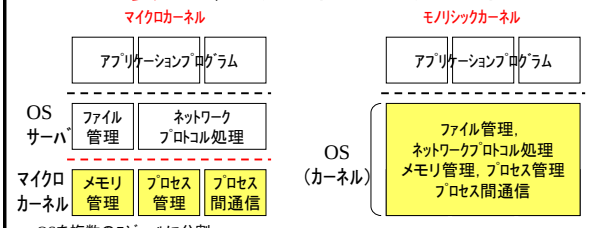
第13回　仮想計算機・プロセス間通信

重要：モノリシックカーネルとマイクロカーネル

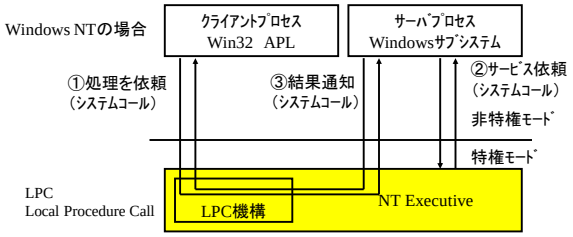


OSを複数のモジュールに分割
・特権モードで動作するマイクロカーネルには
プロセス管理，メモリ管理，プロセス間通信等
OSの最低限の機能のみを残す
・他の付加的機能は，非特権モードで動作
するように階層化する
利点：管理が簡単，デバグし易い
欠点：オーバヘッドが大きい

構成：OS全体を1モジュールで構成
利点：効率が良い（オーバヘッドが少ない）
欠点：
・OSの肥大化により，管理が困難
（機能の追加・変更が難しい）
・特権モード(注)で動作するプログラム量が多く
デバグが難しい（特権命令を使用）

：特権モード（カーネルモード）　：非特権モード（ユーザモード）

重要：マイクロカーネルのシステムコール



図の例では，クライアントプロセスは，Windowsサブシステム（サーバプロセス）に対して，1回のシステムコールを実行．しかし，実際には，サーバプロセス（非特権モードで動作）もカーネルであるNT Executiveに対し，2回のシステムコールを実行している（計3回(注)）．

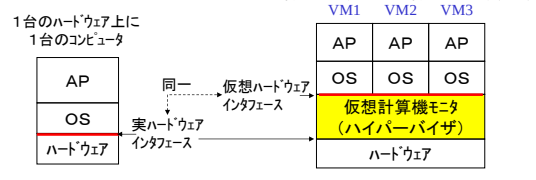
注：モノリシックカーネルでは1回のみ．

マイクロカーネルの欠点：オーバヘッドが大きい
（割り込み分析，モード切替，レジスタの退避など種々の処理が必要）．

重要：仮想計算機（VM：Virtual Machine）

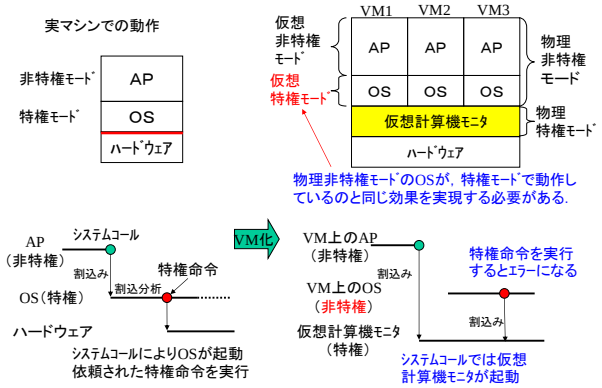
コンピュータのハードウェア資源を仮想的に分割し，複数の独立な実行環境を提供する機構(注)．
仮想計算機（VM）毎にOSをローディングして実行できる．OSは同一でも異なっても良い．
・仮想計算機モニタがVM毎のハードウェアをエミュレートし，実ハードウェアと同一のインタフェースを提供．
・CPUスケジューリングと仮想記憶により，各VM専用の仮想CPUと仮想メモリを提供．
・ディスクを領域分割し，VM毎に独立したファイル管理などができるようにする．
利点：強力な保護機能を持つ（他の仮想計算機とは，完全に独立）
欠点：命令の解釈実行などのオーベヘッドがある（VMあたりのCPU性能＜1／n）
仮想計算機間で，直接の資源共用ができない（ネットワーク経由の通信などで共用）

注：当初は，OSのデバッグや乗換えを効率化する目的で開発された．



エミュレート（Emulate）：まねる，模倣する

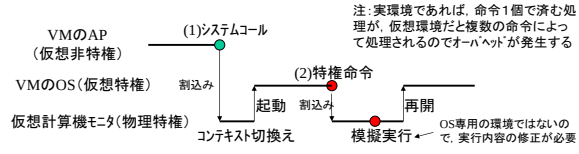
重要：仮想計算機実現の課題



重要：仮想計算機の実現

（1）システムコールのエミュレート(注)
・アプリケーションプロセス（仮想非特権モード）がシステムコールを発行
・内部割込み（原因：システムコール命令）により，仮想計算機モニタが起動される．
・コンテクスト切換えを行い，仮想特権モードにしてVMのOS（割込み分析）を起動．
以上により，システムコールによってVMのOSが直接起動されたのと同じ効果を実現．
（2）特権命令のエミュレート(注)
・OS（仮想特権モード）＝物理非特権モード）はシステムコールの処理を開始し，特権命令実行．
・内部割込み（原因：不正命令実行）により，仮想計算機モニタが起動される．
・OSに代わって，特権命令を模擬的に実行
・再び，仮想特権モードにして，OSを再開（特権命令以降の命令を実行させる）
以上により，OSが直接特権命令を実行したのと同じ効果を実現

注：実環境であれば，命令1個で済む処理が，仮想環境だと複数の命令によって処理されるのでオーバヘッドが発生する

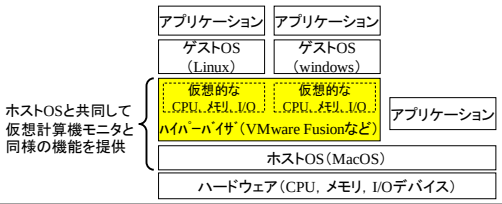


ホストOS型ハイパーバイザによる仮想化

以下に示すようなホストOS型のハイパーバイザが普及してきている.

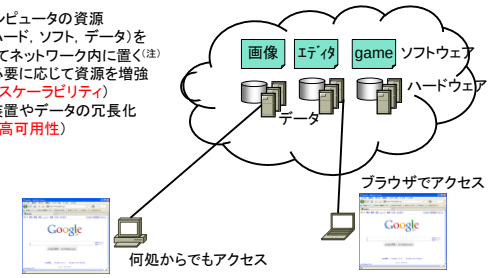
ハイパーバイザをホストOSのアプリケーションとしてインストール
ホストOSと共同して，各ゲストOSに対して仮想的な物理ハードウェア機能を提供する.
・ハードウェア環境を変更しないで（ディスクのパーティションなど），新しいOSを試用できる
・デバグ中のゲストOSがクラッシュしてもホストOS，ハードウェア本体に影響しない
・ホストOSによる処理が余分に必要なのでベアメタル型よりもさらにオーバヘッドが大きい.

本来の仮想計算機モニタ型のハイパーバイザをベアメタル型とも言う　bare metal：裸の金属
後述するクラウドコンピューティングの仮想サーバではベアメタル型を使う.　（実ハードウェアという意味）



クラウドコンピューティングの概念

コンピュータの資源
（ハード，ソフト，データ）を
全てネットワーク内に置く(注)
・必要に応じて資源を増強
（スケーラビリティ）
・装置やデータの冗長化
（高可用性）



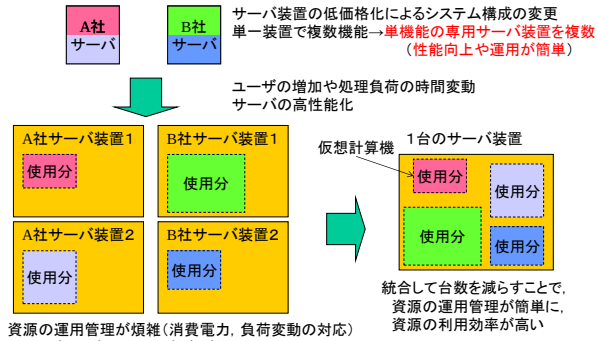
PC上には，ソフトをインストールする必要が無い．PC上にデータを置かない．
どこからでも，どのPCからでも同じ環境，同じ機能で安全に作業ができる.

注：正確にはネットワークのバックボーンに接続されたデータセンタ内に置く

クラウドコンピューティングの要素

- ハウジング
  - 高速なネットワークに接続されたデータセンタに自分のサーバを預かってもらい，保守や管理をアウトソースする.
- ホスティング
  - データセンタ内のサーバおよびソフトウェアを借りて使う.
  - 特に，複数のユーザでwebサーバの機能をシェアして使う.
- SaaS（software as a service）
- ASP（アプリケーション・サービス・プロバイダ）
- SOA（サービス指向アーキテクチャ）
  - アプリケーションプログラムを借りて使う
- サーバ計算機の仮想化（仮想サーバ，仮想マシン）
  - 1台のサーバ計算機を複数の仮想計算機として各ユーザに提供
- グリッドコンピューティング
  - 複数のコンピュータを1台のコンピュータのように見せる

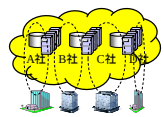
サーバ仮想化(注)の必要性



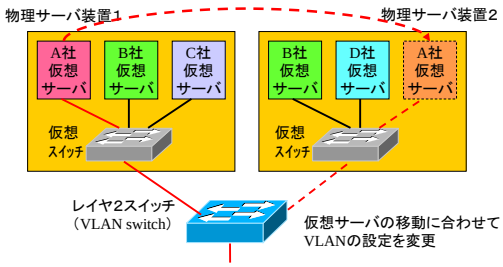
注：サーバはプロセスを指すため，仮想計算機というのが正しいが，世間ではサーバ仮想化と言っている.

プライベートクラウドの提供とその運用

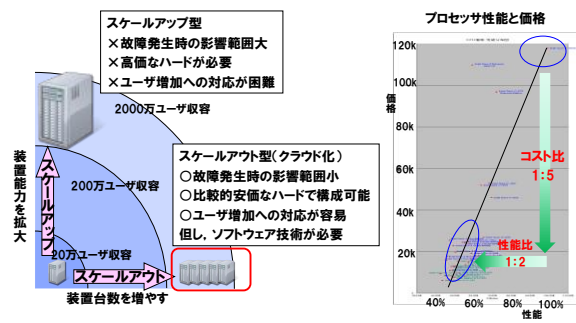
プライベートクラウド：自社専用のクラウド
最近はデータセンタの設備を共有してプライベートクラウドを提供するソリューションがトレンドとなっている.



必要な性能を維持するため，物理サーバ装置を移動



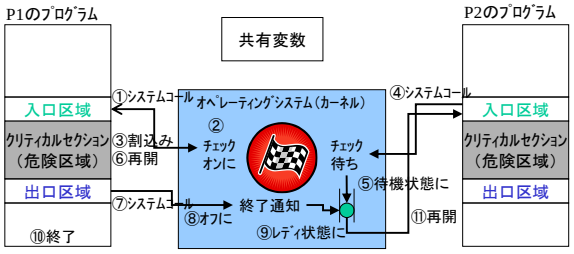
性能向上の手法



ｸﾗｳﾄﾞｺﾝﾋﾟｭｰﾃｨﾝｸﾞﾌﾟﾗｯﾄﾌｫｰﾑ
Google App Engine，Amazon EC2，Windows Azure・・・

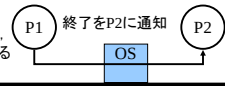
第3回のスライド

排他制御の実現方法とﾌﾟﾛｾｽ間通信



問題点1の解決法：ｼｽﾃﾑｺｰﾙにより，OSがﾌﾗｸﾞを操作する②④⑧
なお，OSがﾌﾗｸﾞを操作中は，割込み禁止（またはﾌﾗｸﾞのﾃｽﾄとｾｯﾄを一度に行う命令を使う）

問題点2の解決法⑤⑨⑩
ﾌﾗｸﾞｵﾝの場合，ﾌﾟﾛｾｽ（P2）を待機状態にし，
P1のｸﾘﾃｨｶﾙｾｸｼｮﾝ終了後にﾚﾃﾞｨ状態にする
（左図のようなﾌﾟﾛｾｽ間通信とみなせる）

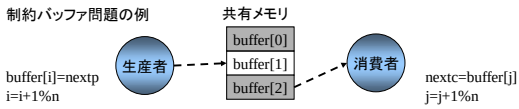


プロセス間通信

- 協調が必要なプロセス間でデータの授受（通信）を行う
  - 同期問題はプロセス間通信の一部（危険区域を出た通知）
- 共有メモリ方式（APがプログラムの中で通信機能を実現）
  - 複数のプロセスがアクセス可能な共有メモリを用意し，この領域を利用してデータの受け渡しを実現する
- メッセージシステム方式（OSが提供する通信機能を利用）
  - ｼｽﾃﾑｺｰﾙを用いて，ﾒｯｾｰｼﾞの形で通信する
  - send（destination, message)/receive（source, message）
  - 直接通信と間接通信

プロセス間通信（共有メモリ方式）

共有メモリ方式（APがプログラムの中で通信機能を実現）
　複数のプロセスがアクセス可能な共有メモリを用意し，この領域を利用してデータの受け渡しを実現する



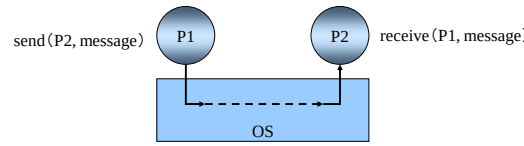
buffer[0]～buffer[n]は，生産者，消費者が共有するメモリ

lock/unlock，ｾﾏﾌｫなどを用いた排他制御も必要であり，APが複雑となる
メモリへのアクセスのみで通信が可能なため，効率が良い

プロセス間通信（ﾒｯｾｰｼﾞｼｽﾃﾑ方式：直接通信）

OSが提供する通信機能を利用：ｼｽﾃﾑｺｰﾙを用いて，ﾒｯｾｰｼﾞの形で通信する

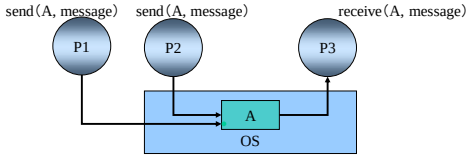
直接通信では，通信相手のプロセス名を指定して，ﾒｯｾｰｼﾞを送受する
　通信相手が固定的に決まっている1：1通信を簡単に実現
　ﾓｼﾞｭｰﾙ性が悪い：プロセス名を変更すると相手プロセスに影響する




```
producer：  /*生産者*/
do{
  …
  ﾃﾞｰﾀを読んでnextpへ
  …
  send (consumer, nextp)
} while (1)
```

```
consumer：  /*消費者*/
do {
  receive (producer, nextc)
  …
  nextcのﾃﾞｰﾀを出力
  …
} while (1)
```

プロセス間通信（ﾒｯｾｰｼﾞｼｽﾃﾑ方式：間接通信）



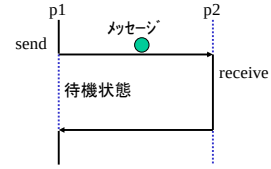
間接通信では，ﾁｬﾈﾙ（channel）(注)を介してﾒｯｾｰｼﾞを送受する
　ﾁｬﾈﾙを共有するﾌﾟﾛｾｽ間にﾘﾝｸが作られ，通信ができる
　1：多通信（受け手が複数），多：1通信（送り手が複数），多対多通信（左記の組み合わせ）が可能

ｸﾗｲｱﾝﾄ～ｻｰﾊﾞ間の通信（多対1通信）を実現
ﾒｯｾｰｼﾞの受信（ｻｰﾊﾞ）は，通信相手（送信ﾌﾟﾛｾｽ名）を指定しない方式
（ﾁｬﾈﾙの変数名を指定し，受信時に送信ﾌﾟﾛｾｽのidを得る）
直接通信方式よりもﾁｬﾈﾙを介した間接通信方式の方が柔軟である．

注：ﾎﾟｰﾄ（port），ﾒｰﾙﾎﾞｯｸｽ（mailbox）とも言う
　　TCP，UDPで用いるﾎﾟｰﾄ番号は，ﾁｬﾈﾙ識別子である

参考：同期式通信と非同期式通信

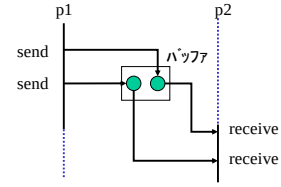
同期式通信



受信ﾌﾟﾛｾｽがﾒｯｾｰｼﾞの受信処理を終えるまで待たされる（待機状態となる）
受信処理完了により，send命令が終了．
（ﾒｯｾｰｼﾞが届いたことが分かる）

通常のクライアント・サーバの通信はこの形式

非同期式通信

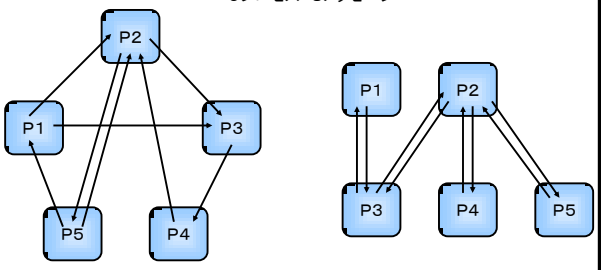


受信ﾌﾟﾛｾｽがﾒｯｾｰｼﾞの受信しなくてもﾒｯｾｰｼﾞがﾊﾞｯﾌｧﾘﾝｸﾞされ，送信ﾌﾟﾛｾｽは，待たずに実行を続けることができる．
但し，send命令で，ﾒｯｾｰｼﾞが届いたかどうかは分からない

短時間に処理を終わらせる必要がある場合は，この形式が採られる．

プロセス間通信の構造化

5プロセス　8メッセージ

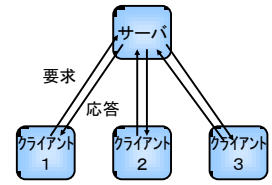


構造化されていない通信　　構造化された通信

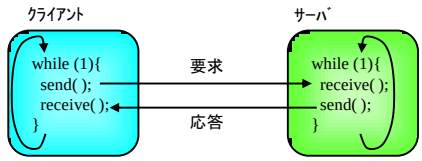
各プロセスの機能分担を見直し，見通しを良くする

重要：クライアントサーバモデル

- プロセスを2種類に分ける
- クライアント
  - サービスを受けるプロセス．単一ユーザが利用．
  - 要求メッセージを送る（能動的）
- サーバ
  - サービスを提供するプロセス．複数のクライアントが利用．
  - 応答メッセージを返す（受動的）



クライアントサーバの通信パターン

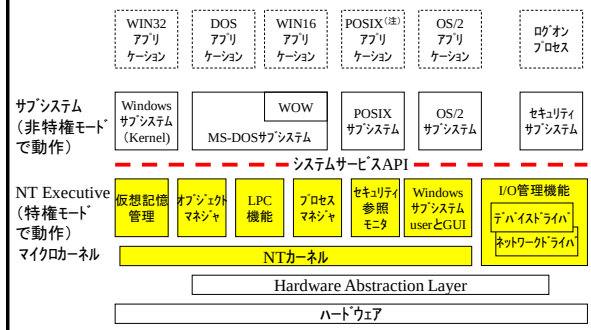


クライアントは，要求メッセージを1回送信（send）し，応答メッセージを1回受信（receive）
サーバは，要求メッセージを1回受信（receive）し，応答メッセージを1回送信（send）
双方，このパターンを守り，必要な回数繰り返す．

①先ず，クライアントが動く．（サーバはクライアントが要求するまで待つ．）
②次に，サーバが動く．（クライアントは，サーバが応答するまで待つ．）
このように，単純な構造で，プロセス間の同期をとる．

前回のスライド

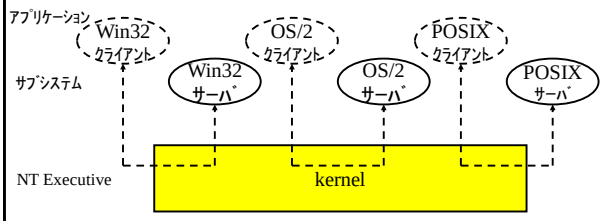
Windows NT/2000/XP



注：POSIX（Portable Operating System Interface：Unixの標準の一つ）

前回のスライド

（重要）WindowsNTクライアント-サーバ構造



マイクロカーネルでは，アプリケーション（利用者プログラム，クライアント）は，カーネルを経由したメッセージ通信を行うことによりOSのサブシステム（サーバ）の機能を利用する（クライアント，サーバともに利用者モードで動作する）

クライアント，サーバという用語はプロセスの機能分担を指す．コンピュータ間の関係ではない．（クライアント≠PC等の装置，サーバ≠ワークステーション等の装置）

クライアントサーバのシステム構成