MACH

Proseminar Konzepte von Betriebssystemkomponenten

Katrin Giese

katti@andariel.informatik.uni-erlangen.de

12. Juni 2006

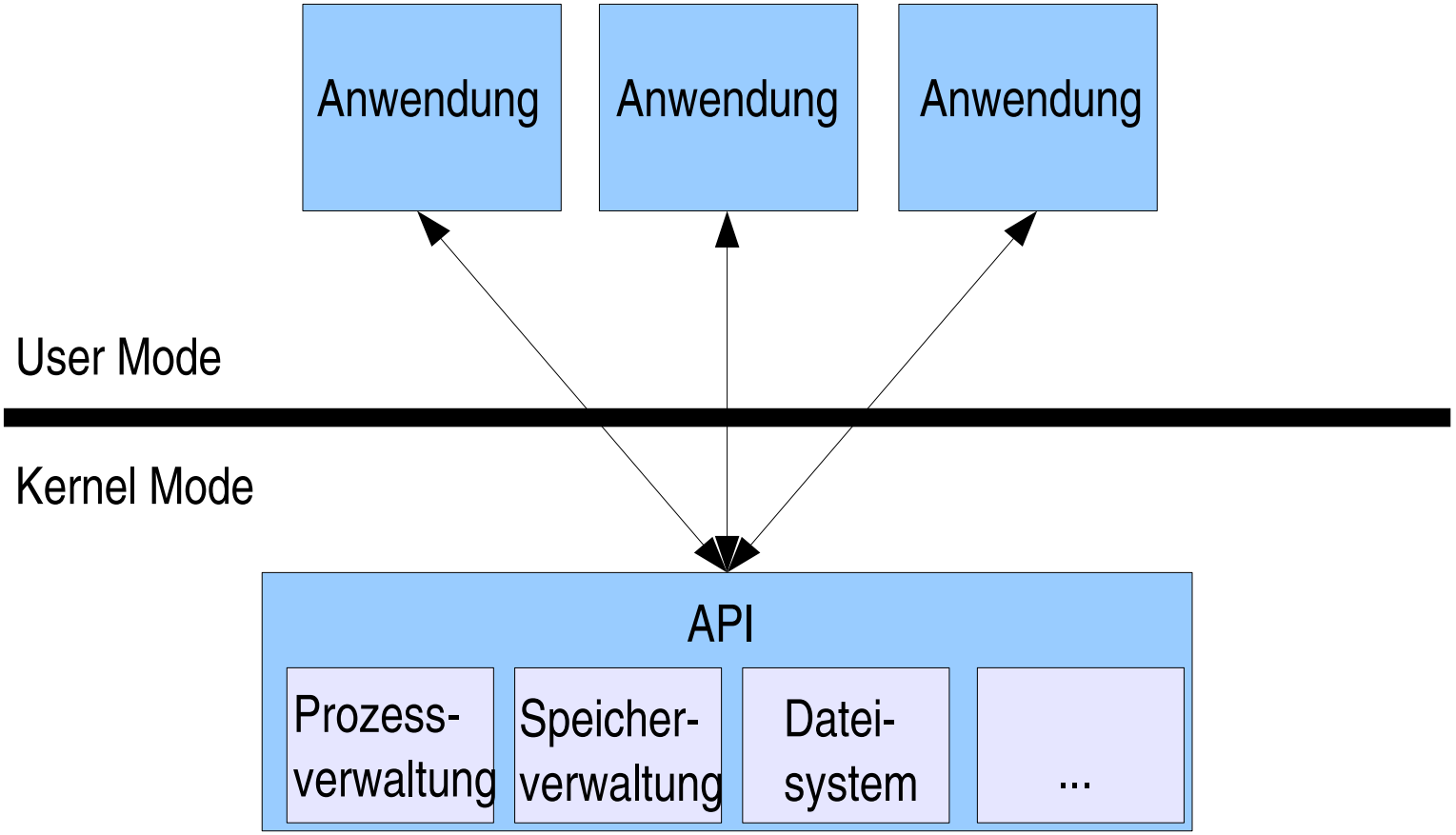
Übersicht

Übersicht

Wozu?

- Entwicklung von monolithischen Kernen, die immer größer und komplexer wurden
- monolithischer Kernel:
  - keine Struktur
  - alle Funktionen von überall aus dem Kernel aufrufbar
  - $\Rightarrow$ kleine Änderungen können zu unbeabsichtigten Nebenwirkungen an anderen (unabhängigen) Stellen führen
  - $\Rightarrow$ stark fehleranfällig (steigt mit erhöhter Komplexität)
  - vom Kernel belegter Speicher wird gehalten (auch wenn unbenutzt)

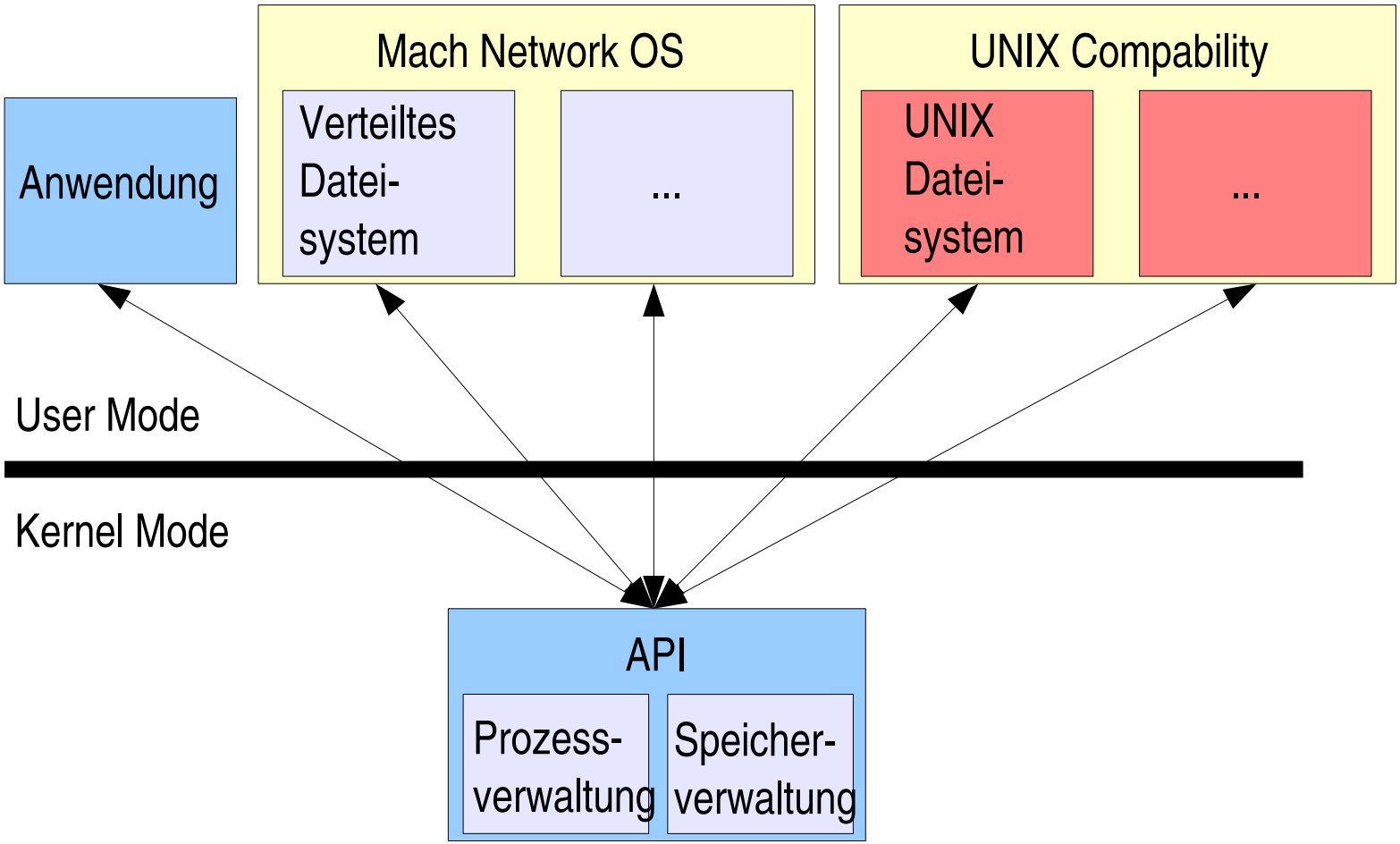
Wozu?


Anwendung
Anwendung
Anwendung
User Mode
Kernel Mode
API
Prozess-
verwaltung
Speicher-
verwaltung
Datei-
system
...

Wozu?

- Entwicklung eines Mikrokernels
- $\Rightarrow$ Reduzierung der Funktionen eines Kernel
- $\Rightarrow$ alle anderen Funktionalitäten von anderen Programmen ausführen
- Beispiel eines Mikrokernel: MACH
- Wurzel von Mach: RIG (Rochester Intelligent Gateway) von Richard Rashid
  - Betriebssystem als Menge kooperierender Prozesse

Wozu?


Mach Network OS
UNIX Compability
Anwendung
Verteiltes Datei-system
...
UNIX Datei-system
...
User Mode
Kernel Mode
API
Prozess-verwaltung
Speicher-verwaltung

Mach-Entstehung

- Rashid's Ziel: Entwicklung eines Nachrichten-orientierten Betriebssystem auf moderner Hardware
- Verwendung von PERQ-Rechnern $\Rightarrow$ Accent, ein Betriebssystem für mikroprogrammierbaren PERQ-Rechner (besser als RIG)
- 1984 Accent auf 150 PERQ-Rechnern $\rightarrow$ schlechter als Unix
- somit Weiterentwicklung von Accent $\rightarrow$ Entstehung von MACH

Mach-Entstehung

- kompatibel zu Unix $\rightarrow$ Verfügbarkeit von Unix-Software
- Verbesserungen:
  - Threads
  - bessere IPC (Interprozesskommunikationsmechanismen)
  - Unterstützung von Mehrprozessorsystemen
  - einfallsreiches virtuelles Speichersystem

Mach-Entstehung

- DARPA (U.S. Department of Defense Advanced Research Projects Agency) suchte Betriebssystem, das Mehrprozessorsysteme unterstützt
  - System muss zu 4.2BSD Unix kompatibel sein
  - Folge: Kern wurde groß→ aber garantierte absolute Kompatibilität
- 1988: MACH 2.5 ist nun ein großes monolithisches System (mit großen Teilen des Berkley UNIX Codes im Kern)
- 1989: Entfernung des Berkley-Codes aus dem Kernel und Verlagerung in den Benutzer-Adressraum → Mikrokernel übrig (bestand aus reinem Mach - Version 3.0)

Ziele

1. Bereitstellen einer Basis zur Entwicklung anderer Betriebssysteme (z.B. UNIX)
2. Unterstützung großer, spärlich bzw. wenig benutzter Adressräume
3. transparenter Zugriff auf Netzwerkbetriebsmittel
4. Ausnutzen von Parallelität sowohl im System als auch in den Anwendungen
5. Übertragbarkeit von Mach auf eine Vielzahl von Maschinen

Übersicht

MACH-Grundlagen

- Mach entwickelt für Emulation von vorhandenen Betriebssytemen
  - Emulation durch Software-Schicht
  - Ausführung außerhalb des Kernels im Benutzermodus
- Idee: Bereitstellung von Mechanismen
  - benötigt für lauffähiges System
  - verlagern von Prozessen zur Benutzerebene

MACH-Grundlagen

- Mikrokernel:
  - Prozessverwaltung
  - Speicherverwaltung
  - I/O-Dienste
  - Interprozesskommunikation
- Benutzeradressraum:
  - Dateien
  - Verzeichnisse
  - andere Funktionen, die von einem Betriebssystem bereitgestellt werden

MACH-Grundlagen

- 5 grundlegende Konzepte (Abstraktionen) stellt Kernel zur Verfügung:
  1. Prozesse
  2. Threads
  3. Ports
  4. Nachrichten
  5. Speicherobjekte

Übersicht

Prozesse

Prozesse

Prozesse

- bestehend aus einem Adressraum, Ports und einer Menge von Threads
- Ports zur Kommunikation mit dem Kernel
- Nutzen der Dienste des Kernels über Senden von Nachrichten an den Prozess-Port (nicht durch Systemaufrufe)
- somit senken der Anzahl der Systemaufrufe auf das Minimum
- Dienste besitzen Stellvertreter-Prozeduren in Programmbibliothek
- Generierung der Prozeduren aus einer Dienstdefinition durch MIG (MACH Interface Generator)

Systemaufrufe der Prozessverwaltung

Aufruf	Beschreibung
Create	erzeugt einen neuen Prozess (erbt gewisse Eigenschaften)
Terminate	zerstört einen angegebenen Prozess
Suspend	inkrementiert den Suspendiert-Zähler
Resume	dekrementiert den Suspendiert-Zähler. Wenn Zähler 0 dann wird Prozess deblockiert
Priority	setzt Priorität für aktuelle und zukünftige Prozesse
Assign	legt Prozess fest, in dem neue Threads ausgeführt werden
Info	liefert Informationen über die Ausführungszeit, Speicherbelegung usw.
Threads	liefert Liste der Threads eines Prozesses

Prozesse

- Emulationsadresse:
  - teilt Kernel mit an welcher Stelle das Emulationspaket im Adressraum des Prozesses platziert ist
  - notwendig zum Emulieren von Unix-Systemaufrufe
  - einmalige Festlegung der Adresse (Weitervererbung der Adresse an Kinder)
- Prozess besitzt nicht:
  - uid, gid, Signalmaske, Wurzelverzeichnis, Array von Dateideskriptoren

Threads

- Darstellung durch Register (enthalten IP, Stackpointer, Zustandsinformation)
- Thread-Zustände: ausführbar, blockiert und rechnend
- nur Kernel hat Kontrolle über Änderungen in Adressräumen $\rightarrow$ keine Inkonsistenz
  passiert über Kernel-thread und/oder Verdrängung
- Thread-Ports für thread-spezifische Kerndienste

Scheduling

- Scheduling-Parameter:
  - Festlegung, welche Threads eines Prozesses auf welchem Prozessor zur Ausführung kommen
- jede Prozessorgruppe verbunden mit Array von Ausführungsschlangen
- ein Array hat 32 Schlangen (korrespondieren mit Thread mit entsprechender Priorität)
- lokale Ausführungsschlange: für Threads, die ständig an CPU gebunden sind
- globale Ausführungsschlange: mit Mutex, Zähler (Gesamtanzahl der Threads), Hinweis (Stelle vom höchstprioren Thread)

Scheduling

- Handoff-Scheduling
  - Thread gibt nach dem Beenden die CPU einem anderen Thread
  - $\Rightarrow$ umgehen der Ausführungsschlangen
  - $\Rightarrow$ wenn klug eingesetzt $\rightarrow$ erhöhte Performance
  - Kernel für Optimierungsmöglichkeiten

Scheduling

- Affinitäts-Scheduling
  - allgemein ausgeschaltet
  - Kernel teilt Thread in der CPU ein, in der Thread schon war in der Hoffnung, dass Teile des Adressraums noch im Cache der CPU eingelagert sind

C Thread-Paket

Aufruf	Beschreibung
fork	erzeugt neuen Thread mit dem selben Code wie der Vater
exit	terminiert den aufrufenden Thread
join	suspendiert den Aufrufer bis ein angegebener Thread terminiert
detach	Ankündigung, dass auf Thread nicht gejoined wird
yield	gibt CPU freiwillig ab
self	liefert Identität des aufrufenden Threads

Kommunikation

Kommunikation

Ports

- Port ist geschützter Briefkasten
- Realisierung der Ports innerhalb des Kernels
- kann geordnete Liste von Nachrichten speichern (Liste nicht in Länge fixiert)
- Capability: Ein Prozess kann anderen Prozessen die Zugriffsrechte auf seinen Ports einräumen
  - jegliche Kommunikation in MACH beruht darauf

Interrupt

- Interrupts sind IPC-Nachrichten
- Nachrichten bestehen nur aus Sender-ID
- Threads stellen Hardware dar (besitzen spezielle Thread-ID)
- Kernel transformiert Interrupts in Nachrichten

Ports

- Port-Typen:
  1. Bootstrap-Ports
     - zur Initialisierung, wenn Prozess startet
  2. Exception-Ports
     - zur Weiterleitung von Fehlermeldungen an Prozessen
  3. registrierte Ports
     - zur Kommunikation von Prozessen mit Standard-System-Servern

IPC

- IPC muss vom Mikrokernel unterstützt werden
- klassische Methode: Nachrichten zwischen Threads über Mikrokernel verschicken
  - Prinzip von Sender-Empfänger
  - Kommunikation unidirektional (entweder lesen oder schreiben vom Port)
- Datenaustausch ist asynchron $\Rightarrow$ Nachrichten müssen *zuverlässig* und *sortiert* sein

IPC

- Nachrichten aufgeteilt in Kopf- und Datenteil
  - Kopfteil: enthält alle Informationen für Transport und Zustellung der Nachricht
  - Größe, Zielport, Antwortport, Funktionscode, Nachrichtenart
  - Datenteil: bestehend aus einem oder mehreren Datenobjekten (Datenbeschreibung und den eigentlichen Datenelementen)
  - Datenbeschreibung: Anzahl der Datenelemente, Größe eines Datenelements in Bits, Typ der Datenelemente
- Empfänger stellt Puffer für Nachricht bereit

RPC

- andere Kommunikationsmöglichkeit:
  - remote procedure call (RPC) → Aufrufen von Funktionen anderer Prozesse über Nachrichten
  - zur Unterstützung: MIG (Mach Interface Generator)
  - MIG erzeugt mit Hilfe einer Definitionsdatei eine Unterroutine
  - Unterroutine schickt RPCs als Nachrichten an anderes Programm
  - danach werden die Nachrichten von Routinen (von MIG erzeugt) ausgewertet und entsprechende Funktion wird aufgerufen

IPC

- Nachrichtenversand über Netzwerk
  - Prinzip von Sender-Empfänger
  - Nachrichtenversand wird von Servern erledigt (nicht über Kernel)
  - Datentypänderungen werden automatisch von Systemen vorgenommen $\Rightarrow$ für Sender und Empfänger irrelevant
  - Kernel stellt Nachricht an den lokalen Nachrichtenserver zu

IPC

- Nachrichtenversand über Netzwerk
  - Nachrichtenserver sucht Zielrechner und Vorbereitung der Nachricht für Versand
  - Vorbereitungen:
    - Verpacken der Nachricht in Netzwerkprotokoll
    - Typumwandlungen
    - Verschlüsseln der zu versendeten Daten
  - Nachrichtenserver ist für Authentifizierung (abhängig vom Nachrichtenserver) des Zielsystems zuständig

IPC

- Nachrichtenversand über Netzwerk
  - wenn Prozess X vom Rechner A über Port Z auf Prozess Y von Rechner B zugreifen will ist Sendcapability nötig
  - da Capabilities nur lokal → *Proxyport* auf Rechner A → Prozess X besitzt Sendecapability und lokaler Nachrichtenserver Receivecapability
  - Nachrichtenserver von Rechner B besitzt Sendecapability auf Zielport Z und Prozess Y die Receivecapability auf dem Port
  - für Zielport Z existiert globale eindeutige Nummer → Netzwerkname
  - somit kann Nachrichtenserver A den Rechner B als Zielrechner bestimmen
  - Nachrichtenserver B nimmt Netzwerknamen entgegen und schickt die Daten als Nachricht an Zielport Z

IPC

- Nachrichtenversand über Netzwerk
  - wenn Prozess X vom Rechner A über Port Z auf Prozess Y von Rechner B zugreifen will ist Sendcapability nötig
  - da Capabilities nur lokal → *Proxyport* auf Rechner A → Prozess X besitzt Sendecapability und lokaler Nachrichtenserver Receivecapability
  - Nachrichtenserver von Rechner B besitzt Sendecapability auf Zielport Z und Prozess Y die Receivecapability auf dem Port
  - für Zielport Z existiert globale eindeutige Nummer → Netzwerkname
  - somit kann Nachrichtenserver A den Rechner B als Zielrechner bestimmen
  - Nachrichtenserver B nimmt Netzwerknamen entgegen und schickt die Daten als Nachricht an Zielport Z

IPC

- Nachrichtenversand über Netzwerk
  - wenn Prozess X vom Rechner A über Port Z auf Prozess Y von Rechner B zugreifen will ist Sendcapability nötig
  - da Capabilities nur lokal → *Proxyport* auf Rechner A → Prozess X besitzt Sendecapability und lokaler Nachrichtenserver Receivecapability
  - Nachrichtenserver von Rechner B besitzt Sendecapability auf Zielport Z und Prozess Y die Receivecapability auf dem Port
  - für Zielport Z existiert globale eindeutige Nummer → Netzwerkname
  - somit kann Nachrichtenserver A den Rechner B als Zielrechner bestimmen
  - Nachrichtenserver B nimmt Netzwerknamen entgegen und schickt die Daten als Nachricht an Zielport Z

IPC

- Nachrichtenversand über Netzwerk
  - wenn Prozess X vom Rechner A über Port Z auf Prozess Y von Rechner B zugreifen will ist Sendcapability nötig
  - da Capabilities nur lokal → *Proxyport* auf Rechner A → Prozess X besitzt Sendecapability und lokaler Nachrichtenserver Receivecapability
  - Nachrichtenserver von Rechner B besitzt Sendecapability auf Zielport Z und Prozess Y die Receivecapability auf dem Port
  - für Zielport Z existiert globale eindeutige Nummer → Netzwerkname
  - somit kann Nachrichtenserver A den Rechner B als Zielrechner bestimmen
  - Nachrichtenserver B nimmt Netzwerknamen entgegen und schickt die Daten als Nachricht an Zielport Z

IPC

- Nachrichtenversand über Netzwerk
  - wenn Prozess X vom Rechner A über Port Z auf Prozess Y von Rechner B zugreifen will ist Sendcapability nötig
  - da Capabilities nur lokal → *Proxyport* auf Rechner A → Prozess X besitzt Sendecapability und lokaler Nachrichtenserver Receivecapability
  - Nachrichtenserver von Rechner B besitzt Sendecapability auf Zielport Z und Prozess Y die Receivecapability auf dem Port
  - für Zielport Z existiert globale eindeutige Nummer → Netzwerkname
  - somit kann Nachrichtenserver A den Rechner B als Zielrechner bestimmen
  - Nachrichtenserver B nimmt Netzwerknamen entgegen und schickt die Daten als Nachricht an Zielport Z

IPC

- Nachrichtenversand über Netzwerk
  - wenn Prozess X vom Rechner A über Port Z auf Prozess Y von Rechner B zugreifen will ist Sendcapability nötig
  - da Capabilities nur lokal → *Proxyport* auf Rechner A → Prozess X besitzt Sendecapability und lokaler Nachrichtenserver Receivecapability
  - Nachrichtenserver von Rechner B besitzt Sendecapability auf Zielport Z und Prozess Y die Receivecapability auf dem Port
  - für Zielport Z existiert globale eindeutige Nummer → Netzwerkname
  - somit kann Nachrichtenserver A den Rechner B als Zielrechner bestimmen
  - Nachrichtenserver B nimmt Netzwerknamen entgegen und schickt die Daten als Nachricht an Zielport Z

IPC

- Nachrichtenversand über Netzwerk
  - wenn Prozess X vom Rechner A über Port Z auf Prozess Y von Rechner B zugreifen will ist Sendcapability nötig
  - da Capabilities nur lokal → *Proxyport* auf Rechner A → Prozess X besitzt Sendecapability und lokaler Nachrichtenserver Receivecapability
  - Nachrichtenserver von Rechner B besitzt Sendecapability auf Zielport Z und Prozess Y die Receivecapability auf dem Port
  - für Zielport Z existiert globale eindeutige Nummer → Netzwerkname
  - somit kann Nachrichtenserver A den Rechner B als Zielrechner bestimmen
  - Nachrichtenserver B nimmt Netzwerknamen entgegen und schickt die Daten als Nachricht an Zielport Z

Speicher

Speicher

Speicherobjekte

- Darstellung einer Datenstruktur → Abbildung im Adressraum eines Prozesses
- Speicherobjekte belegen eine oder mehrere Seiten
- Speicherobjekte enthalten eine Datei oder eine andere spezialisierte Datenstruktur
- Basis für Verwaltung eines virtuellen Speichers in MACH
- bei falschem Zugriff → Seitenfehler
  - Kernel fängt Seitenfehler ab
  - Kernel kann durch eine Nachricht einen Server auf Benutzerebene damit beauftragen, die fehlende Seite einzulagern

Speicherverwaltung

1. pmap-Modul (Ausführung im Kernel): Verwaltung der MMU
2. maschinen-unabhängiger Kernelcode: Verarbeitung der Seitenfehler, Verwaltung der Speicher-Funktion *address map* und Ersetzung der Seiten
3. Speicherverwalter / externer Pager (Ausführung im Benutzeradressraum): Verwaltung des logischen Speichers und Hintergrundspeicher

- Kommunikation zwischen Speicherverwalter und Kern über Protokoll $\rightarrow$ Realisierung eigener Speicherverwalter

Speicherverwaltung

- Vorteile
  - spezielle Seitenersetzungsstrategien für Systeme mit speziellen Anforderungen
  - Verkleinerung und Vereinfachung des Kernels
- Nachteile
  - Speicherverwaltung durch eigene Speicherverwalter kann komplizierter sein
    - Selbstschutz des Kernels vor fehlerhaften oder böswilligen Speicherverwaltern
    - Durchsetzung von wechselseitigem Ausschluss (Gefahr: race condition)

Speicherverwaltung

- Virtueller Speicher
  - großer, linearer Adressraum
  - Einteilung des Speichers in Regionen (*regions*) (Belegung bzw. Freigeben)
  - Belegung
    1. Anfangsadresse und Größe
    2. nur Größe (Anfangsadresse wird zurückgegeben)

Speicherverwaltung

- gültige virtuelle Adressen sind nur in den Regionen vorhanden
- Besonderheit:
  - Speicherobjekte stellen ein Schlüsselkonzept dar
  - Abbildung des Speicherobjekts auf unbenutzte Regionen des virtuellen Adressraums
  - Dateien im Adressraum können mit gewöhnlichen Maschineninstruktionen gelesen und geschrieben werden

MACH-Aufrufe zur virtuellen Speicherverwaltung

Aufruf	Beschreibung
allocate	allokiert eine Region im virtuellen Adressraum
deallocate	invalidiert eine Region im virtuellen Adressraum
map	Einbinden eines Speicherobjekts in virtuellem Adressraum
copy	Kopie einer Region in einem anderem virtuellen Adressraum
inherit	setzen der Vererbungsattribute für eine Region
read	lesen der Daten aus virtuellem Adessraum eines anderen Prozesses
write	schreiben der Daten aus virtuellem Adressraum eines anderen Prozesses

Speicherverwaltung

- gemeinsam benutzter Speicher
  - Threads innerhalb eines Prozesses: sehen gleichen Adressraum
  - gemeinsame Benutzung eines Speicherobjekts
  - im Multiprozessorsystem:
    - Menge kooperierender Prozesse, die parallel auf verschiedenen CPUs zur Ausführung kommen (anstatt quasiparallel/time sharing auf einer CPU)

Speicherverwaltung

- Erzeugung eines Kindprozesses: muss nicht der erzeugende Prozess (Prototyp) sein
- Vererbungsattribute von Regionen:
  1. Region wird im Kindprozess nicht benötigt
  2. Region wird von Prototyp und Kindprozess gemeinsam benutzt
  3. Region im Kindprozess ist Kopie der Region des Prototyps

Speicherverwaltung

- wenn Region des Kindprozesses Kopie der Region des Prototyps ist → copy-on-write
  - Seiten des Kindprozesses nur lesbar
  - wenn Kindprozess schreiben will: Seite rauskopieren und im Adressraum einbinden
  - aber: Verwaltung komplizierter (Unterscheidung zw. Seitenfehler und copy-on-write)

Speicherverwaltung

- Zuordnung jedes Speicherobjektes zu einem externen Speicherverwalter (zur Überwachung)
- Einbindung eines Speicherobjekts in den Adressraum (map)
  - Prozess sendet Nachricht an Speicherverwalter (als Aufforderung)
  - dafür stehen Ports zur Verfügung
    - Objekt-Port, Kontroll-Port, Namens-Port

Speicherverwaltung

1. Objekt-Port
   - Erzeugung vom Speicherverwalter
   - Informationsweitergabe an Kernel über Seitenfehler und andere Ereignisse
2. Kontroll-Port
   - Erzeugung vom Kernel
   - Speicherverwalter sendet Antworten zum Port
3. Namens-Port
   - Erzeugung vom Kernel
   - Objektidentifizierung und Angabe des Bereichs

Speicherverwaltung

- Einlagerung eines Objekts in den Adressraum über Speicherverwalter, der Capability für Objekt-Port liefert
- Kernel erzeugt Kontroll-Port und Namens-Port
- Kernel sendet initiale Nachricht an Objekt-Port (informiert über Namens- und Kontroll-Port)
- Speicherverwalter sendet Nachricht zurück (Auskunft über Objekt-Attribute)
- alle Seiten eines Objekts sind initial nicht lesbar und schreibbar $\rightarrow$ bei Erstbenutzung: *Trap*

Speicherverwaltung - Beispiel

- Thread greift auf Seite eines Objekts zu $\Rightarrow$ Seitenfehler $\Rightarrow$ *Trap*
- Kernel sendet Nachricht an Speicherverwalter über Objekt-Port
- Nachricht ist Mitteilung über zuzugreifende Seite und Aufforderung, die Seite bereitzustellen
- Speicherverwalter überprüft, ob Zugriff rechtmässig ist
  - nein: Speicherverwalter schickt Fehlermeldung zurück
  - ja: Speicherverwalter beschafft Seite und fügt sie in den Adressraum ein (Kernel bekommt Zeiger auf Seite)
- wenn Seiteneinlagerung erfolgreich blendet Kernel die Seite in den Adressraum des Threads ein

Speicherverwaltung

- Überprüfung von genügend freien Seitenrahmen über Erweckung von Kern-Thread (*Paging-Dämon*)
- wenn nicht genügend: alte, veränderte („*dirty*“) Seiten zum Speicherverwalter des Objekts gesendet
- Speicherverwalter schreibt Seiten auf Platte und teilt Beendigung mit
- Ersetzungsalgorithmus: second chance

Übersicht

Vor- und Nachteile

- Vorteile
  - fast alles im Benutzeradreßraum
  - $\Rightarrow$ nur Mikrokernel ist Maschinenabhängig
  - $\Rightarrow$ extreme Flexibilität
    - möglich: Hinzufügen von Operationen für Echtzeitbetrieb
    - verschiedene Betriebssysteme laufen gleichzeitig
  - Schutz zwischen Prozessen
- Nachteile
  - mehr Umschaltung zwischen Kernel- und Userspace
  - mehr Threadwechsel (damit Adreßraumwechsel)

MACH in existierenden Systemen

- bekannteste Serversammlungen: HURD und Darwin
- HURD mit MACH $\rightarrow$ Kern des GNU-Systems (nicht stabil genug für Einsatz auf Produktivsystemen)
- Darwin ist kompatibel zu 4.4BSD und Kern von MacOS X
- ebenso wird Mach in MIKE und Sprite verwendet

Danke für eure Aufmerksamkeit

Noch Fragen?


M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian and M. Young. Mach: A New Kernel Foundation for UNIX Development. USENIX Association, June 1986.


A. Tanenbaum. Moderne Betriebssysteme. Hanser Verlag. 1994


http://www.usenix.org/publications/library/proceedings/mach3/bernadat.html


http://www.usenix.org/publications/library/proceedings/mach3/castro.html


http://www.usenix.org/publications/library/proceedings/mach3/kupfer.html