Betriebssystem 5.2

Altenative Lösungen

- Chapter Struktur
  - TLB (一种高速缓存，专门用来加速计算机从虚拟到物理的地址转换过程)
  - mehrstufige Seitentabellen
  - Swapping

1. TLB

   - Motivation:
     - Seitentabelleneintrag aus dem Speicher lesen ist teuer
     - wenn ein Program grieft wiederwolt die selbe Seiten zu
   - TLB Hit: if the adress information is stored in TLB
   - TLB Miss: not sotred. Then the program must look it up in page table, load the adress to TLB.
     - Anzahl Zugriff auf unterschiedliche Seiten
   - TLB Fehlerrate:
     - TLB missrate = TLB-miss / TLB Hit
     - Treffrate = 1 - Fehlerrate
   - TLB Performance
     - Bei gegebene Anzahl, und vergrößern die Seiten:
       - improve the TLB Trefftrate
       - weniger neue Übersetzung für die gleiche Menge an Speicher notwendig
       - Reichweite: TLB Einträge ∗ Seitengröße
       - Sequentielle Arrayzugriffe fast immer Hit(daher schnell)
       - Zufallig, ohne wiederholete Zugriffe sind langsam
   - Workload Locality
     - Spatial Locality: Zukünftige Zugriffe auf nahegelegene Adressen
       - Zugriff auf die gleiche Seite, daher gleicher Übersetzung Seitenummer auf Rhamennummer benötigt
         - gleiche TLB Eintrag wird wiederverwendent
     - Temporal Locality: Zukünftige Zugriffe sind Wiederholungen
       - ... wird bald wiederverwendent
     - TLB Ersetzungsstrategien
       - LRU(Least Recently Used): den am längsten ungenutzten TLB Eintrag entfernen (`evict`)
         - 需要额外的存储和开销
       - Random(简单，实现成本小)
     - Beim Kontextwechsel:
       - 新进程的虚拟地址空间和之前进程的虚拟地址空间不同，因此，之前进程的 TLB 条目就不再有效。
       - Option 1: Bei jedem Kontextwechsel TLB flushen
         - Zeitaufwändig, verliert alle Übersetzungen(die in nächstes Mal nützlich sind)
         - Einträge mit Prozess assoziieren
           - Jeder TLB Eintrag wird mit einem 8-bit Adress Space Identifier getagged

    - 区分不同进程或不同线程的虚拟地址空间
    - 避免进程间的地址映射冲突，提高缓存效率
    - 减少每次上下文切换时完全清空 TLB 的需要，从而提高系统的性能

2. Mehrstufige Seitentabllen

- Motivation: Seitentabllen nicht mehr zusammenhängend speichern
- Idee: Seitentabellen slebst in Seiten aufteilen
  - Seitenverzeichnis: *Page Directory*. die äußere Ebene
  - Nur die Seitentabelle allokieren, die mindestens einen gültiugen Eintrag enthalten
  - in x86 Architektur mit Hardwareunterstützung implementiert
- Adressformat: Wie viele Bits pro Ebene?
  - Ziel: jede einzelne Seitentabelle passt auf eine Seite
    - example:

```
Annahme Größe Eintrag: 4 Bytes
Seitengröße = 2^12 Bytes
Daher Anzahl Einträge = 2^10
Daher Anzahl der Bits für innere Seite : 10
Wenn 30 Bit virtuelle Adresse: Offset 12 Bit, Innere Seite 10, Äußere Seite 8
bits
```

    - 分层计算同理，注意每层都会汇总上一层的页面条数

3. Swapping

- 当物理内存不足时，将整个进程或部分数据暂时移到磁盘，等需要时再调回内存。
  - `eg. In code section of virtual memeory, there're many large libraries. Some of shich are rarely/never used.`
- Ziel des BS: Unterstützung der Prozess, wenn nicht genug physischer Speicher vorhanden ist
  - Einzelner Prozess mit sehr großem Adressraum
  - Mehrere Peozess mit kombiniertem hohem Speicherbedarf
  - User-code sollte von der Menge an physischem Speicher unabhängig sein
  - Virtuelle Speicher: Betriebssystem schafft Illusion von zusätzlichem physischem Speicher
- Locality of reference
  - Geschätzt: Prozess verbringe 90% ihre Zeit in 10% Teil des Codes.
    - Prozess verwendet zu jedem Zeitpunkt nur einen kleinen Teil des Adressraums
    - Nur ein kleiner Teil des Adressraums muss im physischen Speicher liegen
      - Prozess kann laufen ohne dass alle Seiten in den Hauptspeicher geladen sind

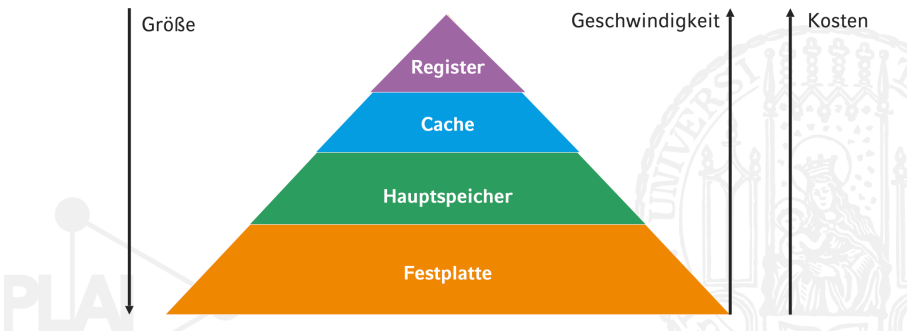

Figure 1: unterschiedliche Speicherebene.

4. Zusammenfassung von Virtual memeory
   - OS und Hardware schaffen Illusion eines großen und schnellen Hauptspeichers
     - gleiches Verhalten als wäre der gesamte Adressraum im Hauptspeicher(Idealerweise ähnliche Performance)
     - Voraussetzung:
       - BS muss den aktuellen Ort jeder Seite kennen(Hauptspeicher oder Festplatte)
       - BS braucht eine Policy, welche Seiten im Speicher und welche auf der Platte abgelegt werden sollen
   - Jede Seite im virtuellen Adressraum ist in einen der folgenden drei Orten:
     - Physischer Hauptspeicher: Klein, schnell, teuer
     - Festplatte: Groß, langsam, teuer
     - Nirgendwo(Fehler): frei
   - Einträge in Seitentabellen erhalten ein weiteres Bit: `present` (用于指示该页面是否在物理内存中)
     - Schutz(r/w), gültig, `present`
     - im Speicher: `present` bit wird im Seitentabelleneintrag gesetzt
     - auf der Platte: `present` bit wird gelöscht
       - Einträge in Seitentabelle zeigt auf Block auf der Festplatte
       - Sobald auf die Seite zugegeriffen wird, löst die MMU einen `page fault` aus: `trap` ins Betriebssystem
         - example

```
1. 页表查找：发现 present = 0，页面不在内存，存储在 磁盘 Block 5
2. MMU 触发 Page Fault：由于该页不在内存，MMU（内存管理单元）发出页面缺失（Page
Fault）异常，触发陷入（Trap）操作系统
3. 操作系统处理 Page Fault：
  3.1 操作系统在磁盘中找到该页面（Block 5）
  3.2 在内存中分配新的空闲页框
  3.3 从磁盘加载该页面到物理内存
  3.4 更新页表，将 present 位改为 1，并设置新的物理地址
4. 进程继续执行：CPU 重新执行引发 Page Fault 的指令
以上操作进程无感知，继续执行，但会有 I/O 延迟
```

Zusammenfassung

- Paging hat viele Vorteile, aber Seitentabellen können zu großem Overhead führen
- TLB speichert bekannte übersetzungen zwischen (caching)
  - Performance hängt vom Workload ab
  - Sequentielle Workloads oder solche mit temporaler Lokalität funktionieren gut
- Mehrstufige Seitentabellen erlauben, nicht die gesamte Seitentabelle in einem durchgängigen Speicherbereich vorzuhalten
  - Für große Anzahl an Seiten sonst nicht realisierbar (64 Bit!)
- Wenn Adressraum nicht in physischen Speicher passt, kann Betriebssystem die Illusion von virtuellem Speicher schaffen
  - Seitentabelleneintrag erhält present Bit
  - Betriebssystem behandelt Seitenfehler (page fault / miss) indem die fehlende Seite von der Festplatte geladen wird

Figure 2: an overall conclusion of chapter 5