情報処理の基礎・第 15 回

2014 年 1 月 15 日

今回の内容

15.1 オペレーティングシステム (続き)

メモリ管理

メモリ空間とアドレス空間　計算機システム中のプロセスは、CPU のロード命令やストア命令などを利用してメモリ中に記憶された特定のデータにアクセス (読み書き) することができます。CPU がデータの記憶に使用することのできる記憶領域全体を**メモリ空間**と呼び、メモリ空間の中の特定の記憶場所を指定するのに使用されるアドレス (番地) の取り得る範囲を**アドレス空間**と呼びます。一般的な 32 bit CPU では、メモリ空間を 8 bit 単位で区切り、区切られた各領域を区別するために 32 bit の符号なし整数で表現されたアドレスを使用するのが普通です。この場合、CPU は最大約 4 GB までメモリ空間に直接アクセスできることになります。このことを指して「(この CPU は) 4 GB のアドレス空間を持っている」と言うときもあれば、「32 bit のアドレス空間を持っている」と言ったりするときもありますので注意が必要です。また、4 GB のアドレス空間があるからといって、必ずしも 4 GB の記憶領域 (メモリ空間) が利用できるとは限りません。アドレス空間には、実際には使用できない (記憶装置内の記憶領域と対応していない) アドレスも含まれています。このことを敢えて無視して「メモリ空間」という言葉を「アドレス空間」と同じ意味で用いることもあります。

メモリ管理の役割　1 つの入出力装置を複数のプロセスが使用する場合に交通整理が必要なように、計算機のメモリ (主記憶装置) も複数のプロセスが使用しますので、これを管理する必要があります。たとえば、ある機械語プログラムを実行しているプロセスが、(たとえば) `ST A, 1000` というストア命令によって、レジスタ `A` の内容をメモリの 1000 番地に記憶したとします。ところが、このプロセスと並行して実行されている別のプロセス[1]が、同様の機械語命令を実行してしまうと、1000 番地のデータは上書きされて、先のプロセスが記憶したかったデータは失われてしまうことになります。このような問題を解決するためにオペレーティングシステムが行う仕事の 1 つが**メモリ管理**です。オペレーティングシステムがメモリ管理を行う主な目的は以下のようなものです。

アドレス空間の多重化 — 同じアドレスのデータでも、プロセスが異なれば、物理的に独立した領域に記憶されるようにします。アドレス空間の多重化によって、各プロセスは独立したアドレス空間を持つことができ、他のプロセスのことを気にしないでメモリ空間にアクセスすることができるようになります。

メモリ空間の拡大 — 主記憶装置の実際の記憶容量以上のメモリ空間をプロセスに提供します。CPU のキャッシュメモリの部分で説明したように、短い時間だけをとって見ると、プロ

[1] 同じプログラムでも別のプログラムでも構いません。

セスがメモリ空間全体に均等にアクセスしているのは稀で、いくつかの特定のアドレスの周辺にしかアクセスしていないのが普通です。そこで、そのプロセスが最近頻繁にアクセスしている部分にだけ主記憶装置 (メモリ) の割り当てておき、あまりアクセスされていない部分のデータはハードディスクなどの補助記憶装置を使って記憶しておくようにします。ハードディスクの方に記憶されている部分のアドレスに対するアクセスが起こった場合、主記憶装置 (メモリ) の一部分をその部分に割り当て直してから、ハードディスクに退避してあったデータを戻し、それから CPU によるメモリ空間へのアクセスを続行するようにします。主記憶装置は CPU から高速にアクセスすることはできますが、その容量は限られています。計算機システムで多くのプロセスが並行して動作していると、各プロセスが利用できるメモリの容量は限られたものになってしまいがちですが、機械語プログラムの実行にさしあたり関係のない部分のデータを、主記憶装置から、より低速ではあるが容量に余裕のある補助記憶装置に移動することで、主記憶装置の実際の記憶容量以上のメモリ空間を各プロセスに提供することができるわけです。

データの保護 — 別のプロセスやカーネル自身が使っている記憶領域のデータを、正当な権限なく覗き見たり、書き換えたりすることができないようにします。

仮想記憶システム

前節で説明した「アドレス空間の多重化」や「メモリ空間の拡大」を実現するメモリ管理の仕組みは**仮想記憶システム**と呼ばれます。仮想記憶システムの実現方法はいくつかありますが、この節では最も一般的な**ページング方式**[2]と呼ばれている仮想記憶の方法の概略を紹介します。ページング方式の仮想記憶システムは、Windows や Linux など、私たちが日頃お世話になっているオペレーティングシステムでも採用されています。

ページ　ページング方式の仮想記憶システムでは、メモリ空間を、数 KiB 程度 (たとえば 4096 B) の大きさの**ページ**と呼ばれる単位に分割して管理します (図 1)。通常、ページの大きさは 2 の冪乗 byte となるようにしますので、アドレスの上位の bit 列を見るだけで、どのページに属するアドレスなのかがすぐに判別できます。たとえば、1 ページの大きさが 4096 B の場合、$4096 = 2^{12}$ ですから、CPU が (1 byte のデータを単位とした) 32 bit のアドレスを使用している場合、アドレスの上位 20 bit をページを識別するための番号として使うことができます。

短い時間だけをとって見ると、各プロセスは、いくつかの特定のページ (に属するアドレス) にしかアクセスしていないので、そのプロセスが最近頻繁にアクセスしている (いくつかの) ページだけに主記憶装置 (メモリ) のページを割り当てておき、あまりアクセスされていないページの内容は、ハードディスクなどの補助記憶装置に退避しておきます。

仮想アドレスと物理アドレス　主記憶装置 (メモリ) が割り当ててあるページであっても、機械語命令で使われているアドレスと、そこに割り当てられている物理的なメモリ (メモリモジュールなど) で使われているアドレスとは必ずしも一致しませんので、機械語命令によって指定されているアドレスを、それに対応する物理的なメモリのアドレスに変換する必要があります。この変換は

[2] **デマンドページング方式**とも呼ばれます。

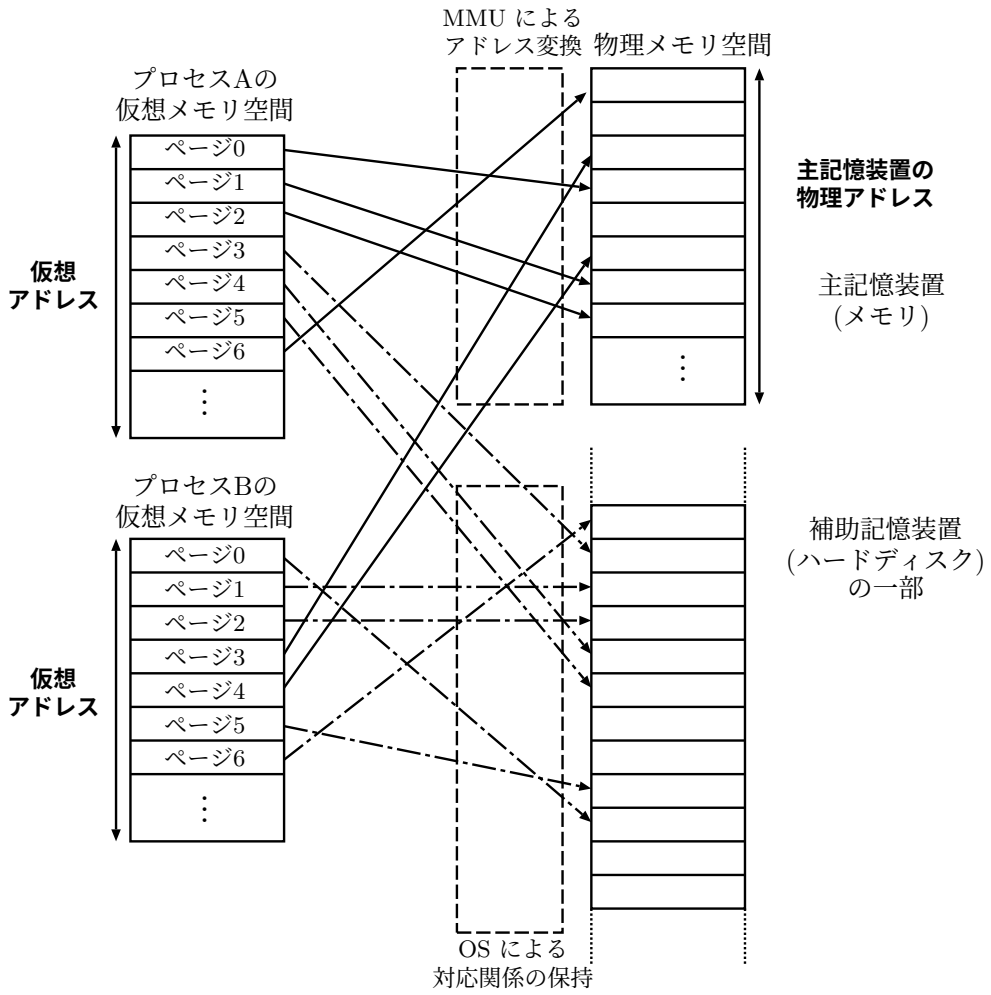

図 1: ページング方式による仮想記憶の仕組み

MMU と呼ばれる装置[3]がハードウェア的に高速に行ってくれます。機械語命令で使用されている変換前のアドレスを**仮想アドレス**と呼び、そのアドレス空間を**仮想アドレス空間**と呼びます[4]。一方、物理的なメモリに対して与えられる変換後のアドレスを**物理アドレス**と呼び、そのアドレス空間を**物理アドレス空間**と呼びます。

仮想アドレス空間はプロセス毎に用意して、プロセスが違えば、同じ仮想アドレスでも異なる物理アドレスに対応するようにしておきます。これによってアドレス空間の多重化が実現できます。オペレーティングシステムがプロセスを実行中の状態にする際には、あらかじめそのプロセス固有の変換が行われるように MMU を制御します。

ページフォルトとページイン、ページアウト　物理的なメモリが割り当てられていない仮想アドレスに CPU がアクセスすると、MMU によるアドレス変換が失敗します。これを**ページフォルト**と呼びます。ページフォルトが発生すると、そのメモリアクセスをとりあえず棚上げにしておいて、オペレーティングシステムは、ハードディスクに退避しておいたそのページのデータを主記憶

[3]MMU (Memory Management Unit) は CPU に内蔵されているのが普通です。CPU のピン (電気的な端子) からは変換後のアドレス (物理アドレス) が出力されます。

[4]**論理アドレス**と言ったり**論理アドレス空間**と言ったりすることもあります。

装置 (メモリ) の使っていないページにコピーし、その (物理) ページをページフォルトが発生した (仮想) ページに割り当てます。もし主記憶装置に空きページがない場合は、最近アクセスしていないページをハードディスクに退避して空きページを作ってから同じことを行います。このようにして、ページフォルトが発生した (仮想) ページに無事主記憶装置の (物理) ページが割り当てられると、オペレーティングシステムは棚上げにしておいたメモリアクセスを続行させます。

ハードディスクなどの補助記憶装置に退避されていたページに主記憶装置のページが割り当てられることを**ページイン**、逆に、主記憶装置からの割り当てがされていたページがハードディスクなどの補助記憶装置に退避されることを**ページアウト**と呼びます[5]。

ページインやページアウトが起こると、ハードディスクなどの補助記憶装置へのアクセスが発生します。ハードディスクは主記憶装置 (メモリ) に比べると非常に低速ですから、ページインやページアウトの処理には非常に時間が掛かってしまい、その間、ページフォルトを起したプロセスの実行は中断されてしまいます。その計算機に搭載されている主記憶装置 (メモリ) の容量に対して、実行されているプロセスが必要としているメモリの容量が大きければ大きい程、また、各プロセスがアクセスするアドレスのばらつき具合が大きければ大きい程、ページインやページアウトの頻度が高まり、計算機が行わなければならない余計な仕事が増えて、その分、各プロセスが本来やるべき仕事が滞ることになります。ひどい場合は、計算機はページインやページアウトに伴うハードディスクの読み書きの処理に掛かり切りになり、本来の仕事はほとんど進行しないといった状態になります。このような状態は**スラッシング**と呼ばれます[6]。

15.2 演習問題

1. 64 MiB の主記憶装置 (物理メモリ) を持つ計算機で、ページの大きさが 4 KiB であるような仮想記憶システムを持つオペレーティングシステムが稼働しているものとします。また、主記憶装置のメモリのレイテンシは 15ns、ページアウトやページインのための補助記憶装置として使っているハードディスクの平均レイテンシは 15ms であり、これらの記憶装置のスループットは十分大きくて、その影響は無視できるものとします。このような計算機システム上で 1 つのプロセスが実行されていて、このプロセスは 64 MiB の物理メモリの内、50 MiB を占有できているものとします。ただし、1 KiB $= 2^{10}$ B、1 MiB $= 2^{20}$ B です。

   (a) このプロセスが、仮想アドレス空間中の 500 MiB のメモリ領域に、全くでたらめな順序 (アドレス) で 4 byte ずつ何度も繰り返しアクセスする場合、1 回のメモリアクセスでページフォルトが起こる確率はどの程度と考えられるでしょうか。また、この状況で、CPU が (仮想アドレス空間での) 1 回のメモリアクセスを行うのに必要な時間は平均してどの程度と考えることができるでしょうか？

   (b) このプロセスが、仮想アドレス空間中の 500 MiB のメモリ領域の先頭から、アドレスの小さい順に 4 byte ずつ、500 MiB すべてのデータにアクセスすることを何度も繰り返

[5] それぞれ、**スワップイン**、**スワップアウト**と呼ぶこともあります。

[6] 本来行いたい仕事の内容から考えると、特にファイルの読み書きを多くする必要はないはずなのに、パソコンがハードディスクへのアクセスばかりを行ってアプリケーションソフトウェアの実行が進まない場合は、このような状態になっていることが考えられます。このような場合は、並行して実行しているプロセスの数を減らしたり、パソコンにメモリモジュールを追加するなどの措置が必要となります。

す場合、1回のメモリアクセスでページフォルトが起こる確率はどの程度と考えられるでしょうか。また、この状況で、CPU が（仮想アドレス空間での）1回のメモリアクセスを行うのに必要な時間は平均してどの程度と考えることができるでしょうか？

15.3　付録：ファイル管理

ファイル　計算機の記憶装置に格納されたプログラムやデータ等のまとまりの1つ1つを一般に**ファイル**と呼びます。機械語プログラム、ワープロの文書、画像ファイルなど、いろいろなディジタル情報がファイルとして記憶されますが、それぞれのファイルは、ある長さのビット列に過ぎません。通常のオペレーティングシステムでのファイルの大きさ (長さ) は1 byte (8 bit) 単位になります。

1つのファイルの内容 (どのようなビット列であるのか) は、オペレーティングシステムによって、ハードディスクや CDROM、フラッシュメモリなどの補助記憶装置に格納されます。ワープロなどのアプリケーションプログラムは、オペレーティングシステムを介して、この内容を読み書きします。CDROM のような読み込み専用の記憶装置 (媒体) にファイルが置かれている場合は、当然、その内容を変更することはできませんが、ハードディスクやフラッシュメモリのように、書き込みもできる記憶装置の場合は、すでにあるファイルの内容を書き換えたり、ファイルを削除したり、あるいは、新しくファイルを作成したりすることができます。

ファイルの名前空間　計算機の補助記憶装置にはたくさんのファイルを作成することができますが、これらを区別するために、それぞれファイルには、それを指し示すための名前 (文字列) が付けられるようになっています。1つのオペレーティングシステムにおいて、ファイルを指し示すために使われる名前 (文字列) 全体を、ファイルの**名前空間**と呼びます。名前空間に属する文字列[7]が、それぞれどのように特定のファイルを指し示すのかの決まりが定められています。

最も単純な名前としては「レポート `1.doc`」とか「`test.c`」のようなものが考えられますが、補助記憶装置には膨大な数のファイルを作ることができますので、このような単純な名前だけで、すべてのファイルを区別するのは困難です。そこで、補助記憶装置に記憶されているファイルをグループ化し、「どのグループのどのファイル」というような指し示し方ができるようになっているのが普通です。この「グループ」は、一般に**ディレクトリ**と呼ばれ、ファイルの「置き場所」として働きます。異なるディレクトリに置かれているのであれば、異なる2つのファイルが同じ「`test.c`」という名前を持つことができます。ディレクトリは「フォルダ」と呼ばれることもあります。

多くのオペレーティングシステムでは、1つディレクトリの中に、さらに別のディレクトリを置くことができるようになっています。1つのディレクトリに対して、そのディレクトリが置かれているディレクトリのことを**親ディレクトリ**と呼び、逆に、1つのディレクトリに対して、そのディレクトリに置かれているディレクトリのことを**子ディレクトリ**と呼びます。1つのディレクトリの子ディレクトリは複数ある場合もありますし、まったく無い場合もあります。

パス名　次ページの図2は、Linux におけるディレクトリ階層 (一部) の例です。すべてのファイルやディレクトリの祖先となっているディレクトリは**ルートディレクトリ**と呼ばれます。ディレクトリの構造が階層的な場合、ファイルの名前空間も階層的な構造を持つことになります。このとき使われる文字列は「... というディレクトリに置かれた ... というディレクトリに置かれた ... というディレクトリに置かれた ... というファイル」いったような意味を持っており、一般に**パス名**と呼ばれます。「パス名」は、ファイルやディレクトリの住所の書き方ようなものです。Linux の

[7]後述する「パス名」などのことです

例では、/ で始めて、ルートディレクトリを起点として目的のファイルやディレクトリに至るまでのディレクトリの名前を / で区切って並べて「/usr/bin/ls」のように書いた**絶対パス名**[8]や、カレントディレクトリ (そのプロセスが動いているディレクトリ) を起点として、そこから目的のファイルやディレクトリに至るまでのディレクトリの名前を / で区切って並べた「bin/ls」のように書かれる**相対パス名**[9]が用いられます。あるプロセスのカレントディレクトリの (絶対) パス名が /usr である場合、そのプロセスにとっては、/usr/bin/ls も bin/ls も同じファイルを指すパス名となります。

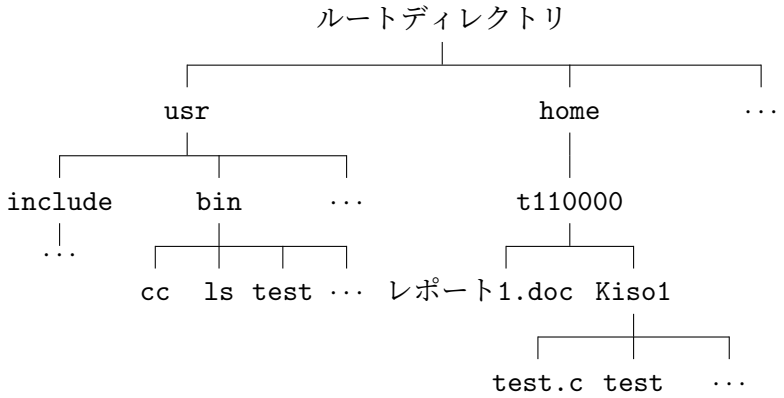

図 2: Linux におけるディレクトリ階層の例

ファイルシステム　1 つのファイルは、補助記憶装置 (たとえばハードディスク) の中で、連続した領域を占めるとは限りません。1 つのファイルの内容は、オペレーティングシステムによって、512 B ～ 32 KB くらいの大きさの**ブロック**と呼ばれる単位に分割され、このブロックを単位として、次ページの図 3 のように (ハードディスク内の) いくつかの場所を使って記憶されます。

このためオペレーティングシステムは、どのファイルのどの部分が (ハードディスクなどの) 補助記憶装置のどこに置かれているのかを知っていなければなりません。また、それぞれのディレクトリにどのような名前のファイルが置かれているかが分からないと、与えられたパス名によって指し示されたファイルの記憶場所に辿り着くことが出来なくなってしまいます。このような (ファイル管理のための) 付加的な情報も、一定の約束事に従って補助記憶装置の一部に格納されます。

いろいろなファイルの内容を補助記憶装置にどのような仕組みで記憶するのかについて約束事、あるいはその約束事に従って補助記憶装置の中に構築されたデータの構造を**ファイルシステム**と呼びます。ハードディスクなどの補助記憶装置の全体、あるいはその一部を、ファイルシステムの約束事にしたがって初期化する (決まったデータ構造を構築する) ことを「**(論理) フォーマットする**」と言います。補助記憶装置を使ってファイルを記憶するためには、まずフォーマットを行って、そこにファイルシステムを構築することが必要ですが、このとき (多くの場合) 補助記憶装置 (の該当部分) に記憶されていたデータは上書きされて消えてしまいますので注意が必要です。

1 つのオペレーティングシステムでも、(状況に応じて) 複数の方式のファイルシステムを使用することがあります。Windows の FAT32 や NTFS、Linux の ext2 や ext3 は、このようなファイル

[8]絶対パス名は必ず / で始ります

[9]相対パス名は / 以外の文字で始ります

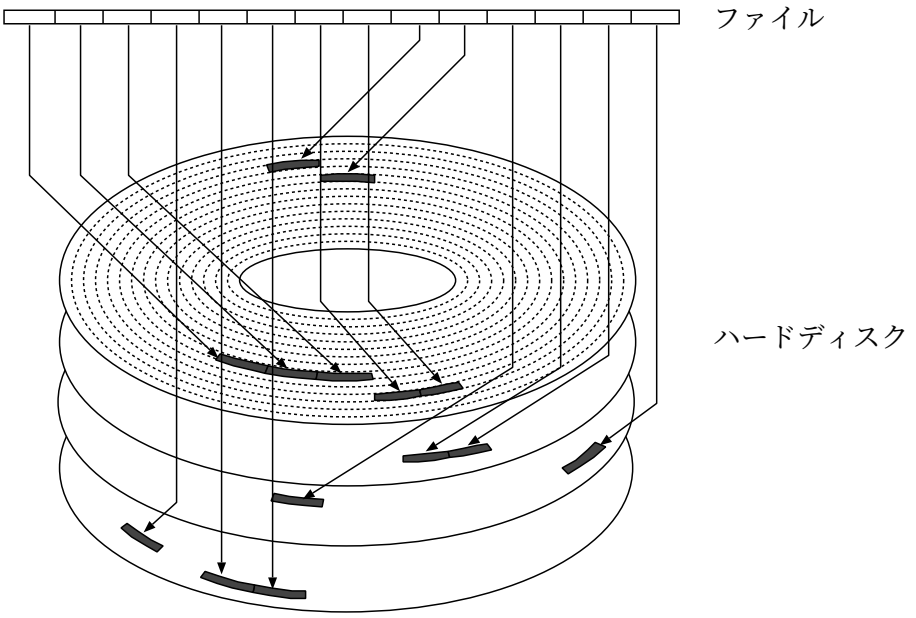

図 3: ハードディスク中で1つのファイルが占める領域の例

システム (の方式) に付けられた名前です。ファイルシステムが違えば、そこに記憶されているファイルの読み書きのやり方が異ってきますので、そのファイルシステムに対応しているオペレーティングシステムでないと、ハードディスク中のファイルの内容を読み書きすることはできません。

断片化　ハードディスクなどの補助記憶装置の構造上、できるだけ連続した領域に記憶する方が、より高速にファイルの内容全体を読み書きできるようになります。しかし、アプリケーションプログラムの指示によって、多くのファイルが作成、削除されたり、各ファイルの大きさが変化することで、どうしても補助記憶装置の記憶領域は虫食い状態となり、必ずしも連続した領域を 1 つのファイル全体のために使用することができなくなります。オペレーティングシステムは、できるだけ連続した領域を 1 つのファイルに割り当てようとしますが、これが困難な場合は (図 2 のように) いくつかのブロックに分割して 1 つのファイルの内容を記憶するわけです。

ファイルの内容が細く分割されて、(ハードディスクなどの) 補助記憶装置の記憶領域内に散らばってしまうことを**断片化 (フラグメンテーション)** と呼びます。たくさんのファイルが作られて、補助記憶装置の記憶容量が残り少なくなった状態で、ファイルの作成や削除が繰り返されたり、ファイルの大きさの変更が繰り返されると断片化が起きやすくなりますが、その程度は、使われているファイルシステムが何であるかによって違ってきます。たとえば、FAT32 は、NTFS や ext2、ext3 に比べると断片化が起きやすいという性質を持っています。断片化がひどくなると、1 つのファイル内容を読み書きするためにハードディスクのあちこちにアクセスする必要が出てきますので、ファイルの読み書きにより時間が掛かるようになります。

演習問題

あるハードディスクの全体を論理フォーマットして、ある方式のファイルシステムを構築して使っているものとします。このファイルシステムでは、ファイルを 2 KB の大きさのブロックに分割して記憶することになっています。このハードディスクは 80 GB の記憶容量を持っており、そのた

めに 2 枚のプラッタの両面 (計 4 面) が使われています。このプラッタは 7200 rpm (毎分 7200 回転) で回転しています。また、このハードディスクのスループットは 40 MB/s であり、(プラッタの回転待ち時間を含めた) レイテンシは、平均 12 ms、最大 25 ms となっています。

1. このハードディスク (ファイルシステム) に、大きさが 100 MB のファイルが記憶されているとします。このファイルの断片化が全く起こっていないものとすると、このファイルの内容を先頭から順にすべて読み出すのに、どの程度の時間が掛かると考えられるでしょうか？
2. 同じファイルが、この上なく断片化しているとして、オペレーティングシステムが、このファイルの内容を先頭から順にすべて読み出すのに、どの程度の時間が掛かると考えられるでしょうか？

情報処理の基礎・第 15 回・終り