計算機アーキテクチャ講義
ノート５
（129頁～162頁）

平成19年12月17日配布

今瀬　真

メール：imase@ist.osaka-u.ac.jp
http://www.ispl.jp/~imase/lecture/comp-arch/2007/

４　システム・アーキテクチャ

CPUシステム・アーキテクチャ

- オペレーティングシステムとハードウェアの接点となるアーキテクチャ
- オペレーティングシステム：　キーボード入力や画面出力といった入出力機能やディスクやメモリの管理など、多くのアプリケーションソフトから共通して利用される基本的な機能を提供し、コンピュータシステム全体を管理するソフトウェア。「基本ソフトウェア」とも呼ばれる。アプリケーションソフトウェアの開発者は、OSの提供する機能を利用することによって、開発の手間を省くことができ、アプリケーションの操作性を統一することができる。また、ハードウェアの仕様の違いはOSが吸収してくれるため、あるOS向けに開発されたソフトウェアは、基本的にはそのOSが動作するどんなコンピュータでも利用できる。

4　システム・アーキテクチャ

オペレーティングシステムの機能で最も重要な機能は次の2つである。ハードウェアとの依存関係が強いものに次の3つの機能がある。

1. 多重処理：1台の計算機を複数のプログラムで共同使用
   - プロセッサの仮想化：プロセッサを仮想的に専有
   - 主記憶の仮想化：主記憶を仮想的に専有

   物理的な資源（プロセッサ、主記憶など）を時分割、空間分割で使用する。このための資源（リソース）管理を実現する方法が*システムアーキテクチャ*
2. 入出力の仮想化：様々な外部記憶装置をファイルシステムとして仮想化。物理的な資源（入出力装置など）を時分割、空間分割で利用するのは1と同様である。

4．1　プロセッサの仮想化

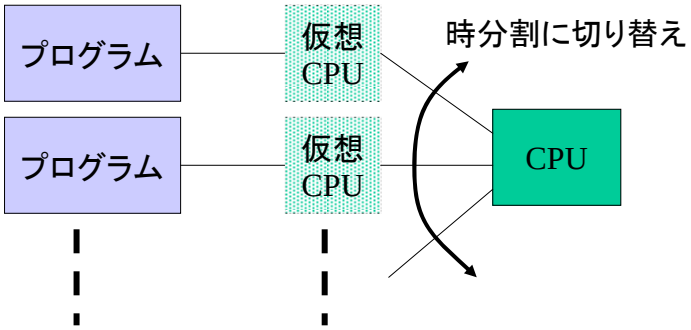
本節を完全に理解するためには、3年次前期の「オペレーティングシステム」の授業を受けないと困難である。詳細の仕組みの理解よりは、概念の理解に専念すること。また、オペレーティングシステムの授業を理解するためには、この講義の割り込み機能を理解は必須である。（教科書130頁から131頁まではよく読んで完全に理解すること）

1. プロセスとは
2. 割り込みの機能とその必要性
3. 並行に動作する場合の問題点：排他制御、デッドロック、状態合わせ

4.1.1プログラムの多重処理とプロセス

- 多重処理の必要性：CPUに比べて入出力装置の動作は非常に遅い。入出力装置に指示を出した後、処理の完了まで他のプログラムを実行することにより、CPUを効率的に利用できる。

プロセス：
仮想プロセッサで処理中のプログラム
プログラムからは、あたかも計算機を仮想的に占有しているように見える。これを仮想化と言う



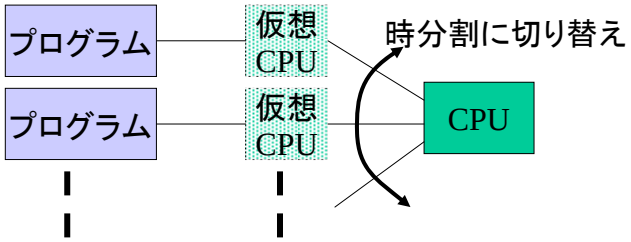
4.1.1プログラムの多重処理とプロセス

多重処理の方式（切り替えの契機）

- タイムスライシング方式：時間（数十ms～数百ms）ごとに次々にプログラムの実行を切り替える。
- 事象駆動（エベントドリブン）方式：事象（入出力要求の発生、入出力処理の終了など）を契機に切り替える。

通常は、両者と同時に実現

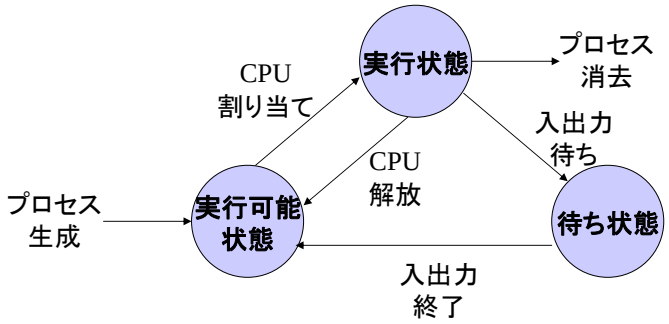
プロセス：
仮想プロセッサで処理中のプログラム



4.1.1プログラムの多重処理とプロセス

プロセスの状態
- 実行状態：CPUが割り当てられてプログラムが実行されている状態
- 待ち状態：入出力装置などの動作の終了を待っている状態
- 実行可能状態：実行可能であるがCPUが割り当てられていない状態

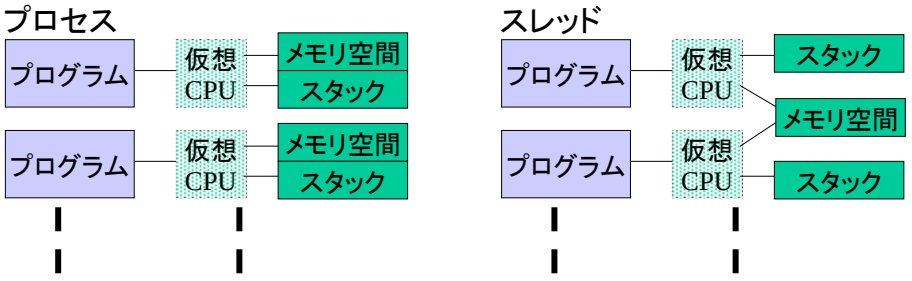
オペレーティングシステムがCPUの割り当てを実現



4.1.1プログラムの多重処理とプロセス

プロセスとスレッド
- プロセスは固有のメモリ空間・固有のスタック上で動作する
- スレッドは共有のメモリ空間・固有のスタック上で動作する

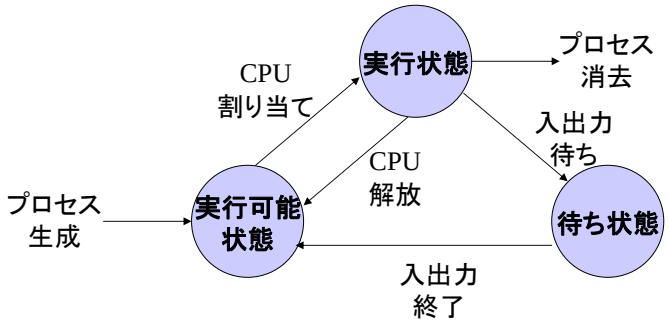
システムが管理するそれぞれのプログラムが1つのプロセスであり、1つのプロセスのなかにある並列実行の単位をスレッドと呼ぶ。たとえば Macintosh や Windows ではワープロなどのアプリケーションが1つのプロセスとして実行され、そのなかのウィンドウがそれぞれ1つのスレッドになっている



4.1.1プログラムの多重処理とプロセス

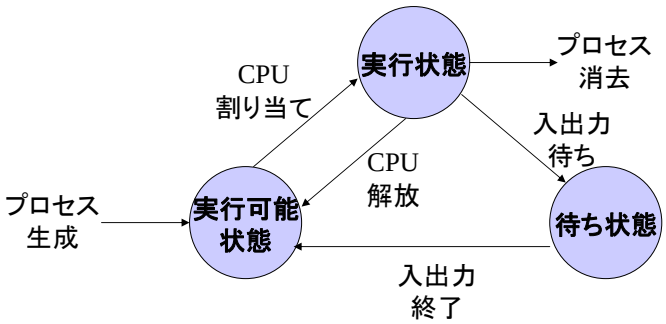
CPUの割り当てをどのように実現するか？

- 実行中のプログラムが時々入出力の動作やタイマーを調べる→非現実的
- 割り込み機能の利用



4.1.2　割り込み

割り込み機能を用いることによりプロセスの状態遷移の管理プログラムが容易に実現できる。

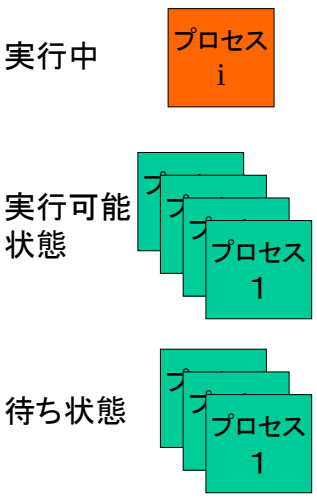


4.1.2　割り込み

- 割り込み：あるプログラム（割り込まれたプログラム）が実行の途中に**何かの契機**で（プログラムの内部要因でのジャンプでなく）実行を中断し他のプログラム（割り込んだプログラム）が走行すること。所定の処理が終了した際に、割り込まれたプログラムが動作を復旧できる。
- 割り込みの契機：障害、プログラム割り込み（桁あふれ、命令コードやアドレスが不正）、外部（タイマー）割り込み、入出力割込み、リスタート割り込み、SVC(Supervisor call)割り込み（何故サブルーチンでないかは4.1.3参照）など 。
- 割り込みをトラップ(TRAP)と呼ぶこともある。その場合外部要因におこるトラップを割り込みトラップと呼ぶ。

サブルーチンと同様、他のプログラムに実行が移り処理が終了するともとのプログラムに戻る。
他のプログラムへ実行が移る契機が、サブルーチンとことなり外部要因により強制的であることが異なる。

4.1.1　多重処理の実現法



SVC 割り込み

1. 入出力装置に動作を要求する。
2. 該当プロセスを待ち状態にする。
3. 実行可能なプロセスから次に実行するプロセスを選択する。
4. 選択したプロセスに制御を移す

タイマー 割り込み

1. 実行可能なプロセスから次に実行するプロセスを選択
2. 実行可能なプロセスに制御を移す
3. 選択したプロセスに制御を移す

入出力 割り込み

1. 割り込み要因を分析し、該当プロセスを待ち状態から実行可能状態に変更する
2. 割り込んだプロセスに制御を移す

4.1.2　割り込み

- 割り込みの機構：実行を中断して復旧できることが条件。ハードとソフトの組み合わせで実現。
- PS（プログラムの状態）を保存しないと割り込み後もとのプログラムに復帰できない

割り込み処理の手順（P.138の簡易版 ）

- 割り込み要因を識別するために詳細情報を割り込み原因レジスタに取り込むH
- **旧PSをセーブし、割り込みプログラム用のPSを現PSとする。H**
- **PCを設定して、割り込みプログラムを走行させる。H**
- 割り込みプログラムの走行。最初に割り込まれたレジスタのセーブS
- 詳細情報に従って、割り込み処理を行う。S
- 割り込まれたプログラムのレジスタのアンセーブS
- 旧PSを現PSとする。H
- (1)でセーブしたPCをアンセーブし、割り込まれたプログラムが走行H

サブルーチン処理とほぼ同様であるが、サブルーチンの場合はPSのセーブ、アンセーブまでは行わない。1の処理はサブルーチンの引数の受け渡しにあたる。

4.1.2　割り込み

PS（プログラム状態）とは

- PC（プログラムカウンター）
- CC（コンディションコード）：演算結果により立つフラッグ

通常の計算機ではこれ以外にも各種ある。これは計算機のアーキテクチャによる。以下はSPARCの例

- インプリメンテーションクラスとバージョン：機種を示す（機種により実行できる命令が異なる）
- 利用可能命令を示すフラッグ（コアプロセッサ機能、浮動少数点演算機能、特権命令実行機能）
- 割り込みレベル
- 割り込み禁止フラッグ
- カレントウィンドポインター（主記憶でアクセス可能な領域を指定するため）

4.1.2　割り込み

PS（プログラム状態）とは

- 広義な意味でのPS（プログラムの実行を再開するために必要なCPU内の状態）には汎用レジスターも入る。
- 汎用レジスタ（など）は、ソフトウェアでセーブ、アンセーブが可能である。一方PCやCCは本質的にハードウェアでのセーブ、アンセーブが必要である。
- 通常はハードウェアでサーブ、アンセーブを行う必要があるものをPSという。
- セーブ先は、ほとんどの場合主記憶で、固定された番地にセーブされたり、スタックにつまれたりする。

4.1.2　割り込み

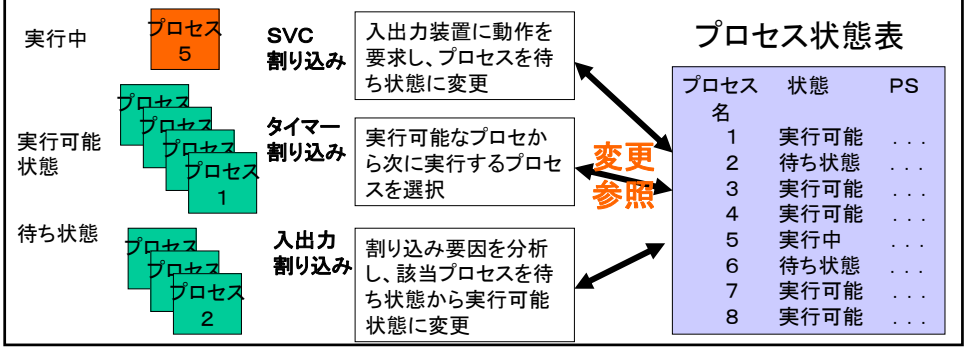
割り込み要因と割り込みレベル

- 割り込み要因は多岐に渡る（障害、プログラム割り込み（桁あふれ、命令コードやアドレスが不正）、外部（タイマー）割り込み、入出力割込み、リスタート割り込み、SVC(Supervisor call) など ）。
- 割り込み要因の重要度から、多重割り込みを許す必要がある。（例えば電源異常が起こった時は、即座計算機をシャットダウンする（ハードウェアやデータの保護）ための処理を行う必要がある。）
- 通常は要因毎に割り込みレベルが規定されている。現状のレベル（前述のPS内に規定）と同じまたはそれ以下の優先度の割り込みは禁止される。

4.1.2　割り込み

割り込み禁止

- 割り込み処理ルーチン間でデータを共有する。処理の途中で、他の割り込みが処理ルーチンが走行すると困る場合がある。（下記の例では、プロセス状態表の書き換え中）
- これを排除するために、割り込み禁止をプログラムで設定、解除する機能が必要となる。→割り込み禁止フラッグ



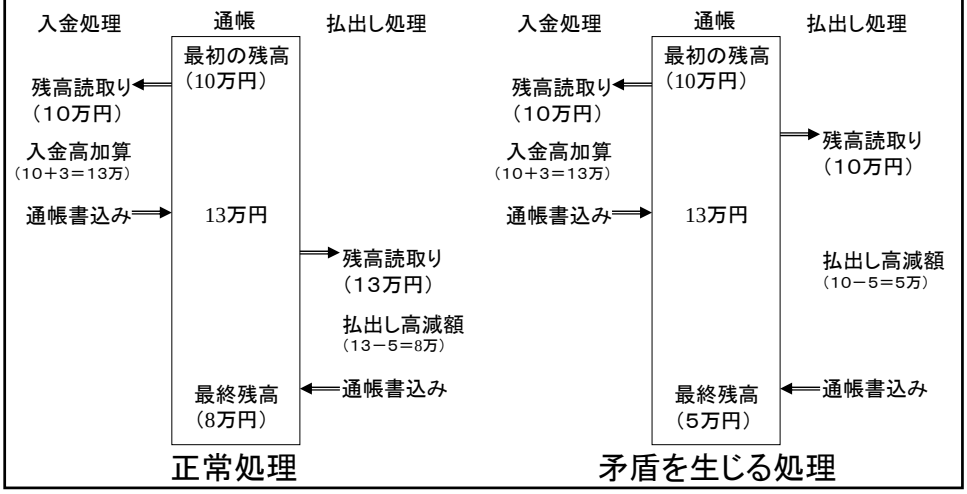
特別：並行動作の諸問題

複数プロセスが並行に動作する環境（1台のオペレーティングシステム上の複数プロセスや分散システム）では、「プロセス間で何らかの連携をとろうとすると」様々なややこしい問題がおこる。ここでは、教科書を離れて、この問題について説明する。しばしば遭遇するややこし問題の本質はほとんど下記の3つである。

- 排他制御
- デッドロック
- 状態合わせ

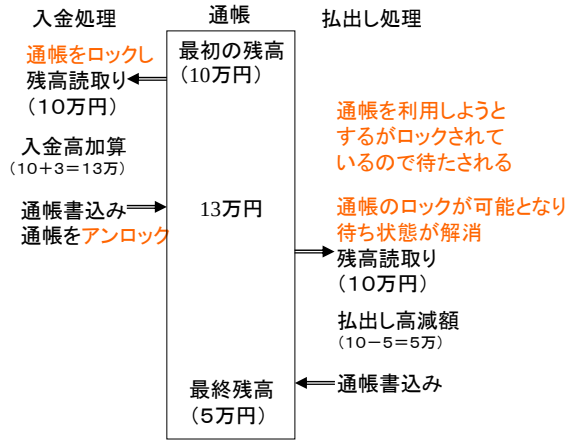
排他制御

- 複数のプロセスがデータを共有する場合の例：残高が10万円の預金通帳に3万円の入金と5万円の払い出しが独立して並行に実行される。



排他制御

- 矛盾が生じる原因：入金処理と払出し処理が同時に行われた。
- 解決策：通帳（資源）を処理の間占有して他の処理に使わせなくする。資源のロックとアンロック



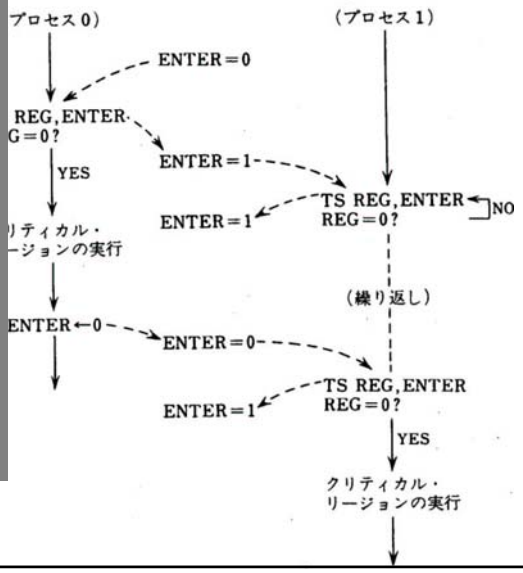
ではロック/アンロック機構の実現法は？

通帳に1ビットのフラッグ
　1：ロックされている
　0：ロックされていない
リードとライトが分離していては、実現できない（前述の矛盾が起こる）

排他制御

TS(Test and Set)命令：排他制御を実現するためのもっとも基本的なハードウェア機構

- アドレス部で指定された主記憶の内容(1ビットの錠前)をレジスタに読み込むとともに読み出した値の如何にかかわらず1を書き込む(鍵をかける(ロック))
- 読み出しと書込みは1メモリサイクルとして実行され、その間にはどのような処理も入らない。

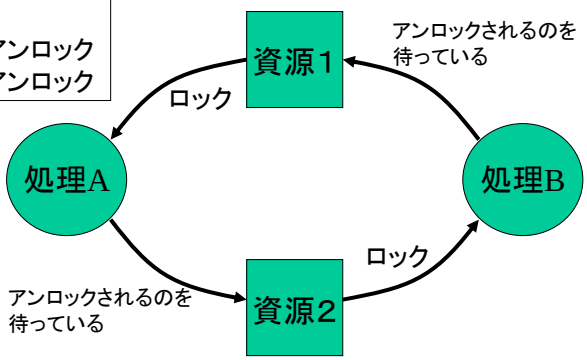


デッドロック

デッドロック：複数の資源をロック、アンロックする処理を入れると下記のような現象が起こる。

処理Aの手順	処理Bの手順
資源1をロック	資源2をロック
資源2をロック	資源1をロック
処理	処理
資源1をアンロック	資源2をアンロック
資源2をアンロック	資源1をアンロック

処理Aは処理Bが資源2をアンロックするのを待ち
処理Bは処理Aが資源1をアンロックするのを待って
処理が永遠に進まない状態

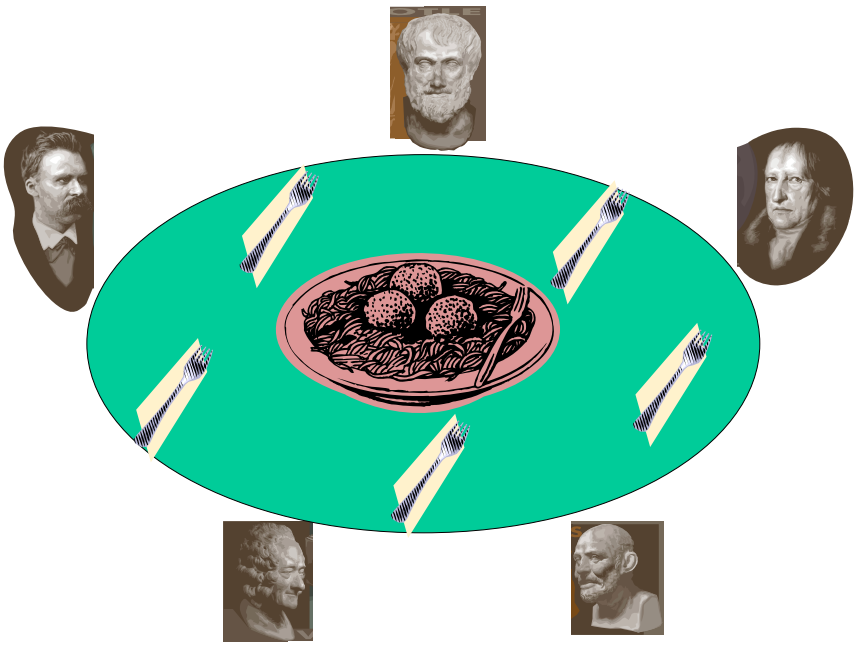


デッドロック

「食事する哲学者」の問題

- 並行処理の研究課題としてダイクストラにより提案された。その内容は以下の通り。
- 円卓を囲んで5人の哲学者が思策にふけりつつ、時に応じて食事を取る。食事は円卓中央の皿に山盛りにされたスパゲッティである。しかし、このスパゲッティはあまりにもからまりあっているので、食べるときには両手にフォークを持って食べなければならない。 フォークは、哲学者と哲学者の間に1本ずつ、都合5本が置いてある。つまり、右隣の哲学者が食事をしている最中は、右側のフォークは使用中であり、自分では使えないことになる。左についても同じ。
  ここで問題は、5人の哲学者が一人も飢えることがないように、食事をさせるプログラムを書くことである。

この問題は並行処理のモデル化であり、フォークが共有資源にあたる。デッドロックを表現する有名な問題で、良く引き合いにだされる。



デッドロック

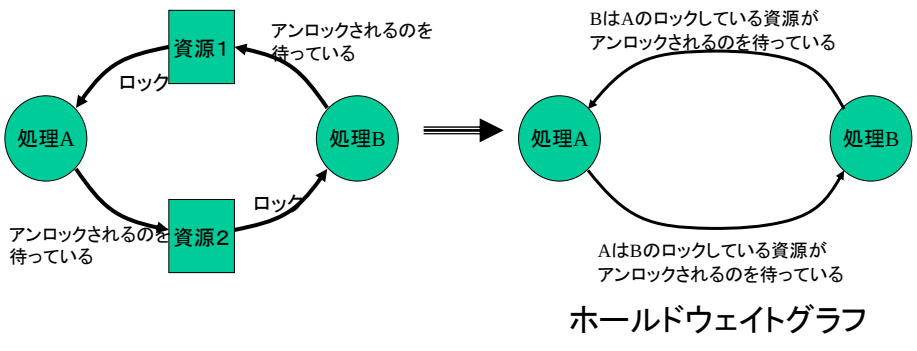
- デッドロックを解決する決まりきった手法はなく、様々な手法が状況に応じて提案されている。一つだけ紹介する。

時刻印（タイムスタンプ）法

- 時刻印：各処理に付加される番号で計算機番号と処理の発生時刻を組み合わせた数（システム内で一意の数となることが本質）
- 時刻印Taをもつ処理Aが、時刻印Tbを持つ処理Bがロックしている資源をロックしようとした場合下記の処理を行う。
  1. Ta＞Tbのとき、AはBがアンロックするまで待つ
  2. Ta＜Tbのとき、Aは自分がロックした資源をすべて解放し、処理を中止する。Aは時刻印を付け直して、処理を最初からやり直す。
- デッドロックがおこらないのは容易に証明できる。（宿題）
  ヒント：ホールドウェイトグラフ（後述）に有向枝の閉路ができないことを利用

デッドロック

- ホールドウェイトグラフ：処理Bが処理Aのロックしている資源がアンロックされるのを待っている時にBからAへ向かう有向枝を定義
- デッドロック発生の必要十分条件は、ホールドウェイトグラフに有向枝の閉路が存在することである。

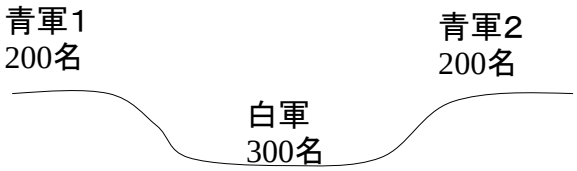


ホールドウェイトグラフ

状態合わせ

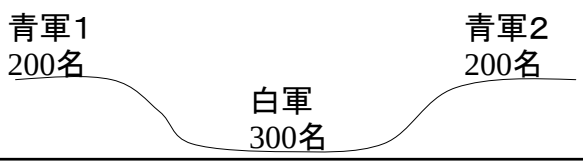
更に話は脱線するが、並行に動作するものの間の通信にエラーが起こる場合は、更にいやらしい問題が起こる。2つのプロセスの間で、お互いの情報を確実に共有することさえできない。これを抽象化した問題として、2つの軍隊問題がある。

- 2つの軍隊問題(two-army problem)
  - 図のように谷に白軍が野営していて、両側の山腹に青軍がいる。
  - 両方の青軍が同時に白軍を攻撃すれば勝てるが、一方の青軍だけで攻撃すれば負ける。青軍は同時に攻撃を仕掛けたい。
  - 通信手段は伝令を谷にやるしかない。伝令は白軍に捕まってしまうかもしれない(青軍はエラーのある通信手段しかない)。
  - 青軍は同時に攻撃をしかけることができるか？



状態合わせ

- 2つの軍隊問題(two-army problem)
  - 青軍1の司令官が「明日の10時に攻撃をしかけたい」という伝令を青軍2に送る。→この状態では青軍1は攻撃を仕掛けることができない。何故なら伝令が青軍2に届いている保証がないから。
  - 伝令が到着し青軍2の司令官が「明日の10時に攻撃をしかける」ことに合意し合意したことを知らせる伝令を青軍1に送る。→この状態では青軍2は攻撃を仕掛けることができない。何故なら伝令が青軍1に届いている保証がないから。
  - 以下、いくら伝令を送りあっても、伝令を最後に送った司令官はその伝令が相手に届いた保証がないので攻撃をしかけることができない。
- 見切り発車しか方法はない。計算機の場合は、軍隊全滅のような取り返しのつかない惨事がおこるわけではないので、起こる惨事の復旧コストとどこまで手間をかけて惨事を減らすかのトレードオフを考えて方式を決定する。



４．２　主記憶の仮想化

- ハードディスクをメインメモリの代用として利用するOSの機能。実際のメモリ容量以上のメモリ領域を確保する技術。「仮想記憶」とも言う。
- ハードディスク上に「スワップファイル」と呼ばれる専用の領域を用意して、メモリ容量が不足してきたら使われていないメモリ領域の内容を一時的にハードディスクに退避させ、必要に応じてメモリに書き戻すことで実現される。
- スワップファイルの内容を入れ替える動作を「スワップ」という。メモリ容量が少ないとスワップ動作が頻繁に発生し、性能の低下につながる。

４．２　主記憶の仮想化

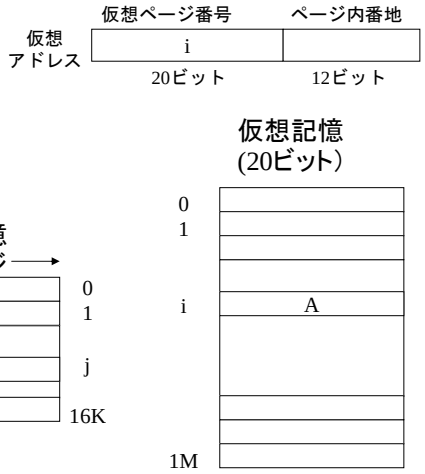
- 主記憶の仮想化には2つの方式がある。
  - ページ方式とセグメント方式
- キャッシュメモリと同様であるが、オペレーティングシステム（ソフト）で動作させるので、比較的複雑な処理が可能である。

	キャッシュメモリ	仮想記憶	
		ページ方式	セグメント方式
実記憶場所	ｷｬｯｼｭﾒﾓﾘ	主記憶	
仮想記憶場所	主記憶	補助記憶（ディスクなど）	
入替え単位名	ブロック	ページ	セグメント
入替え単位量	固定	固定	可変

４．２．１　ページ方式

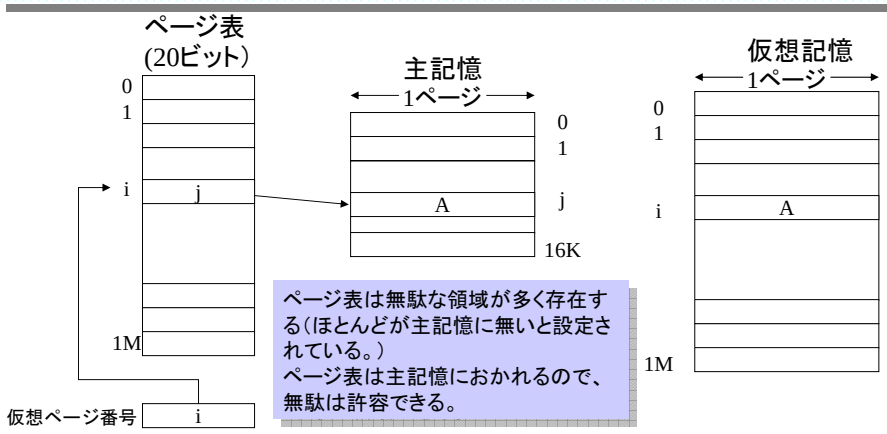
- ディレクトリ方式（キャシュメモリと同様に仮想記憶を用いる方式）をとると20bit×16Kエントリの仮想メモリが必要となり非現実的

例	容量	アドレス指定
ﾍﾟｰｼﾞｻｲｽﾞ	4 Kbyte	12 bit
仮想記憶	4 Gbyte (1Mpage)	32 bit 20 bit
主記憶	64 Mbyte (16Kpage)	26 bit 14 bit



４．２．１　ページ方式
（１）ページ表とアドレス変換

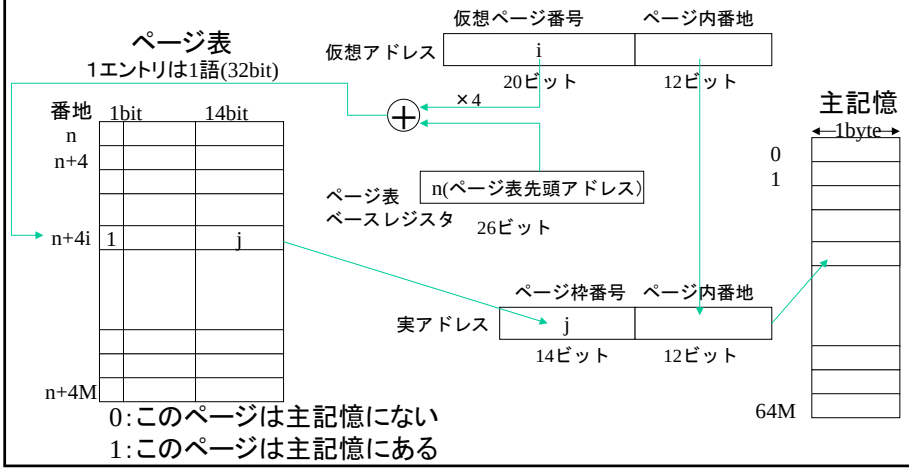
- ページ表方式（仮想記憶ページ数分の表を用いる方式）をとるのが通常
- ページ表の管理、主記憶と仮想記憶の間のスワップはオペレーティングシステムの仕事
- ストアスルー方式は現実的でない。



ページ表は無駄な領域が多く存在する（ほとんどが主記憶に無いと設定されている。）
ページ表は主記憶におかれるので、無駄は許容できる。

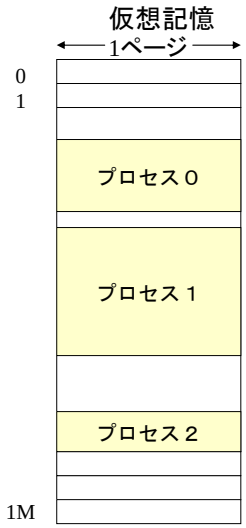
4．2．1　ページ方式
（1）ページ表とアドレス変換

- ページ表は主記憶に配置され、1エントリーに1語(32ビット)を使う（無駄なビットは発生する）
- 仮想記憶アドレスから主記憶アドレスの変換は下図



4．2．1　ページ方式
（2）単一仮想記憶と多重仮想記憶

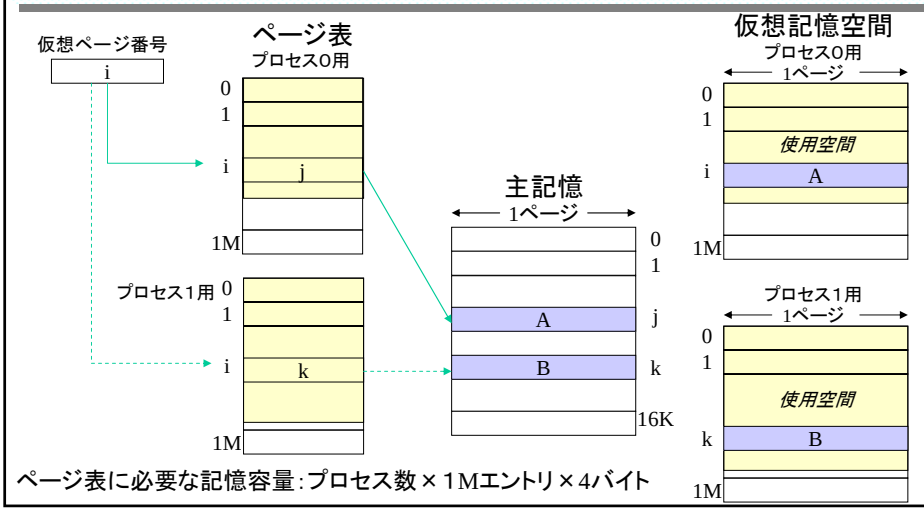
- 単一仮想記憶：1台の計算機が一つの仮想記憶空間をもつ。
  - プロセス間で仮想アドレスが重複しないように割り当てる制御が必要。（仮想メモリを使わない場合にくらべれば割り当ては格段に容易にはなっている。）
  - プログラムの再配置が必要（プログラムの開始アドレスが変わるために機械語のアドレスを変更する必要がある）。



(2)単一仮想記憶と多重仮想記憶

多重仮想記憶:プロセス単位に仮想記憶空間をもつ。

– ページ表に必要な記憶領域が大きくなる。しかし、実際に使われている部分は少ない

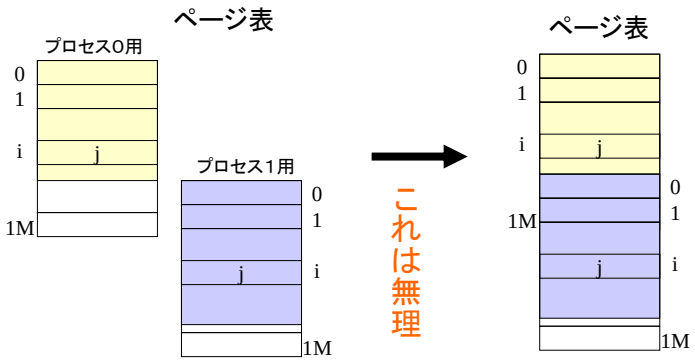


ページ表に必要な記憶容量:プロセス数×1Mエントリ×4バイト

(2)単一仮想記憶と多重仮想記憶

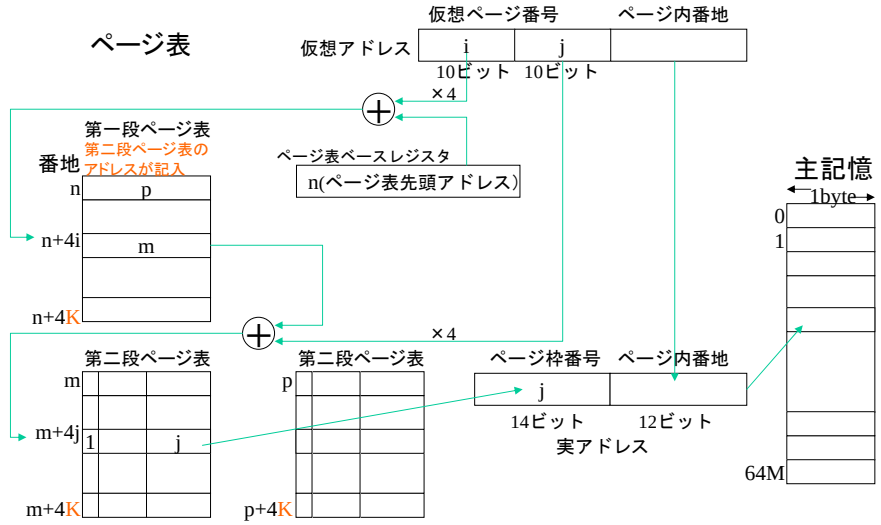
- 実際に使われている部分は少ない→ページ表の後半が使われないのでつめるという考えはあるが、これは無理

理由:プロセスが使うメモリ量は、プロセス生成時には分からず、実行中に大きくなる。



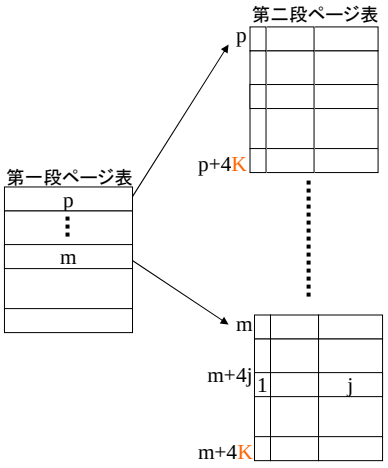
4．2．1　ページ方式
（1）ページ表の多段構成

- 無駄なページ表を削減する方式



4．2．1　ページ方式
（1）ページ表の多段構成

- 2段構成で必要となるページ表の記憶容量

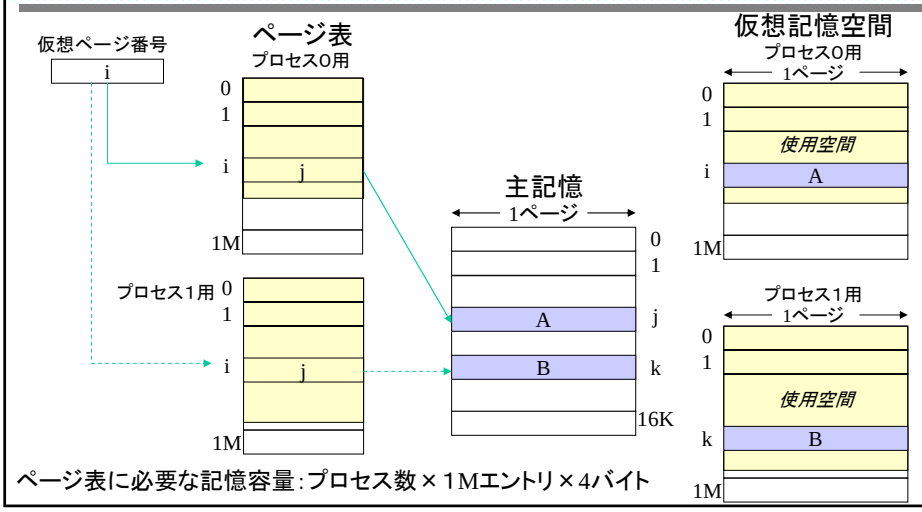


プロセスが最終的に使用するメモリ量：
n Mバイト

•第一段ページ表は必ず必要（4Kバイト）
•第二段ページ表は1個で1Kエントリであり4Kバイト×1K個で4Mバイトを表現可能であり、***n*** / 4個（***n*** Mバイト/ 4Mバイト）で表現可能
•第二段ページは、プロセスの実行中に不足すれば追加が可能である

(2) 多重仮想記憶の付随的効果

- 他のプロセスの領域を読み書きすることができない。
  - 理由：ページ表にないものにはアクセスできない。
- プログラムの保護につながる。



ページ表に必要な記憶容量：プロセス数×1Mエントリ×4バイト

4．2．2　セグメンテーションと2次元アドレス

- プログラムを**ユーザ**が分割（分割した各々をセグメントと呼ぶ）。
- 仮想アドレスをセグメント番号とセグメント内アドレスで表現。セグメント単位にスワップを行う。（2次元アドレス）
- ページは固定長であるが、セグメントはユーザが分割した意味のある単位（例えば、一つのサブルーチン）

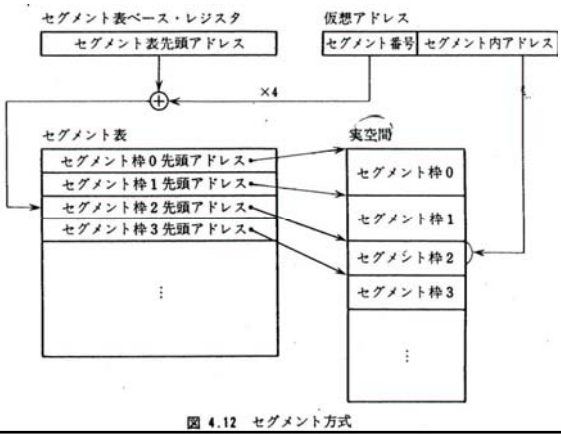


図 4.12　セグメント方式

４．２．２　セグメンテーションと2次元アドレス

- プロセス間でデータを共有するのに便利
- アクセスの保護単位とできる。

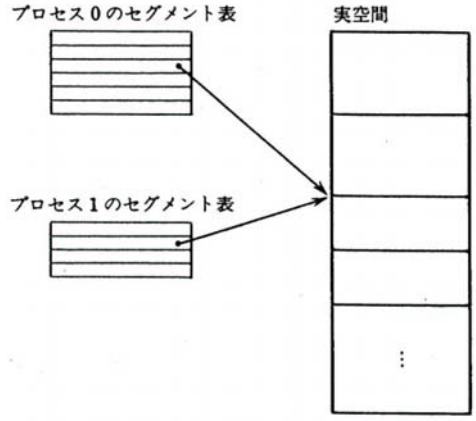


図 4.13　プロセス間でのセグメントの共有

アクセスの保護
•セグメント表のエントリに読み出し・書き込み・実行の可否のビットを表すビットを設ける
•例えばプロセス0は読み書きできるがプロセス1は書込みしかできないなどの制御が行える（他のプロセスのバグで影響をうけることが防げて信頼性の高いソフトを構成できる）

（2）ページ化セグメント方式

- 各セグメントを複数のページに分割
- 2段仮想記憶と同様の制御を行う

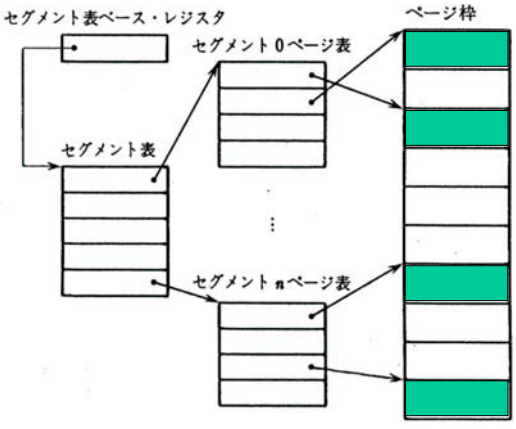


図 4.14　ページ化セグメント方式

Fragmentation（セグメントを主記憶にスワップインしようとした時、主記憶の空き容量はあるが、連続した領域が存在しない）問題を解決する。

•記憶保護のチェック
•指定されたアドレスがセグメント協会を超えないかのチェックを行う

４．２．３　アドレス変換の高速化

- ページ表やセグメント表は主記憶におかれる
  - 1回の主記憶参照に2回以上の主記憶参照が必要！（1回は実際のデータの読み書き、残りは表検索）
- 解決策：テーブルそのものをキャッシングする（過去に主記憶でテーブル検索した結果を、高速メモリに覚えておく）。

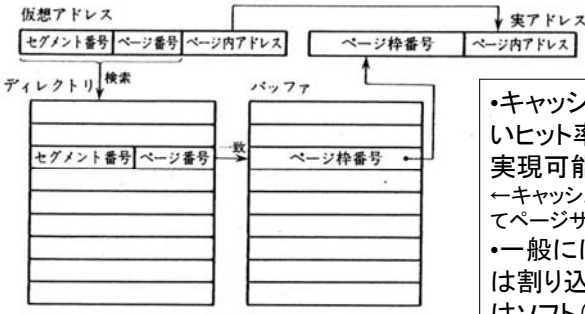


図 4.15　アドレス変換バッファ（TLB）の構成

ディレクトリ：高速連想メモリ
バッファ：高速メモリ

•キャッシュより少ないエントリー数で高いヒット率が得られる。（９９%以上を実現可能）
←キャッシュのブロックサイズ（64バイト）に比べてページサイズは大きい（4Kバイト）
•一般には、TLBにデータがない場合は割り込みがかかり、その後の処理はソフト（オペレーティングシステム）でアドレス計算を行う。