分散システム 第3回
― プロセス ―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

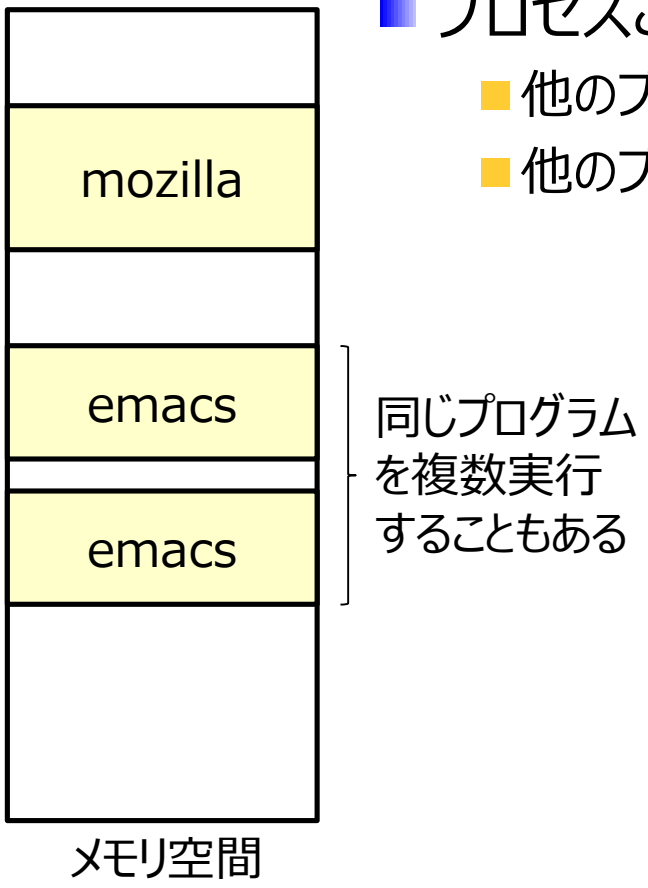
- プロセス
- マルチプロセス・マルチスレッド
- 仮想化
- コード移動

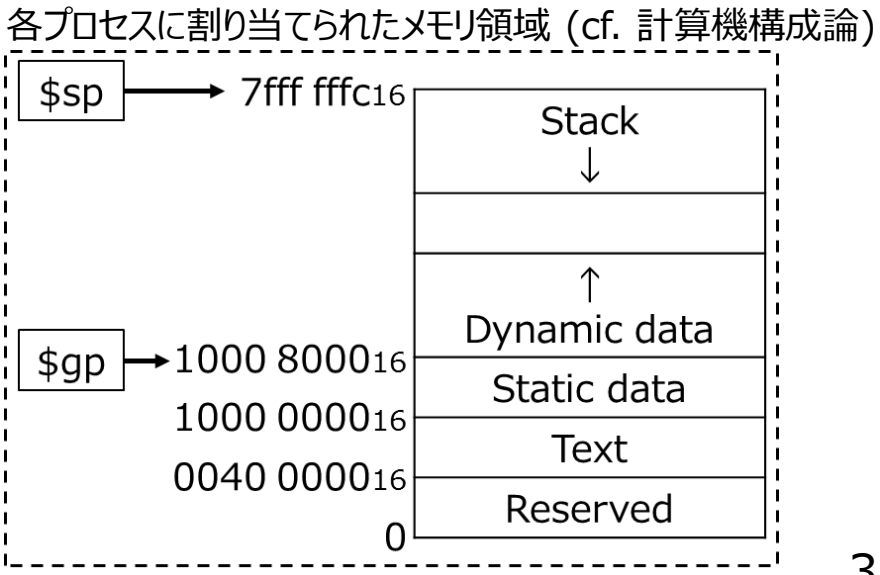
プロセスとプログラム

- プログラム：
  - 処理の手順をプログラミング言語で表したもの (＝ソースコード)
  - 実行可能なアプリケーション、実行ファイル
- プロセス：
  - プログラムに従ってプロセッサが実行する一連の動作・処理
    - プログラムのインスタンス

マルチプロセス

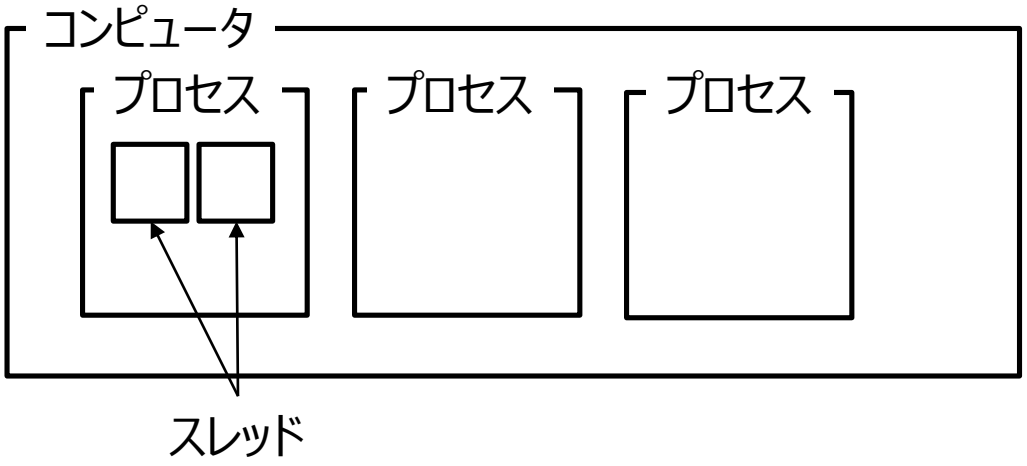
- 一つのOS上では複数のプロセスが実行される
  - プロセスごとに専用の領域が確保される
    - 他のプロセスから参照や書き換えがないように保護される
    - 他のプロセスと共有したい場合は共有メモリを使用





マルチスレッド

- 1つのプロセスが複数のスレッドにより実行される
  - スレッド(thread)：
    プロセス内部の実行の筋(線)
  - 各スレッドは並列的に実行される



プロセスの実行

- OSはプロセステーブルでプロセスを管理
  - レジスタの値、メモリ割り当ての状況、オープンされたファイル 等々
- 複数のプロセスが同一リソースを共有する場合もある → OSが管理
- プロセス生成は高コスト
  - 1プロセスに独立したメモリ空間1つ
  - 複数のプロセスでのCPUの切り替え

スレッドの実行

- スレッドの実行
  - スレッドもプロセスと同様、それぞれ独立して実行
  - 高度な並行透過性は実現しない
    - (例) 複数スレッドから同一データにアクセスする場合、支障がないようにプログラムの開発者が注意する必要
  - ユーザ空間でスレッドを実装
    - OSはプロセス内部に複数のスレッドがあることを知らないので、複数のプロセッサは割り当てられない
  - カーネル空間でスレッドを実装
    - カーネル(kernel)：OSの中核部分
    - スレッドごとにプロセッサを割り当てることも可能だが、OSの設計・実装時にマルチスレッドに対応しておく必要

分散システムにおけるスレッド

- クライアントにおけるマルチスレッドの必要性
  - (例) Webブラウザ
    データの読み取りに時間がかかる(通信遅延)
    → 全データの準備ができる前に表示・操作できるようにするべき
- サーバにおけるマルチスレッドの必要性
  - (例) ファイルサーバ
    - ディスパッチャ(dispatcher)がファイル操作要求を読み取る
      → ワーカスレッド(worker thread)に処理が割り当てられる
      → ワーカスレッドはファイルシステムからの読み取りを行う
      → 読み取りの最中は他のスレッドを実行
    - 単一スレッドであれば、ファイル読み書きの間、他の要求を受け付けられない

確認問題

- 以下の説明に合う語句を答えよ。
  - プログラムに従ってプロセッサが行う一連の処理。
    OSによって実行状態が管理される。
  - プロセス内部の実行の筋。並列的に実行することもできる。
- 空欄に当てはまる語句を答えよ。
  - 一つのマシンで複数の処理を同時に行うためには、
    (1)スレッドや(1)プロセスが有効である。
- 以下の各文は正しいか。○か×で答えよ。
  - クライアントマシンにおいて、通信遅延等により処理に時間がかかる場合、プログラムをマルチスレッド化することで、処理を待たなくてもユーザの操作を受け付け可能となる。
  - マルチプロセス実行においては、マルチスレッドとは異なり、並列処理におけるOSのサポートが得られないため、高度な並行透過性は実現しない。

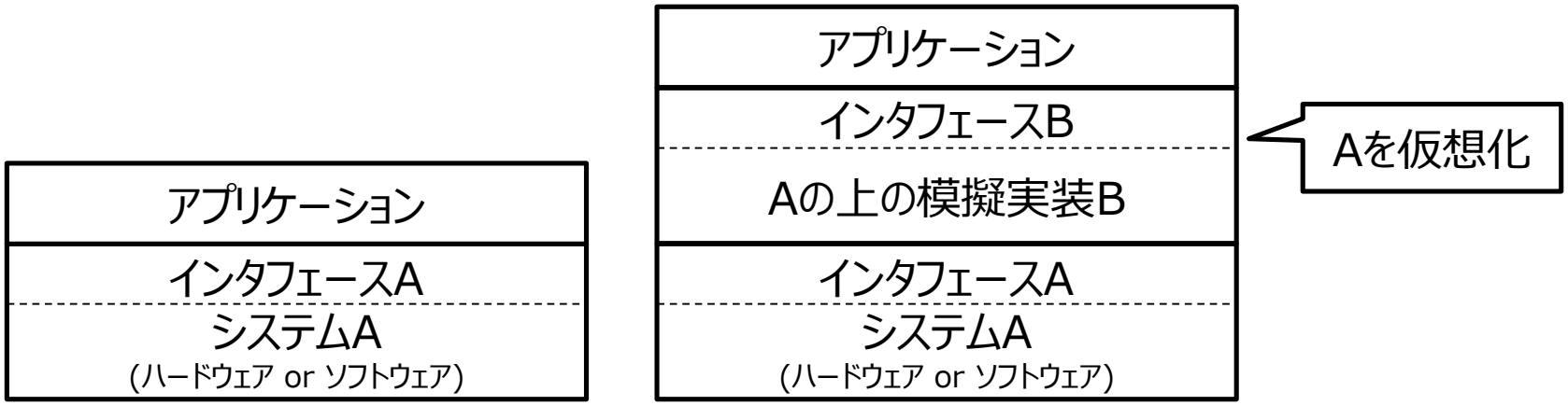
講義内容

- プロセス
  - マルチプロセス・マルチスレッド
  - 仮想化
  - コード移動

仮想化 (virtualization)

- “見せかけ”の処理
  - (例) CPU1つの計算機における
    マルチプロセス、マルチスレッド
    → 1つのCPUを高速にプロセス・スレッド
    間で切り替えて、仮想的に複数のCPUが
    あるように見せかけている

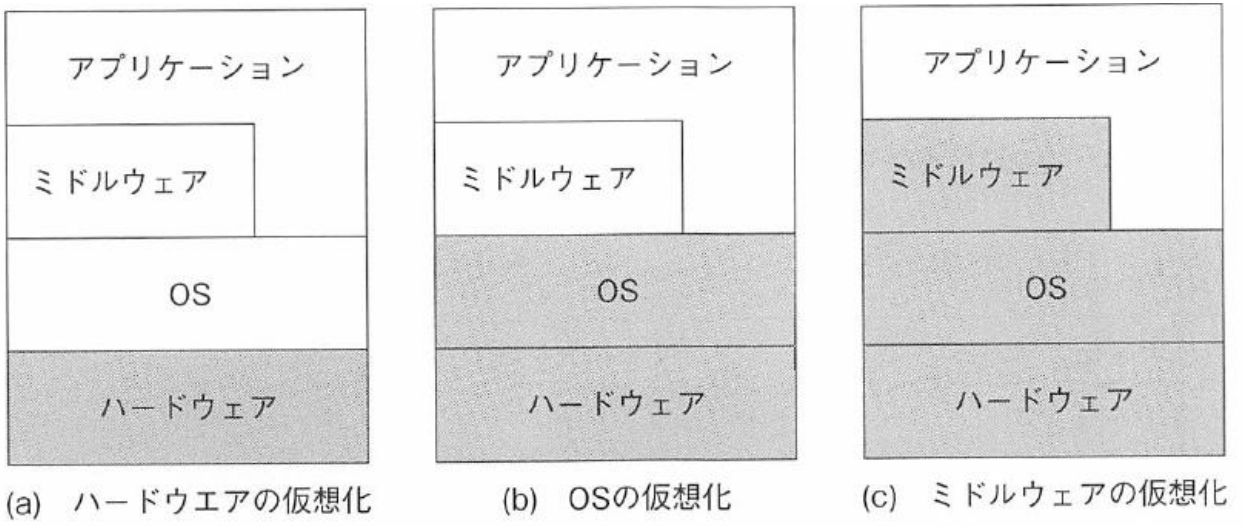
分散システムにおける仮想化



- 仮想化の目的
  - 分散システムにおける種々の透過性の提供
    - (例) 右上の場合、アプリケーションはシステムAについて意識する必要がない
  - 仮想化によって、実際のシステムの違いを吸収することができる
  - 可搬性、柔軟性の向上

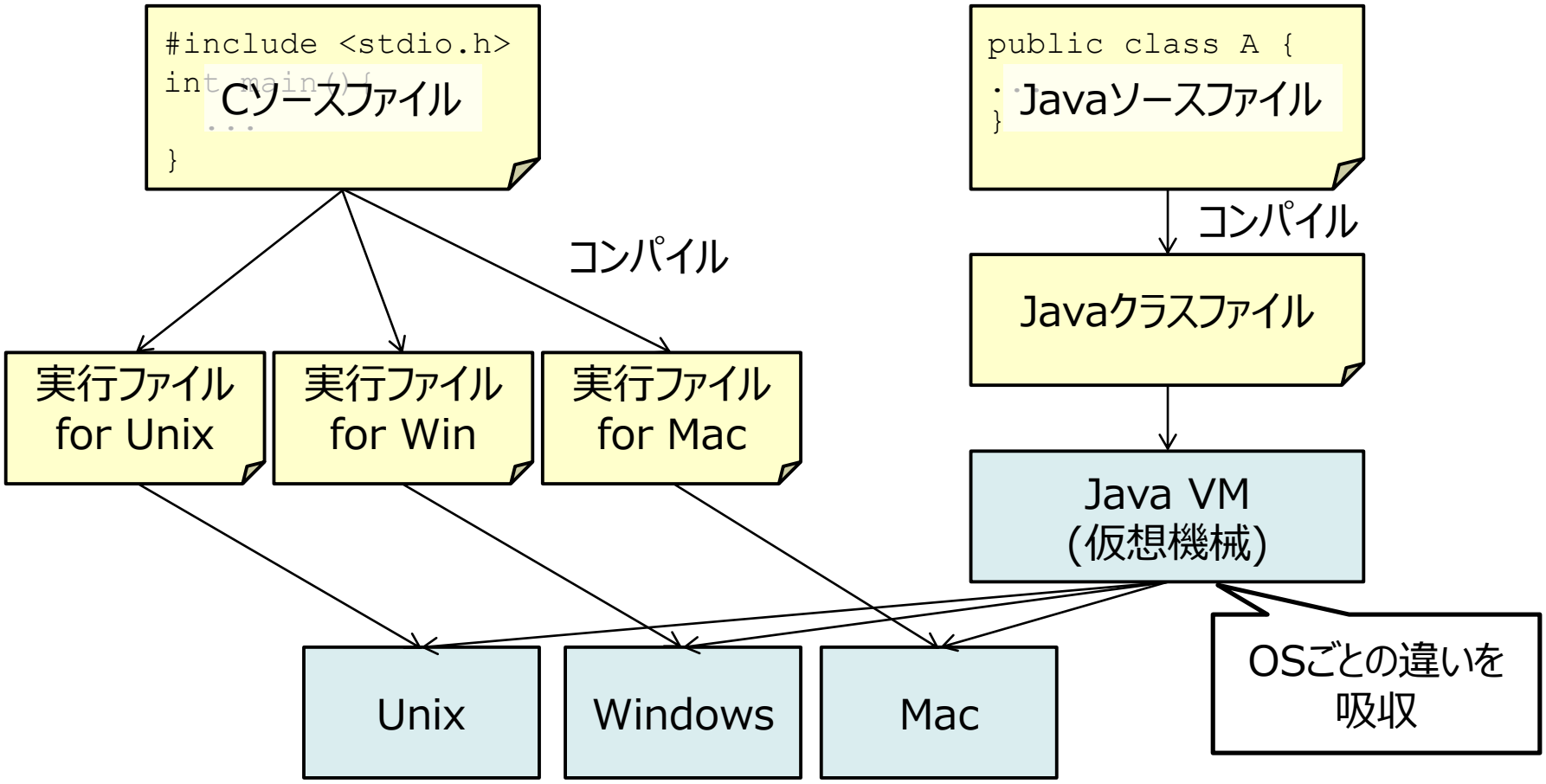
仮想化とインターフェース

- アプリケーション・ミドルウェア・OS・ハードウェア
  それぞれの間のインタフェースを模擬することで
  仮想化を実現できる



どの部分を仮想化するかにより、仮想化の分類ができる
(e.g., ハードウェア仮想化、OS仮想化、ミドルウェア仮想化)

(例) JavaVM


#include <stdio.h>
int main(){
...
}
Cソースファイル
public class A {
...
}
Javaソースファイル
コンパイル
コンパイル
実行ファイル for Unix
実行ファイル for Win
実行ファイル for Mac
Javaクラスファイル
Java VM (仮想機械)
Unix
Windows
Mac
OSごとの違いを吸収

講義内容

- プロセス
  - マルチプロセス・マルチスレッド
  - 仮想化
  - コード移動

コード移動 (code migration)

- プログラムを他のマシンに移動
  - 分散システム内でやり取りされるのは
    データだけではない
  - 実行中にも移動の可能性有
- なぜ必要か
  - 処理待ちを減らしたり、負荷の集中を避けるため、
    CPUが空いている所へプロセスを移動
  - 通信量を削減
    - データの処理はデータの近くで行う
  - 柔軟性の向上
    - システム構成を動的(dynamic)に変化
    - (例) 必要になった時点で最新のクライアント
      アプリケーションをダウンロードする

コード移動のためのモデル

- [Fuggetta-98]のモデル
  - プロセスを構成する3つのセグメント
  - 弱移動性
  - 強移動性
  - 送信者起動、受信者起動
  - リソースの保護

[Fuggetta-98]のモデル

- プロセスを構成する3つのセグメント
  - コードセグメント：実行されるプログラム
  - リソースセグメント：
    ファイル、デバイス等プロセスによって必要とされる外部リソースへの参照
  - 実行セグメント：
    プロセスの実行状態、メモリ上のスタック、プログラムカウンタ等

[Fuggetta-98]のモデル

- 弱移動性
  - コードセグメントのみを移動
    (初期化データを含むこともある)
  - (例) Javaアプレット
- 強移動性
  - 実行セグメントも移動
  - 実行中のプログラムを停止し、別のマシンに移動し、実行再開
  - 汎用性に富むが、実行が困難

[Fuggetta-98]のモデル

- 送信者起動
  - クライアントがサーバにコードを送信
- 受信者起動
  - サーバがクライアントにコードを送信
- リソースの保護
  - 特に送信者起動の場合、
    サーバのリソースの保護が重要

コード移動とリソース

- リソースは必ずしも伝送できない
  - コード移動時、プロセス・リソースバインディングは変更してはならない
- プロセス・リソースバインディング [Fuggetta-98]
  プロセスからリソースにどのようにアクセスするか
  - 識別子によるバインディング
  - 値によるバインディング
  - 型によるバインディング
- リソース・マシンバインディング [Fuggetta-98]
  リソースがマシン上で移動可能か、拘束されているかどうか
  - 非結合リソース
  - 拘束リソース
  - 固定リソース

コード移動とリソース

- プロセス・リソースバインディング [Fuggetta-98]
  - 識別子によるバインディング
    - (例) URLによるアクセス、IPアドレスによるアクセス
  - 値によるバインディング
    - リソースの値のみが必要な場合
  - 型によるバインディング
    - 特定種類のリソースが必要な場合
      (例) モニタ、プリンタ等への参照
- リソース・マシンバインディング [Fuggetta-98]
  - 非結合リソース
    - 移動可能なリソース
      (例) 移動するプログラムのみで使うファイル
  - 拘束リソース
    - 移動可能だが高コスト (例) ローカルデータベース
  - 固定リソース
    - 移動不可能 (例) ローカルデバイス

コード移動とリソース

リソース・マシンバインディング

プロセス・リソースバインディング	非結合リソース	拘束リソース	固定リソース
識別子による	MV(またはGR)	GR(またはMV)	GR
値による	CP(またはMV, GR)	GR(またはCP)	GR
型による	RB(またはMV, CP)	RB(またはGR, CP)	RB(またはGR)

GR　グローバルなシステムにわたるレファレンスを確立する
MV　リソースを移動させる
CP　リソースの値をコピーする
RB　ローカルに利用できるリソースにプロセスをバインドし直す

図 3.19　コードが別のマシンに移動したとき、ローカルのリソースへのレファレンスに関して取りうるアクション

グローバルリファレンス：
マシンの境界を越えることのできる参照　(例) URL

コード移動の注意点

- 異種システムへの移動
  - 分散システムは一般的に異種プラットフォームの集まりで構成
  - 移動先のプラットフォームが移動したコードの実行をサポートしている必要
  - 抽象機械の利用によりプラットフォームの差異を吸収するのが一つの解決策
    - (例) Java VM、スクリプト言語
- 実行環境全体を移動
  - マシンのシャットダウン等への対策
  - メモリページの送り出し(pushing)等

確認問題

- 以下の各文はどの語句について述べたものか。
  最も適したものを語群から選べ。
  - 1つのCPUを高速に複数プロセス間で切り替え、
    仮想的に複数のCPUがあるように見せる。
  - プログラムを他のマシンに移動して実行する。

  語群： 仮想化、仮想機械、コード移動、遠隔プロセス呼び出し

- 分散システムにおけるコード移動の利点を3つ挙げよ。
- Fuggettaのモデルにおける弱移動性と強移動性の
  違いを説明せよ。

参考文献

- 「分散システム」
  水野 忠則 監修、共立出版、2015
- 「分散システム 原理とパラダイム 第2版」
  アンドリュー・S・タネンバウム 他 著、
  ピアソン・エデュケーション、2009