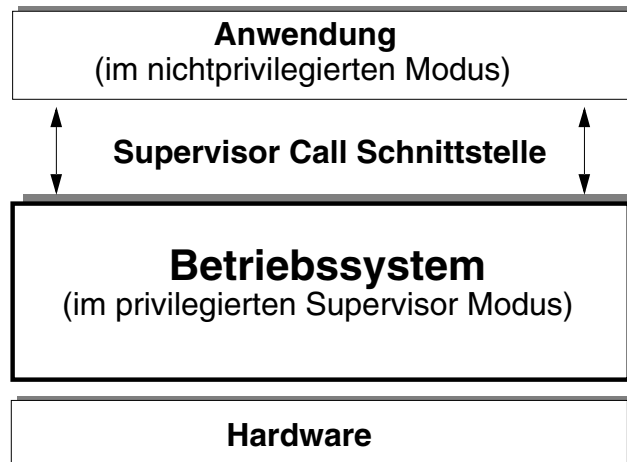


Dienste in Betriebssystemen (1)

- Struktur eines *klassischen Betriebssystems*

- monolithischer Aufbau

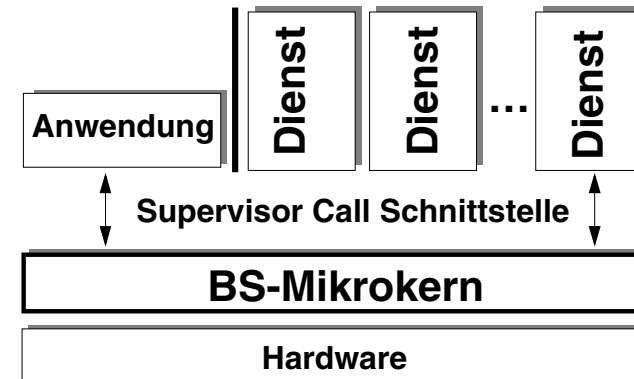


Zusätzlich:

- Dienstprogramme
- Libraries

Dienste in Betriebssystemen (2)

- Client/Server-orientierte, mikrokernbasierte Systeme:



- Funktionale Kapselung in "autonome" Dienstmoduln

- z.B. Dateiverwaltung, Prozessmanagement, Kommunikation, Speichermanagement (paging), Schutz / Sicherheit (Authentisierung)...
- klare, nachrichtenorientierte Schnittstelle zwischen den Diensten

- Mikrokern: nur noch minimale Basisfunktionalität

- low-level I/O, basic memory management, basic interrupt handling,...

- Vorteile gegenüber monolithischer Struktur

- bessere Wartbarkeit, Weiterentwickelbarkeit, Anpassbarkeit
- potentiell bessere Fehlertoleranz (Ausfall eines einzelnen Dienstes gefährdet nicht unbedingt andere Dienste; redundante Dienste...)
- im Prinzip sehr einfach verteilbar (kein wesentlicher Unterschied zwischen zentralistischer und verteilter Architektur)

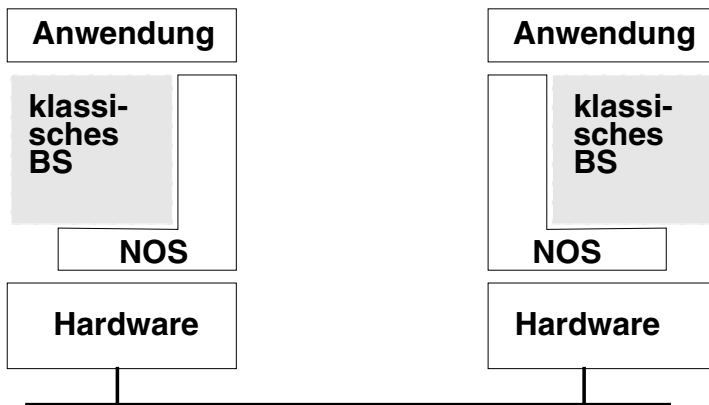
- Potentielle Nachteile

- Effizienzminderung (insbes. bei Verteilung)

- *Aufgaben:*
 - Verwaltung der Betriebsmittel
 - Prozessor, Speicher, Uhr, E/A-Geräte, Dateien...
 - Virtuelle Maschine für Anwendungen
 - Virtualisierung der Hardware
 - Abschottung "paralleler" Anwendungen
- Einzelne BS-Funktionen nicht isolierbar / austauschbar
- Ergänzung neuer BS-Dienste sehr aufwendig
- BS-Schicht sehr „dick“

Netzwerk-Betriebssysteme

- Historisch: Schritt zum verteilten Betriebssystem mittels Network Operating System (NOS)
 - "Kapselung" klassischer, nicht netzwerkfähiger Betriebssysteme

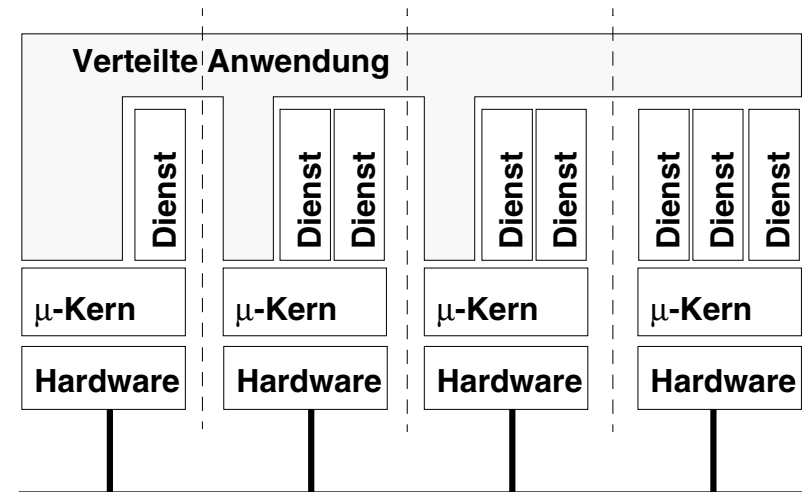


- NOS steuert Netzwerk und Netzwerkzugang, benutzt dazu u.a. Dienste des lokalen Betriebssystems
 - jeder Rechner besitzt weiterhin Kontrolle über seine Betriebsmittel
 - keine unmittelbare, gemeinsame Nutzung von Betriebsmitteln
- NOS stellt Anwendungen weitere Dienste bereit
 - file sharing, file transfer
 - Zugriffsschutz (Benutzergruppen...)
 - Namensverwaltung
 - Netzverwaltung
 - remote login
 - E-mail

War z.B. in UNIX als klassisches Mehrbenutzersystem bereits vorhanden!

Verteilte Betriebssysteme

- Wesentliches Charakteristikum: Ortstransparenz
 - einheitliche Systemsicht für Benutzer (Anwendung; Clients)
- Menge der Dienstleistungserbringer ist räumlich verteilt
 - minimale Funktionalität pro Rechner: Mikrokern, Kommunikationsdienst



- Dienste in ihrer Gesamtheit erbringen Leistung eines einzigen Betriebssystems (--> "virtueller Monorechner")
 - Prozesse auf einem Rechner können ("ortstransparent") einen entfernten Service nutzen, der lokal nicht angeboten wird
- Ggf. spezielle Client-Rechner: keine oder wenig globale Dienste; ggf. eingeschränkter Betriebssystemkern
- Realisierung verteilter Betriebssysteme nicht trivial
 - z.B. fehlende unmittelbare globale Sicht

Aufgaben eines verteilten Betriebssystems

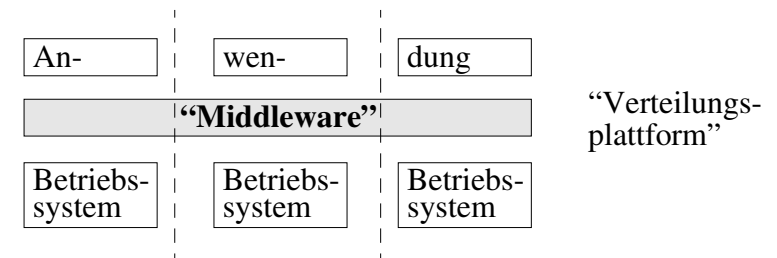
- Klassische BS-Aufgaben
- Herstellung von weitgehender Ortstransparenz
- Lokalisierung von Objekten; Namensdienst
- Anbieten von gegen Fehler gesicherter Kommunikation
- Zugriffsschutz

Und ggf. auch noch auf dem Wunschzettel:

- Uhrensynchronisation
- Lastverteilung
- Prozessmigration
- Konsistenzsicherung replizierter Daten
- Implementierung von "virtual shared memory"
- ...

Middleware

- Kann man durch eine geeignete Softwareinfrastruktur die Realisierung verteilter Anwendungen vereinfachen?
 - wieso ist das überhaupt so schwierig?
 - kann man für viele Anwendungen gemeinsame Aspekte herausfaktorisieren?
- Lösung: Zauberwort "Middleware"



- Aufgabe:
 - Verteilung (für die Anwendung) möglichst transparent machen (z.B. umspannender Namensraum, globale Zugreifbarkeit, Ortstransparenz)
 - zumindest aber die Verteilung einfach handhabbar machen
- Soll insbesondere Kommunikation und Kooperation zwischen Anwendungsprogrammen unterstützen
 - Verbergen von Heterogenität von Rechnern und Betriebssystemen (z.B. durch einheitliche Datenformate)
 - einheitliche „Umgangsformen“: Schnittstellen, Protokolle
- Sollte gewisse Basismechanismen für verteiltes Programmieren anbieten, z.B.
 - Verzeichnis- und Suchdienste (Nameservice, Tradingservices...)
 - automatische Schnittstellenanpassung (Schnittstellenbeschreibungssprache, Stub-Compiler...)

Der Weg zum „Netzwerkrechner“

1. RPC-Pakete: z.B. Sun-RPC

- Client-Server-Paradigma, RPC-Kommunikation
- Schnittstellen-Beschreibungssprache, Datenformatkonversion, Stubgeneratoren
- Sicherheitskonzepte (Authentifizierung, Autorisierung, Verschlüsselung)

2. Client-Server-Verteilungsplattformen: z.B. DCE

- Zeitdienst, Verzeichnis- und Suchdienst
- globaler Namensraum, globales Dateisystem
- Programmierhilfen: Synchronisation, Multithreading ...

3. Objektbasierte Verteilungsplattformen: z.B. CORBA

- Kooperation zwischen gleichberechtigten („peer-to-peer“-) Objekten
- objektorientierte Schnittstellenbeschreibungssprache, Vererbung
- Objekt Request Broker

4. Infrastruktur für spontane Kooperation (z.B. Jini)

- unterstützt Dienstorientierung, Mobilität, Dynamik

Beachte: Der Begriff “Middleware” ist leider im Laufe der Zeit zunehmend verwässert worden

- oft weniger gebraucht im technischen Sinne als Verteilungsplattform und Kommunikations- und Dienstinfrastruktur
- sondern “alles” was nicht gerade Anwendung oder Betriebssystem ist, also auch Datenbanken, Workflow,...