入出力の管理についてです。教科書は、第5章の入出力の制御です。

オペレーティングシステム
（2024年 第７回）

I/Oシステムについて

前回の課題について

詳細は「第6回の課題について.pdf」をみてください

（1）アドレス値をページサイズで除算すると、商がページ番号で余りがページ内オフセットとなるので、以下のようになる

① ページサイズが4096バイトの場合　20716÷4096＝5…236 → 5ページのオフセット236

② ページサイズが8192バイトの場合　20716÷8192＝2…4332 → 2ページのオフセット4332

（2）・時間的なもの … メモリアクセスの増加

プログラムでメモリをアクセスする際に必ずメモリ内のページテーブルをアクセスして実ページ番号を得、それによって再度メモリをアクセスすることになり、一度のメモリアクセスに対して2回のメモリアクセスが必要になる（多段のページテーブルにすると、さらにその分メモリアクセスが増える）

さらに、デマンドページングと組み合わせて仮想記憶を構成すると、ページイン・ページアウトでの二次記憶アクセスに要する時間が大きなオーバーヘッドになる

・空間的なもの … ページテーブルサイズの増加

ページテーブルはメモリ内に置くもので、4KBページで32ビットの仮想アドレス空間の場合、100プロセスで400MBなどと無視できないサイズになる

入出力（I/O）装置

- プログラムと実世界が情報のやりとりを行う装置全般
- 代表的な入出力装置
  - 二次記憶装置 … プログラムやデータの格納
    磁気ディスク装置、光ディスク装置、USBメモリ、SSD など
  - 通信装置 … 他のコンピューターと情報交換を行う
    有線LAN、無線LAN、モデム、ネットワークアダプタ など
  - 表示装置 … データの表示
    ディスプレイ など
  - 操作のための装置 … 指示を与えるための装置
    キーボード、マウス、タッチパネル など
  - 印字装置 … 情報を印刷する装置
  - イメージ入力装置 … 画像情報の読取り
    スキャナ、カメラ など
  - 音声信号入出力装置 … 音声信号の入力、デジタル信号の出力

入出力装置はプログラムと実世界が情報のやり取りをおこなう装置全般で、コンピューティングに不可欠なものです。OSの発展においても資源管理など入出力装置から始まった側面もあります。

CPUやメモリは機能が明確であり、仮想化や抽象化を考えるには単純といえるかもしれませんが、入出力装置は多様です。

入出力装置は実世界のアナログ情報をデジタル情報に変換する機能を持っていますが、装置はただの数値を扱うだけであり、OSでその数値の解釈を行うことはありません。

装置から得られる数値を解釈するのはアプリケーションプログラムの役割です。

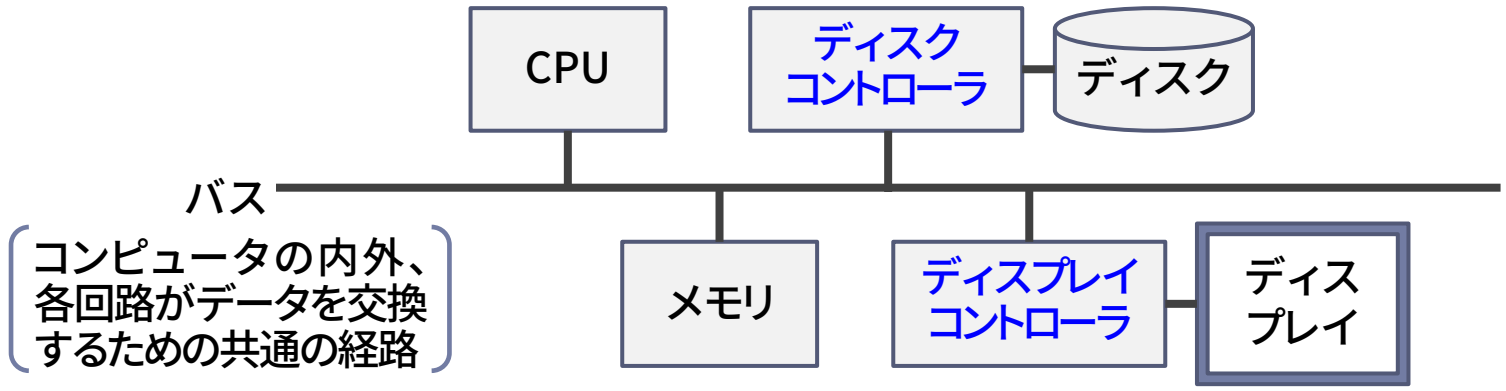
入出力装置（デバイス）とプログラム

- 入出力装置とのデータのやり取りはデバイスコントローラを介して行う

  例：ディスクコントローラ

  … CPUからの指示に対応して、ハードディスクなどのディスク装置を制御する装置。ディスクドライブ自体に内蔵されている場合もある

  - プログラムから指示が与えられると、それを電気的な信号に変換する
  - 入出力操作の結果を参照できる形式に変換する
  - プログラムからアクセスできる次のレジスタ（I/Oレジスタ）を持つ
    - 転送するデータを一時的に格納するレジスタ（データレジスタ）
    - 入出力の結果を示す状態をCPUに示すためのレジスタ（状態レジスタ）



入出力装置との情報のやり取りは、数値の列を扱いますが、入出力装置の状態を確認し、データの入出力を行うというものです。

プログラムからみると、装置側で数値に変換したデータや入出力の状態を保持するレジスタ（I/Oレジスタ）があって、それを参照したり書き込んだりするという形式になります。

I/Oレジスタのどれかに値を書き込むと、たとえば「read」といった機能を指定することになり、別なI/Oレジスタに読み出した値が入ると考えるものです。

このようなやり取りを受け持つものがデバイスコントローラです。

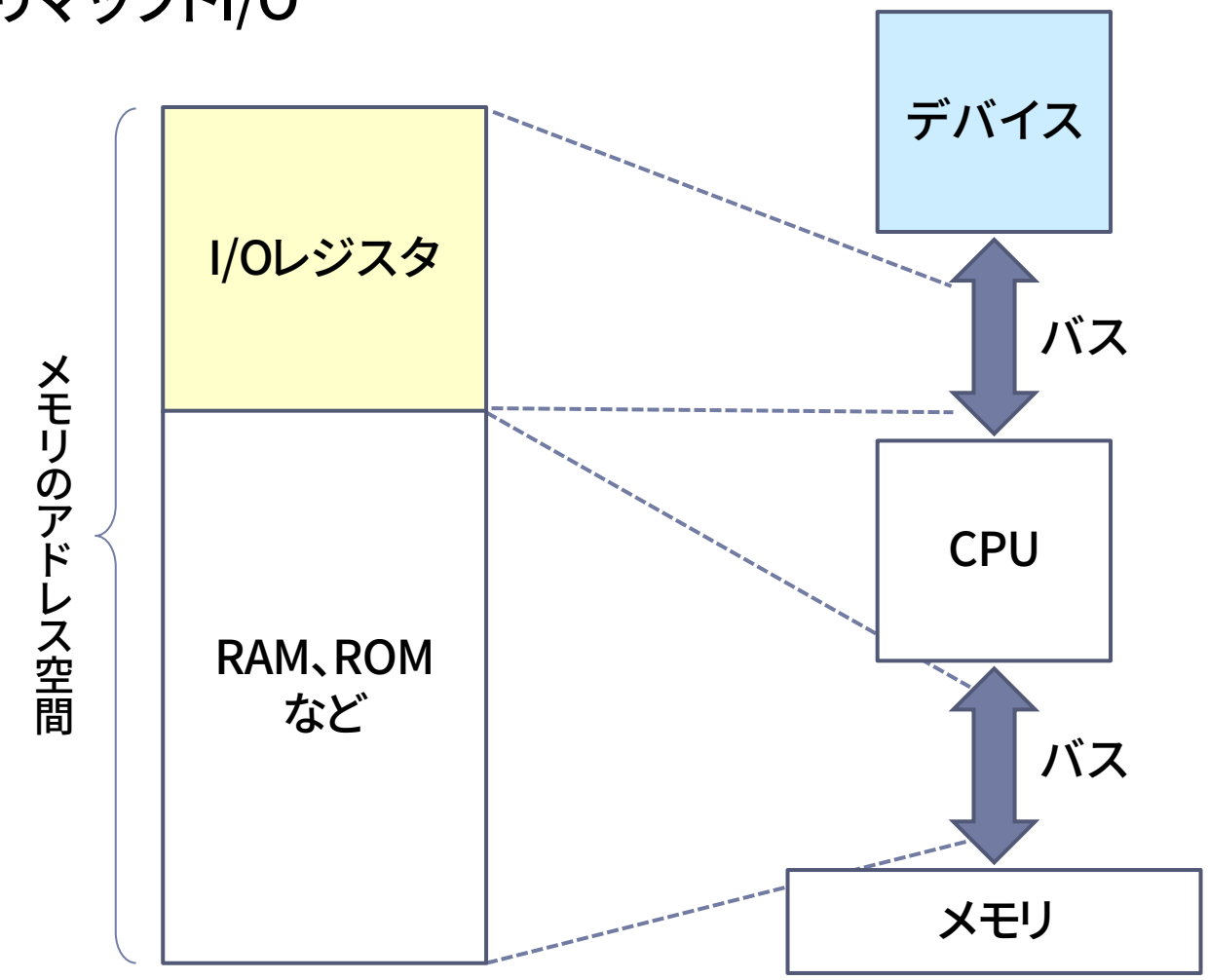
- 入出力装置とのデータのやり取りの手順
  - 状態レジスタやデータレジスタを操作することで、設定の変更や状態の確認、データの入出力ができる
  - 割込みによる処理（入出力操作の完了通知）
- データの転送をすべてCPUで行う方式（プログラムドI/O）
  - メモリマップト I/O（memory mapped I/O）
    - 上記レジスタ（I/Oレジスタ）はメモリ空間の一部に見え、プログラムからは通常のメモリと同じようにアクセスできる→ソフトウェア的にはメモリの一種
    - 多くのプロセッサで使われている
  - ポートマップト I/O
    - メモリ空間とは別のI/O空間に割り付けられたポートに対して、CPUのI/O命令を使ってアクセスする（inやoutという命令）
    - x86で使われている
- データの転送をCPUを経由しない方式
  - DMA（Direct Memory Access）方式

入出力装置とデータをやり取りするには、状態やデータを保持するレジスタを操作し、第3回の講義でみた割込みによって入出力操作が完了したことを通知します。

このデータの転送のやり方によって、入出力の方式が分かれます。

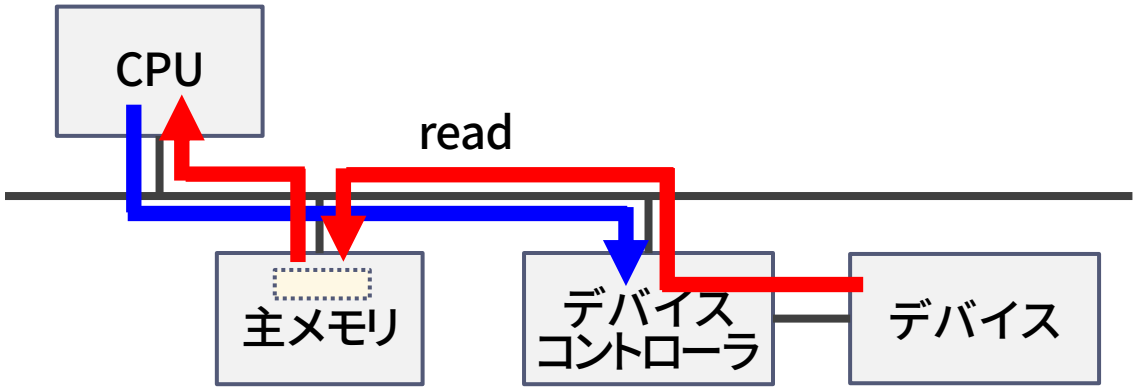
まず、データの転送をすべてCPUが行うプログラムドI/O方式と、データの転送にCPUを経由しないDMA方式とに分かれます。プログラムドI/O方式はさらに、I/Oレジスタがメモリ空間の一部に見えて通常のロード命令やストア命令でアクセスできるメモリマップトI/Oと、メモリ空間とは別のI/O空間があり専用のI/O命令によってアクセスするポートマップトI/Oとがあります。

メモリマップトI/O

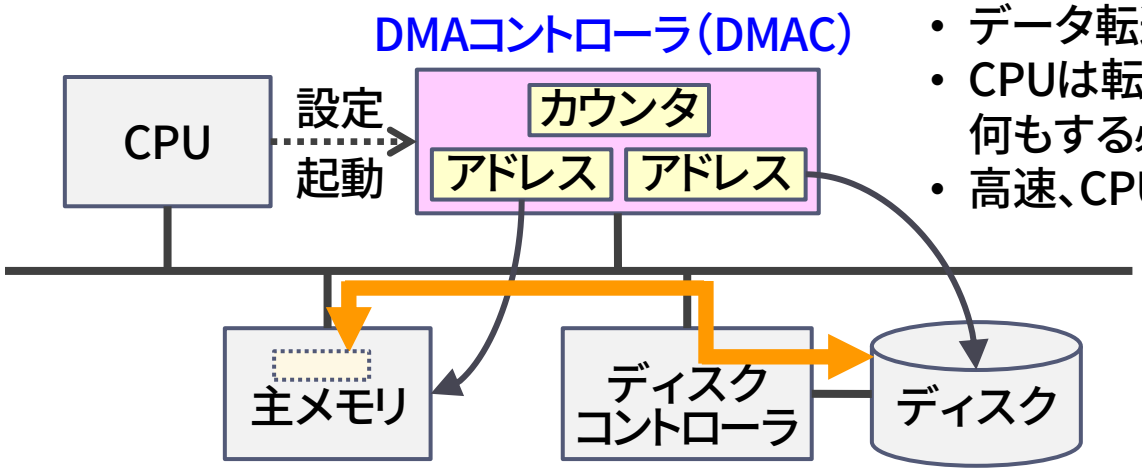


メモリマップトI/Oを図示するとこのようになります。CPUから出るアドレスのうち、あるものはデバイスコントローラのレジスタ（I/Oレジスタ）をアクセスする指定になっているということです。

プログラムド I/O方式



DMA方式



- データ転送はDMACが行う
- CPUは転送が完了するまで何もする必要がない
- 高速、CPUに負荷をかけない

プログラムドI/O方式では、CPUからデバイスコントローラに命令が出されます(メモリマップトI/Oでは、I/Oレジスタの領域に書き込んだりします)。すると、デバイスコントローラが動作してデバイスからデータを得てI/Oレジスタに書き、それがメモリに書き込まれることになります。

DMA方式は、下方の図のように、データの転送をDMAコントローラ(DMAC)が行うもので、CPUは転送が完了するまで他の仕事をすることができます。入出力の完了は割込みで通知します。

この例のようにディスクに対する場合、ディスクのどこ(アドレス)から、どれだけのサイズ(カウンタ)で、メモリのどこ(アドレス)に読み出すかをCPUがDMACに設定し、起動すると、その後はDMACによって、ディスクからメモリにデータが転送されます。

入出力管理の目的

- オペレーティングシステムの入出力管理の機能と目的
  - 入出力装置の仮想化 … 一定の形式に従った操作で装置依存性を排除
  - APIの提供 … 標準的なアクセス方法を定義し、APIを提供
  - 調停と保護 … 許可された操作だけを提供
    マルチプロセス環境での利用時の調停
- CPUやメモリとI/Oの仮想化の違い
  - 装置特性、利用法が多種多様
  - 機能が様々、統一的な概念を提供しにくい
- 共通的な性質の多い入出力装置が標準的な形式で提供されている
  - 二次記憶装置：ファイルとして仮想化
  - 通信装置：ソケットとして仮想化
  - 操作装置と表示装置：ウィンドウシステムとして仮想化

OSが行う入出力管理の目的は、入出力装置を直接アプリケーションに操作させず、仮想化して装置への依存性を少なくする、標準的なアクセス法を定義してAPIを提供する、許可された操作だけをアプリケーションに提供することなどになります。

CPUやメモリと違って入出力装置は様々な機能をもっているため、個々の装置に依存する部分が極めて大です。そこで、共通的な性質の多いものを標準的な形式で提供することになります。二次記憶装置、通信装置、表示装置が主なものになります。

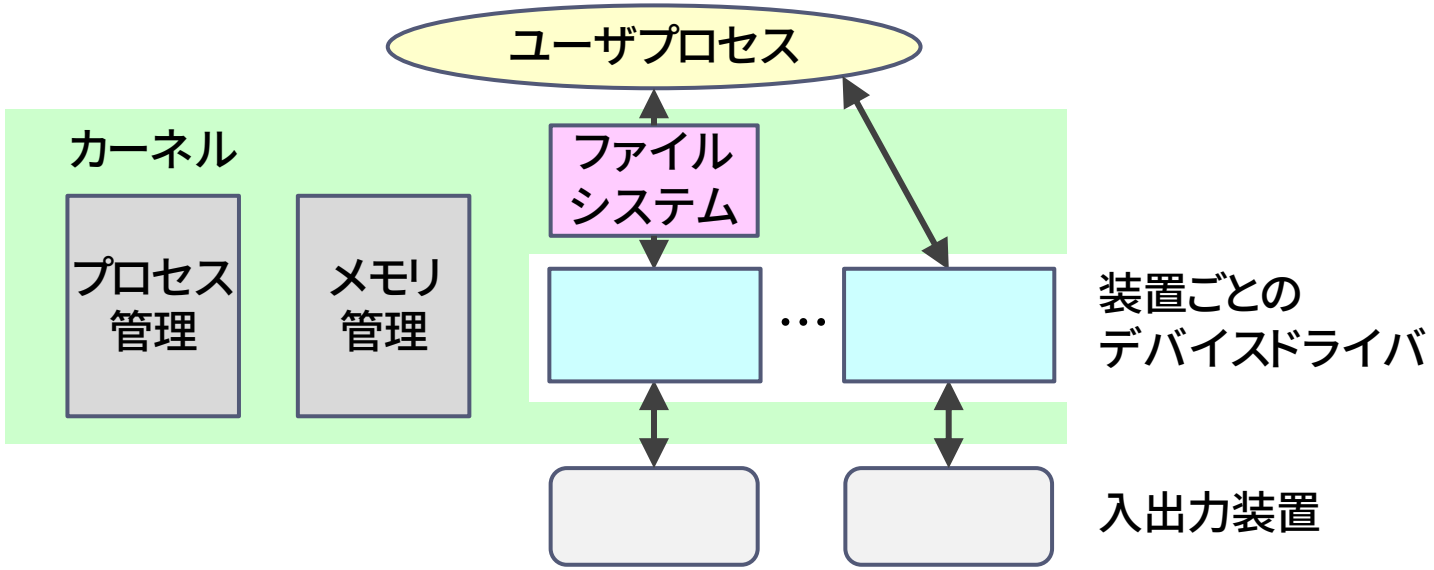
オペレーティングシステムにおける入出力装置の考え方

- オペレーティングシステムで仮想化・抽象化して取り扱う入出力装置の特徴（条件）
  - 利用者やアプリケーションからの要求が高い
  - 装置が持つ共通の性質が比較的明確
  - 仮想化された資源の形式が比較的安定
  - 性能や保護の観点からオペレーティングシステムで扱うことが妥当
  - 仮想化しても性能の劣化を招かない
- オペレーティングシステムとアプリケーションの切分け
  オペレーティングシステムでしか行えないことをオペレーティングシステムで実施
  それ以外の処理はアプリケーション層のサービスで実装
  - 例：プリンタサービス
    - サービス自身はユーザプロセスとして実装、印刷要求をプロセス間通信で受領
    - プリンタ装置そのものへのバイト列などのデータ転送処理はオペレーティングシステムが行う

OSにおける入出力装置の考え方は、要求が高く、共通の性質が明確で、性能や保護が重要であるようなものを仮想化して扱うということであり、装置との基本的なデータ転送処理をOSで行い、サービスと考えられるものはライブラリやサービスプログラムで行うというものです。

入出力管理とデバイスドライバ

- 個々の入出力装置固有の処理については、「デバイスドライバ」で装置制御の処理ソフトウェアをオペレーティングシステムから分離
  - 開発メーカが提供するデバイスドライバをオペレーティングシステムに組み込むことで、装置ごとの処理が行える
    - オペレーティングシステム開発者にとっては入出力装置を組み込む手順が削減
    - 入出力装置の開発者にとってはハードウェア開発の自由度が高まる



入出力装置を制御するソフトウェアは、ハードウェア、OSの両方を知らないと作成できません。
入出力装置の制御ソフトウェアは原理的にはフラグの読み書きやデータの読み書きが基本ですが、メモリアドレスやビット操作、データ転送、割込み制御、エラー処理など煩雑です。
ハードウェアメーカーがハードウェア設計と合わせて入出力装置の制御ソフトウェアを作成することになります。
ほとんどのOSでは、個々の入出力装置固有の処理については、デバイスドライバというものでその処理ソフトウェアをOSから分離しています。
入出力装置の開発メーカーが提供するデバイスドライバをOSに組み込みことで、個別の処理が行えます。
入出力装置を利用するには、そのOS向けのデバイスドライバが存在することが必要条件です。
（この図で、「ファイルシステム」といったレイヤがあるのは、ファイルではさらにファイル管理のために抽象化を行う層があるということです。また、図にはありませんが、通信では別の層があります）

- デバイスドライバ
  … 入出力装置（デバイス）に依存した処理を行う部分
  - デバイスコントローラに処理を与え、装置を直接操作するプログラム
- デバイスドライバ内部には操作内容に応じた処理手順が記述される
  例えば、open を通じて呼び出された場合の処理、
  read でデータを読み出す場合の処理 など、
  各デバイスに応じた手続き
  - プログラムが入出力操作を実行するとデバイスドライバはデバイスコントローラに対して入出力要求を出し、その完了を待つ
  - 入出力装置から割込みがあったときに、それに対応した処理を行う
    - 入出力の完了は割込みによって通知され、それを受けて割込み処理ルーチンが起動され、待ちを解除し必要な処理を行う
- これらの処理手続きはデバイスごとに大きく異なるため、それぞれのデバイスに合わせたデバイスドライバが用意される

デバイスドライバの大まかな処理は左のようになっています。

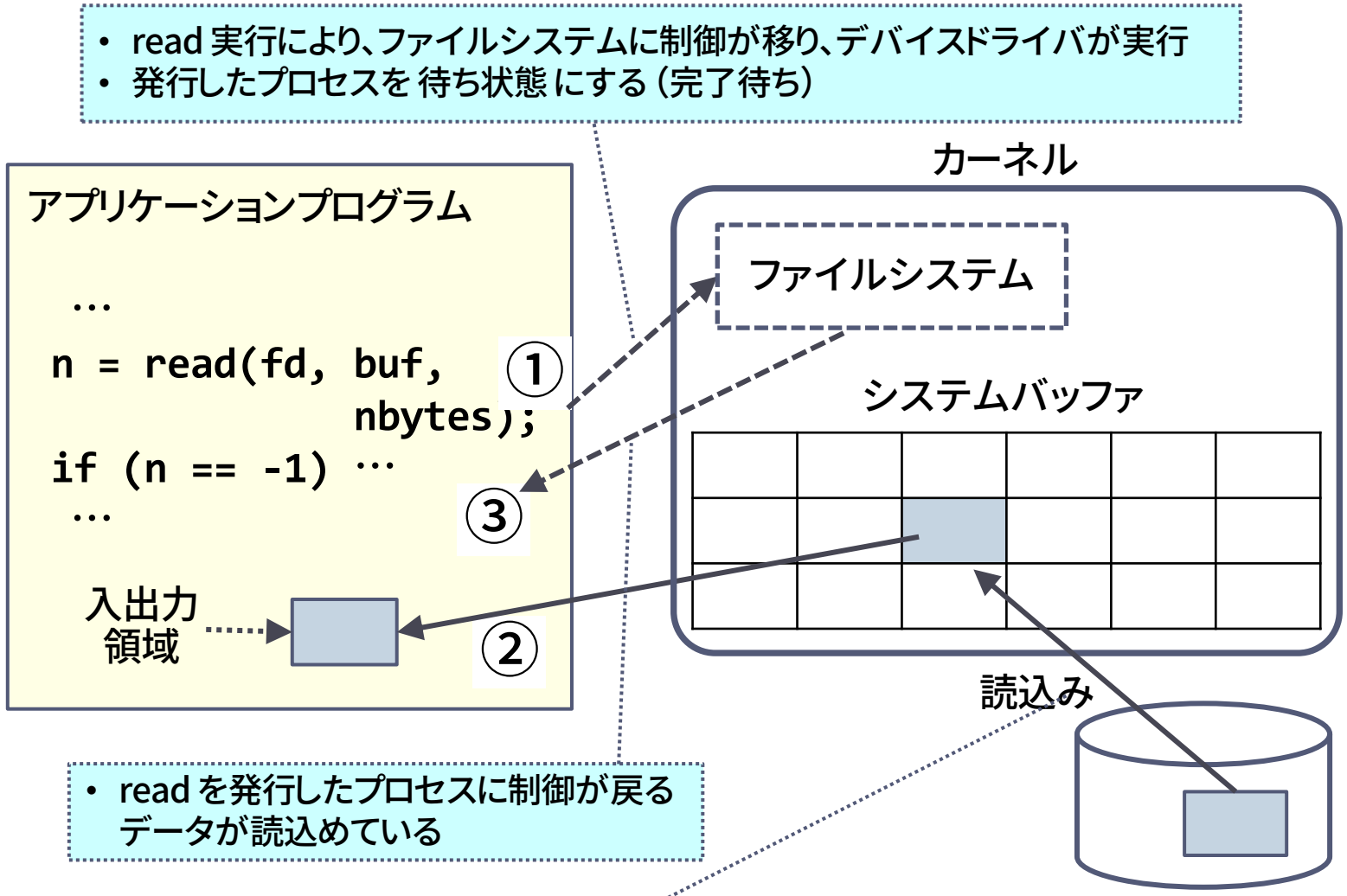
入出力要求と処理

- 入出力の実行
  以下に相当する情報を指定して、システムコールを発行する
  - 対象となる入出力装置
  - 入力または出力の区別
  - 入出力装置上の読み/書きの位置（必要な場合）
  - メモリ内のデータ位置
  - データ長
- 同期入出力
  - 入出力のシステムコール発行後、入出力動作が完了してから発行元に制御が戻る（発行元は入出力動作の完了を待つ）
  - プログラムはOSと同期をとる必要がない
- 非同期入出力
  - 入出力のシステムコール発行後、入出力動作の完了を待たずに発行元に制御が戻る
  - 複数の入出力を同時に起動できる

入出力の要求では、以下のような情報を指定してシステムコールを発行することになります。

システムコールが発行されると、入出力動作が完了してからコール元に制御が戻る同期入出力と、入出力動作の完了を待たない非同期入出力があります。

同期入出力の利点はプログラマにとって処理が単純で分かりやすいことです。
しかし、複数のファイルを扱う高度な処理では性能面で欠点が出てきます。



入出力の実行の流れはこの図のようになります。同期入力の例で、アプリケーションからreadを実行した(①)とします。この結果、ファイルシステムに制御が移り、デバイスドライバが実行されてデバイスコントローラに入力要求が出され、完了を待ちます。

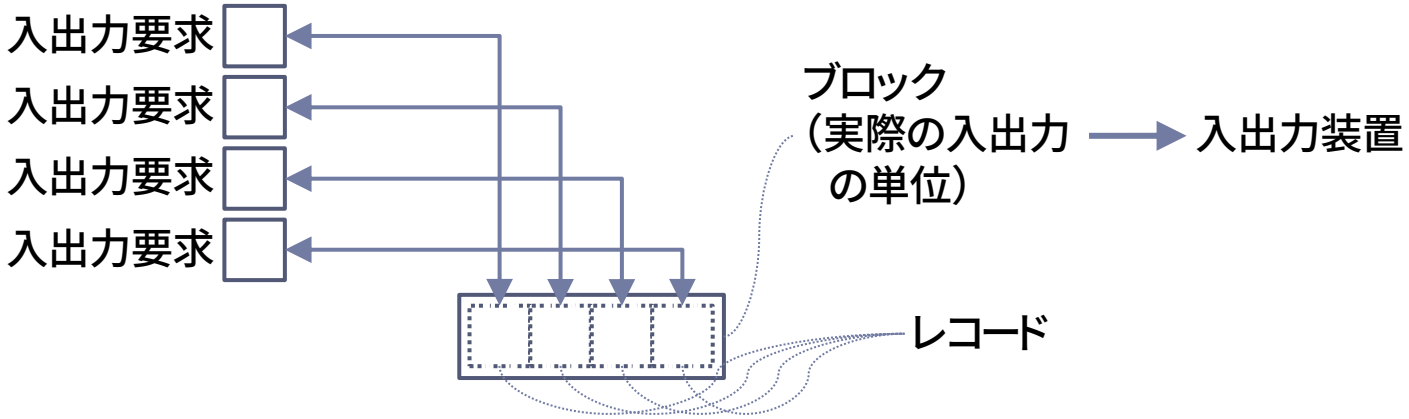
HDDからの読み出しには数十ミリ秒といった程度の時間がかかりますので、readを発行したプロセスを待ち状態にします。(他のreadyなプロセスにCPUを割り当てる)

HDDからの入力完了で割込みが発生し、デバイスドライバに制御が渡ります。

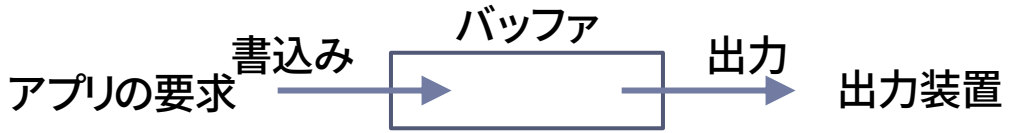
カーネル内のシステムバッファにある読み出しデータをシステムコールで指定した領域に転送します(②)
この操作が完了すると、readを発行したプロセスの待ち状態が解除され、スケジューリングによって制御が戻れば、ファイルからデータを読み込んだ状態(③)になります。

入出力効率化のための技法

- ブロッキング
  複数の入出力要求のデータをまとめて大きなデータ長で入出力を行う
  - 実際の入出力は、何回かの要求について1回ですみ、効率が上がる



- バッファリング
  データの入出力とプロセッサの処理をオーバラップさせるために、バッファと呼ばれる記憶領域に入出力データを一時的に保存しておく
  - 入出力装置とCPUの並列処理を促進する



HDDなど入出力装置の動作時間はCPUに比べてかなり遅く、CPUから頻繁に入出力要求を行うとCPUが待ちになる割合が増え、実行速度が遅くなります。
そこで、入出力を効率化する方式がいろいろ考えられています。

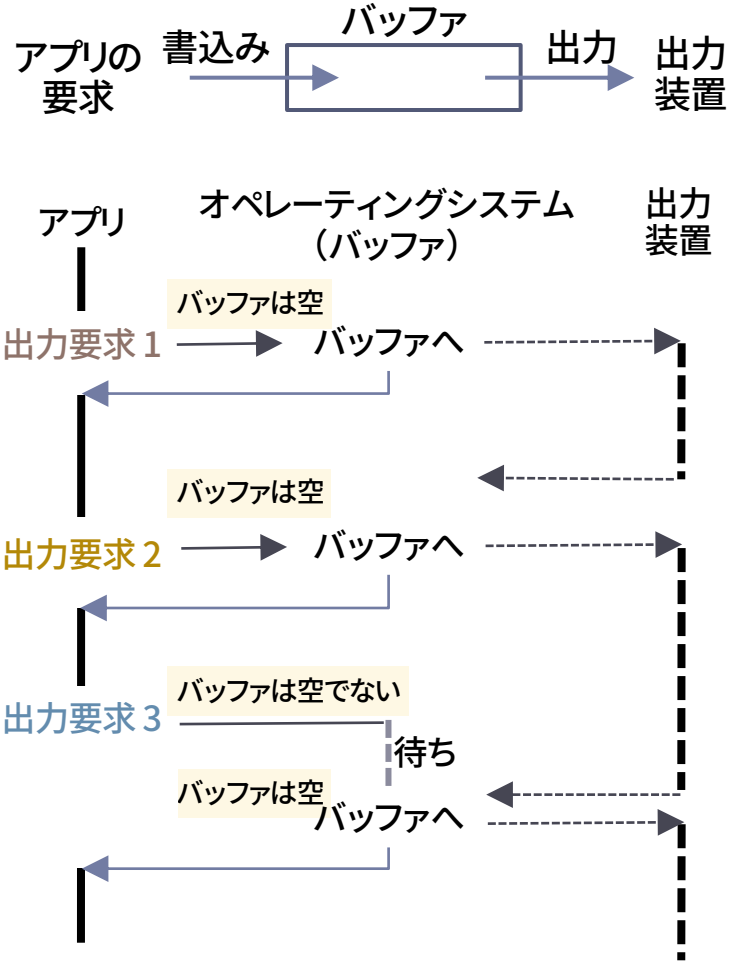
ブロッキングは、複数の入出力要求をまとめて大きなサイズのもの1回にしようとするものです。アプリケーションは論理的な単位（レコード）で入出力を要求しますが、それをまとめてブロックとし、ブロックの単位で実際の入出力を行うものです。

バッファリングは、バッファと呼ぶメモリ内の領域に入出力のデータを一時保存し、入出力とCPU動作を並列に動作させるようにするものです。OS内では、たとえばバッファに書かれたデータの出力を起動するだけでアプリに戻るようにします。

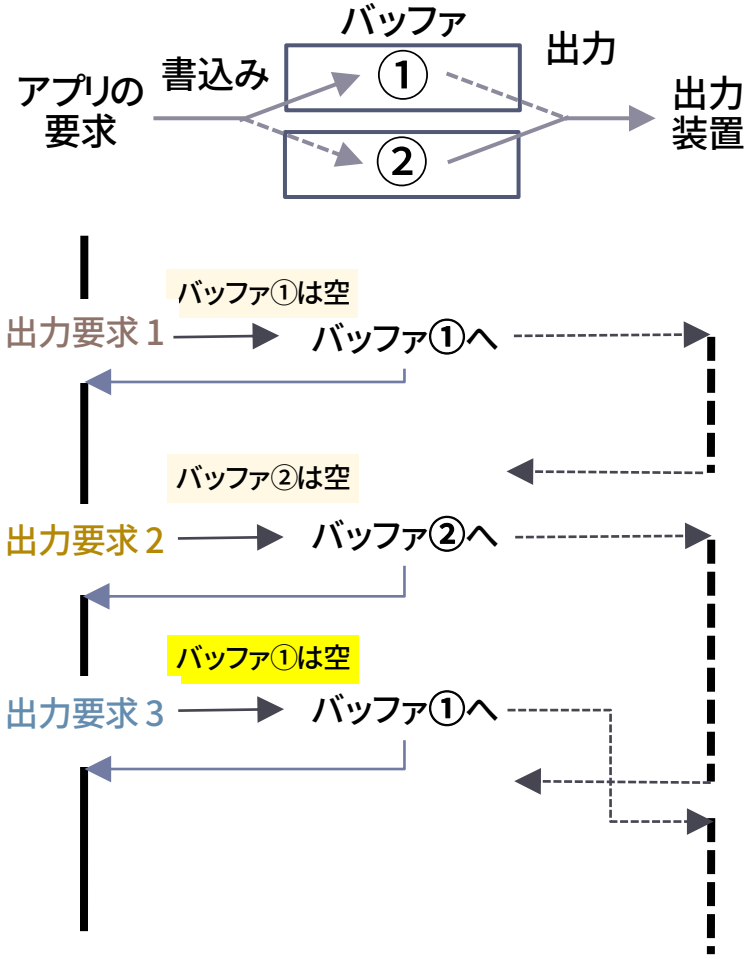
左図でバッファがない場合、出力装置の動作が完了した後にアプリに戻るので、その間アプリの実行は待つことになります。（マルチプロセスでない場合）
バッファがあればアプリは実行を続けることができ、次に書込み要求を出したとき前の出力が完了してバッファが空いていれば、その要求後すぐにアプリに戻れます。

バッファリングによる入出力方式

1面バッファでの出力方式



2面バッファ（ダブルバッファリング）での出力方式



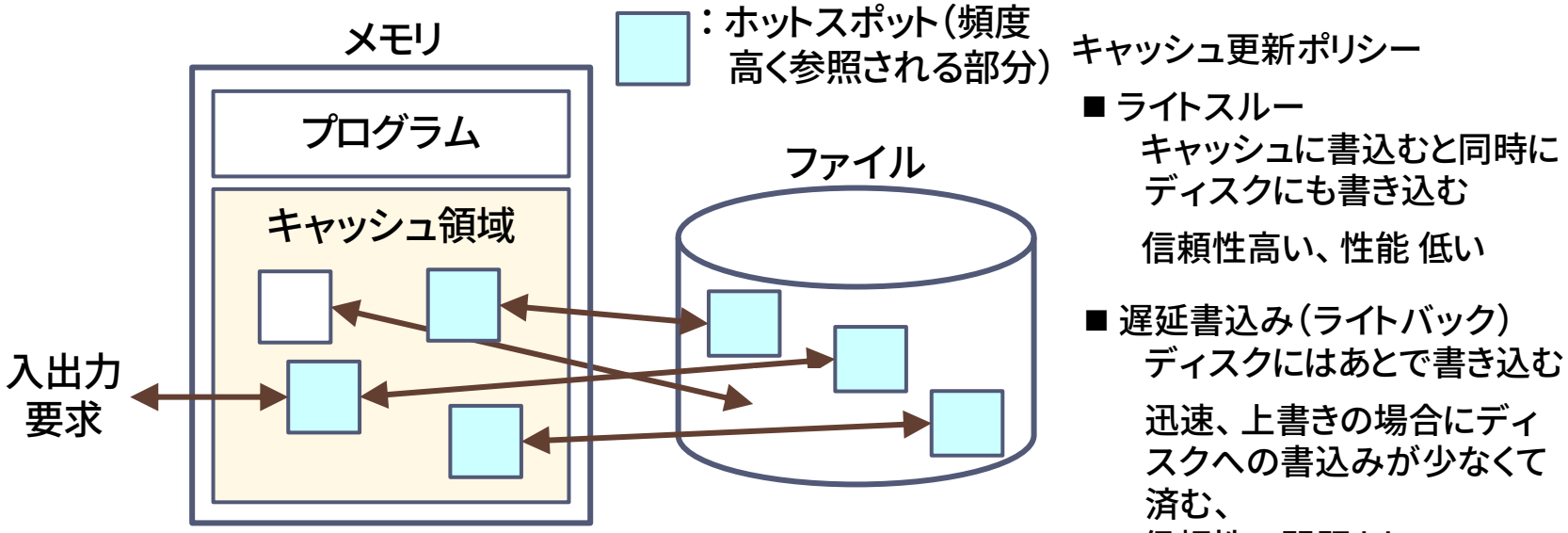
左図、1面バッファで、出力要求1はすぐに終わり、アプリは実行を続けます。次に、出力要求2を出したとき、前の出力動作が完了しているので、出力要求2もすぐに終わりアプリは実行を続けます。出力要求3では、前の出力動作が完了しておらず、バッファが使えないので待ちになります。出力動作完了後、バッファが空けば、出力要求3が終わり、アプリの実行に戻ります。

そこで、バッファを①②2つ用意して交互に使います。1つ目のバッファ①に書込んだ情報が出力されている間に、もう一つのバッファ②に書込み動作を行います。出力要求2はバッファ②に対するものですのですぐに処理され、出力要求3はバッファ①に対するもので、この時バッファ①は空いていてすぐに終了すると期待できます。

もっと多数のバッファ（多面バッファ）を用意すれば、多数の入出力要求が続いておこるような場合でも要求が待ちにならずにすみます。

バッファは「入出力アクセスのギャップを埋める緩衝領域」という意味です。
（ファイル入出力を行う領域をバッファ領域と呼ぶことも多いです）

- キャッシング
  二次記憶から読み出したデータをメモリ上のバッファ（キャッシュ、cache）に一時的に格納し、次に同じデータにアクセスするとき、キャッシュにアクセスすることで高速化



キャッシュ更新ポリシー

■ ライトスルー
キャッシュに書込むと同時にディスクにも書き込む
信頼性高い、性能 低い

■ 遅延書込み（ライトバック）
ディスクにはあとで書き込む
迅速、上書きの場合にディスクへの書込みが少なくて済む、
信頼性の問題あり

- スプーリング（Spooling）
  急を要さない入出力を専用のプロセス（スプーラ）に依頼し、入出力と他のプログラムを並行実行する。また、入出力装置の長時間の占有を避ける
  - プリンタなどによく利用されている

HDDなどから読み出したデータをメモリなどより高速な記憶装置（キャッシュ）に一時的に置き、次に同じデータにアクセスがあった場合に、HDDではなくキャッシュから得ることで高速化する方式です。（参照には局所性があると期待できます）

スプーリング（Simultaneous Peripheral Operations On-Line）とは、メモリやディスク上の特別な領域（バッファ）に入出力の内容を置き、装置が処理を受け付けできる状態になった時に、スプーラ（Spooler）がその領域から装置へ入出力を行なう仕組みです。プリンタに出力する場合、一旦バッファに書き込んで、バッファとプリンタの間でデータをやり取りします。これによってCPUはプリンタの動作完了を待つことなく、次の処理に移ることができます。

教科書との対応

- 磁気ディスク装置、SSD については、ファイルシステムのところで少し触れます
- 「スプール」は教科書 p.11 で出てきています

教科書での説明と記載箇所が違うところがありますので、読むときに注意してください。

第7回の課題

今回（第7回）の課題はありません

事後学習・事前学習

- 今回の講義資料に基づいて内容を振り返り、教科書などの該当箇所を読む
- 教科書第6章（6.1～6.3）に目を通す

今回の講義内容の振り返りと次回の準備をお願いします。