プログラムとメモリ空間

- メモリの基本
- メモリはアドレス（番地）で記録場所を指定
- 0〜Nまでの番地なら、N+1個のデータを分けて管理できる→通常は固定サイズの数値（３２ビット00000000〜FFFFFFFFとか）
- 通常、メインメモリはバイト単位
  →HDDなどの補助記憶はブロック単位
- プログラムの事情
- プログラム内にはメモリのアドレスを直接指定されているものがある
- シングルプログラミングの場合は問題無い
  →プログラムのメモリへの読み込み位置は常に同じ
- マルチプログラミングでは大きな問題
  →メモリ内の配置換えをしないと動かない
  →他のプロセスの使用しているメモリを書き換えてしまう
- 実際の配置位置とは別にプログラムが想定しているメモリ配置を仮想的に用意できれば
  →論理アドレス（仮想アドレス）
- アドレス空間
- 一連の数値によって連続的に管理されるアドレスの集合→アドレス空間
- ハードウェアに依存したアドレス
  →物理アドレス空間
- 仮想的に用意するアドレス
  →論理アドレス空間

論理アドレスの利点と実現

- 論理アドレスの利点
- 利点１：プログラムの物理メモリへの配置位置の自由
- 利点２：他のプロセスのメモリへのアクセスの防止

- システム全体で論理アドレスを使用するという取り組みが必要
- プログラム内→論理アドレス指定
- OS、ハードウェア→実行時には物理アドレスに変換

- アドレスの変換
- 論理アドレスを物理アドレスに変換する仕組み→アドレス変換機構
- アドレス変換は頻繁に行われるためハードウェアでサポート

  →メモリ管理ユニット（MMU）

- リロケータブルコード
- プログラムの構築の際、アドレス変換がきちんと働くよう処置される

  →再配置可能コード

  （リロケータブルコード）
- 論理アドレスによって、メモリのどこに置かれても実行可能

メモリの仮想化とスワッピング

- メモリを効率よく使う手法
  →実状以上のメモリ量を使用したい
  →仮想化
- マルチタスキングの場合，実行中のプロセス以外は物理メモリ上になくて良い
  →実行途中の実行可能なメモリイメージがあれば良い
- 実行可能状態→メモリイメージをハードディスクに移動
- 空き領域ができる＝新しいプロセスが作れる
- 移動したプロセスを実行したい
  →他の実行可能プロセスをハードディスクに退避、空きスペースに入れ替える

- スワッピング
- メモリの入れ替え機構の一つ
  →スワッピング
- メモリイメージの退避領域として、ハードディスクなどに特別な領域（スワップ領域）や特別なファイル（スワップファイル）を用意
- メモリ→退避領域（スワップアウト）
- 退避領域→メモリ（スワップイン）
- プロセスの実行状況を維持しつつ、より多くのプロセスを実行できる

メモリの断片化とその対策

- メモリ確保の注意点
- プロセスによって必要なメモリサイズ＝確保されるメモリサイズは異なる
- 領域は基本的に連続して確保
- プロセス生成、スワップイン
  →メモリ確保
- プロセス終了、スワップアウト
  →メモリ解放

- メモリの空き領域は使うほどにバラバラに
  →メモリの断片化
  （メモリフラグメンテーション）
- プログラムがリロケータブルなら、再配置で空き領域を一つにすれば良い→コンパクション
- 実行中のプロセスは動かせない
- 全てのプロセスの再配置
  →手間が大きくて困難
- 空きメモリを上手に管理する方法が必要

断片化対策と空きメモリ管理

- 空きメモリ管理
- 確保の際どの位置に割り当てるか
- 空き領域がどこにどのくらいあるか
- 固定区画方式：メモリを固定長単位で割り当てる→区画数に実行プロセス数が依存
- 可変区画方式：確保サイズをリストで管理→断片化の恐れ
- 一般的には可変区画方式で割り当て
- メモリの割当方式
- OSはプロセスそれぞれにメモリを割り当てる
- 基本的な方法として３種
- ファーストフィット：空き領域リストの並びに依存
- ベストフィット：小さい領域がより小さく断片化しやすい
- ワーストフィット：どの空き領域も使えない状況が発生する
- 空きリスト管理
- ビットマップによる管理：管理情報をすべてメモリ上に
- 連結リストによる管理：整列状態を保つのは便利

ページング

- 実行できるプログラムの大きさは物理メモリに依存する
- 物理メモリが少ない→小さなプログラム、多重度小
- プログラムの置き場所に依存しない→論理アドレス空間
- 物理メモリの量に依存しない→仮想記憶
- 物理メモリを隠す
- プログラムに物理メモリの配置位置を意識させない→論理アドレスを利用
- プロセスごとに異なる論理アドレスを割り当て→多重仮想記憶
- 基本的に論理アドレス空間の全領域＞＞物理メモリの大きさ
- 物理メモリを論理空間の使う部分だけに割り当てる仕組み→ページング
- ページング

- 論理アドレスをページ（固定長）に分割
- 物理メモリをページフレーム（固定長）に分割
- 使用するページに対して空いているページフレームをマッピング
- マッピング情報はページテーブルというデータで管理→MMUが読んでアドレスを変換
- バラバラの位置にある物理メモリを仮想アドレス空間上で連続領域として扱える
- ページングの機能と補助記憶と組み合わせ
  →物理メモリ以上の領域を物理メモリとして利用
- 実行中だけど処理される可能性の低いページを補助記憶の蓄積領域にコピー（退避）→ページアウト
- 使用するときは補助記憶から読み出し，ページフレームを割り当ててイメージを配置→ページイン
- 補助記憶が主記憶の一部として機能

ページテーブルとページングの効率化

- ページテーブルは論理ページ（ページ）の物理ページ（ページフレーム）の関係を表すデータ
- ページテーブル
- 各ページの状況と参照先を得ることができる
- Vフラグ：ページインの判断
- Pフラグ：アクセス制御
- Cフラグ：ページアウトの判断
- これにページフレームの位置（物理アドレス）が加わる
- アドレスの管理方法
- 簡単な実装例としては、アドレスを（ページ番号）＋（ページ内のオフセット）の形で区切る
- ページ番号に割当てたビット数
  ＝ページ数の最大値
- オフセット指定に割当てたビット数
  ＝１ページの大きさ
- ページテーブルの問題
- ページテーブルはメインメモリ上に保持
  - アドレス変換時に適宜参照→頻繁なメモリアクセス
  - アドレス変換時のメモリアクセスに対する速度の問題
- 仮想アドレス空間はプロセス毎に作られる
  - ページテーブルのエントリ数→（仮想アドレス空間あたりのページ数）×（プロセス数）
  - メモリ量増加、処理能力増加→テーブルの肥大化
- ページ変換の効率化
- TLBは高速化、逆ページテーブルは領域削減が期待できる