システムプログラム概論

メモリ管理 (1)

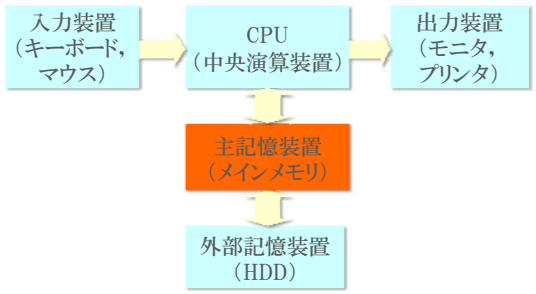
第x講: 平成20年10月15日 (水) 2限 S1教室

中村 嘉隆(なかむら よしたか)
奈良先端科学技術大学院大学 助教
y-nakamr@is.naist.jp
http://narayama.naist.jp/~y-nakamr/

今日の講義概要

- メモリ管理の必要性
- 静的メモリ管理と動的メモリ管理
- スワッピング, 仮想記憶
- ページングとセグメンテーション

メモリはコンピュータの 5 大構成要素

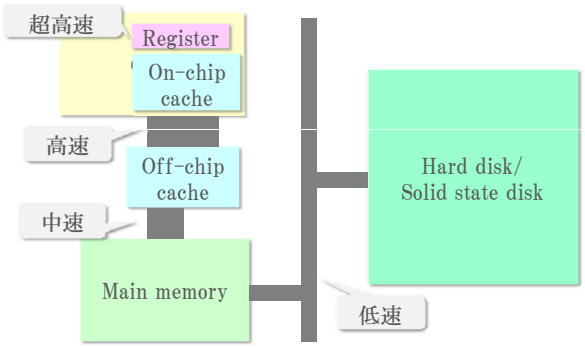


なぜメモリ管理が必要か

- メモリ管理が必要な理由
  - プログラムを実行するためにはメモリが必要
  - メモリ技術の進歩の速さ<プログラムの巨大化の速さ
    - パーキンソンの法則:「プログラム量は与えられたメモリのスペースを満たすまで膨張する」
    - 1980 年代: 4MB VAX で 10 人以上が同時ログイン
    - 現在: 512MB PC でシングルユーザ(Vista)
  - 理想: 高速メモリが無制限に使用可能
  - 現実: 少量・高速のキャッシュメモリ, 中容量・中速のメインメモリ(RAM), 低速・大容量のディスクストレージ

→ OS によるメモリ管理が必要

メモリ階層



メモリ階層

- メモリ技術
  - アクセス速度, 容量, 価格のトレードオフ

	Latency	Size	Price
Register	0.13 ns	512 bytes	On chip
On-chip cache	4.7 ns	2 MB	On chip
Main memory	20 ns	1 GB	$0.1 /MB
HDD	13 ms	500 GB	$0.3 /GB

メモリ管理機構（メモリマネージャ）

- メモリ階層を管理する OS 内の機構
  - メモリの使用部分と未使用部分を管理
    - プロセスへのメモリの割当て，解放のため
  - メインメモリとディスク間の**スワッピング**を実行
    - メインメモリだけでは必要容量が足りないとき，メインメモリ内のあまり使用されていない部分をディスクに退避したり必要な部分をディスクからメモリに復元

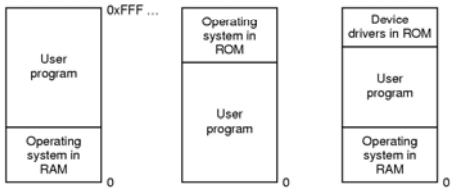
メモリ管理機構の種類

- モノプログラミング
- マルチプログラミング（固定区画）
- スワッピング
- 仮想記憶

（モノプログラミング，マルチプログラミング（固定区画）：静的な割当／スワッピング，仮想記憶：動的な割当）

モノプログラミング

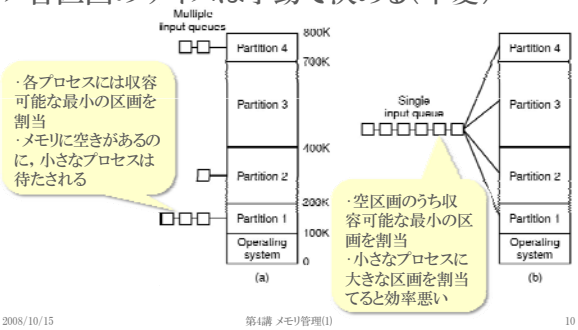
- 一度に 1 つのプログラムを実行する最も原始的なメモリ管理
  - アドレス固定，境界チェックなし
  - メモリ上に他のプログラムがない → Protection はない



(a) メインフレーム　(b) PDA，組込みシステム　(c) 初期の PC（MS-DOS）

マルチプログラミング（固定区画）

- 各区画のサイズは手動で決める（不変）

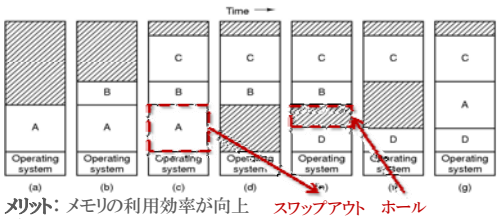


リロケーションとプロテクション

- **リロケーション**
  - プログラムはどの区画で実行されるか分からない
  - → その区画で実行するかで，プログラム内のジャンプ先アドレスの書き換えが必要
- **プロテクション**
  - 区画 1 が割り当てられているプロセスが，区画 2（別の実行中プロセスに割当済）を読み書きできてしまう
  - → マルチユーザシステムでは特に問題
- 解決策
  - **ベースレジスタ**：区画の先頭アドレスを入れると，プログラム内のジャンプ先，メモリアクセス先に自動的に足してくれる
  - **リミットレジスタ**：区画の長さを指定して，それを超えるアドレスへのアクセスはハードウェア的にできなくしてくれる
    - 初のスパコン CDC6600 で採用
    - 初代 IBM PC の Intel 8088 はベースレジスタのみ

スワッピング

- 各プロセスについて，メモリにロードし，一定時間実行し，ディスクに退避（**スワップアウト**）する，を繰り返す方法
- メモリサイズの制限を受けないが，ディスク I/O のオーバヘッドが問題



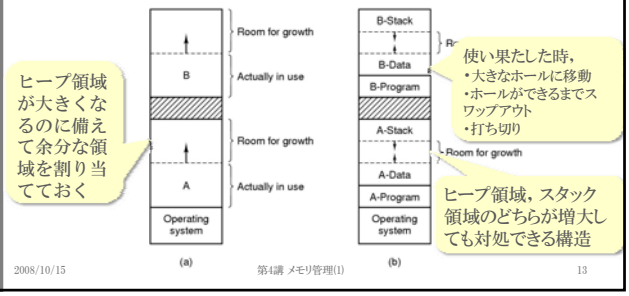
スワップアウト　ホール

メリット：メモリの利用効率が向上

デメリット：メモリの管理が複雑に
- スワップアウトで領域に**ホール**ができる → どう解消？
- ある区画のデータ（ヒープ）領域が増大する場合 → どう増やす？

スワッピングにおけるメモリ管理

- スワップアウトでホール(空き)ができてしまう
  → **メモリコンパクション**(Disk のデフラグに類似) → 処理重い
- ある区画のデータ(ヒープ)領域が大きくなる時(プロセスが動的にメモリ割当てした時など)

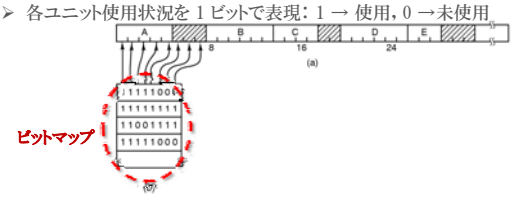


メモリの動的割当

- 動的なメモリ区画の割当
  → OS によるメモリの使用/空き状況の追跡・管理
- ビットマップを用いたメモリ管理
- 連結リストを用いたメモリ管理

ビットマップを用いたメモリ管理

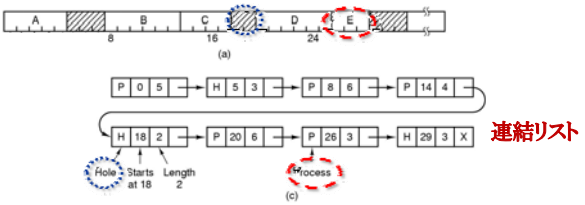
- メモリ全体 → 割当ユニット(数バイト~数 K バイト)に分割
- 各ユニット使用状況を 1 ビットで表現: 1 → 使用, 0 →未使用



- 割当ユニットのサイズが小さい → ビットマップ大きくなる
- 割当ユニットのサイズが大きい → プロセスへのメモリ割当サイズがユニットの倍数でないと, より大きな無駄が生じる
- 問題点: k 個の割当ユニット必要 → ビットマップ内で k 個の連続した 0 を検索 → 時間がかかる

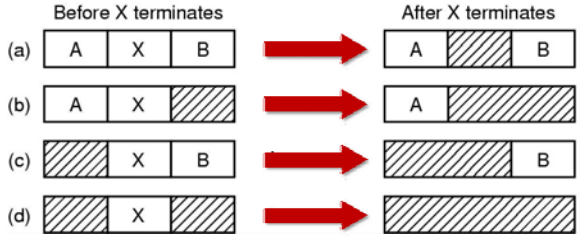
連結リストを用いたメモリ管理

- メモリ区画(プロセスまたはホール)を連結リストで管理



リストの更新

- 以下の場合が存在



(a) の場合, P → H に書き換えるだけ
(b)-(d) の場合, エントリが一つ削除される

連結リストでのメモリ割当アルゴリズム

- First fit
  - リストを先頭から順に辿り, 最初に見つかった十分な空き容量のホールを割り当てる
- Next fit
  - 前回ホールを見つけた場所(リストの途中)を覚えておき, 次回の割り当ては, 途中から探す
  - First fit より若干性能が悪い
- Best fit
  - リスト全体を検索して十分かつ最小のホールを割り当てる
  - ホールのサイズが小さい順にソートしておくと, 効率が良くなる(リストを最後まで見なくて良い)

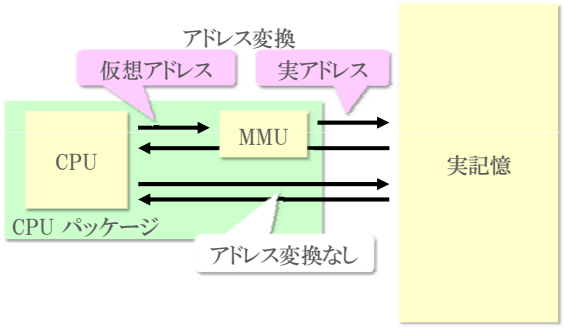
仮想記憶(Virtual Memory)

- 登場の背景
  - 利用可能メモリ容量より大きいプログラムの実行
  - **オーバレイ**: プログラムを手動で断片に分割し，順にメモリにロードして実行
    → 自動化(1961年)したものが**仮想記憶**
- 仮想記憶の基本アイデア
  - プログラム，データ，スタックの合計サイズが物理メモリ(実記憶)容量を超えても良い
  - プログラムの実行に必要な一部だけをメモリに置き，残りはディスクにスワップアウトする

仮想記憶と実記憶のマッピング

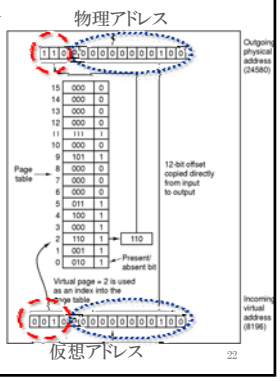
- 仮想記憶(virtual memory)
  - 仮想アドレス，仮想アドレス空間
  - 実記憶(物理メモリ)より広大，各部分は物理メモリ，ディスクなどから構成
- 実記憶(physical memory)
  - 実アドレス，実アドレス空間
  - 物理メモリのみから構成
- 仮想アドレス空間でプログラム実行
  → 仮想メモリへの読み書きが発生 → アドレス変換が必要
- Address translation
  - 仮想アドレスから実アドレスへのマッピング
  - Memory Management Unit (MMU) が実行

アドレス変換



ページテーブル

- 仮想アドレス空間をページに分割(ページサイズは固定)
- ページを単位としたアドレス変換
  - (仮想ページ番号，オフセット)
  → (物理ページ番号，オフセット)
- ページテーブル(変換表)を実メモリ上に保持



ページングの問題点

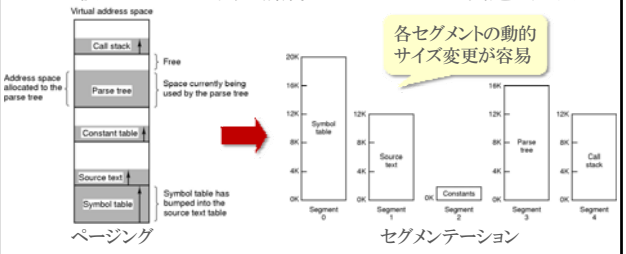
- ページテーブルサイズ
  - 主メモリを消費．プロセス毎に必要
  - ページサイズが小さいとテーブルサイズ増大
    - アドレス空間 2^{64}bytes，4KB ページ → 2^{52} エントリ必要
- 要求ページの検索
  - ページテーブル大きい場合に，目的ページの検索に時間がかかる
  → TLB (translation look-aside buffer)

TLB (translation look-aside buffer)

- 変換早見表
  - よく使うページテーブルをキャッシュ
  - 主メモリへのアクセスなしにアドレス変換
- 機構
  - ハッシュ関数を用いて実現
    - ハッシュ関数: 検索の平均オーダ $O(1)$
  - ミスすると MMU を使ってページテーブルから検索
- 小さなサイズの TLB でも 90% のアドレス変換を高速化可能
  - TLB の検索 10ns，ページテーブル の検索 500ns，ヒット率 90% とすると，
    平均 10ns × 0.9 + (500ns + 10ns) × 0.1 = 60ns

セグメンテーション

- 可変長の物理メモリ区画(セグメント)を確保，それらを組合せて仮想アドレス空間を構成 → ページングの問題を解決

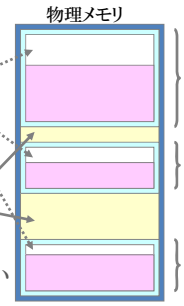


ページング　　セグメンテーション

メモリへのアクセス → セグメント番号 ＋ オフセットを指定

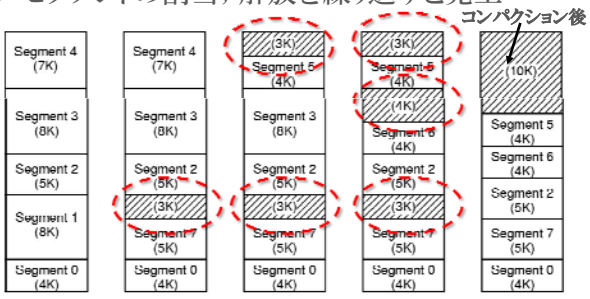
セグメンテーションの問題

- 物理メモリの連続領域を確保する必要がある
  - 断片化(fragmentation)を招く
- 断片化
  - 内部断片化
    - セグメント内部の空き領域
  - 外部断片化
    - 主メモリのセグメント間の空き領域
    - ページングでは外部断片化が起きない



外部断片化の例

- セグメントの割当，解放を繰り返すと発生



ページングとセグメンテーション

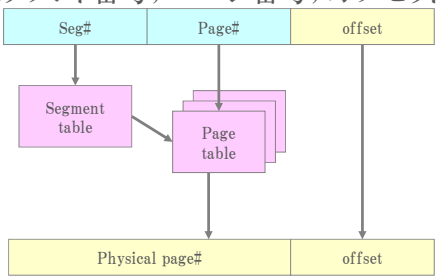
- ページング
  - ページサイズは固定(多くの OS では 4KB)
  - 割当てる領域が連続していなくてもよい
    - 外部断片化が起きない
  - ページ単位でスワッピングでき，メモリを有効利用可能
  - ページテーブルサイズ，メモリ動的割当時の問題
- セグメンテーション
  - セグメントサイズは可変
  - セグメント間でデータの保護が可能
    - セグメントとして，ひとまとまりになっているため
  - 外部断片化(external fragmentation)

ページ化セグメンテーション

- セグメントとページを併用
  - 今日のプロセッサアーキテクチャでの主流
- プロセスなどに割り当てるメモリ区画はセグメントで確保
  - メモリ領域間の保護が容易
- セグメント内の領域はページング
  - メモリの有効利用
  - TLB を用いて，検索効率化

ページ化セグメントのアドレス変換

- 仮想アドレス＝
  セグメント番号，ページ番号，オフセット



まとめ

- メモリ管理の必要性
  - メモリ管理機構の種類: 静的管理, 動的管理
- スワッピング
  - プロセスに割当てたメモリをディスクに退避
  - 動的なメモリ割当:ビットマップ, 連結リスト
- 仮想記憶
  - 物理メモリ容量より大きなプログラムの実行
  - アドレス変換
  - ページング
  - セグメンテーション
  - ページ化セグメンテーション