

オペレーティングシステム — イントロ

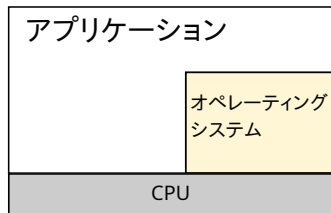
田浦健次郎

オペレーティングシステム (Operating System; OS) とは

- ▶ 実例: Windows, MacOS, Linux, BSD, iOS, Android, etc.
- ▶ アプリケーションを動かすためのソフト (基本ソフト)
- ▶ 存在理由 (一般的な言葉で):
 - ▶ 抽象化: 簡単にプログラムできるようにする
 - ▶ 効率化: 簡単なプログラムで高速に動作するようにする
 - ▶ 資源保護・管理: 資源 (CPU, メモリ, etc.) の独占を防ぎ, 公平に割り当てる
- ▶ 具体的には OS がないとどうなるかを知る・考えるのが良い

オペレーティングシステム (Operating System; OS) とは

- ▶ 実例: Windows, MacOS, Linux, BSD, iOS, Android, etc.
- ▶ アプリケーションを動かすためのソフト (基本ソフト)
- ▶ 存在理由 (一般的な言葉で):
 - ▶ **抽象化:** 簡単にプログラムできるようにする
 - ▶ **効率化:** 簡単なプログラムで高速に動作するようにする
 - ▶ **資源保護・管理:** 資源 (CPU, メモリ, etc.) の独占を防ぎ, 公平に割り当てる
- ▶ OSがないとどうなるかを知る・考えるのが良い



OSがないと...

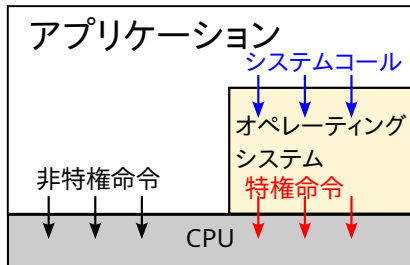
- ▶ CPU (プロセッサ) 上に直接ユーザのプログラムが動く
 - ▶ 例えば以下のようなことが非常に困難になる
 1. CPU (計算のための資源) を公平に分け合う
 2. メモリ (記憶のための資源) を安全に分け合う
 3. 外部ストレージを安全に分け合う
 4. 入出力
- etc.

OSがない場合の問題点とOSの機能

- ▶ CPU を分け合う
 - ▶ OSなし: 1つのプログラムでCPUを独占できてしまう
 - ▶ OS: プロセス, スレッド
- ▶ メモリを分け合う
 - ▶ OSなし: 1つのプログラムが他の人の(メモリ上の)データをのぞき見・破壊できてしまう; 大量のメモリを独占できてしまう
 - ▶ OS: プロセス(アドレス空間), 仮想記憶
- ▶ ストレージを分け合う
 - ▶ OSなし: 1つのプログラムが他の人のデータをのぞき見・破壊できてしまう
 - ▶ OS: ファイルシステム, システムコール
- ▶ 入出力
 - ▶ OSなし: 入力監視, 割り込み処理など複雑, かつ機器依存
 - ▶ OS: ファイルシステム, プロセス間通信

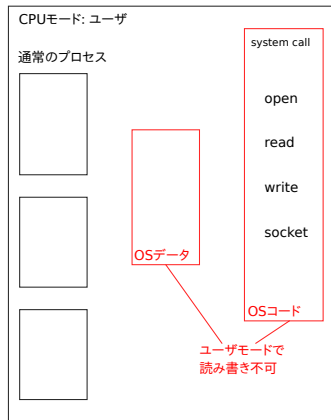
資源保護・管理のための基本的仕組み

- ▶ 命題: 資源 (CPU, メモリ, etc.) の独占を防ぎ, 公平に割り当てる
- ▶ 悪意のあるプログラムでも OS の破壊, 他のプログラムの破壊, 資源の独占を不可能にする
- ▶ 以降ではまず OS の破壊を不可能にする仕組みを考える
 - ▶ それ以外は次週以降の個別の機能説明にて



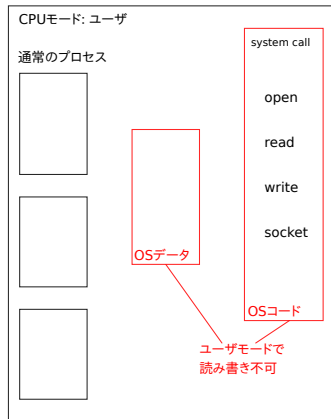
OSは実体としてはどこにどう存在している？

- ▶ その他のプログラムと同様、メモリ上にプログラム+データとして存在
- ▶ ただし OS 以外のプログラムには読み書き不能になっている
- ▶ → どうやって？



CPUの特権モード・ユーザモード

- ▶ CPUの動作モードに(少なくとも)2種類ある
 - ▶ ユーザモード
 - ▶ 特権モード(スーパーバイザモード)
- ▶ 両者の主な違い
 1. 一部の命令が特権モードでしか実行できない(特権命令)
 2. 一部のメモリ領域に「ユーザモードでアクセス不可」という属性をつけられる
- ▶ OSのデータやプログラムがOS以外のプログラムには読み書き不能な仕組み
 - ▶ OSが管理する領域を「ユーザモードでアクセス不可」
 - ▶ OS以外はユーザモードで動作



ユーザモードから特権モードへの移行

- ▶ 通常のプログラムはユーザモードで実行される
- ▶ 一方通常のプログラムも OS の機能呼び出すことが出来る (さもないと OS はいらぬはず)

ユーザモードから特権モードへの移行

- ▶ 通常のプログラムはユーザモードで実行される
- ▶ 一方通常のプログラムも OS の機能呼び出すことが出来る (さもないと OS はいらぬはず)
- ▶ → ユーザモードから特権モードへ移行する仕組みがあるはず

ユーザモードから特権モードへの移行

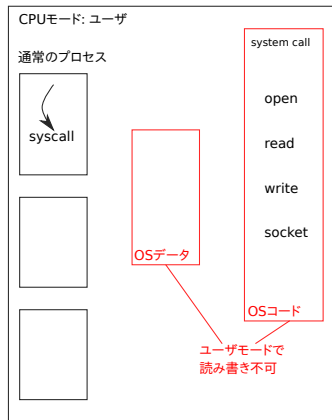
- ▶ 通常のプログラムはユーザモードで実行される
- ▶ 一方通常のプログラムも OS の機能呼び出すことが出来る (さもないと OS はいらぬはず)
- ▶ → ユーザモードから特権モードへ移行する仕組みがあるはず
- ▶ 下手に設計すれば、結局誰でも特権モードで好きな命令を実行可能になる危険

ユーザモードから特権モードへの移行

- ▶ 通常のプログラムはユーザモードで実行される
- ▶ 一方通常のプログラムも OS の機能呼び出すことが出来る (さもないと OS はいらぬはず)
- ▶ → ユーザモードから特権モードへ移行する仕組みがあるはず
- ▶ 下手に設計すれば、結局誰でも特権モードで好きな命令を実行可能になる危険
- ▶ → **トラップ命令**

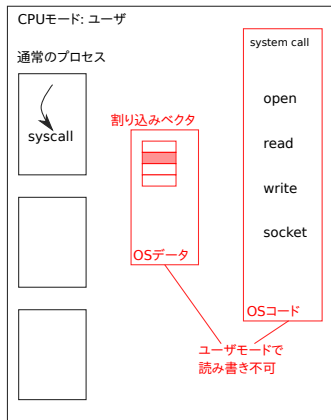
トラップ命令

- ▶ 以下の二つを行う
 - ▶ ユーザモードから特権モードへ移行
 - ▶ ある定められた番地にジャンプ
- ▶ x86 の場合
 - ▶ int 0x80h 命令
 - ▶ syscall 命令



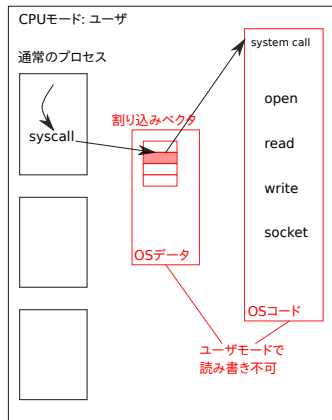
トラップ命令

- ▶ 以下の二つを行う
 - ▶ ユーザモードから特権モードへ移行
 - ▶ ある定められた番地にジャンプ
- ▶ x86 の場合
 - ▶ int 0x80h 命令
 - ▶ syscall 命令
- ▶ ある定められた番地は、「割り込みベクタ」と呼ばれるメモリ上の配列にかかれており, OS が起動時に設定する



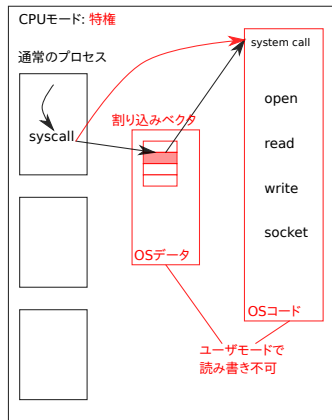
トラップ命令

- ▶ 以下の二つを行う
 - ▶ ユーザモードから特権モードへ移行
 - ▶ ある定められた番地にジャンプ
- ▶ x86 の場合
 - ▶ int 0x80h 命令
 - ▶ syscall 命令
- ▶ ある定められた番地は、「割り込みベクタ」と呼ばれるメモリ上の配列にかかれており, OS が起動時に設定する



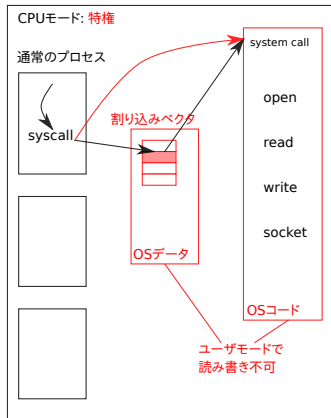
トラップ命令

- ▶ 以下の二つを行う
 - ▶ ユーザモードから特権モードへ移行
 - ▶ ある定められた番地にジャンプ
- ▶ x86 の場合
 - ▶ int 0x80h 命令
 - ▶ syscall 命令
- ▶ ある定められた番地は、「割り込みベクタ」と呼ばれるメモリ上の配列にかかれており、OS が起動時に設定する
- ▶ ユーザプログラムから OS への「入り口」 → システムコール



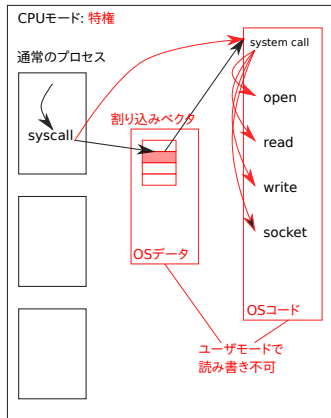
システムコール

- ▶ OSがユーザに対して提供している(根源的な)機能
 - ▶ 実例: open, write, read, close, fork, exec, wait, exit, socket, send, recv, etc.
 - ▶ 通常Cの関数として説明されているがこれは説明の便宜上 + ユーザの利便性のため
- ▶ 本当にシステムコールが呼び出されている瞬間は, トラップ命令(前スライド)でOS内の命令に突入する瞬間



システムコール

- ▶ OSがユーザに対して提供している(根源的な)機能
 - ▶ 実例: open, write, read, close, fork, exec, wait, exit, socket, send, recv, etc.
 - ▶ 通常Cの関数として説明されているがこれは説明の便宜上 + ユーザの利便性のため
- ▶ 本当にシステムコールが呼び出されている瞬間は, トラップ命令(前スライド)でOS内の命令に突入する瞬間



保護とシステムコール(すべての保護の基礎)

- ▶ OS 内には無数の機能が命令列として存在しているが、ユーザプログラムからの「入り口」(特権モードで実行される最初の命令) がひとつしかない
- ▶ その唯一の入り口から分岐してすべての機能ごとのシステムコールが実行されている
- ▶ ユーザプログラムが正規の入り口(システムコール)を通らずに、特権モードに移行することはできない
- ▶ OS 内部の(特権モードで実行される)プログラムをしっかりと書けば、OS を保護可能

