MACH

Überblick

Entwicklung seit 1986 an der Carnegie Mellon Univ. (CMU)

Motivation:

- UNIX-Systemkern heute nicht mehr so einfach aufgebaut und leicht modifizierbar wie früher
  - früher auf PDP11 mit 32k HSP lauffähig, heute 1,5 - 2 MB Speicher (inkl. Puffer etc.) notwendig
  - viele neue Funktionen werden kritiklos in den Kern eingebaut, weil dort die notwendigen Information leicht zugänglich sind

    ⇒ Aufblähung des Kerns, Abstraktionen und Strukturierung werden mehr und mehr zerstört

- für Einsatz von UNIX auf Multiprozessorsystemen sind aufwendige Modifikationen notwendig
  - Kern-Monitor muß in Teile zerlegt werden oder ganz aufgegeben werden
  - an Stellen, an denen die Monitor-Eigenschaft aufgegeben wird, ist aufwendige Koordinierung erforderlich

wesentliche Eigenschaften

- Unterstützung für Multiprozessoren
- neues - hardwareunabhängiges - Konzept der virtuellen Speicherverwaltung
- *Capability*-basierte Interprozeßkommunikation - transparent erweiterbar für Netzwerk-Kommunikation
- Modularer Aufbau, einfache Erweiterbarkeit, nur die wichtigsten Funktionen sind im MACH-Kern realisiert
- die Betriebssystemumgebung wird durch eine Reihe von *Servern* realisiert, die über die Basis-Mechanismen des MACH-Kerns angesprochen werden können
  Beispiele:
  - Dateisystem
  - Netzwerk-Kommunikation (z. B. TCP/IP)
  - Scheduling

Abstraktionen des MACH-Kerns

Task Ablaufumgebung für ***Threads***

- virtueller Adreßraum
- gesicherter Zugriff zu Systemressourcen (Prozessoren, ***Port-Capabilities***, Speicher)

Thread Aktivitätsträger innerhalb einer ***Task***

- beschreibbar durch
  - einen unabhängigen Programmzähler
  - eigenen Stack-Bereich innerhalb des virt. Adreßraums der *Task*
- alle *Threads* einer *Task* haben gemeinsamen Zugriff zu allen *Task*-Ressourcen

Port Kommunikationskanal - Warteschlange für Mitteilungen

- wird vom MACH-Kern verwaltet

Message Menge von Daten-Objekten für die Kommunikation zwischen *Threads*

- Messages besitzen einen Typ
- können max. den gesamten virt. Adreßraum umfassen
- können Zeiger und *Capabilities* für *Ports* enthalten

Operationen

Operationen auf einem Objekt (ausgen. *Messages*) werden durch Versenden von *Messages* an *Ports*, die dieses Objekt repräsentieren ausgeführt

Beispiel: Beim Generieren einer *Task* erhält man die Zugriffsrechte auf einen *Port* dieser *Task*.
Durch *Messages* an diesen *Port* kann die *Task* beeinflußt werden.

Tasks und Threads

Einführung

UNIX-Prozeßkonzept ist für viele heutige Anwendungen unzureichend

- in Multiprozessorsystemen werden häufig parallele Abläufe in einem virtuellen Adreßraum benötigt
- zur besseren Strukturierung von Problemlösungen sind oft mehrere Aktivitätsträger innerhalb eines Adreßraums nützlich
- typische UNIX-Server-Implementierungen benutzen die *fork*-Operation, um einen Server für jeden Client zu erzeugen

  ⇒ Verbrauch unnötig vieler System-Ressourcen (Datei-Deskriptoren, SKTs, Speicher, ...)

Lösungsansätze:

- derzeitige Multiprozessor-Betriebssysteme (z. B. Dynix) ermöglichen gemeinsame Speicherbereiche für mehrere Prozesse

  Nachteile:

  - viele Betriebsmittel zur Verwaltung eines Prozesses notwendig; Prozeßumschaltungen aufwendig → teuer
  - innerhalb einer solchen Prozeßfamilie wäre häufig ein anwendungsorientiertes Scheduling notwendig: schwierig realisierbar

- einige Anwendungen lassen sich mit Hilfe von Koroutinen (auf Benutzerebene) innerhalb eines Prozesses gut realisieren

  Nachteile:

  - Scheduling zwischen den Koroutinen schwierig (Verdrängung meist nicht möglich)
  - in Multiprozessorsystemen keine parallelen Abläufe möglich
  - wird eine Koroutine wegen eines *page faults* oder in einem Systemaufruf blockiert, ist der gesamte Prozeß blockiert

- Threads: leichtgewichtige Prozesse *(lightweighted processes)*
  - •• eine Gruppe leichtgewichtiger Prozesse nutzt gemeinsam eine Menge von Betriebsmitteln
  - •• jeder leichtgewichtige Prozeß ist aber ein eigener Aktivitätsträger
    - – eigener Programmzähler
    - – eigener Registersatz
    - – eigener Stack
  - •• Umschalten zwischen zwei leichtgewichtigen Prozessen einer Gruppe ist erheblich billiger als eine normale Prozeßumschaltung
    - – es müssen nur die Register und der Programmzähler gewechselt werden (entspricht dem Aufwand für einen Funktionsaufruf)
    - – Adreßraum muß nicht gewechselt werden
    - – alle Systemressourcen bleiben verfügbar

Abstraktionen in MACH: Tasks und Threads

Task: Betriebsumgebung (System-Resourcen) für Aktivitätsträger

- virtueller Adreßraum
- Zugriffsrechte
- Betriebsmittel-Informationen
- kein Programm-Ablauf und keine Register

Thread: Aktivitätsträger und seine Ablaufumgebung

- Registersatz
- Stack
- Programmzähler

- Innerhalb einer *Task* können beliebig viele *Threads* existieren
- In einer echten Multiprozessorumgebung können verschiedene *Threads* einer *Task* parallel auf mehreren Prozessoren ablaufen
- Ein herkömmlicher UNIX-Prozeß entspricht einer MACH-*Task* mit einem *Thread*

Interprocess Communication - IPC

Mitteilungs-orientierte Kommunikation

Abstraktionen

- Port
  - geschütztes Objekt des MACH-Kerns
  - kann Mitteilungen ***(messages)*** von einem Sender entgegennehmen und an einen Empfänger ausliefern
  - zu einem *Port* können beliebig viele Sender, aber nur ein Empfänger existieren
  - Zugriff auf einen *Port* erhält man durch Empfang einer Mitteilung, die ein *Port*-Zugriffsrecht ***(Capability)*** enthält
- Message
  - Ansammlung von Datenobjekten, die durch einen Typ näher gekennzeichnet sind
    - Daten (beliebiger Länge)
    - Zeiger
    - *Port*-Zugriffsrechte *(Capabilities)*

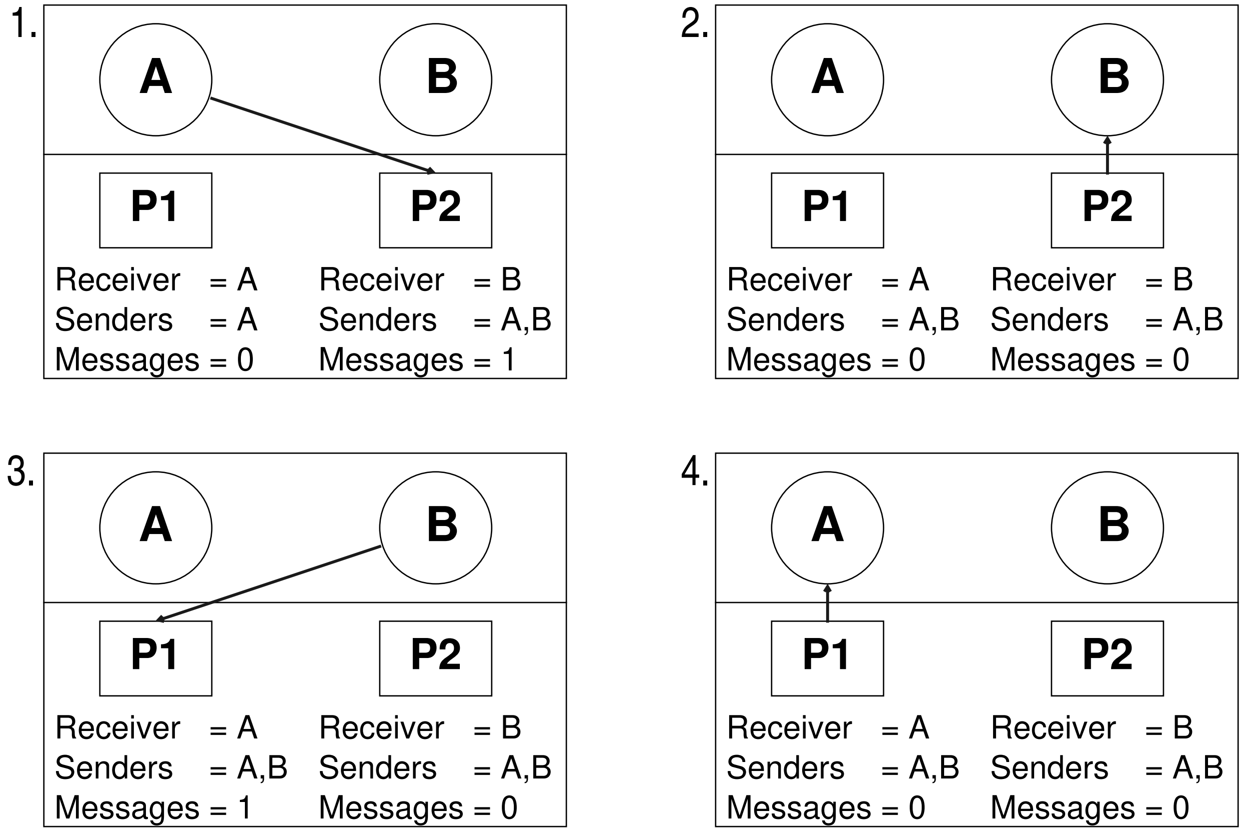
IPC ist in MACH der allgemeine Mechanismus zur Ausführung von Operationen auf Objekten

- Objekten (z. B. *Tasks* und *Threads*) sind Ports zugeordnet
- Operationen auf Objekten werden durch Mitteilungen an diese Ports ausgeführt

Beispiel:

Beim Erzeugen eines Threads erhält man Senderechte auf die *Ports* dieses *Threads*. Durch Mitteilungen an diese *Ports* kann der *Thread* manipuliert werden (suspendieren, deblockieren, vernichten, ...)

Ablauf einer typischen IPC:



- Task A sendet *Message* an Port P2
- Task A hat Senderechte für Port P2 und Empfangsrechte für Port P1
- Task B hat Empfangsrechte für Port P2 und empfängt irgendwann die Mitteilung
- die Mitteilung enthält u. a. Sendrechte für Port P1
- Task B antwortet durch eine *Message* an Port P1

Übertragen einer ”großen” Mitteilung

- *Task* A sendet Mitteilung an *Port* P1
- der Datenbereich der *Message* wird temporär in den Adreßraum des MACH-Kerns abgebildet (= Senden an *Port*)
- im Adreßraum der *Task* A wird der Datenbereich als *”copy on write”* markiert
  (d. h. für Seiten, auf die nachfolgend schreibend durch A zugegriffen wird, wird eine Kopie erzeugt und diese wird in den virtuellen Adreßraum von A übernommen)
- *Task* B empfängt die Mitteilung, indem der Datenbereich in den virt. Adreßraum von B abgebildet wird
- Aus dem Adreßraum des MACH-Kerns wird der Datenbereich wieder entfernt
- Auch bei *Task* B ist der Datenbereich als *”copy on write”* markiert

MACH - virtuelle Speicherverwaltung

Eigenschaften

- portabel
- hardware-unabhängig
- unterstützt Multiprozessoren
  - ➥ parallel ablauffähig implementiert
- unabhängige Pager

Motivation

- heute viele unterschiedliche MMUs verbreitet

- weitgehend äquivalente Funktionalität

  - unterschiedliche virtuelle Adreßräume, beschrieben durch Datenstrukturen (z. B. *page table*)

  - pro Seite:

    - Zeiger auf physikalische Adresse der Seite

    - *valid bit* – zeigt an, ob die phys. Seite gültig ist

    - *protection bits* – (z. B. *read/write, user/kernel*)

    - *modify bit*

    - *reference bit*

- typischerweise zwei mögliche Vorgehensweisen in heutigen Betriebssystemen zur Unterstützung von virtuellem Speicher

  - Mechanismen, die möglichst gut den Eigenschaften der verwendeten MMU entsprechen

    ➥ System nicht portabel (z. B. VMS für VAX)

  - Beschränkung auf Eigenschaften, die einfach auf jeder Hardware realisierbar sind

    ➥ System weniger leistungsfähig und ineffizient (z. B. UNIX in 4.3bsd)

Realisierung

Ziel

- möglichst flexible Unterstützung von virtuellem Speicher auf möglichst vielen Architekturen

Vorgehensweise

- Unterteilung der Implementierung

  ➥ architekturunabhängiger Teil

  - realisiert Benutzerschnittstelle und Funktionsumfang
  - Verwaltung der architekturunabhängigen Datenstrukturen
  - alle notwendigen Abbildungsinformationen für jede Seite des Systems können aus den architekturunabhängigen Datenstrukturen gewonnen werden

➥ architekturabhängiger Teil

– Verwaltung der von der Hardware benötigten Adreßabbildungstabellen

physical address maps – pmaps

entsprechen den MMU-Tabellen
(z. B. VAX-page-table oder eine Menge von Segment-Registern beim IBM PC/RT)

– Abbildung von MACH-Seiten in eine oder mehrere Kacheln

– konstruiert nur einen Teil der Adreßabbildung - Zugriff auf weitere Adressen kann bei Page-fault ermöglicht werden

→ kleine Tabellen trotz großem (dünn besetztem) Adreßraum (vgl. herkömmliche VM-Implementierungen - z. B. VAX)

- VM-unabhängige Pager
  - •• Seiten-Ein- und Auslagerung wird durch VM von Pagern über Messages angefordert

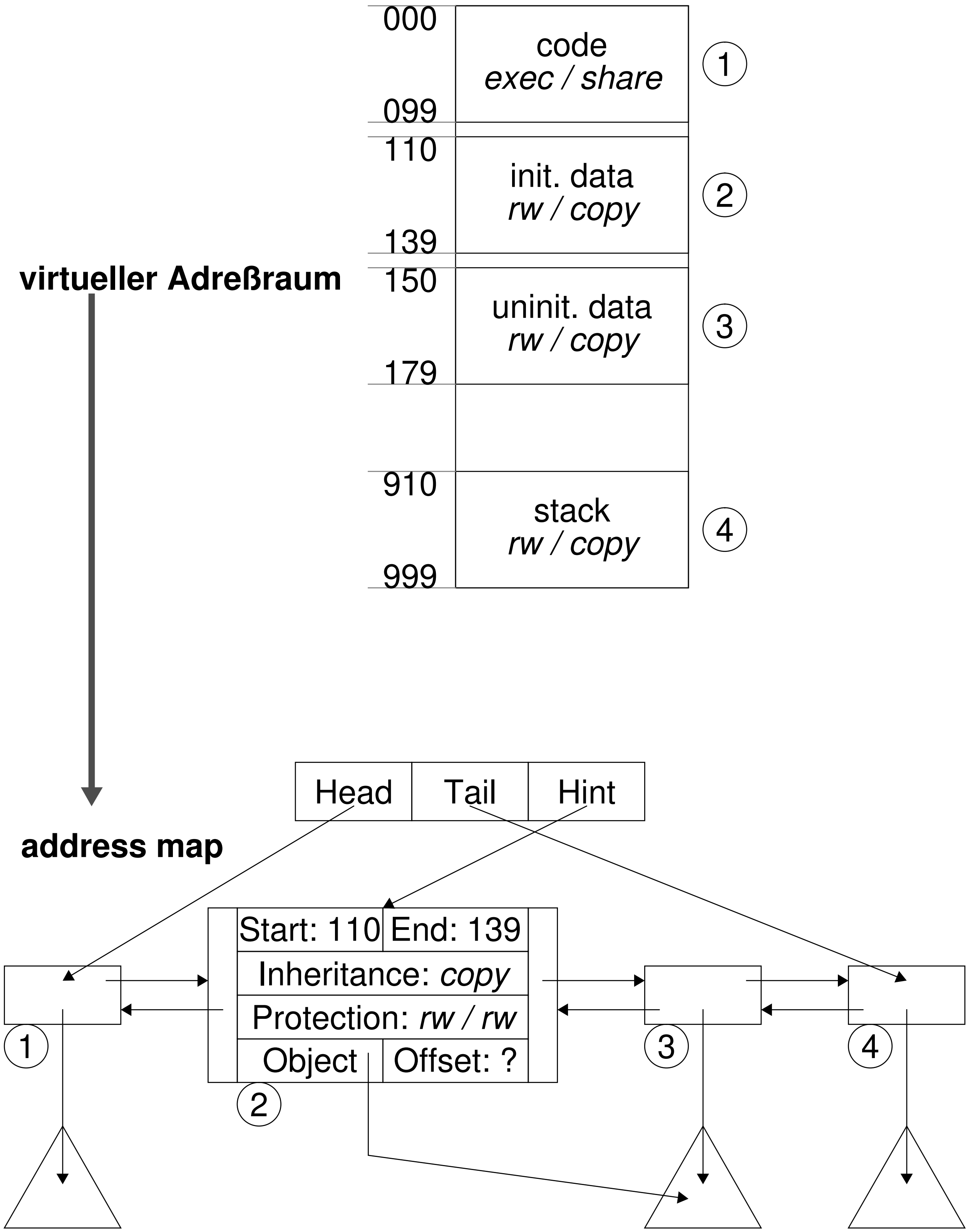
- Lazy evaluation
  - •• Speicher (Haupt- und Hintergrundspeicher!) wird erst belegt, wenn eine Seite benutzt wird
  - •• Map-on-Reference: Page-Tables werden nur für aktuell benötigten Speicher bereitgestellt
  - •• Copy-on-write

Datenstrukturen

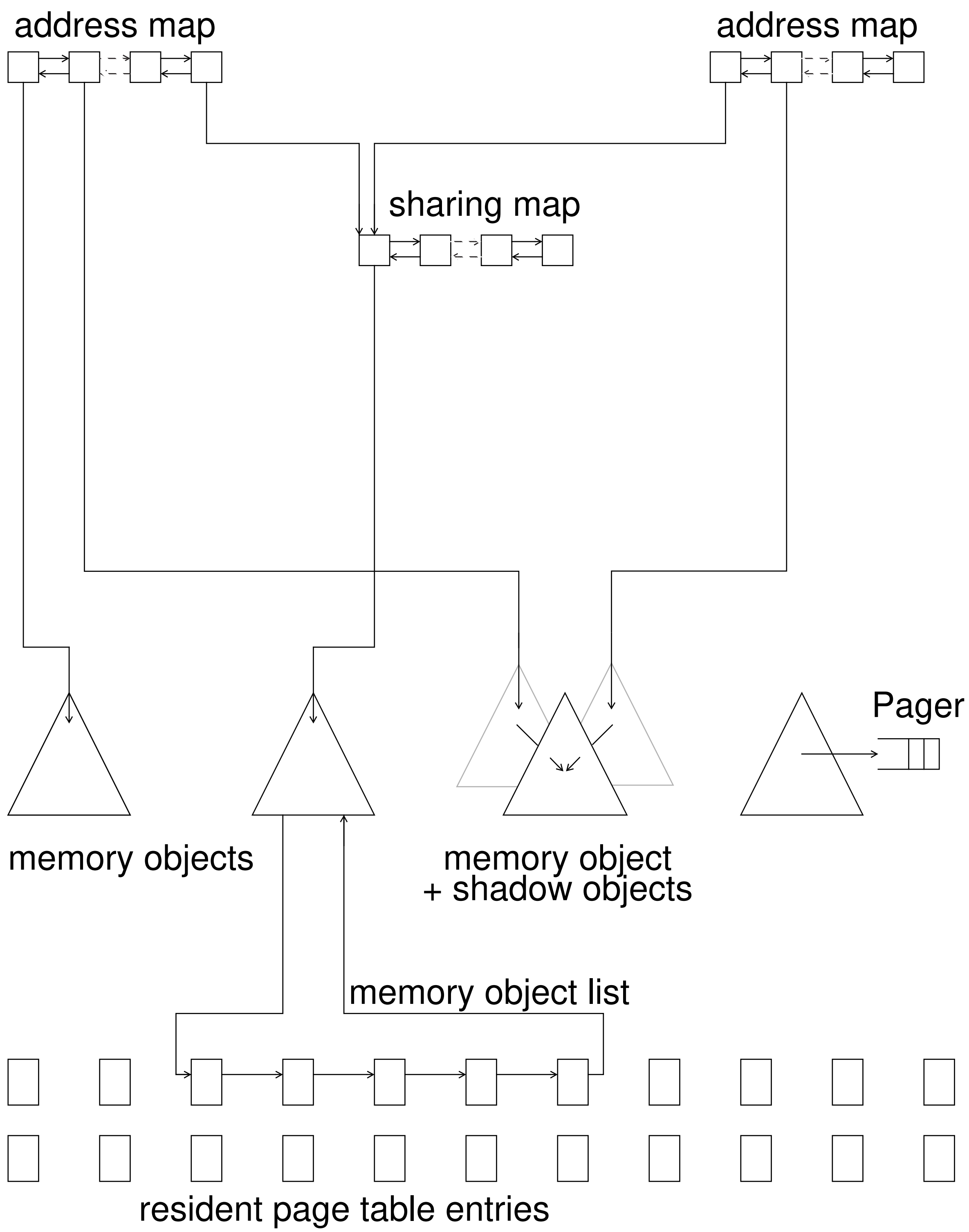
- Address Map
  - Abbildung von (virtuellen) Speicherbereichen mit gleichen Eigenschaften (Schutz, Vererbung) in einen Bereich eines *Memory Objects*
  - als verkettete Liste von *address-map-entries* realisiert
  - ein *address-map-entry* enthält:
    - Vorwärts- und Rückwärts-Verzeigerung
    - Start- und Endadresse des Speicherbereichs im virtuellen Adreßraum
    - aktuelle und maximale Zugriffsrechte
    - Regel für Vererbung des Speicherbereichs an neue Task
    - Zeiger auf *Memory Object*
    - Offset dieses Speicherbereichs im *Memory Object*
- Sharing Maps
  - zur Realisierung von shared Memory
  - Zwischenstufe zwischen *address map* und *Memory Object*
  - um Änderungen im *address map* -Eintrag eines Objekts nicht in den *address maps* mehrerer Tasks nachtragen zu müssen

- Memory Object
  - Bindeglied zwischen virtuellem Adreßraum und Hintergrundspeicher
  - weiß für eine Anzahl Bytes, wie sie zu beschaffen sind und was mit ihnen im pagout-Fall zu tun ist
  - Pager (intern oder extern) realisiert Ein-/Auslagerung von Seiten
  - *Memory Objekt* hat Information, welche seiner Daten gerade im Hauptspeicher *gecached* sind
- Shadow Object
  - zur Realisierung von *copy-on-write*
  - nimmt Seiten auf, die kopiert werden mußten
  - verweist für unveränderte Seiten auf das original-Objekt
- Resident Page Table
  - Verwaltung des **Hauptspeicher-Caches** für *Memory-Object-*Seiten
  - Verkettung in Listen (*memory object, memory allocation, object/offset hash*)

- Datenstrukturen: virtueller Adreßraum - address map



- Datenstrukturen - Gesamtüberblick



Benutzerschnittstelle

MACH-VM stellt folgende Funktionen zur Verfügung:

- virtuelle Speicherbereiche allokieren und freigeben
- Schutzparameter für einzelne virtuelle Speicherbereiche individuell einstellen (innerhalb vorgegebener Schranken: *current / maximum protection*)
- Vererbung an neue Task einzeln einstellbar
  - keine Vererbung
  - Copy (on write) - default
  - Shared
- Zugriffsmöglichkeit auf VM anderer Tasks (Debugger!)
- Status- und Statistikabfragen

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!)

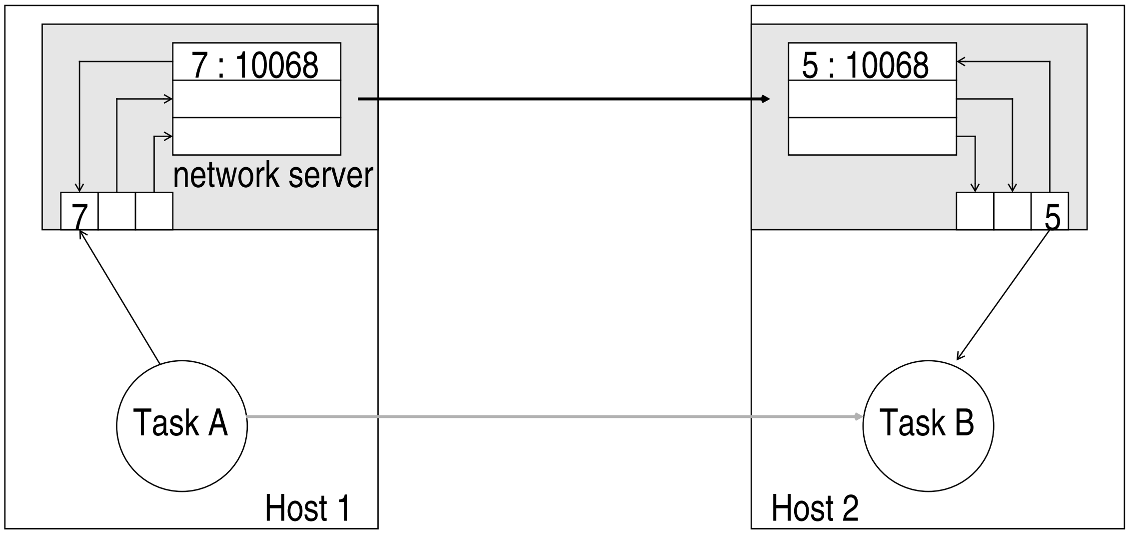
➥ mögliche Lösungen

- 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 Zugriffsrechte erweitert werden)
- Komplexere Algorithmen: z. B. *shootdown* Algorithmus

MACH in verteilten Systemen

Netzwerk-Kommunikation

- Mach IPC nur für lokale Kommunikation konzipiert
- Nachrichten können durch *Network-Server-Tasks* (user-level!) an andere Knoten weitergeleitet werden
- Capability-Funktionalität wird über Netzwerk abgebildet
- Netzwerk-Transfer wird geschützt
  - DES-Verschlüsselung der Daten
  - Sequenznummern und Checksummen für Pakete
  - Authentisierung anderer Netzwerk-Server für Capability-Transfer (*Secret Identifier*)



Distributed Shared Memory

- der *shared memory* Bereich wird durch ein *Memory Object* repräsentiert
- jede beteiligte Task bildet den Speicherbereich in ihren Adreßraum ab
- dem *Memory Object* wird ein *external pager* zugeordnet
- durch die *lazy evaluation* Technik werden die Seiten erst auf Anforderung in den Hauptspeicher (Seiten-Cache) geholt
- im Fall von *page faults* werden die Seiten von dem *external pager* angefordert
  - diese Anforderungen erfolgen über die regulären MACH Kommunikationsmechanismen - sind also auch auf Netzwerk-Kommunikation erweiterbar
- als *external pager* wird ein sog. ***shared memory server*** benutzt, der Buch über das Caching der Seiten führt
- *read-only* kann eine Seite problemlos auf mehreren Knoten gecached werden
- bei Schreibzugriff wird ein *page fault* erzeugt, der *shared memory server* wird informiert, er entzieht den anderen Knoten den Zugriff auf die Seite und erlaubt dann den Schreibzugriff