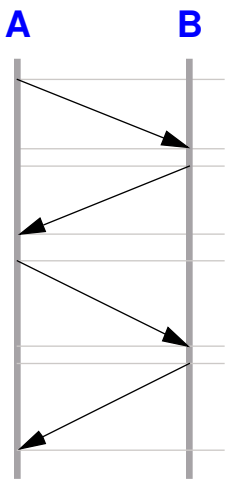
4 Beispiel: Kerberos (4)

- Unterscheidung zwischen Benutzer *(User)* und Benutzerprogramm *(Client)*
  - Wie kann ein Benutzerprogramm seinen Benutzer identifizieren?
  - Geheimer Schlüssel vom Benutzer (K_X) hängt von einem Paßwort ab
  - mittels einer Einwegfunktion wird aus dem Paßwort der Schlüssel K_X erzeugt
  - Benutzerprogramm braucht also das Paßwort zur Verbindungsaufnahme
- Beispiel: *kinit, klogin*
  - Anmeldung beim Authentisierungsdienst mit *kinit* und Paßworteingabe
  - Ticket wird im Benutzerkatalog gespeichert
  - *klogin* erlaubt das Einloggen auf einem entfernten Rechner mit Datenverschlüsselung und ohne Paßwort

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Austausch öffentlicher Schlüssel

- A und B tauschen ihre öffentlichen Schlüssel aus



A schickt an B seinen öffentlichen Schlüssel: K_A

B schickt an A seinen öffentlichen Schlüssel: K_B

A verschickt signierte und verschlüsselte Nachricht:
$E(K_B, E(K_A^{-1}, T_1))$

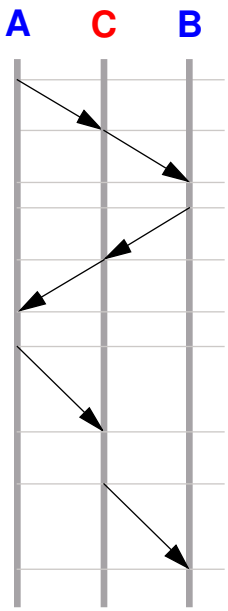
B verschickt signierte und verschlüsselte Nachricht:
$E(K_A, E(K_B^{-1}, T_2))$

- Problem
  - A und B können nicht sicher sein, daß der öffentliche Schlüssel wirklich vom jeweils anderen stammt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Austausch öffentlicher Schlüssel (2)

- Aktiver Mithörer C fängt Datenverbindungen ab



A versucht an B seinen öffentlichen Schlüssel zu schicken: K_A

C fängt Nachricht ab und schickt an B seinen eigenen öffentlichen Schlüssel: K_C

B versucht an A seinen öffentlichen Schlüssel zu schicken: K_B

C fängt Nachricht ab und schickt an A seinen eigenen öffentlichen Schlüssel: K_C

A versucht an A eine Nachricht zu schicken:
$C_1 = E(K_C, E(K_A^{-1}, T_1))$

C dekodiert die Nachricht: $T_1 = D(K_C^{-1}, D(K_A, C_1))$

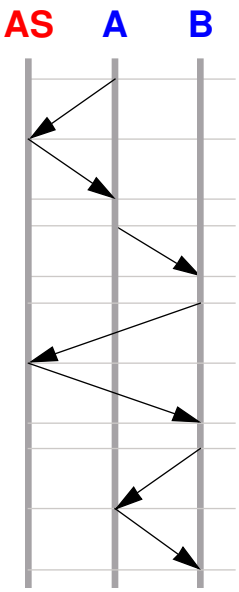
C schickt die Nachricht weiter: $E(K_B, E(K_C^{-1}, T_1))$

B glaubt eine Nachricht von B zu empfangen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Austausch öffentlicher Schlüssel (3)

- Einsatz eines Authentisierungsdienstes



A kontaktet AS und möchte Verbindung mit B: a, b

AS verschickt den Schlüssel von B: $E(K_{AS}^{-1}, <K_B, b>)$

A schickt initiale Nachricht an B: $E(K_B, <a, i_a>)$

B kontaktet AS und möchte den Schlüssel von A: b, a

AS verschickt den Schlüssel von A: $E(K_{AS}^{-1}, <K_A, a>)$

B antwortet an A: $E(K_A, <i_a, i_b>)$

A bestätigt: $E(K_B, <i_b>)$

- Replay-Probleme
  - Hinzunahme von Zeitstempel und Lebendauer

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

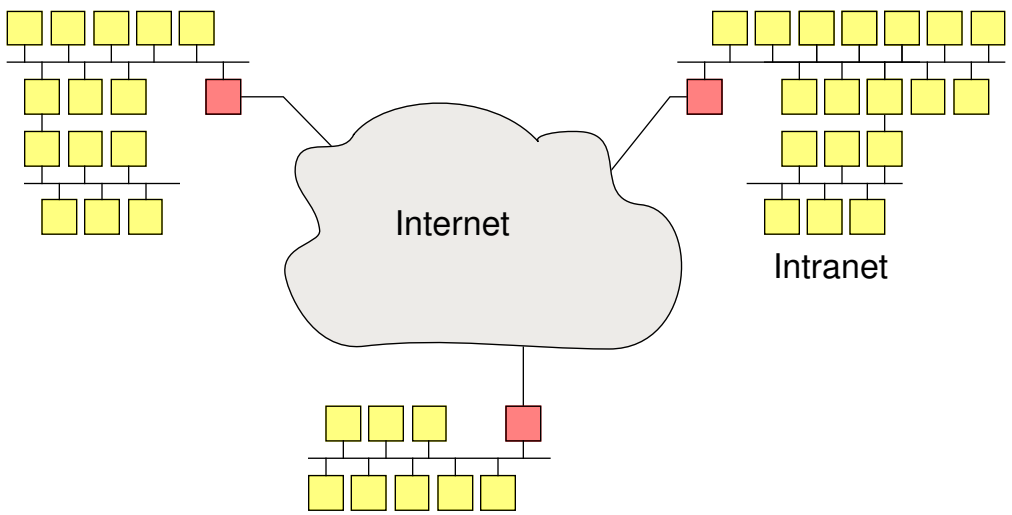
I.8 Firewall

- Trennung von vertrauenswürdigen und nicht vertrauenswürdigen Netzwerksegmenten durch spezielle Hardware *(Firewall)*
  - Beispiel: Trennen des firmeninternen Netzwerks (Intranet) vom allgemeinen Internet
- Funktionalität
  - Einschränkung von Diensten
    - von innen nach außen, z.B. nur Standarddienste
    - von außen nach innen, z.B. kein Telnet, nur WWW
  - Paketfilter
    - Filtern „defekter“ Pakete, z.B. SYN-Pakete
  - Inhaltsfilter
    - Filtern von Pornomaterial aus dem WWW oder News
  - Authentisieren von Benutzern vor der Nutzung von Diensten

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

I.8 Firewall (2)

- Virtual private network
  - Verbinden von Intranet-Inseln durch spezielle Tunnels zwischen Firewalls
  - getunnelter Datenverkehr wird verschlüsselt
  - Benutzer sieht ein „großes Intranet“ (nur virtuell vorhanden)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J Mach

- Fallbeispiel: *Mach* Betriebssystem
  - Mikrokern
  - 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)
  - Betriebssystemumgebung wird durch eine Reihe von *Servern* realisiert
- Geschichte
  - Entwicklung 1985–1994 an der Carnegie Mellon Univ. (CMU)
  - Basis für OSF/1 (Open Software Foundation), NeXT-OS (Next/Apple)
  - Bedeutung heute eher gering

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

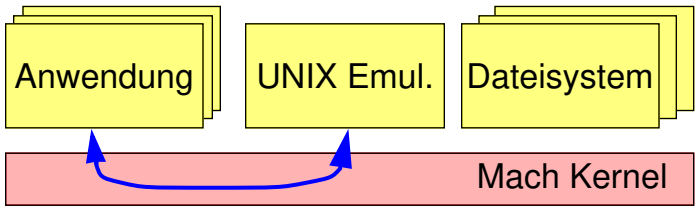
J.1 Motivation

- Entwicklung des UNIX-Systemkerns
  - 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.1 Motivation (2)

- Architektur von Mach
  - Mikrokern
  - 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
    - UNIX-Emulation

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.2 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 mit Warteschlange für Nachrichten
  - wird vom MACH-Kern verwaltet

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.2 Abstraktionen des Mach-Kerns (2)

- *Message:* Menge von Datenobjekten für die Kommunikation zwischen *Threads*
  - Messages können typisiert werden
  - können max. den gesamten virt. Adreßraum umfassen
  - können Zeiger und *Capabilities* für *Ports* enthalten
- Operationen
  - Operationen auf einem Objekt (ausgenommen *Messages*) werden durch Versenden von *Messages* an *Ports* ausgeführt, die dieses Objekt repräsentieren.
  - Beispiel:
    - Beim Generieren einer *Task* erhält man die Zugriffsrechte auf einen *Port* dieser *Task*.
    - Durch *Messages* an diesen *Port* kann die *Task* beeinflußt werden.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

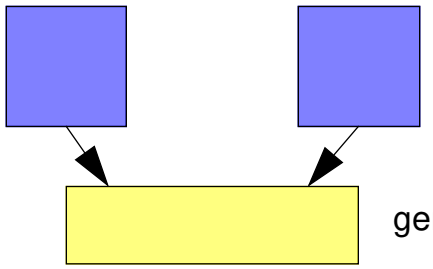
J.3 Tasks und Threads

- 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 Systemressourcen (Dateideskriptoren, SKTs, Speicher etc.)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.3 Tasks und Threads (2)

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

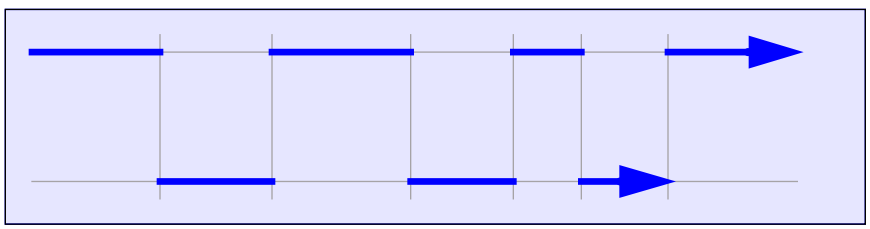


- Nachteile
  - viele Betriebsmittel zur Verwaltung eines Prozesses notwendig; Prozeßumschaltungen aufwendig
  - innerhalb einer solchen Prozeßfamilie wäre häufig ein anwendungsorientiertes Scheduling notwendig, aber schwierig realisierbar

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.3 Tasks und Threads (3)

- Lösungsansatz: Koroutine
  - 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.3 Tasks und Threads (4)

- Lösungsansatz: 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Task

- Betriebsumgebung (System-Ressourcen) für Aktivitätsträger
  - ◆ virtueller Adreßraum
  - ◆ Zugriffsrechte
  - ◆ Betriebsmittel-Informationen
  - ◆ kein Programmablauf und keine Register

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 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*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Interprocess Communication – IPC

- nachrichtenorientierte Kommunikation
- Port
  - geschütztes Objekt des MACH-Kerns
  - kann Nachrichten *(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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Interprocess Communication – IPC (2)

- Message
  - ◆ Ansammlung von Datenobjekten
  - ◆ unstrukturiert oder strukturiert (durch einen Typ näher gekennzeichnet), z.B.:
    - Daten mit Typinformation
    - Zeiger auf Adreßraumbereiche
    - *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 Nachrichten an diese Ports ausgeführt
  - ◆ zuverlässige Nachrichtenübertragung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

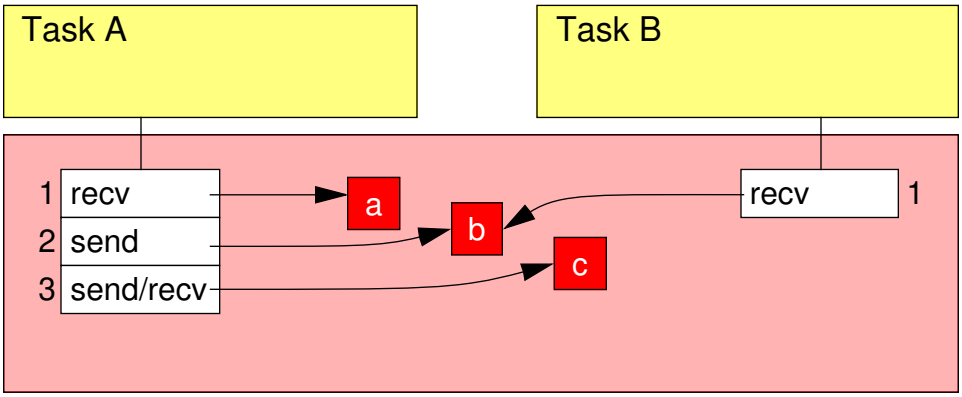
3 Interprocess Communication – IPC (3)

- 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 Interaktion
  - ◆ Ein Thread von Task A schickt eine Nachricht an einen Port
  - ◆ Task B ist der Empfänger für diesen Port

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Interprocess Communication – IPC (4)

- Capabilities werden ähnlich wie Dateideskriptoren verwaltet

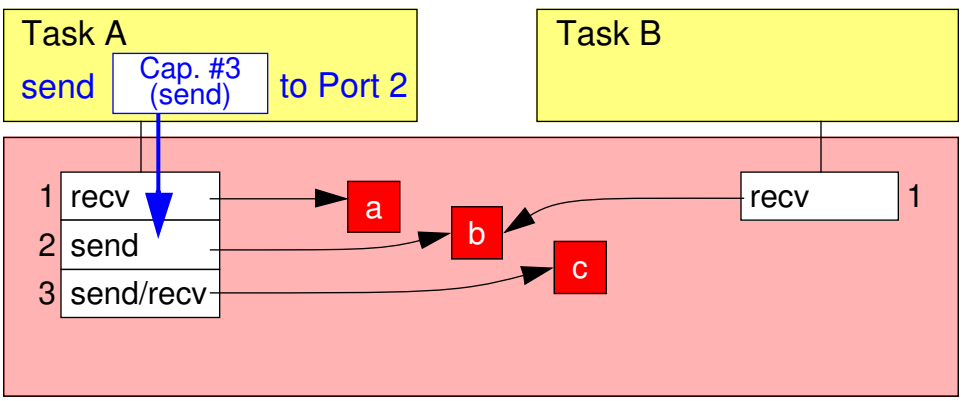


- ◆ Jeder Task hat einen privaten Deskriptor für eine Portcapability
- ◆ Deskriptor ist mit Rechten verbunden:
  Senderecht (send), Empfangsrecht (recv)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Interprocess Communication – IPC (5)

- Task A erzeugt eine Nachricht und sendet sie

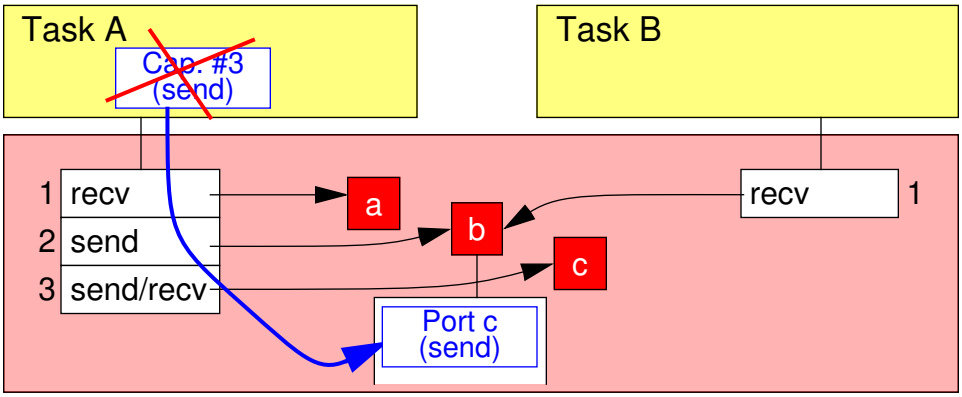


- ◆ Adressat ist Port mit Deskriptor 2, Port b

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Interprocess Communication – IPC (6)

- Einstellen der Nachricht in der Port-Warteschlange

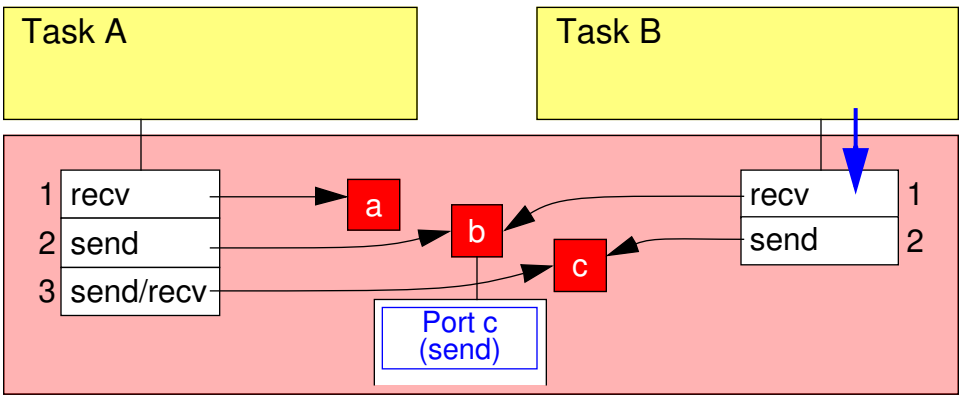


- ◆ Nachricht wird in die Warteschlange des Port b eingefügt
- ◆ übermittelte Capability wird in Portadresse umgesetzt
- ◆ Nachricht steht jetzt zum Empfang bereit
- ◆ Task A kann den Nachrichtenspeicher löschen oder wiederverwenden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Interprocess Communication – IPC (7)

- Task B möchte eine Nachricht empfangen

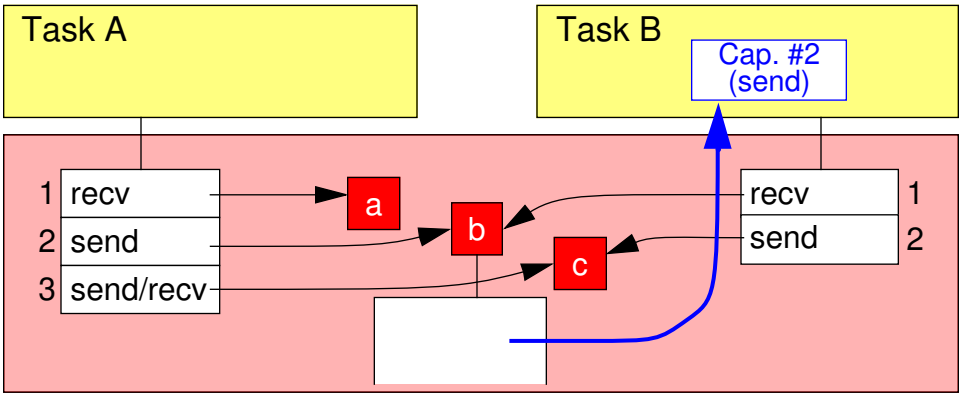


- ◆ Task B benötigt neuen Deskriptor für die Capability in der Nachricht
- ◆ Deskriptor 2 mit dem entsprechenden Recht wird erzeugt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Interprocess Communication – IPC (8)

- Task B empfängt Nachricht



- ◆ Task B erhält die Nachricht
- ◆ Portadresse wird in Deskriptor übersetzt
- ◆ Nachricht wird aus der Warteschlange entfernt
- ◆ Task B hat nun Zugriff auf den Port c

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Übertragen einer „großen“ Nachricht

- Ablauf
  - ◆ Task A sendet Message 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.4 Virtuelle Speicherverwaltung (VM)

- Eigenschaften
  - portabel
  - hardwareunabhängig
  - unterstützt Multiprozessoren: parallel ablauffähig implementiert
  - unabhängige *Pager*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 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 physikalische Seite gültig ist
    - *Protection bits* (z. B. *read/write, user/kernel*)
    - *Dirty bit* (Seite ist modifiziert)
    - *Reference bit* (Seite wurde angesprochen)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Motivation (2)

- Zwei mögliche Vorgehensweisen zur Unterstützung von virtuellem Speicher
  - Mechanismen, die möglichst gut den Eigenschaften der verwendeten MMU entsprechen
    - System nicht portabel (z. B. VMS für VAX)
    - effizient
  - Beschränkung auf Eigenschaften, die einfach auf jeder Hardware realisierbar sind
    - System weniger leistungsfähig und ineffizient (z. B. UNIX in 4.3bsd)
    - aber portabler

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Realisierung

- Ziel
  - möglichst flexible Unterstützung von virtuellem Speicher auf möglichst vielen Architekturen
- Vorgehensweise
  - Unterteilung der Implementierung
    - architekturabhängiger Teil
    - architekturunabhängiger Teil
    - externe Pager

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Realisierung (2)

- 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 Segmentregistern 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Realisierung (3)

- 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
- externe Pager
  - ◆ Seitenein- und -auslagerung wird durch die virtuelle Speicherverwaltung (VM) von sogenannten *Pagern* über Messages angefordert
    - entscheiden nicht über Ein- und Auslagerungsstrategie
    - entscheiden wohin ausgelagerte Seiten transportiert und woher eingelagerte Seiten bezogen werden (z.B. Platte, Netzwerk etc.)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Realisierung (4)

- 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:* Seiten werden solange gemeinsam benutzt wie möglich; bei modifizierendem Zugriff werden Nutzer voneinander getrennt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Datenstrukturen

- Memory Object
  - repräsentiert einen Speicherbereich, der ganz oder teilweise in einen Adreßraum eingeblendet werden kann
  - Bindeglied zwischen virtuellem Adreßraum und Hintergrundspeicher
  - das *Memory object* kennt oder implementiert einen Pager, der weiß woher die Seiten des Speicherbereichs beschafft werden können und was im Falle eines *Pagout* zu tun ist
  - Pager (intern oder extern) realisiert Ein-/Auslagerung von Seiten
  - *Memory object* hält Liste von Seiten, die momentan im Hauptspeicher vorhanden sind *(cached pages)*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Datenstrukturen (2)

- Address Map
  - repräsentiert einen Adreßraum
  - als verkettete Liste von *Address map entries* realisiert
  - jeder Eintrag bildet einen (virtuellen) Speicherbereich mit gleichen Eigenschaften (Schutz, Vererbung) in einen Abschnitt eines *Memory objects* ab
  - *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 entsprechenden *Memory object*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

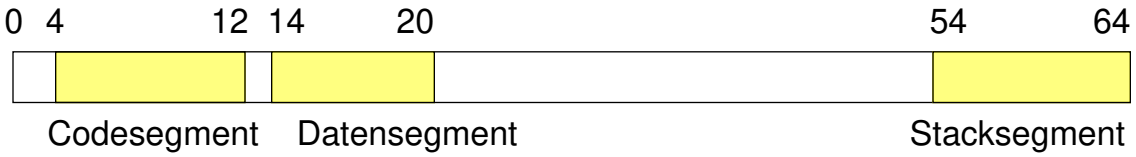
3 Datenstrukturen (3)

- Sharing Maps
  - zur Realisierung von *Shared memory*
  - Zwischenstufe zwischen *Address map* und *Memory object*
  - Änderungen im *Address map*-Eintrag eines Objekts müssen nicht in den *Address maps* mehrerer Tasks nachgetragen werden
- Shadow Object
  - wird wie ein *Memory object* benutzt
  - zur Realisierung von *Copy-on-write*
  - nimmt Seiten auf, die kopiert werden mußten
  - verweist für unveränderte Seiten auf das Originalobjekt

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Datenstrukturen (4)

- Resident Page Table
  - ◆ Verwaltung des Hauptspeicher-Caches für *Memory object*-Seiten
  - ◆ Verkettung in Listen
- Beispiel eines Adreßraums (z.B. einer Task)

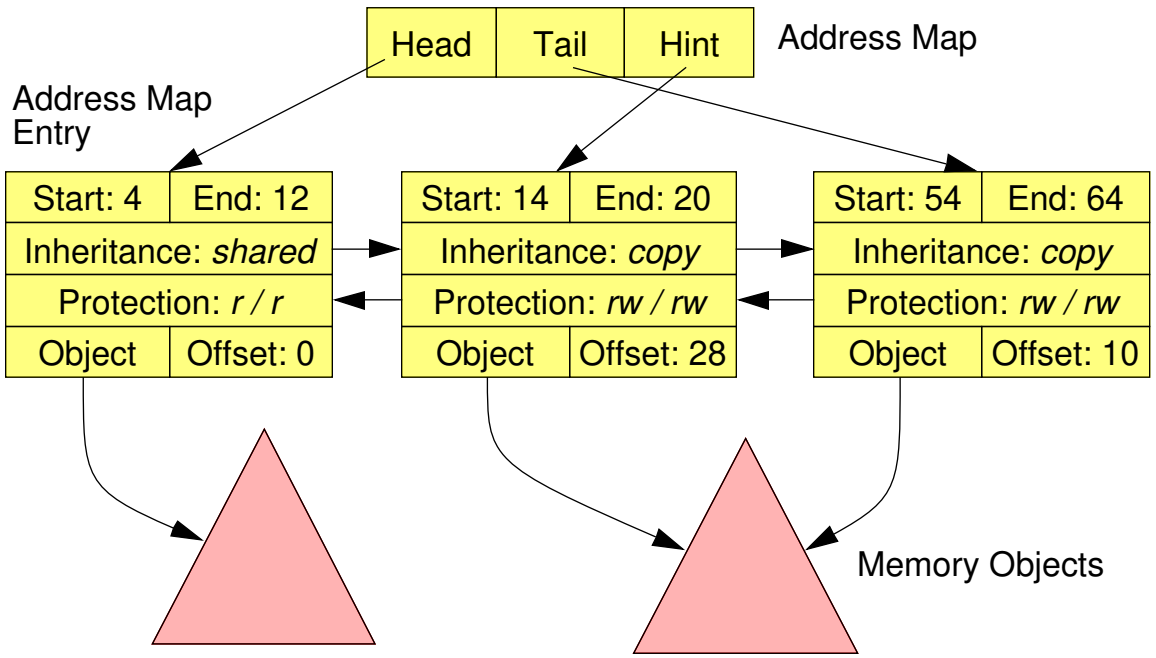


  - ◆ Abbildung des Codesegments in eine Programmdatei
  - ◆ Abbildung des Daten- und Stacksegments in einen *Swap space*

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Datenstrukturen (5)

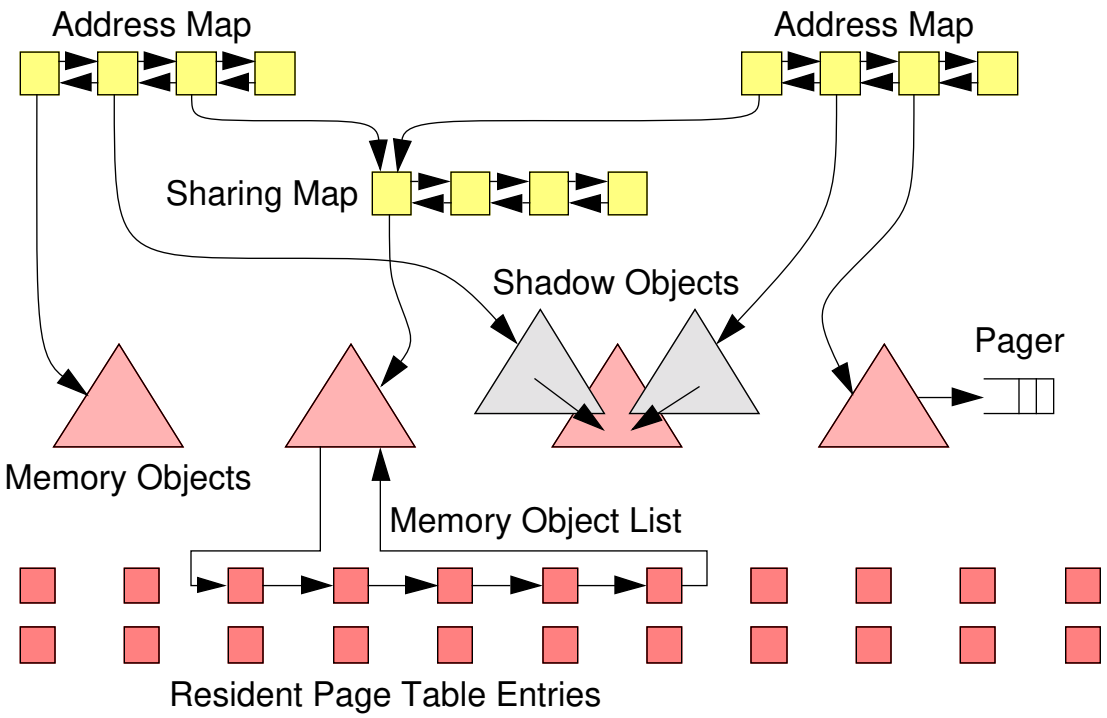
- Datenstrukturen der Adreßraumabbildung: *Address map*



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Datenstrukturen (6)

- Beispielhafter Gesamtüberblick



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Benutzerschnittstelle

- MACH-VM stellt folgende Funktionen zur Verfügung:
  - ◆ virtuelle Speicherbereiche belegen 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 Speicherabbildung anderer Tasks (Debugger!)
  - ◆ Status- und Statistikabfragen

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Benutzerschnittstelle (2)

- Anwendungsbeispiel
  - ◆ *Memory mapped files* für Benutzer und Kern
  - ◆ Realisierung: Memory object mit entsprechendem Pager
  - ◆ Kern-Anwendungen:
    - Programme laden
    - *Shared libraries*
    - Dateisystem
  - ◆ Benutzeranwendungen:
    - neue *Stdio* mit Dateizugriff über *Mapped file*
    - Dateizugriff über Pointer

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Benutzerschnittstelle (3)

- Erfahrungen mit verschiedenen Hardware-Architekturen
  - ◆ Virtuelle Speicherverwaltung von Mach 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 VM auf Multiprozessoren

★ Problem: *Translation Lookaside Buffer (TLB)* -Inkonsistenz

- ◆ TLB = 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 VM auf Multiprozessoren (2)

- ◆ viele heutige Multiprozessorarchitekturen ignorieren dieses 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 VM auf Multiprozessoren (3)

▲ Problem

- ◆ systeminitierte Aktualisierung der pmap-Strukturen überschneidet sich mit der prozessorinitierten Aktualisierung (z.B. Setzen des *Reference bit*)
- ◆ inkonsistente Menge von TLB-Einträgen in der MMU möglich
- ◆ Überschreiben der pmap-Strukturen durch die MMU möglich

■ 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

J.5 Mach in verteilten Systemen

1 Netzwerk-Kommunikation

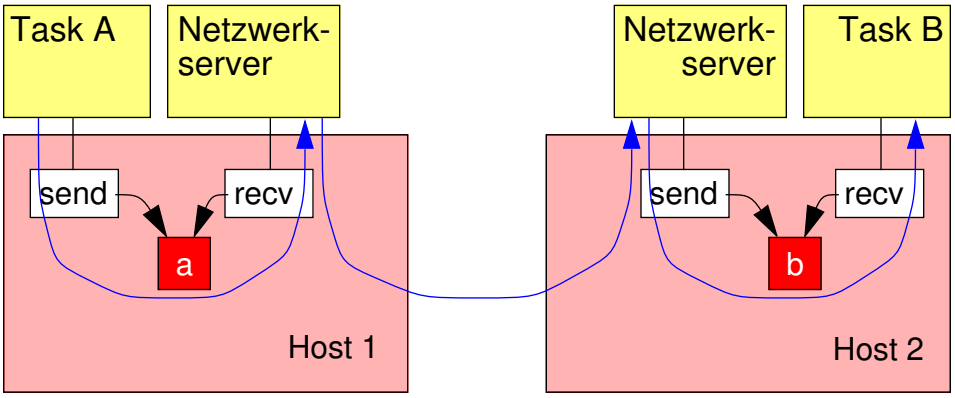
■ Mach IPC nur für lokale Kommunikation konzipiert

- ◆ Nachrichten können durch *Network-Server-Tasks* (im User-level!) an andere Knoten weitergeleitet werden
- ◆ Capability-Funktionalität wird über Netzwerk abgebildet
- ◆ Netzwerktransfer wird geschützt
  - DES-Verschlüsselung der Daten
  - Sequenznummern und Checksummen für Pakete
- ◆ Authentisierung anderer Netzwerk-Server für den Capability-Transfer (*Secret Identifier*)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Netzwerkkommunikation (2)

- Schaubild einer verteilungstransparenten Taskkommunikation



  - Netzwerkserver fungieren als Vermittler
  - sie stellen jeweils lokale Ports bereit und tunneln Kommunikation durch die Ports über ein Netzwerk

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 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 Netzwerkkommunikation erweiterbar

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Distributed Shared Memory (2)

- als *External pager* wird ein sog. ***Shared memory server*** benutzt, der Buch über das Caching der Seiten führt
- *read-only* Seiten können 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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.