハードウェアにかかわるプログラムは作らずに，自分本来のプログラミングに専念した方がプログラムの生産性が向上する．

図 2･4 にはプロセスよりファイル読込み要求の例を示す．プロセスはファイル読込みの操作をシステムコールとして OS に依頼する．このとき，OS は特権状態になる必要があるため，システムコールは特定の命令を用いて割込みを発生させ，特権状態に遷移する．それはスーパバイザコール命令，システムコール命令，ゲートウェイ命令などと呼ばれており，割込み（割り出し：Exception と呼ぶ場合もある）として動作する．つまり，OS の機能を呼び出すために割込みを用い，CPU の状態遷移を行うのである．

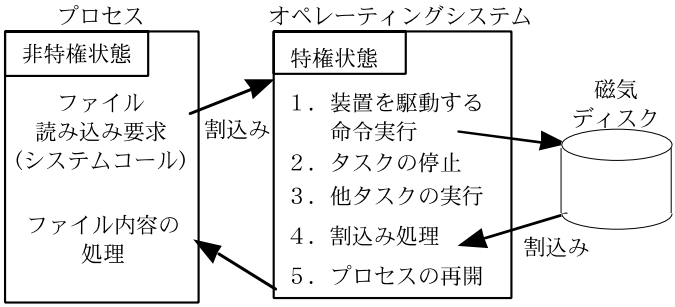

図 2・4　システムワイドな資源の操作を行う OS 機能呼出し機構

■7群 － 3編 － 2章

2-2 主記憶装置

（執筆者：吉澤康文）［2013年2月 受領］

2-2-1 利用可能となった大容量記憶

初期のコンピュータは主記憶容量が少なく，ユーザはメモリ使用に制約があった．そこでOS の課題は，限られたメモリ領域を有効に利用し，多重プログラミングの多重度向上を図る点にあった（詳しくは6章で述べる）．しかし，半導体記憶の出現により事情は一変し．むしろあり余るメモリを使用してコンピュータの性能を向上させる技術が求められるようになったのである．そして，1980年代には大容量メモリを用いた性能向上技術開発が進んだ．

この代表的な例がRAM（Random Access Memory）ディスクである．主記憶に使用しているRAMの一部を磁気ディスクとみなし，頻繁にアクセスされる情報をRAM内に格納しておくことでアクセス時間を短縮する．このような技術を一般的にキャッシング（Caching）と呼び，OSにより入出力のシミュレーションが行われている．1980年代にはキャッシング技術がコンピュータの多くの部分に利用された．つまり，外部記憶装置に対する機械的な動作から生じる遅延時間を短縮するために，電気的な操作で代替えする技術である．

2-2-2 仮想記憶の出現

プログラマにとって「メモリ容量の制限から開放されたい」という願望はとても強かった．この要求に応えたのがページ化された主記憶の発明である．メモリ管理の技術ならびにノウハウ（Know How）はシステムプログラムのほかの分野にも広く流用されている．仮想記憶実現方式と管理方法は5章のハイライトとして後述する．

2-2-3 記憶保護

企業の情報中枢として活躍するコンピュータシステムでは,機密保持は極めて重要である．機密保持の基本的な機能はハードウェアに備えられているが，それをユーザに利用しやすい機能として提供するのもOSの使命である．

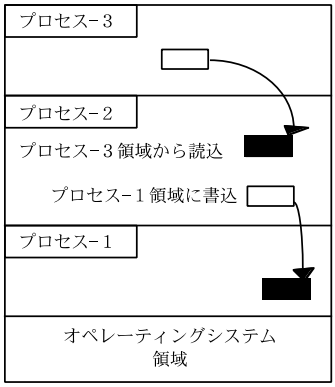

図2・5　多重プログラミング環境での不当アクセス

記憶保護の基本的な機能は，多重プログラミングにおけるプロセス間のメモリ領域の保護であり，他のプロセスからの不当なアクセスを防ぐことにある．**図 2･5** にはプロセス間の不当なメモリアクセスの例を示した．

この図のように，プロセス-2 がほかのプロセス領域に対して書き込みをし，読み出しをしてしまうと正しい処理が行えなくなってしまう．書き込みを行えば，書き込まれたプロセスは正常に処理できないかもしれないし，また不当な書き込みを行ったプログラムも正しい処理を行っていない可能性が高い．

不当なメモリアクセスをしているプログラムでも，たまたま未使用な領域への書き込みを行ったときは，一見正しく動作してしまう．「バグが潜在している」というのはこのようなプログラムのことである．条件が整わないとバグは露呈しないことになる．

■7 群 － 3 編 － 2 章

2-3 入出力装置

（執筆者：吉澤康文）［2013 年 2 月 受領］

2-3-1 共通課題の解決

入出力を行うプログラムは，多くのプログラマに共通の問題として存在していたため，初期の OS は入出力ユーティリティ（Utility）として発展してきた．今日，入出力機器は多種多様に存在するため，それらの制御プログラムをサポートすることは OS 開発において最も多くの労力をとられるところである．プリンタ，モデム，CD-R/RW，MO，DVD，Ether Adapter など多くのベンダが多種の製品開発を行っている．これらに対するプログラム開発は，ハードウェアの詳細な知識が求められるため，一般的に煩雑で難易度が高い．

2-3-2 対話形式のプログラム開発例

対話形式のプログラムを作成しようと考える．**図 2･6** のようにパラメータをキーボードから入力し，プログラムの計算結果をディスプレイに出力する．場合によると，計算過程でファイルから情報を読み出したり，書き込んだりするかもしれない．このようなケースでは，キーボード, ディスプレイ, 磁気ディスクへの入出力を前提にプログラムを作ることになる．更に，入力はキーボードからだけではなく，マウスなどを利用することもある．

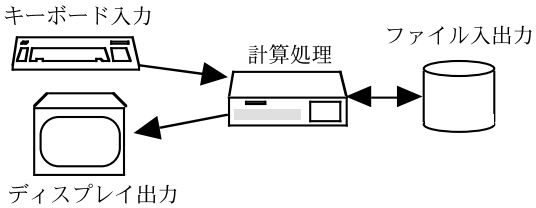

図 2・6　対話処理での入出力使用例

OS はこれらの入出力機器に対する操作をファイルという論理的なインタフェースとしてプログラマに提供する．したがって，プログラマは入出力機器が変わっても，ファイルに関するプログラムインタフェースを変更する必要がない. 高性能な磁気ディスクを導入しても，プログラムを一切変更することなくハードウェアの進歩を享受することができるのである．物理的なインタフェースの変更は OS が担っており，同時に OS はハードウェアの能力を最大限引き出す使命をもっているのである．

2-3-3 入出力の構成例

図 2･7 にパーソナルコンピュータやワークステーションなどの基本的な入出力接続例を示す．コンピュータの各構成要素は共通のデータバスに接続されている．このとき演算装置と主記憶間のデータバスは頻繁なデータ転送を必要とし，演算性能に密接な関係があるため，特別のデータバスを用意している場合が多い．

入出力機器は商用的な意味から標準インタフェースを備えているものが多く，機器の選択の自由度を高めている．OS は入出力機器をプログラマに抽象度の高いファイルとして提供

するが，図 2･7 に示すような各装置の詳細な制御，管理を行う義務をもつ．

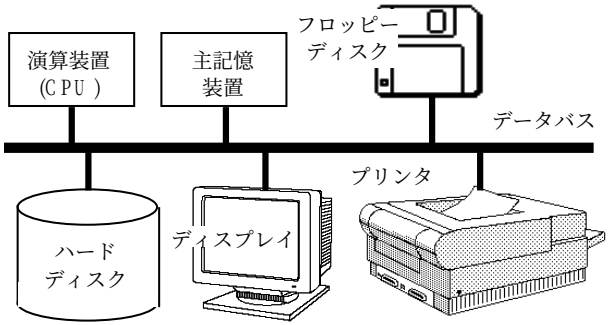

図 2・7　入出力の典型的な接続形態

2-3-4 入出力管理・制御のソフトウェア階層

図 2･8 は入出力ソフトウェアの階層構造を示している．最上位にはユーザプロセス（プロセス）が位置し，ファイルシステムが提供するインタフェースにより入出力要求を行う．ファイルシステムが提供している機能については 3 章で述べることにし，ここではファイルシステムよりも下位に位置する OS の機能について述べる．

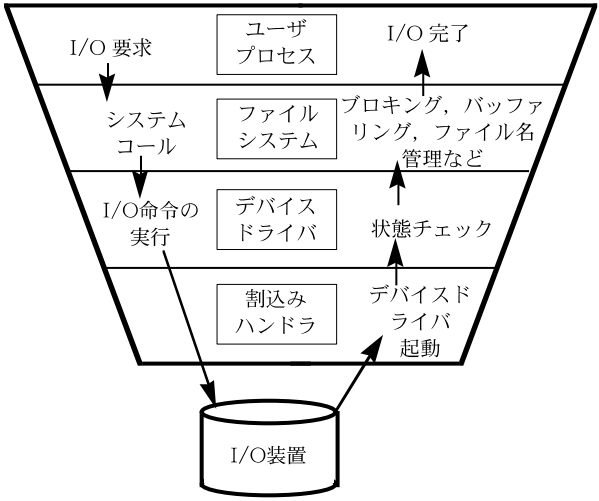

図 2・8　入出力ソフトウェアの階層構造

デバイスドライバ階層の役割は入出力装置に対する起動制御である．したがって，この層では各装置の状態を把握しておく必要がある．つまり，装置がすでに別のプロセスからの入出力要求をサービスしている場合には新たな要求は受け付けるが，入出力操作は延期される．そのために，入出力要求は一般的に待ち行列に入れられ，その先頭がサービスされていると

いう状況になる．

入出力装置への起動命令は 2-1-5 項で示した CPU の特権状態でしか実行できない．つまり，特権命令（Privileged Instruction）なのである．したがって，デバイスドライバはユーザプロセスとは異なる CPU 状態で実行されねばならない．図 2･8 ではファイルシステムの中からシステムコールが実行されて非特権状態から特権状態になったわけである．

入出力装置はデータ転送などの一連の動作が完了すると，割込みを CPU に通知する．この結果，割込みハンドラが起動し，該当するデバイスドライバに制御が渡ってくる．デバイスドライバは入出力操作が正常に行われたか否かをチェックし，それらの情報に基づく処理を行う．入出力が正常に動作しなかったときは，何回か同じ I/O 命令を繰り返し実行することがある．これを，ロールバック（Rollback）と呼ぶ．

デバイスドライバは正常な動作が行われたならば，該当する入出力装置に対する待ち行列をチェックし，待ち状態の I/O 要求を実行する．そして，今完了した入出力要求をファイルシステムに通知する．この段階でファイルシステムは一つの入出力処理が完了したことになる．その後，ユーザプロセスの要求を満たす処理を行いユーザプロセスに制御を返す．

2-3-5 入出力待ち行列管理

一般的に，多重プログラミング環境では複数のプロセスが独立に実行されている．磁気ディスクには複数のファイルが格納されているので，プロセスからの I/O 要求が特定のディスクに集中する可能性がある．このため，デバイスドライバでは**図 2･9** に示すような I/O 要求の待ち行列を作りこれを管理する．

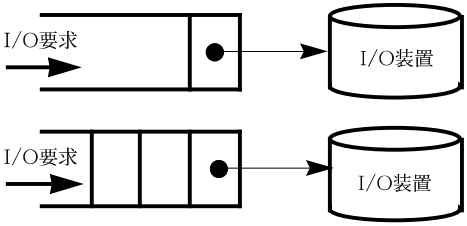

図 2・9　入出力装置への要求待ち行列

待ち行列を設けることにより，各装置がビジーか否かを判定できるばかりでなく，先頭の要求が完了したなら直ちに当該装置に対する次の入出力要求を実行できる．また，優先度の高い要求を先頭の待ち行列に入れることができるので入出力のスケジューリングが可能になる．更に，磁気ディスクなどでは磁気ヘッドの位置の移動時間を最小とするように I/O 要求を待ち行列上に並べ，入出力のスループットを高める工夫が可能になる．

2-3-6 磁気ディスクの機構とアクセス

磁気ディスクはアルミニュームやガラスの円盤上に磁気膜を塗布し記録する媒体である．3.5, 2.5 インチ径の物があり容量が年々増大している．多くのコンピュータで 2 次記録媒体の主力として利用されている．**図 2･10** にその構造を示す．大容量化のために，円盤を数枚重ねる場合がある．同心円上にトラックがあり，トラックに記録の単位としてのセクタ（Sector）

が複数ある．セクタは 512 B（バイト）のように固定長である．

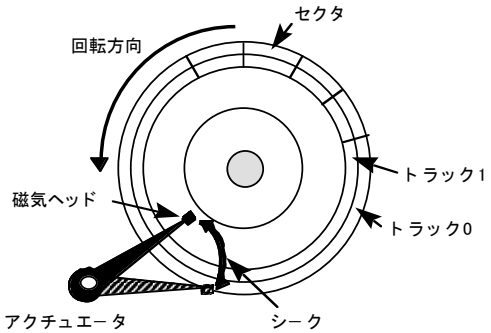

図 2・10　磁気ディスクの構造

磁気ディスクに対する読み出しや書き込みをアクセス（Access）という．アクチュエータが機械的に動作して目的とするトラックまで磁気ヘッドを移動させ，目的のセクタが磁気ヘッドの下を通過するまで待ち，アクセスすることになる．これをもう少し一般的な磁気ディスクについて考察する．

2-3-7 磁気ディスクアクセス時間

図 2･10 の円盤が複数重ねられた場合，ある特定のトラックに注目すると複数のトラックが上下に存在することになる．**図 2･11** に示すように，それらのトラックの集まりをシリンダ（Cylinder）と呼ぶ．これが一般的な磁気ディスクの構造である．

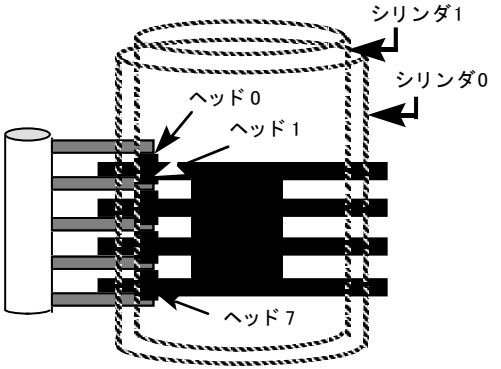

図 2・11　横から見た磁気ディスクとシリンダ

以上から，磁気ディスクへのアクセス過程とその時間は次の四つの操作時間の合計となる．

(1) 目的とするシリンダへアクチュエータを移動：シーク時間（Seek Time）
(2) シリンダ上の目的のヘッドへの切替え時間
(3) 目的のセクタが磁気ヘッドの下まで到着する時間：サーチ時間（Search Time）

(4) 転送時間（Transmission Time）

ここで，機械的な操作時間はシーク，サーチ，そして転送であり，アクセス時間の大半を占めている．これらの時間の割合を実測した例を**図 2･12** に示す．このケースでは，シーク時間が 60 %を占め，サーチが 32 %を占めている．そして転送時間はわずか 8 %である．

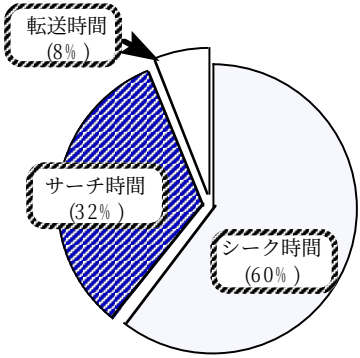

図 2・12　磁気ディスクへのアクセス時間の内訳例

磁気ディスクのアクセスではシーク時間を最小化する必要があり，次にサーチ時間の短縮が望まれる．2-3-5 項で述べたように，磁気ヘッドのスケジューリングを行う目的はこのような分析結果に基づいている．

■7群 － 3編 － 2章

2-4 演習問題

（執筆者：吉澤康文）［2013年2月 受領］

(1) 割込みの具体例をあげ，それらの使用目的を述べよ．
(2) 割込みのもたらす利点を述べよ．
(3) 割込みマスクとはいかなる機能か説明せよ．
(4) 特権状態がある理由と，どのような状態かを説明せよ．
(5) 割込み時に必要な情報は何か考え，なぜ必要かを述べよ．
(6) 割込みは多重プログラミングにどのような利点をもたらしたか述べよ．
(7) チャネルの目的は何かを説明せよ．
(8) 割込みを2種類に分けるとするといかなる分類か．そして上記がどのように分類されるか．
(9) コンピュータが割込み禁止状態でかつ命令をフェッチしない状態はいかなるときか説明せよ．
(10) 割込み禁止で実行しなくてはならない場合はいかなるときか．
(11) 割込み禁止状態が長く続くといかなる状況になるか考えてみよ．
(12) メモリ保護機構が必要な理由をあげよ．また，その機構を2種類あげ方式を説明せよ．
(13) 磁気ディスクのアクセスタイムの要素を述べ，その各々の時間のオーダを確認せよ．
(14) メモリには空き領域がたくさんあり，多くのプロセスが入出力の処理完了を待っている状態にある．そして，CPU の利用率が 30 %であるとき，このコンピュータの運用はどのようにすべきか．