オペレーティングシステム
（2024年 第２回）

OSの基本構造と機能について

問題

プログラム（非常に簡単なもので、例えば以下のもの）

```
#include <stdio.h>
int main(void)
{
  printf("Hello World!¥n");
  return 0;
}
```

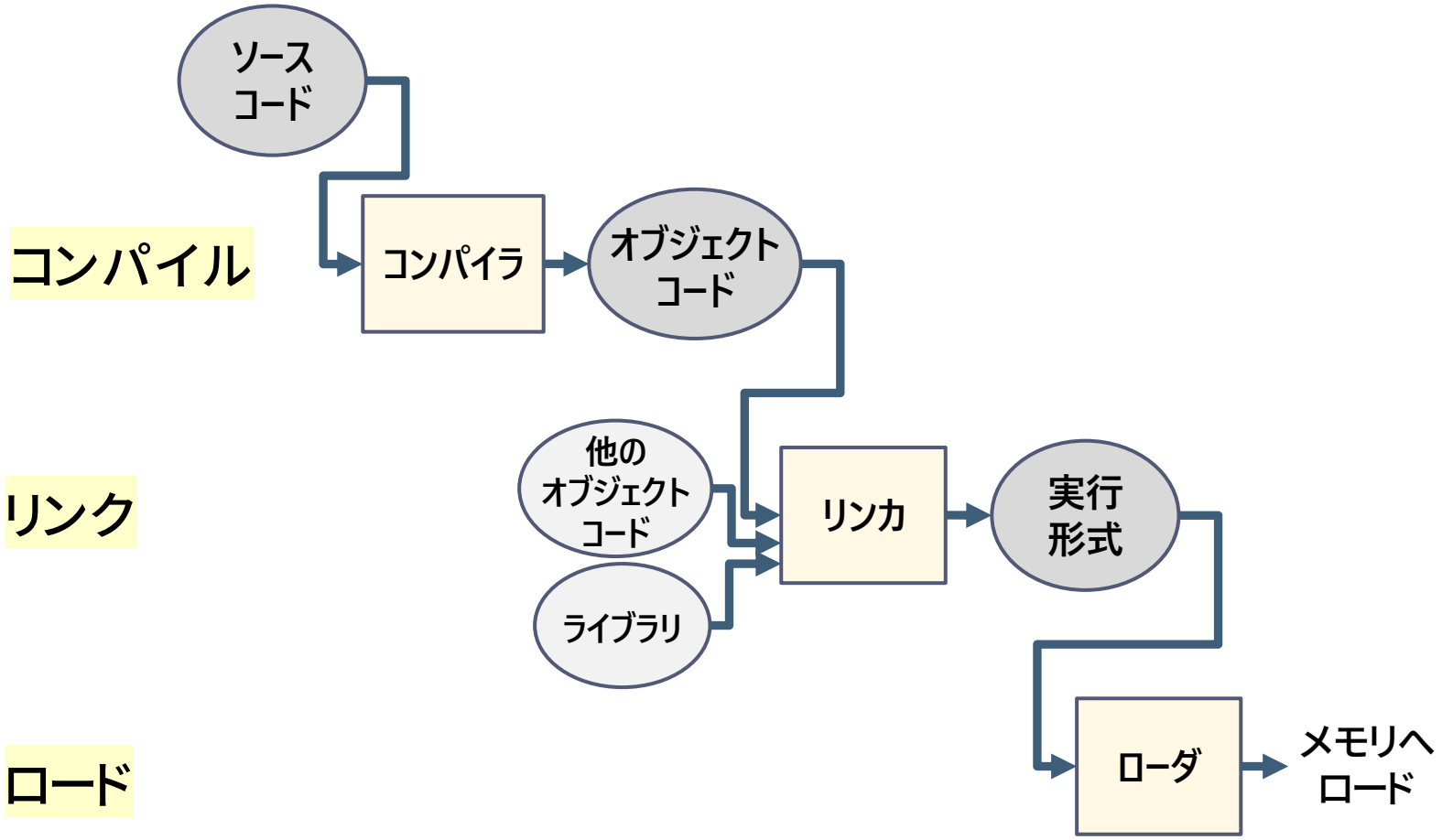
をPCで作成して動作させるときに、OSはどのような機能を提供していると考えられるか

検討する項目

- カーネルが行うものとして、プログラムの実行についてのもの
  - 例題プログラムの実行形式をメモリにロードする、
  - そのために、メモリ領域を確保する、
  - ロードした実行形式プログラムを実行する、
  - プログラムで行っている出力に対して、ディスプレイに文字列を出力する、
  - 終了した後に、メモリを解放する、

  など
- それ以外に、広義のOSの機能も考えると
  - ソースコードを作成する際のテキストエディタ、コンパイラ、
  - 出力のための関数 printf をリンクするリンカ
- エディタでテキスト入力をする際のキー入力、プログラムを保存するためのファイル出力や、コンパイルやリンクでのファイルの入出力などもOSの機能になり、これらはカーネルの機能

プログラムの開発・実行

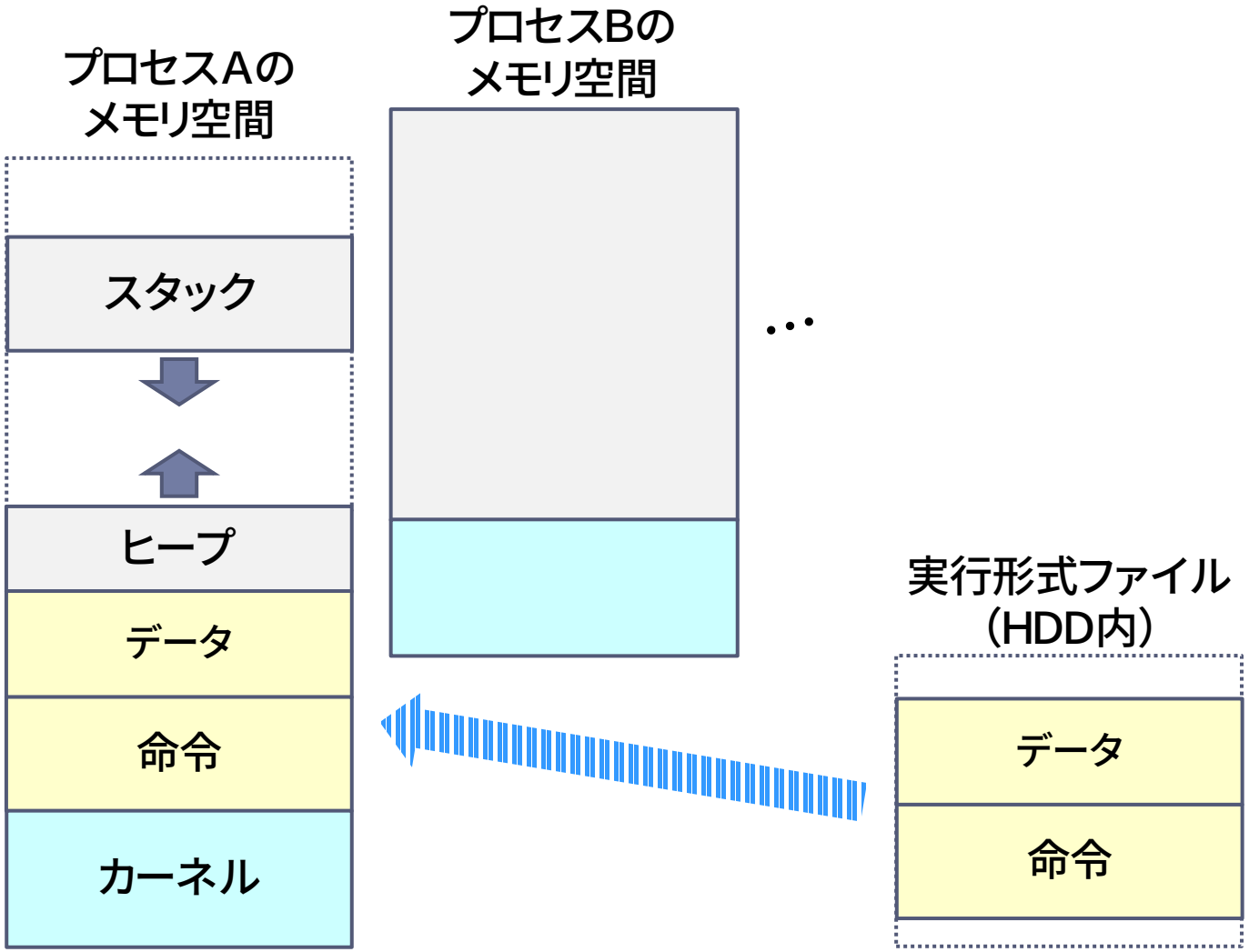


ここで、プログラムの開発について触れておきます。
ユーザがプログラムを作成して実行するには、ここに示すようなコンパイル・リンクの過程を経て、ソースプログラムをコンピュータが実行可能な形式に変換し、OSが提供するサービスを受けることになります。

C言語などで記述されたソースプログラム(ソースコード)はコンパイラによって機械語であるオブジェクトコードに変換されます。リンカによって、オブジェクトを他のオブジェクト、また先にありましたprintfなどのようなあらかじめ提供されているライブラリと結合し、さらに、CPUで実行可能な形式である実行形式プログラムが生成されます。

この実行形式プログラムはハードディスクなどに保存され、実行時にローダによってコンピュータのメモリにロードされて実行されます。

プログラムのメモリ配置

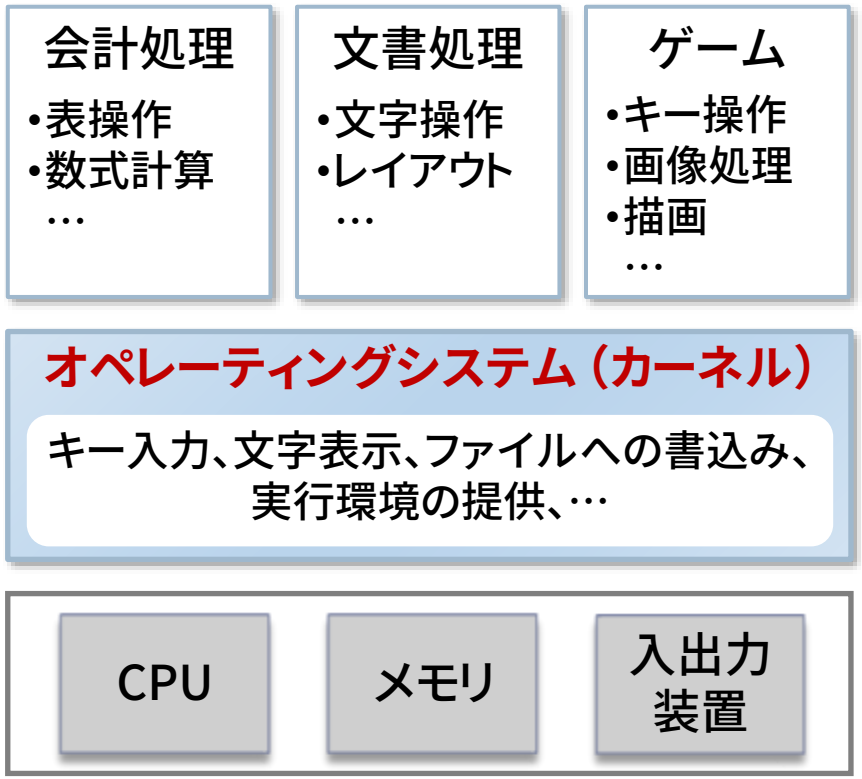


後の回で詳しく説明しますが、実行形式のプログラムは、ハードディスクのような二次記憶装置内のファイルにデータと命令が配置されて保存されています。

実行時には、これが、メモリ上に展開・配置されます。
メモリ上では、このデータや命令以外にスタックやヒープと呼ばれる、実行時に必要となるデータを配置する領域なども確保され、
それらを合わせて「プロセス」と呼ぶモノを構成してメモリ内に配置します。

オペレーティングシステムの機能

- OSの目的を実現し、仮想化された資源を実行環境としてアプリケーションに提供する
  - CPU（プロセス）
    - プロセスの生成・消滅
    - 同期などプロセス間の調停
    - プロセス間の通信
  - メモリ
    - メモリの確保、解放
  - 入出力装置
    - キー入力
    - ファイルのオープン、クローズ
    - 読出し・書込み



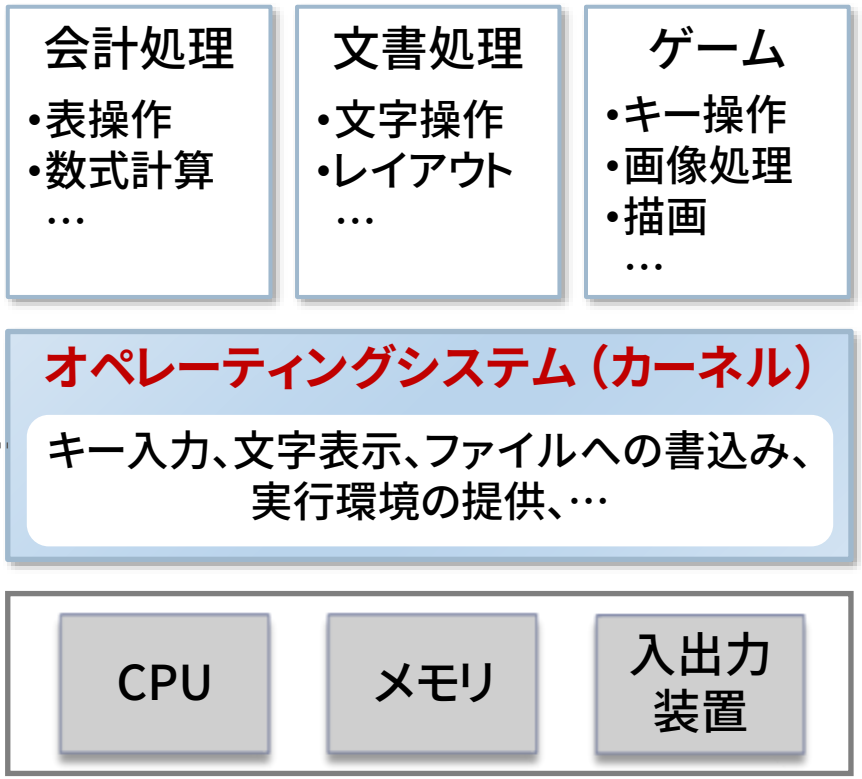
前回示したOSの構成の図にそれぞれのソフトウェアがもっている機能を入れて再度示します。

OSが提供する機能は、キー入力を行ったり文字を表示すること、ファイルをオープンしたり、そこへ書き込んだりすることなどで、これらは入出力であってユーザとのやり取りにかかわることで、直接わかりやすいものだと思います。

それ以外には、CPUを仮想化したものと考えるプロセスに関して、それを生成したり複数のプロセス間の連絡を行ったりする機能などがあります。
また、メモリに対して、そのある部分を確保したり解放したりするといった機能を考えることができます。

カーネル

- カーネル
  - 提供するさまざまなサービス（OS機能）を実現するためのオペレーティングシステムの基本的な部分



以上のような様々な機能が提供されますが、これらの機能を提供する部分がOSの基本部分カーネルです。

システムコールによるOS機能の提供

- OS（カーネル）はアプリケーションプログラムに対して抽象化したインタフェース（API）によって機能を提供する
  - OSのAPIを システムコール（system call）と呼ぶ
- アプリケーションからは、OSはシステムコールやライブラリの集合としてみえる
  - APIは、関数の形式で利用できる
  - ハードウェアに独立した機能を提供する
- システムコールの具体的な仕様はOSごとに異なる
  - OSが違えば、アプリケーションはそのままでは動かない

こういった様々なOS機能は、関数呼び出しの形で利用することになります。すなわち、理解しやすい名前をつけて、それで表現できるようにするわけです。
そのインタフェースをAPIと呼びます。
また、特にOSに対するAPIをシステムコールと呼びます。

アプリケーションから見ると、カーネルは、システムコールの集合として捉えることができます。抽象化によって個々のハードウェアから独立している、すなわちハードウェアが何であれ、例えば、「いくらのサイズのメモリを確保する」、「ファイルに出力する」といったレベルで考えることができるわけです。

システムコールはOSごとに違っています（似てはいるのですが）。したがって、OSが違えば、知らないシステムコールがあることになるので、アプリケーションはそのままでは動かないことになります。

参考

- API（Application Programming Interface）アプリケーションプログラミングインタフェース
  - プログラミングの際に使用できる命令や規約、関数等の集合のこと
  - あるコンピュータプログラム（ソフトウェア）の機能や管理するデータなどを、外部の他のプログラムから呼び出して利用するための手順やデータ形式などを定めた規約のこと
  - ソフトウェアからOSの機能を利用するための仕様、またはインタフェースのこと
  - APIには、サブルーチン（関数）、データ構造、オブジェクトクラス、変数などの仕様が含まれる
- インタフェース
  - 二つのものが接続・接触する箇所や、両者の間で情報や信号などをやりとりするための手順や規約を定めたもの

参考までに、APIの定義を示しておきます。

API使用のイメージと実現方法

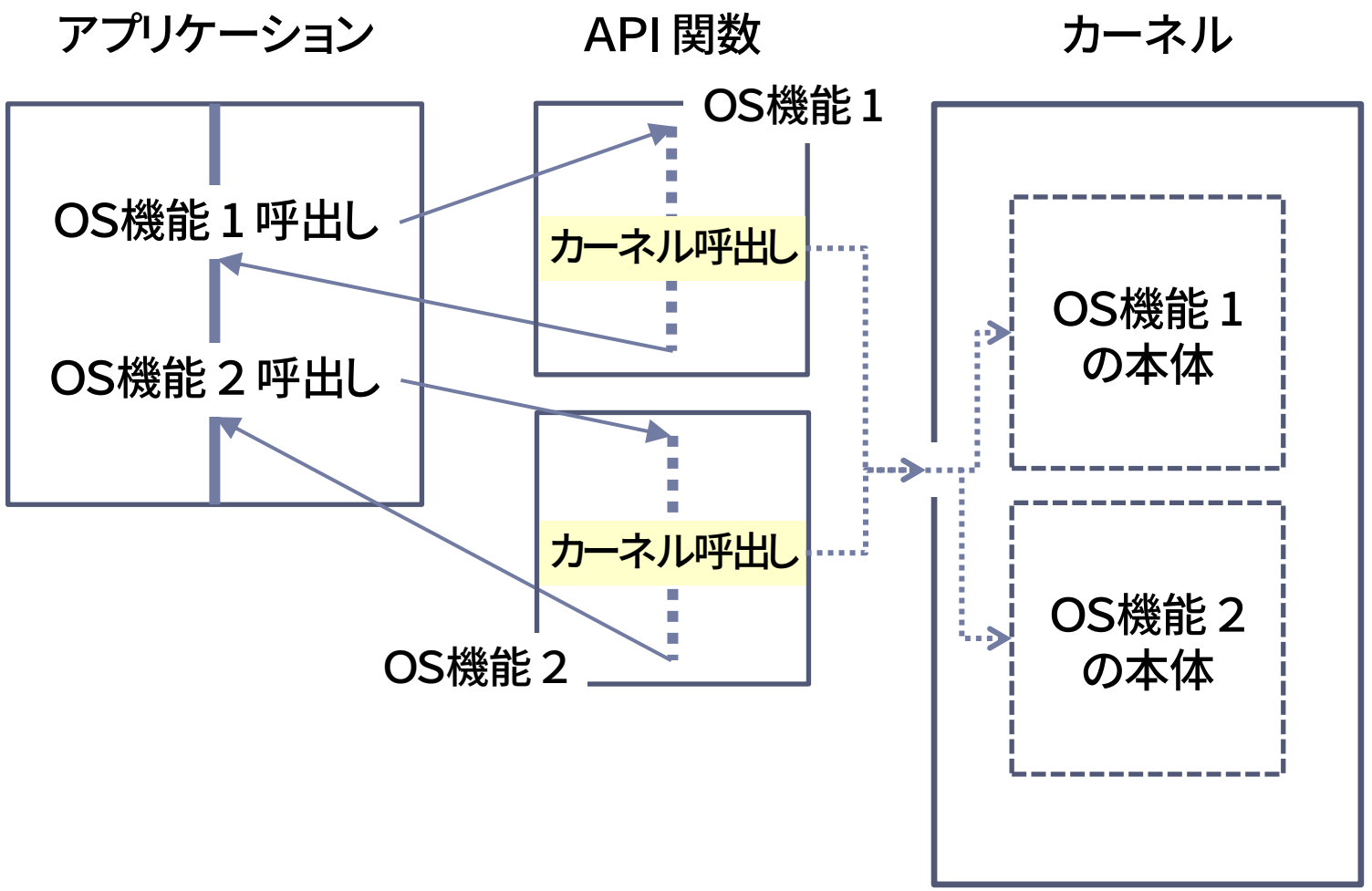
- API関数を呼び出す
  - アプリケーションプログラムからは関数呼び出しの形で利用できる
- カーネルとアプリケーションは分離されている
- OSの基本機能はカーネルで実現される
  - アプリケーションからは、特別な命令を実行して呼び出す

APIは関数の形式なので、それを呼び出すことで必要な機能の提供を受けることができます。
APIの機能の中身はカーネルの中にあり、呼び出す方はアプリケーションです。
一般に、アプリケーションとカーネルは分離、すなわちそれぞれ独立したものとして構成されています。

同じプログラムの中にある別の関数を呼び出すことと比べますと、関数を呼ぶということで書き方は同じですが、内部の動作が違います。

そこで、特別な機構を使用して呼び出すことになります。

API使用のイメージ



カーネルの中にはAPIが規定している機能が入っており、API関数が呼び出されると、このカーネル呼出しが行われて、カーネルの中に入って、対応する機能が実行されます。

この詳細については、次回以降で説明します。

APIの例

- C言語規格の標準ライブラリ
  - 入出力やメモリの操作など、OSに関係の深いものがあり、もともとUNIXの機能の一部として提供されていたもの
    `fopen, printf, getchar, malloc, time, …`
- UNIXのAPI
  - システムコール関数
    `read, lseek, mkdir, fork, exec, …`
  - ライブラリ関数
    システムコールにほぼ1対1に対応
- Windowsのシステムインタフェース
  - Win32 API：Windows 95以降、Windows NT用のAPI
    `CreateProcess, CreateFile, ReadFile, CreateDirectory, …`
  - WinRT（Windows Runtime）：Windows 8 以降で実装されている
- POSIX（Portable Operating System Interface）
  UNIXを始めとする異なるOSの実装に共通のAPIを定めた国際規格

APIの例は次のようなものです。C言語の標準ライブラリもAPIですし、UNIXのシステムコールやWindowsのシステムコールなどがあります。

教科書p.36、37、39の表3.1、3.2、3.3に例があります。

OSごとにAPIが違っているのは不便なので、共通のAPIを定めた国際規格があり、POSIX（ポジックス）と呼ばれています。

システムコールとライブラリ

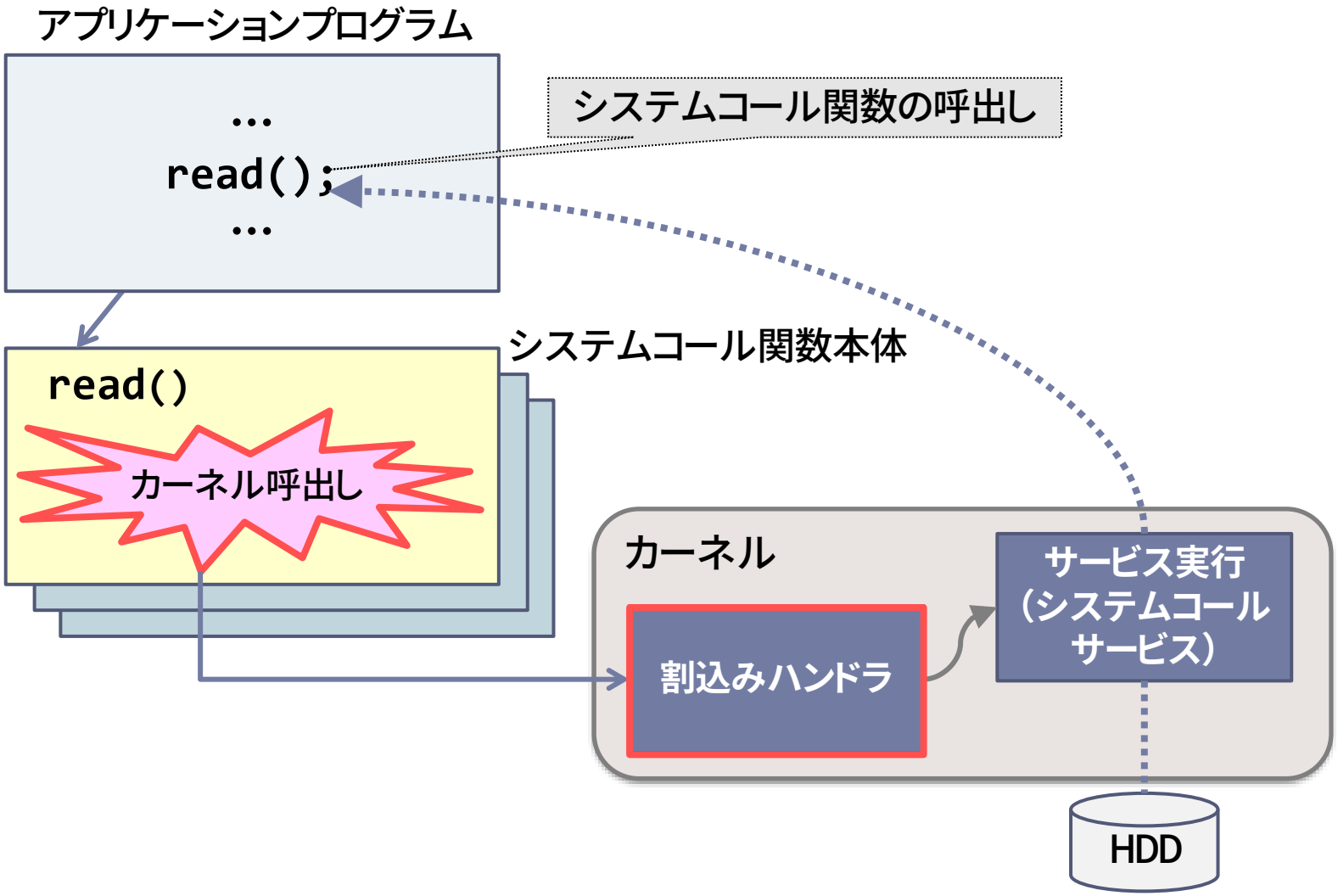
- システムコール（スーパバイザコール）
  カーネルの機能を呼出すために使用される機構
  関数（API）呼出しによって実現される
  - Linuxでは300個程度
  - 呼出し専用の機構（ソフトウェア割込み）を用いる
- ライブラリ関数
  よく使用されるプログラムの共通部分を抜出して、他のプログラムから利用できるようにしたもの。次の2タイプがある
  - 最終的にシステムコールを呼出すもの
    - C言語のライブラリ関数 fopen，fread などは内部で open，read などを呼出す
  - 操作や計算を行うだけで、カーネルの提供する機能を使わないもの

システムコールはスーパバイザコールとも呼ばれます。
OSという名前が一般的になるまで、いろいろな名前がつけられていました。スーパバイザというのは、その中でもよく使用されていたものです。

システムコールは、呼び出し専用の機構を用いて実現されています。

ライブラリ関数とは、よく使用される機能を抜き出して、あらかじめ提供して他のプログラムから利用できるようになっているプログラムです。
これには、先に出てきたC言語のライブラリ関数printfのように、入出力やメモリ操作など最終的にシステムコールを呼び出すものと、
計算（例えば、配列の演算など）を行うだけで、カーネルが提供する機能は使わないもの、
の二つのタイプがあります。

システムコールの動作概要



先に、アプリケーションからは特別な命令を実行して呼び出すといいましたが、システムコールは特別な機構で実行されます。詳細については、次回以降で説明しますので、今回は、「カーネル呼出し」という割込み（これも次回説明）を利用した特別な機構を使ってカーネルに入り、カーネルの中にあるOS機能（サービス）を実行します。

そして、例えば図のread()の場合、HDDの内容が読みだされてアプリケーションに渡される、ということで理解しておいてください。

p.11と対比して見比べてください。p.11で右側のカーネルの箱の左横に入り口のようなものを書いていますが、この図で割込みハンドラがそれに対応するものということになります。

プロセスとは

- プロセス
  プログラムの実行コード、全ての変数、その他の状態を保持しているもの
  - プログラムに対応する命令コード
  - 変数など操作されるデータ
  - プロセスに固有のデータ
    - 実行中の関数の管理領域
    - 動的に確保するメモリ領域
    - …
  - CPUの状態
    - レジスタの内容など

今回の最初でも出てきました「プロセス」については次々回に学ぶことにしますが、プログラムに対応する命令や変数などのデータ、そして処理中のCPUの状態を保持しているもので、処理の実体というべきものです。

詳しい学習

- APIの詳細な説明について、別途添付する資料を理解すること
  添付する資料は、参考書：「オペレーティングシステムの概念」からの抜粋
  （ただし、「システム呼出し」を「システムコール」、「標準的なCライブラリ」を「標準Cライブラリ」と変更している）

  次の事項に注意すること
  - ファイルコピーの例が、一連のシステムコールで実現されるようす
  - ここで述べているAPIとシステムコールとの違い
  - システムコールインタフェースの説明と今回のスライド資料との対応
  - 「通常、各システムコールには数字が割り当てられており、システムコールインタフェースはこの数字で索引付けを行ったテーブルを持っている。そして、システムコールインタフェースは、オペレーティングシステムカーネル内で意図されたシステムコールを呼び出し・・・」という記述がp.11でカーネル呼出しからカーネルに入って中で対応するサービスに分かれるところに対応していること

別途提供する資料を読んで、APIとシステムコールについて理解を深めてください。

open() にはたとえば、「2」が割り当てられており、図2.2内の i で指されるテーブルの2番目の要素が open() の実装の先頭を指しているようなイメージです。

カーネルの動作、サービス提供

- 通常は 待機している
- イベントの発生によって「動き出し」、機能（サービス）を実行する
  - 入出力 (ファイルの読出し など)
  - 別プログラムの実行
  …
- イベントとなるもの
  - システムコール
  - 割込み
  - 例外
- CPUには、動作する状態が複数ある（CPUモード）
  - 特権モード/ユーザモード

  カーネルが動作するモードは特別なモード

それぞれの項目について、後で説明

カーネルは通常、待機していて、ある出来事（イベント）が発生すると動いて機能を提供します。
ここで、機能とは、先のHDD読み出しや別のプログラムを実行したりすることです。

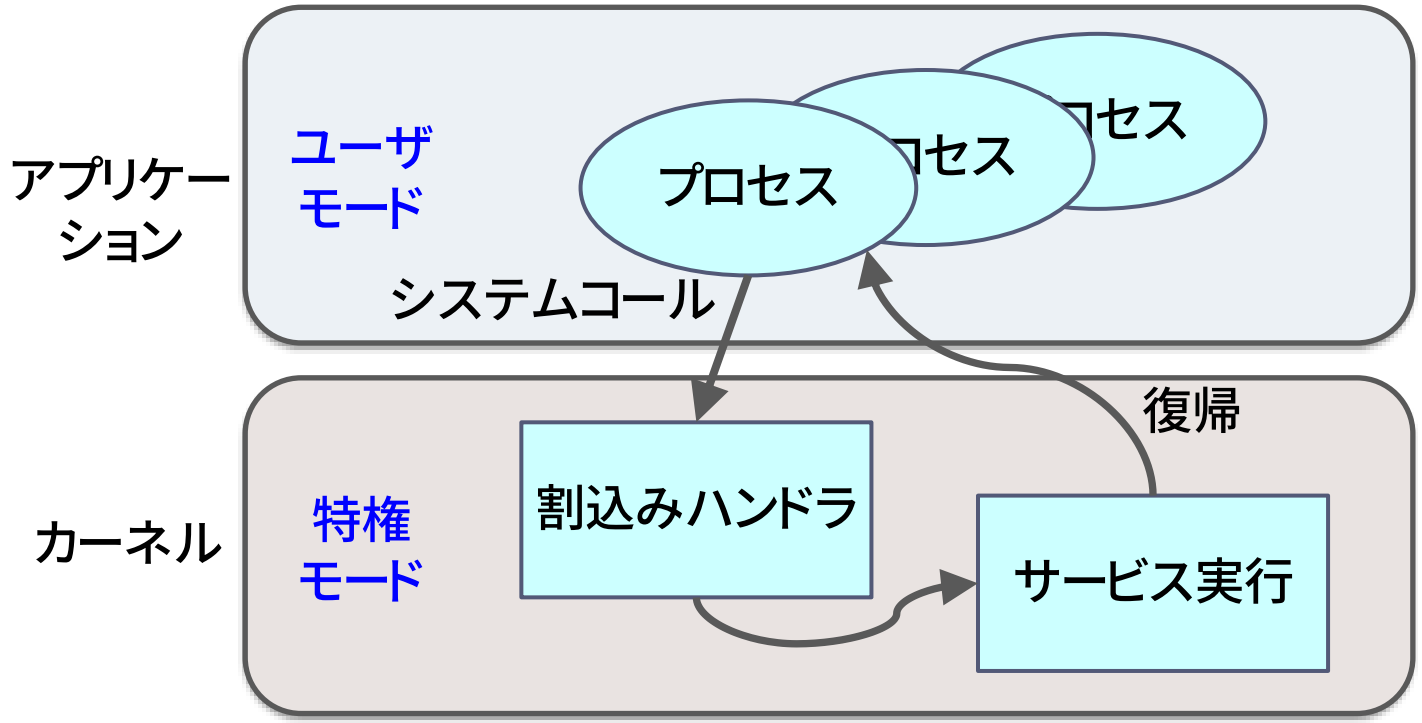
このイベントになるものは、システムコール、割込みや例外、になります。割込みや例外については、次回に学びます。

また、カーネルが動作するCPUの状態というものが用意されています。すなわち、アプリケーションプログラムとカーネルは動作するCPUのモードが異なるということです。

これらの項目については後で説明して行きます。

プログラム実行の状態

- カーネルが特権モードで動作する場合

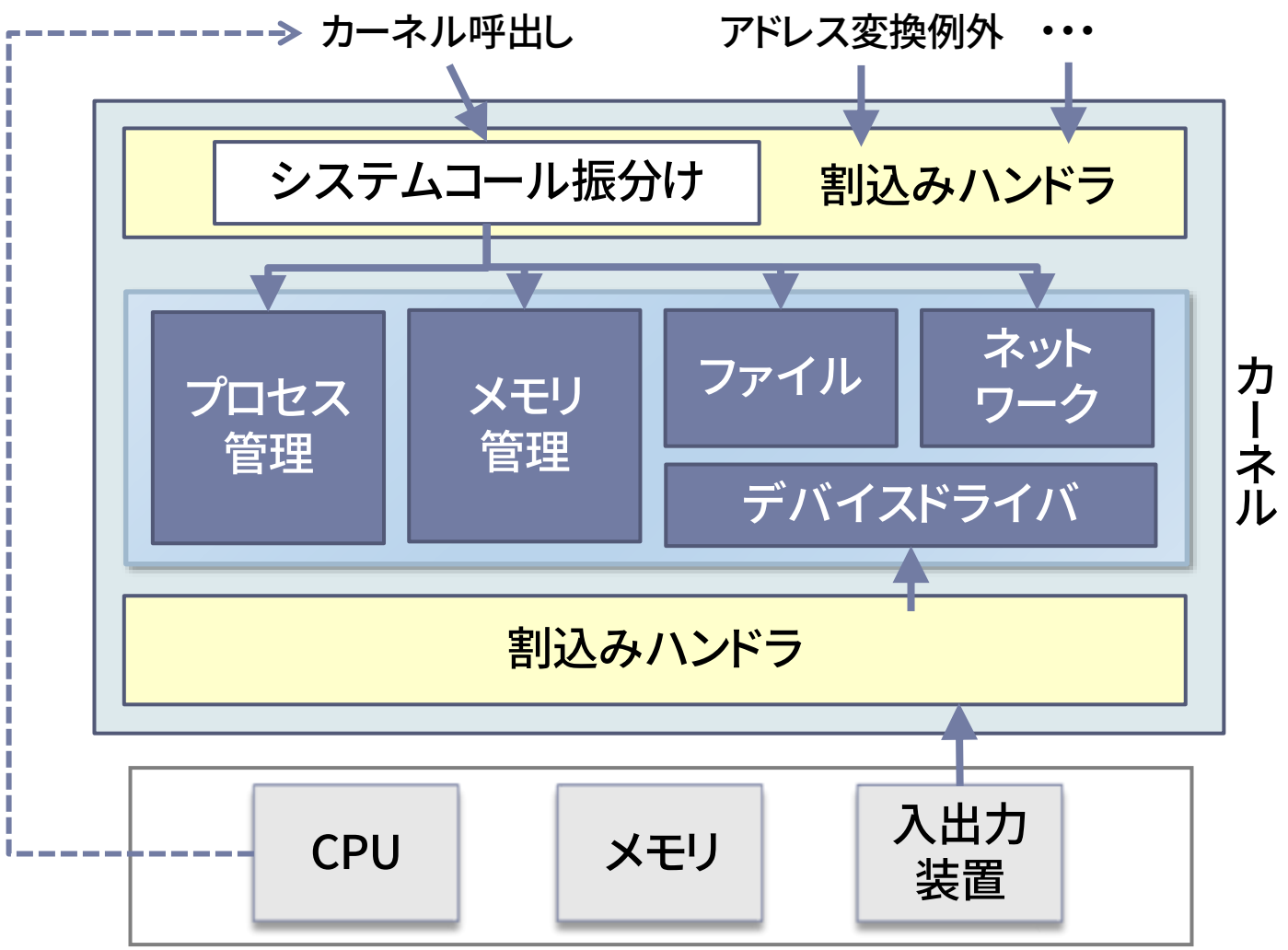


分かりやすい例として、カーネルが特権モード、アプリケーションがユーザモードで動作する場合のプログラムが実行される概要です。

アプリケーションがユーザモードで動作する複数のプロセスで構成されていて、カーネルが特権モードで動作する場合のイメージです。
アプリケーションとカーネルが分離された領域にあり、実行モードも違っています。特権モードは、名前のとおりコンピュータのすべての資源を操作できるもので、ユーザモードは、そうではなく、実行できない命令があるなど制約があるものです。

アプリケーションからは、システムコールによってカーネルのサービスを受けることになります。

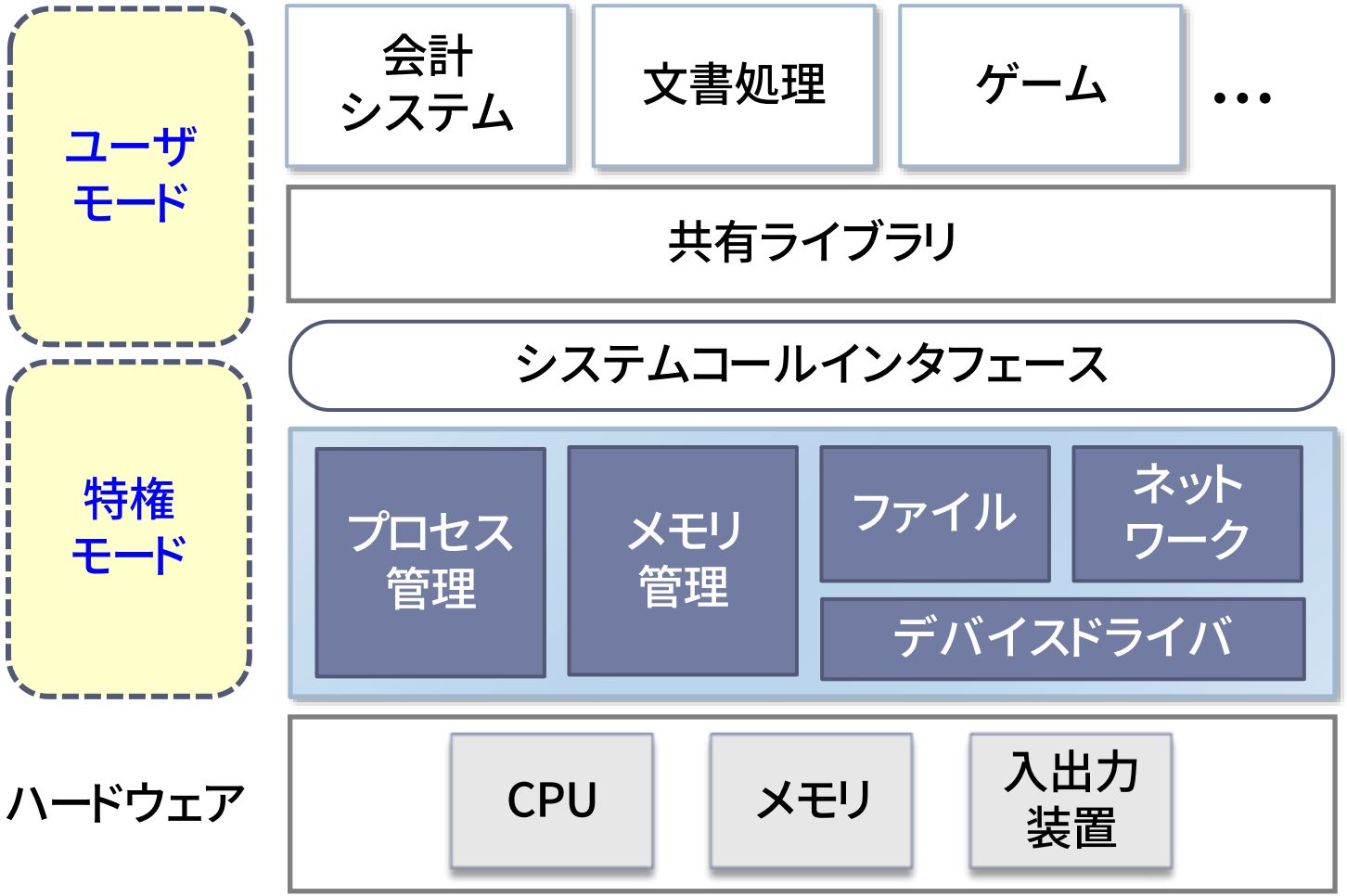
カーネルの構成と呼出し



上部の「割込みハンドラ」がp.14にあるものと対応します。

アドレス変換例外や入出力装置と割込みハンドラの関係などにつきましては、次回以降説明します。

オペレーティングシステムの構造



アプリケーションや広義のOSを含めて構造を見ると、この図ようになっており、システムコールインタフェースによってユーザモードと特権モードが区切られ、
アプリケーションはカーネルの提供する機能を受けるということになります。

第2回の課題

1. OSにおけるAPIの説明として適切なものはどれか（基本情報技術者平成14年春期）

   ア　アプリケーションソフトがハードウェアを直接操作して，各種機能を実現するための仕組みである

   イ　アプリケーションソフトから，OSが用意する各種機能を利用するための仕組みである

   ウ　複数のアプリケーションソフト間でデータを受け渡すための仕組みである

   エ　利用者の利便性を図るために，各アプリケーションソフトのメニュー項目を統一する仕組みである

2. 第1回の「資源管理の着想」で 時分割 や 空間分割 などを挙げている
   時分割について考えると、1つのCPUを時間枠ごとに複数のプログラムに交互に割り当てることで、あたかもCPUが複数あって 、複数のプログラムが同時に動いているように実行することができるが、このような処理を行う際に発生するオーバーヘッドとして、どのようなものが考えられるか

理解の確認のための今回の課題です。

4/20の午前中までに、解答をクラスウェブのレポートで、提出してください。

選択肢から選ぶものは、それが選ばれる理由や選ばれない理由を記述してください。

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（第1章）などの該当箇所を読む
  - 教科書第2章については、第12回の授業で触れます
- コンピュータアーキテクチャとプログラムについて概要を確認しておく
- 教科書第4章（4.1）、第10章（10.3）に目を通しておく

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。