システムプログラム概論

メモリ管理 (2)

第5講: 平成20年10月20日 (月) 1限 S1教室

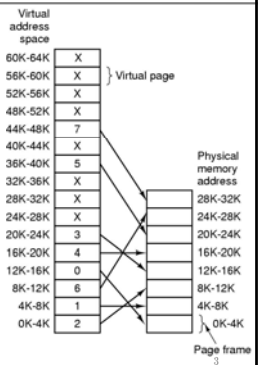
中村 嘉隆(なかむら よしたか)
奈良先端科学技術大学院大学 助教
y-nakamr@is.naist.jp
http://narayama.naist.jp/~y-nakamr/

今日の講義概要

- ページ管理方式
- ページ置換アルゴリズム

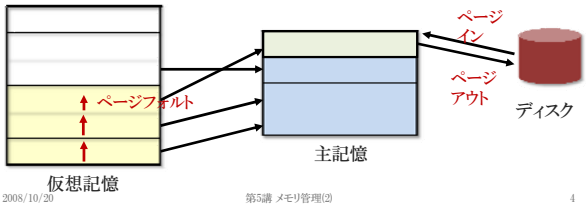
ページング(復習)

- 仮想アドレス空間, 主記憶(実アドレス空間)を固定サイズのページに分割
- 仮想アドレス空間のページを主記憶(メモリ)のページに対応させる
- **ページテーブル**(変換表)を実メモリ上に保持
- 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)
  - 変換は MMU がハードウェアで行う



ページ管理方式

- **ページフォルト**(Page Fault)
  - 参照したページが主記憶(メモリ)にない時に発生
  - プロセスを起動・実行 → ページフォルト発生!
- 主記憶にあるページを**ページアウト**(主記憶から除く)して, そこに参照したいページをディスクから読込む(**ページイン**)
  - ページアウトするページが修正されていたら, ディスクへの書込みが必要(修正されていない → 単に破棄で OK)



ページ置換(Page Replacement)

- 物理ページ数には限りがある
→ あまり使っていないページを退避させて, 主記憶に空きをつくる(ページアウト)
- どのページを選んで退避させるか?
さまざまなアルゴリズムがこれまでに考案
  - NRU, FIFO, second chance, LRU, …

ページ置換アルゴリズムの戦略

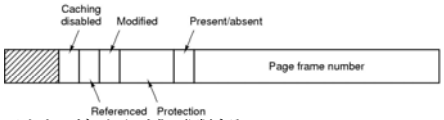
- ページフォルト時の性能向上
  - × ランダムに選択したページを取り除く
  - ○ あまり参照されないページを取り除く
- ページアウト時の処理
  - 変更が加えられたページはディスクへ保存
  - 未変更のページは破棄
- 頻繁に参照されるページを取り除くのは良くない
  - すぐにまた呼び戻される

最適ページ置換(Optimal Replacement)

- ページフォルト発生
  →今後，最も遠い未来で参照されるページを取り除く
- 実現方法
  - 各ページに何命令後に参照されるかを記録
  - 何命令後にページが参照されるか知ることは不可能
    →実現不可能
- 他のアルゴリズムの性能の比較対象として利用

NRU: Not Recently Used Algorithm

- 各ページは Reference bit と Modified bit を持つ
  - ページが参照/変更された時，対応するビットが1 にセットされる
  - 参照ビットは，クロック割込発生(20ms)毎に 0 にリセットされる

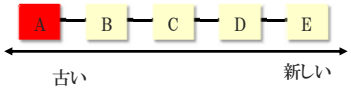


- NRU ではページを以下のように分類する
  1. 参照も変更もされていない
  2. 参照されていないが，変更はされた
  3. 参照されたが，変更されていない
  4. 参照も変更もなされた

  (参照ビットは**定期的に(0 に)リセット**されるので，2. があり得る)
- NRU では，ページフォルト時に上記の順でページを取り除く(複数ある時はランダムに選択)
  →そこそこ良い性能を達成

FIFO: First-In, First-Out Algorithm

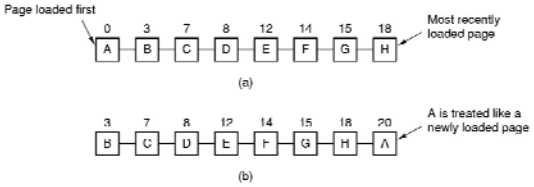
- すべてのページの連結リストを保持
  - 順番は，メモリに読み込まれた順
- 先頭(最古)のページから順に交換



- 欠点
  - よく使われるページを削除してしまうかも知れない
  - 古くからメモリに読み込まれているページはよく参照されることが多い

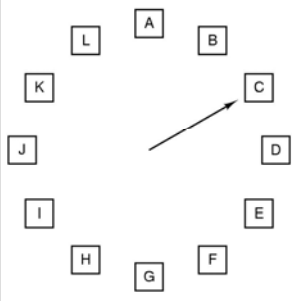
Second Chance Algorithm

- 再チャンスを与える(FIFO の改良)
  - ページは FIFO と同じ順でリンクされている
  - 時刻 20 でページフォルトが発生
    - ページ A の参照ビットが 1 なら，A のロード時間を 20 に，参照ビットを 0 に更新して，リストの最後尾に追加
    - B 以降のページを探す



問：A～Hが全て参照ビット 1 の時，どうなる？

The Clock Algorithm



- Second Chance の改良版
- ページフォルトが起こったときに指しているページの参照ビットを確認
  - 0 なら，そのページを取り除く
  - 1 なら，その参照ビットを 0 にして，ポインタを 1 つ時計回りに進めて，同じ処理を繰り返す

LRU: Least Recently Used

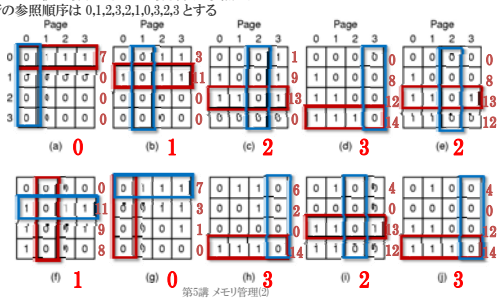
- 最適ページ置換アルゴリズムの近似手法
  - 最近使われたページは，また使われる可能性が高い
  - 長時間未参照のページは今後も使われない → 取り除く
- 実現は可能だがコストが高い
  - 連結リストでページを管理・維持することで実現可能
    - 最近使用したページが先頭，最も昔に使用したページが最後
    - ページ参照のたびに更新が必要 → コスト高い
  - ハードウェアによる LRU の実現
    - 高速だが，全てのマシン/OS で使えない
  - ソフトウェアによる LRU の模倣
    - 近似的な手法によりコストを抑える

LRU の実現法 (1) ハードウェアでの実現

- LRU は行列でページの参照を表現
- ページフレーム k を参照時，k 行すべての要素を 1 にセットし，その後，k 列すべての要素を 0 にする

→ 各行のバイナリ値の最小のものが，最も古くに参照されたページ

- ページの参照順序は 0,1,2,3,2,1,0,3,2,3 とする

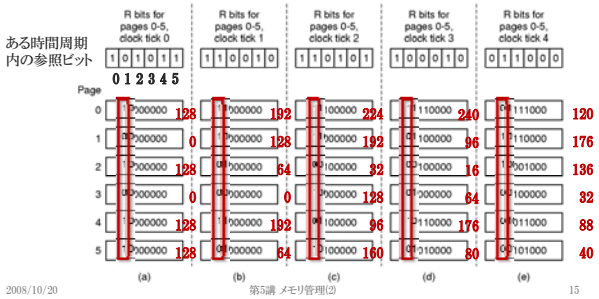


LRU の実現法 (2) ソフトウェアでの実現

- NFU (Not Frequently Used) アルゴリズム
  - 全ページにカウンタ(初期値 0)をそれぞれ用意
  - 各割り込みクロックで OS はメモリ内の全ページをスキャン
  - 参照 (R) ビットの値をカウンタに加算
  - ページフォルト時，カウンタ値が最小のページを削除
- NFU の問題点
  - ページが参照された回数のみ考慮
    → 過去に頻繁にページが使用され，最近はあまり使用されていないページは削除されない
  - 例：マルチパスコンパイラ
    - 同じプログラムを複数回走査
    - 1回目の走査が最も長時間であったとすると，1回目の走査で参照頻度の高かったページはずっと高いカウント値を保持
      → 実はその後不要だとしても全く削除されない

LRU の実現法 (2) ソフトウェアでの実現

- Aging アルゴリズム(NFU の改良)
  - クロック割込毎に，カウンタ値を右にシフト
  - 参照があった時 → 右にシフト後，最上位ビットを1に
- 参照ビットが以下のように変化 → LRU を上手く近似

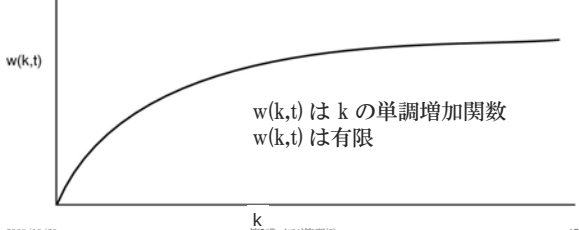


The Working Set Algorithm (1)

- ワーキングセット
  - プロセスが現在使用しているページの集合 → 時間の経過とともに変化する
  - メモリ内にすべてある場合
    - ページフォルトが起こらない
  - メモリ容量が少ない → ワーキングセットが一度にメモリにロードできない
    - ページフォルトが頻繁に起こる
  - ほとんどのプログラム → 参照の局所性を持つ
    - 実行の各フェーズで，ワーキングセットはごく一部のページ群
- スラッシング(thrashing)
  - 数命令毎にページフォルトを起こすプログラム

The Working Set Algorithm (2)

- ワーキングセットモデル
  - プロセスのワーキングセットを追跡し，プロセスの実行前にワーキングセットをメモリ内にロード → ページフォルト削減
- w(k,t): 時刻 t における最近 k 回の命令で参照されたページの集合(ワーキングセット)の大きさ

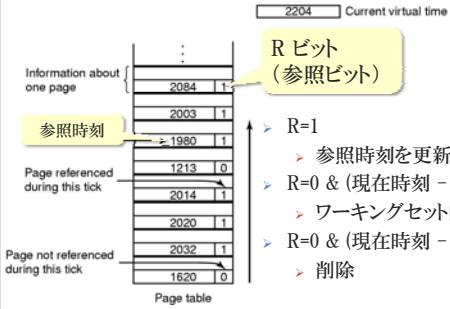


The Working Set Algorithm (3)

- マルチプログラミングシステム
  - プロセスは他プロセスが CPU が使用するときにディスクに退避
  - ワーキングセットの解析を行うことで，プロセスが再始動する時に必要とするページに関する推測が可能
- プレページング
  - プロセスが再始動する前に必要となるページをロード
- ワーキングセットモデルによるページ置換
  - ワーキングセット内のページか否かの判断が必要
  - ワーキングセット外のページを選択し削除

The Working Set Algorithm (4)

- 近似アルゴリズム

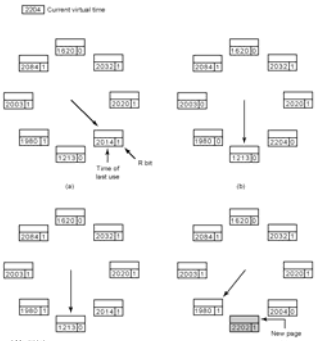


閾値

- R=1
  - 参照時刻を更新．次へ．
- R=0 & (現在時刻 − 参照時刻) < τ
  - ワーキングセット内．次へ．
- R=0 & (現在時刻 − 参照時刻) ≧ τ
  - 削除

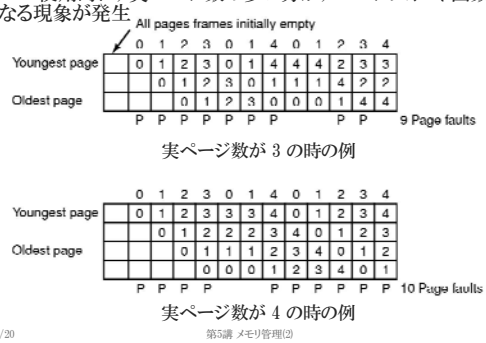
The WSClock Algorithm

- Working Set アルゴリズムと Clock アルゴリズムの併用



Belady's Anomaly

- FIFO 使用時に，実ページ数の多い方が，ページフォルト回数が多くなる現象が発生

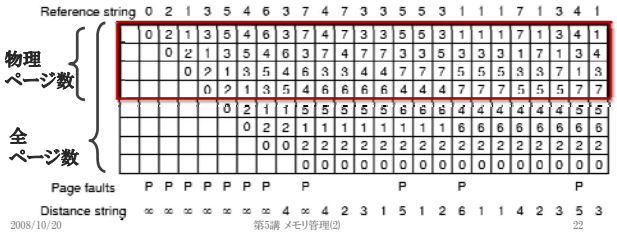


実ページ数が 3 の時の例

実ページ数が 4 の時の例

Stack Algorithms

- Belady's Anomaly を回避するアルゴリズム
  - 物理ページ（上段）＋仮想ページ（下段）で，ページの順番を保持
  - 物理ページ数 k が k+1 になったとき，ワーキングセット w(k,t) ⊆ w(k+1,t) となる
  - 使用するページ置換アルゴリズムは LRU，FIFO 等どれでも良い

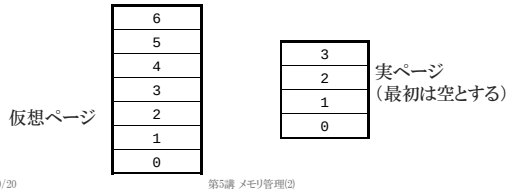


ページ置換アルゴリズムの一覧

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

第 2 回ミニレポート
（期限：10/27テスト開始時まで，形式: A4）

- 実ページ数が 4 の時，仮想ページ 0-6 を以下の順で参照する（変更はしない）とする．
  - 0, 1, 4, 5, 6, 1, 2, 1, 3, 6, 0, 1, 0, 6, 2, 0, 5, 1
- FIFO, Second Chance, LRU（Aging，カウンタは 4bit）でページ置換を行う場合に，実ページにロードされている仮想ページの移り変わりがどうなるか，P.21 の記法で表せ．また，それぞれ何回のページフォルトが発生するか答えよ．



まとめ

- ページ管理
  - ページフォルトが発生?ページの置換が必要
- ページ置換アルゴリズム
  - NRU, FIFO, Second Chance, Clock, LRU, WS, WSClock
  - Belady's Anomaly, Stack Algorithms