UNIX-Speicherverwaltung unter MACH

UNIX-Speicherverwaltungsoperationen können aus den MACH-Operationen aufgebaut werden

- dynamische Stackerweiterung
  - •• virtuellen Speicher der maximalen Stackgröße allokieren (Ressourcen werden erst bei Bedarf belegt)
- brk
  - •• virtuellen Speicher im Adreßbereich hinter dem Datensegment-Bereich allokieren
- exec
  - •• ausführbare Datei in Adreßbereich einer Task mappen
- fork
  - •• wird von der Operation *task_create* erfüllt
- shared library
  - •• Teil einer Datei in Adreßraum mappen

Anwendungsbeispiel

memory mapped files für Benutzer und Kern

- Realisierung: VM-Objekt mit entsprechendem Pager

- Kern-Anwendungen:

  - Programme laden

  - shared libraries

  - Dateisystem

- Benutzeranwendungen:

  - neue *stdio* mit Dateizugriff über mapped file

  - Dateizugriff über Pointer

Erfahrungen mit verschiedenen HW-Architekturen

- MACH-VM ist
  - portabel
  - stellt wenig Anforderungen an die Hardware-Plattform
  - und wurde auf einer Reihe von Architekturen implementiert

➥ gute Basis für Untersuchungen von VM-Problemen bei verschiedenen Architekturen

VM auf Monoprozessoren

- VAX-Architektur
  - theoretisch 2GB Adreßraum
  - volle Nutzung des Adreßraums würde aber sehr große SKT bedeuten (8 MB) – da nur 512 Byte Seiten
  - UNIX-Systeme haben deshalb normalerweise den verfügbaren virt. Adreßraum auf 16 - 64 MB begrenzt
  - unter VMS können auch SKTs im virtuellen Adreßraum des Systemkerns gepaged werden
  - MACH erzeugt nur einen Teil der SKT und generiert bei Bedarf *(page fault)* weitere Teile aus den architekturunabhängigen Datenstrukturen

- IBM RT PC
  - •• unterstützt nicht SKT pro Prozeß, sondern eine globale ***inverted page table***
  - •• eine *inverted page table* ordnet jeder physikalischen Adresse die zugehörige virtuelle Adresse zu
  - •• Adreßauflösung erfolgt durch Suche in der IPT - unterstützt durch eine Hash-Funktion
  - •• 4 GB Adreßraum können ohne zusätzlichen Speicheraufwand verwaltet werden
  - •• Problem in MACH:
    - – für jede Kachel existiert nur eine Adreßabbildung
    - – bei *shared memory* wird eine Kachel aber auf mehrere virtuelle Adressen in versch. Adreßräumen abgebildet
    - – Folge: bei Zugriff auf *shared page* durch andere Task wird page fault erzeugt – die Seite muß *umgemapped* werden

VM auf Multiprozessoren

Problem: *Translation Lookaside Buffer (TLB)* -Inkosistenz

TLB ein Cache für die Adreßdaten der SKT, um eine schnelle Adreßübersetzung ohne zusätzliche Hauptspeicher-Zugriffe durch die MMU zu ermöglichen

- bei *shared-memory*-Multiprozessoren werden normalerweise Strategien implementiert, um die Caches der Prozessoren konsistent zu halten

- analog ist es notwendig die TLBs der Prozessoren konsistent zu halten, wenn Threads einer Task parallel auf verschiedenen Prozessoren ablaufen

  – wenn sich die architekturabhängigen VM-Datenstrukturen einer Task ändern, müssen die TLBs aller Prozessoren aktualisiert werden, auf denen Threads der Task ablaufen

  – in anderen Betriebssystemen ist dies meist kein Problem, weil in einem Adreßraum nicht mehrere Aktivitätsträger existieren

  – die *lazy evaluation* Techniken in MACH führen dazu, daß sehr häufig die pmap-Strukturen angepaßt werden müssen

- viele heutige MPS-Architekturen ignorieren dieses Problem!

das TLB-Inkonsistenz-Problem

➥ intuitive Lösungsidee:

- nach einer Änderung der VM-Abbildung werden durch einen Befehl die TLBs der anderen Prozessoren invalidiert
- Problem:
  – die meisten MPS-Systeme erlauben es einem Prozessor nur, seinen eigenen TLB, aber nicht die anderer Prozessoren zu invalidieren

➥ weitergehende Lösungsideen:

- Hardware-Unterstützung implementieren oder
- durch Software-Interrupts die anderen Prozessoren zum Invalidieren ihrer TLBs bewegen
- Probleme:
  – werden die pmap-Strukturen nach dem *flushing* der TLBs aktualisiert, können die Hardware-*reload*-Mechanismen inkonsistente Einträge nachladen
  – werden die pmap-Strukturen vorher aktualisiert, können die Strukturen beim *flushing* der TLBs durch inkonsistente TLB-Einträge überschrieben werden (reference + modified bits!)

➥ mögliche Lösungen

- Prozessoren werden durch Interrupt informiert und führen ein TLB-Konsistenz-Protokoll durch
- die Nutzung geänderter Abbildungen wird verzögert, bis alle TLBs *geflushed* wurden
- Inkonsistenzen werden in unkritischen Fällen toleriert (z. B. wenn die Zugrifsrechte erweitert werden)
- Komplexere Algorithmen: z. B. *shootdown* Algorithmus