Zu zeichnen: Aufbau des OS, wartend/ ausführend Schaltung
Was soll ein Betriebssystem leisten?
ist Schnittstelle zwischen Anwendungen und Hardware
Erweiterung / Veredelung der Hardware
Hardware muß billig, schnell und zuverlässig sein
BS stellt komplexe Funktionen bereit, die die Anwendungsprogramme verwenden können
Abstraktion der Hardware
Rechner sind trotzähnlicher Architektur im Detail sehr unterschiedlich
Fallunterscheidung wird vom BS vorgenommen
BS realisiert einheitliche Sicht für Anwendungen (kein Unterschied ob CD Laufwerk, Diskette,Externspeicher, Netzlaufwerk,….)
Verwaltung von Betriebsmitteln
Fairness: .gerechte. Verteilung der Betriebsmittel
Sicherheit: Schutz der Anwendungen und Benutzer voreinander
Anwendungen können nicht direkt (d.h. *unkontolliert*) auf die Hardware zugreifen
Mechanismen:
Betriebsmodi des Prozessors (System- / Benutzermodus)
Systemaufrufe, Ausnahmen, Unterbrechungen
Adreßumsetzung (durch spezielle Hardware,

Was ist ein Prozeß? Prozeß = Programm, das sich in Ausführung beendet
Woraus besteht ein Prozess? (Prozeßkontext): Programmcode (im Hauptspeicher)/ Daten im Hauptspeicher/ Eingabe, (Zwischen-)Ergebnisse, Keller, .../ Daten in Prozessorregistern/ Befehlszähler, Rechenwerksregister, Statusregister, .../ Information über genutzte Ressourcen (im BS)/ z.B. geöffnete Dateien, Netzwerkverbindungen/
Gründe für den Mehrprozeßbetrieb: bessere Auslastung des Rechners (Warten auf E/A!)/ Abarbeitung von Aufgaben .im Hintergrund./ gleichzeitige Nutzung mehrerer Anwendungen/ gleichzeitige Bedienung mehrerer Benutzer

Einschub: Leichtgewichtige Prozesse (*Threads*):Jeder Prozeß hat seine eigenen, privaten Ressourcen/ Prozessorzustand, Speicher, offene Dateien, .../ Das Umschalten von einem Prozeß zum nächsten ist relativ aufwendig/ In vielen BSen daher auch **Threads**:/ Prozeß weiter unterteilt in (quasi-)parallele Abläufe/ jeder Thread hat seinen eigenen Prozessorzustand (Register, einschließlich Befehlszähler)/ alle anderen Ressourcen (Speicher, offene Dateien, ...) sind allen/ Threads (desselben Prozesses) gemeinsam/ das Umschalten zwischen Threads (desselben Prozesses) geht daher sehr schnell

Prozeßtabelle:rechnend (jeweils nur ein Prozeß)/ bereit/ wartend (BS weiß, auf welches Ereignis der Prozeß wartet)
Tabelle oder Liste von Objekten (**Prozeßdeskriptoren**)
Prozeßdeskriptor wird ausschließlich vom BS verwaltet
Inhalt eines Prozeßdeskriptors:
Verwaltungsinformation: Identi_kation des Prozesses, Priorität, Benutzeridenti_kation, Rechte, ...
Zustand des Prozesses (rechnend, bereit, ...)
Inhalt der Prozessorregister (Befehlszähler, ...)/ benutzte Speicherbereiche/ benutzte Systemressourcen (offene Dateien, ...)/ Mit den letzten drei Informationen (Prozeßkontext) wird Prozeß in Zustand .rechnend. gebracht

Ende von Prozessen ...
Prozeß endet i.a. freiwillig (z.B. System.exit(0))
Manchmal Terminierung des Prozesses von außen
durch Betriebssystem
z.B. bei Ausnahmen, unzulässigen Zugriffen, ...
durch andere Prozesse
z.B.: UNIX-Kommando kill, Windows Taskmanager
BS arbeitet als Vermittler (Prüfung der Rechte!)
i.a. noch letzte Reaktion des Prozesses gestattet
z.B. für saubere Freigabe von Betriebsmitteln
z.B. konsistentes Beenden einer Datei- ¨ Anderung
für .Notfälle. aber auch zwangsweise Terminierung
Aktivierung eines Prozesses ...
Am Ende jeder BS-Aktivität: Ansprung des **Schedulers**
Auswahl eines Prozesses, der .bereit. ist
Zustand des Prozesses auf .rechnend. setzen
Prozeßkontext wiederherstellen, insbesondere
Prozessorregister mit Werten aus Prozeßdeskriptor
laden
auch Programmzähler wird dabei geändert
Umschalten in Benutzermodus
Kontrolle geht an den ausgewählten Prozeß über
Beachte: während einer BS-Aktivität kann (in einem Einprozessorsystem)
kein Prozeß im Zustand rechnend sein
Deaktivierung eines Prozesses ...
å Durch Systemaufruf (insbes. Zugriff auf E/A-Gerät)
å Prozeß gibt Kontrolle (d.h. Prozessor) freiwillig ab
å Interne Unterbrechungen (Ausnahmen)
å Prozeß verursacht einen Fehler (z.B. unzulässiger Zugriff)
å Kontrolle wird zwangsweise entzogen
å Externe Unterbrechungen
å z.B. Fertigmeldung eines E/A-Geräts
å Kontrolle wird vorübergehend entzogen (BS muß die Unterbrechung behandeln)
Ablauf eines Systemaufrufs
å Z.B. fork(), E/A-Funktionen, exit(), ...
å Durch Hardware: Einsprung ins BS (Systemmodus)
å Ablauf im BS:
å Sichern des Prozeßkontexts (insbes. Registerinhalte)
å Prozeß in Zustand .bereit. setzen
å Ausführung bzw. Initiierung des Auftrags
å evtl. Prozeß in Zustand .blockiert. setzen
å Sprung zum Scheduler
å Aktivieren eines (anderen) Prozesses
Behandlung von internen Unterbrechungen (Ausnahmen)
å Z.B. privilegierter Befehl, arithmetischer Fehler, Seitenfehler
(+ **4.3**)
å Durch Hardware: Einsprung ins BS (Systemmodus)
å Ablauf im BS:

å Sichern des Prozeßkontexts (insbes. Registerinhalte)
å Je nach Art der Ausnahme:
å Beenden des Prozesses
å Blockieren des Prozesses (bei Seitenfehler)
å Behebung der Ursache der Ausnahme
å Sprung zum Scheduler

Scheduler
å Teil des BSs, der entscheidet,
å wann ein Prozeß von .rechnend. nach .bereit. muß
å wann ein Prozeß von .bereit. nach .rechnend. darf
å Speichert / nutzt Prozeßinformation im Prozeßdeskriptor
å Wird immer aufgerufen, wenn das BS die Kontrolle an Prozesse abgibt
å Wichtig: gute Scheduling-Strategie
å d.h.: wie trifft der Scheduler die Entscheidung?

Ziele aus Benutzersicht
å Minimierung der Bearbeitungszeit
å Prozeß erhält hohen Anteil der Prozessorleistung
å Minimierung der Antwortzeit (Latenz)
å Prozeß wird nicht zu lange blockiert
å wichtig für interaktives Arbeiten
å Vorhersehbarkeit (Realzeit-Anforderungen)
å garantierte Rechen- / Antwortzeit
å z.B. Multimedia: Mindestleistung notwendig
å z.B. Notabschaltung AKW
å benötigt Prioritäten: garantierte Prozessorzeit, egal wieviele Prozesse sonst noch da sind

Ziele aus Betreiber-/Systemsicht
å Optimierung der Prozessorauslastung
å Prozessor sollte immer nutzvolle Arbeit tun
å BS (Scheduler) sollte Prozessor möglichst wenig nutzen
å Optimierung der Geräteauslastung
å bevorzugte Behandlung von Prozessen, die wenig ausgelastete Geräte nutzen
å Fairness
å jeder Benutzer sollte gleich behandelt werden
å Einfache Realisierung des Schedulers
å Zuverlässigkeit, Ressourcennutzung

Nicht präemptive Verfahren:Einem laufenden Prozeß kann der Prozessor nicht entzogen werden
å First Come First Served (FCFS)
å Shortest Job First
å **Präemptive Verfahren:** Einem laufenden Prozeß kann der Prozessor entzogenwerden
å Round Robin (RR) (wie FCFS)
å Static Priority Scheduling (Zuteilung immer an Prozess(e) mit der höchsten Priorität)
å Dynamic Priority Scheduling (Verbindung von Fairness und Realzeit-Tauglichkeit)

Fragen / Probleme f ¨ ur das BS
å Wie werden die Speicherbereiche an Prozesse zugeteilt?

å BS muß genügend großes Stück Speicher _nden
å was ist, wenn laufender Prozeß mehr Speicher anfordert?
å Programme enthalten Adressen
å kann man Programme an beliebige Adressen schieben?
å Schutz der Prozesse gegeneinander
å wie kann man verhindern, daß ein Prozeß auf den Speicher eines anderen Prozesses (oder des BSs) zugreift?
å Begrenzte Größe des Hauptspeichers
å was ist, wenn nicht alle Prozesse in den Hauptspeicher passen?

Fragen / Probleme f ¨ ur das BS ...

å Transparenz für die Anwendungen
å im Idealfall sollte die Anwendung annehmen dürfen, sie hätte den Rechner für sich alleine
å Programmieraufwand sollte sich durch Speicherverwaltung nicht erhöhen

Wie ist der Adreßraum eines Prozesses strukturiert?

å Wichtige Unterscheidung / Begriffe
å **Adreßraum**: Menge aller verwendbaren Adressen
å **logischer (virtueller) Adreßraum**: aus Sicht eines Programms / Prozesses gesehen
å die Adressen, die der Prozeß verwendet / verwenden kann
å **physischer Adreßraum**: aus Sicht der Hardware
å die Adressen, die die Speicher-Hardware verwendet / verwenden kann
å nicht immer identisch! (siehe später: *Paging*)
å **Speicher**: das Stück Hardware, das Daten speichert ...

Ein Prozeß enthält: Code/ Daten/ Keller/ getrennt von anderen Daten verwaltet!/ Jeweils in getrennten Segmenten abgelegt/ **Segment** = logisch zusammengehöriger Teil des Adreßraums

Der Binder (*Linker*)

å Größere Programme bestehen in der Regel aus mehreren **Modulen** (Paketen, Bibliotheken)
å Module werden getrennt entwicklet und übersetzt
å jedes Moduls besteht aus Codesegment und initialisierten Teilen des Datensegements
å Binder hängt die Segmente der Module hintereinander
å Ergebnis: Ausführbare Datei/ Probleme:
å Code der Module muß im Adreßraum verschoben werden
å Unterprogrammaufruf (Prozedur / Methode) benötigt als Operanden die Adresse des gerufenen Unterprogramms
å die Adresse ist vor dem Binden noch unbekannt!

Der Binder: Lösung der Probleme

å Binder muß Code im Speicher verschieben (*relocate*)
å ein Modul enthält eine Liste aller Befehle, die Adressen als Operanden besitzen
å der Binder paßt diese Adressen entsprechend an
å Der Binder setzt nach dem Verschieben die Adressen für Aufrufe von Unterprogrammen in externen Modulen in den Programmcode ein (**Au_ösung externer Referenzen**)
å Unterprogramme werden durch Namen identi_ziert
å jedes Modul enthält eine Liste mit der Zuordnung von Namen zu Adressen

Der Lader (*Loader*)

å Lädt ausführbare Datei in den Hautpspeicher
å kann Code / Daten verschieben, falls nötig

å reserviert Platz für nicht-initialisierte Daten und Keller

Dynamischer Binder

å Ermöglicht das Einbinden von Modulen zur *Laufzeit*
å Funktionsweiseähnlich dem normalen Binder
å Aufruf entweder beim Start des Programms oder erst beim Aufruf eines Unterprogramms in einem neuen Modul
å Vorteil: dynamisches Binden ermöglicht die gemeinsame Nutzung von Modulen durch mehrere Prozesse

Swapping:

- Idee: Falls Prozeß längere Zeit blockiert ist, wird sein Speicherbereich (***Image***) auf Externspeicher ausgelagert Keine wirklich brauchbare Lösung!

å Probleme / Nachteile:
å das Ein-/Auslagern des gesamten Adreßraums dauert sehr lange
å der Prozeß benötigt aber evtl. gar nicht alles
å *Image* muß beim Einlagern evtl. verschoben werden
å was ist, wenn Prozeß zur Laufzeit mehr Speicher braucht?
å was ist, wenn das *Image* größer als der Hauptspeicher ist?
å kein Speicherschutz!

Virtueller Speicher (*Paging*)

å Grundlage: strikte Trennung zwischen
å logischen Adressen (die der Prozeß sieht / benutzt)
å physischen Adressen (die der Hauptspeicher sieht)
å Idee: *bei jedem Speicherzugriff* wird die vom Prozeß erzeugte logische Adressen auf eine physische Adresse abgebildet
å muß durch Hardware erfolgen, da sonst viel zu langsam
å Vorteile:
å kein Verschieben beim Laden eines Prozesses erforderlich
å auch kleine freie Speicherbereiche sind nutzbar
å Speicherschutz ergibt sich (fast) automatisch
å ermöglicht, auch Teile des Adreßraums auzuslagern

***Paging*: Seitenbasierte Speicherabbildung**

å Abbildung von virtuellen auf physische Adressen erfolgt über Tabelle
å Tabelle wird vom BS erstellt und aktualisiert
å Problem: Tabelle wäre bei byteweiser Abbildung viel zu groß!
å ein Eintrag für jede Adresse im virtuellen Adreßraum!
å Daher: Abbildung erfolgt für Speicherblöcke fester Größe
å **Seite** (*page*): Block im virtuellen Adreßraum
å **Kachel** (*page frame*): Block im physischen Adreßraum
å Typische Seitengröße: 4 KByte

***Paging*: Seiten/Kachel-Tabelle**

å Jeder Seite muß eine Kachel zugeordnet werden
å Problem: Speicherplatzbedarf der Tabelle
å z.B. bei virtuellen Adressen mit 32 Bit Länge: 220 Seiten
å für jeden Prozeß wäre 4 MByte große Tabelle nötig
å bei 64-Bit Prozessoren sogar 252 Seiten, d.h. 9000 TByte!
å Aber: Prozeß nutzt virtuellen Adreßraum i.a. nicht vollständig
å daher: mehrstu_ge Tabellen sinnvoll
å z.B. zweistu_ge Tabelle beim x86 (32-Bit Adressen)

å 1. Stufe: *page directory*, 1 Kachel, 1024 Einträge
å 2. Stufe: _ 4096 *page tables*, je 1 Kachel mit 1024 Einträgen
Paging*: einige Anmerkungen:** Adreßabbildung ist ein Zusammenspiel von BS und Hardware/ BS setzt für jeden Prozeß Seiten/Kacheltabelle auf/ Tabelle wird bei Prozeßwechsel ausgetauscht/ Hardware (**MMU**, ***Memory Management Unit) macht die Adreßumsetzung
å falls für eine Seite keine Kachel im Hauptspeicher vorhanden ist: MMU erzeugt Ausnahme (**Seitenfehler**): Speicherschutz ist gegeben, da Seiten/Kacheltabelle nur auf Kacheln verweist, die dem Prozeß zugeteilt sind/ MMU enthält ein Cache für Adreß- ¨ Übersetzungen/ sonst wären für jeden Speicherzugriff eines Prozesses mehrere Speicherzugriffe der MMU notwendig!
Working Set: zu jeder Zeit arbeitet ein Prozeß nur mit einem Teil seines Adreßraums (räumliche und zeitliche Lokalität), nur das *Working Set* muß im physischen Speicher vorhanden sein
***Paging*: Vollständige Speicherhierarchie: Verwaltung/ Art des Speichers/ Datentransport (zur Schicht nach „unten“):**
Betriebssystem/ Hintergrundspeicher/ automatisch durch BS
BS + Hardware (MMU)/ Hauptspeicher/ automatisch durch Hardware
Hardware/ L2 – Cache/ automatisch durch Hardware
Hardware/ L1 – Cache/ "manuell" durch Anwendung
Anwendungsprogramm/ Prozessorregister
Beim Swapping sind die Daten eines Prozess entweder vollständig ausgelagert oder vollständig im Hauptspeicher enthalten. Diese Eigenschaft unterscheidet das Swapping vom Paging, bei dem nur einzelne Speicherseiten aus- und eingelagert werden.

Geräte-Treiber

å Ziel: BS-Kern soll unabhängig von den E/A-Geräten sein
å Daher: Zugriff auf E/A-Geräte erfolgt über einheitliche Schnittstelle:
å u.a.: Lesen, Schreiben, Positionieren, Unterbrechungsbehandlung
å Geräte-Treiber realisieren diese Schnittstelle für ein bestimmtes Gerät
å in der Regel vom Gerätehersteller bereitgestellt
å werden beim Start des BSs oder sogar zur Laufzeit in das BS eingebunden

Objekte in Betriebssystemen

å BS verwaltet eine Vielzahl von Objekten (Betriebsmitteln), z.B.
å Prozesse, Dateien, Geräte, Kommunikationskanäle, ...
å Da viele Funktionen für verschiedene Objekteähnlich sind:
å oft Nutzung objektorientierter Konzepte
å Zugriff auf Objekte nur über Referenzen (***H*andles**) und Systemaufrufe
å aber i.a. keine Nutzung objektorientierter Sprachen
å Im folgenden betrachtet:
å allgemeine Aspekte der Betriebsmittelverwaltung
å d.h. gemeinsame Methoden der Objekte

Zugriffsmatrix

å legt für jedes Paar (Benutzer, Objekt) die erlaubten Operationen fest
å Einträge erfolgen i.d.R. durch Eigentümer der Objekte
In der Praxis zwei Varianten zur Speicherung der Matrix:
å ***Access Control List*** (ACL): Rechte aller Benutzer an einem Objekt, zusammen mit Objekt gespeichert
å ***Capability***: alle Rechte eines Benutzers (bzw. Prozesses)
Verklemmungen: Das Warten auf Freiwerden von Betriebsmitteln kann zu Problemen führen, wenn der Prozeß bereits andere Betriebsmittel belegt hat/ Beispiel: Prozeß A hat den Drucker

belegt/ Prozeß B hat Datei xyz gesperrt und will drucken, wartet also auf den Drucker/ Prozeß A stellt während des Druckens fest, daß er auf Datei xyz zugreifen muß/ Datei ist aber durch B gesperrt) Prozeß A wartet/ Beide Prozesse warten nun ewig: **Verklemmung (*Deadlock*)**
Voraussetzungen f ¨ ur Verklemmungen: Wechselseitiger Ausschluß/ Betriebsmittel (BM) sind exklusiv; können nur von jeweils einem Prozeß benutzt werden/ Belegung und Warten/ ein Prozeß besitzt BM, während er auf andere wartet/ Kein Entzug von Betriebsmitteln/ einem Prozeß kann ein BM nicht weggenommen werden/ Zyklisches Warten/ es gibt eine geschlossene Kette von Prozessen, so daß jeder mindestens ein BM hat, auf das der Nachfolger in der Kette wartet
Behandlung von Verklemmungen
å Verklemmungserkennung und -au_ösung
å Au_ösung durch Abbruch bzw. Rücksetzen von Prozessen
å Rücksetzen in üblichen BSen nicht möglich
å Verklemmungsvermeidung
å BM nur zuteilen, wenn es dadurch in keinem Fall zu einer Verklemmung kommen kann
å benötigt zusätzliches Wissen; meist nicht praktikabel
å Verklemmungsvorbeugung
å durch Design dafür gesorgt, daß eine der vier Voraussetzungen für Verklemmungen nicht erfüllt werden kann
å z.B. Einsatz eines Drucker-Spoolers: Drucker ist kein exklusives BM mehr
Semaphore sind vom BS bereitgestellte Objekte mit zwei Methoden:/ P(): Prüfen und Belegen des Semaphors/ falls Semaphor bereits belegt:/ Blockieren des Prozesses/ Eintrag des Prozesses in Warteschlange/ sonst: Semaphor belegen/ wichtig: Prüfung und Belegung erfolgt unteilbar/ V(): Freigeben des Semaphors/ falls wartende Prozesse in Warteschlage:/ einen davon aufwecken
Wechselseitiger Ausschluß mit Semaphoren: Semaphore s = ...; // zum BM gehöriges Semaphor____s.P(); // Prozeß wird evtl. blockiert___ . . ; // Benutze Betriebsmittel____s.V();
å Erweiterung: allgemeine Semaphore/ besitzen einen Zähler statt des .belegt.-Flags/ P() zählt herunter und blockiert, falls Zähler < 0/ V() zählt hoch, weckt blockierten Prozeß, falls Zähler _ 0/ können auch für allgemeinere Synchronisationsaufgaben genutzt werden, z.B. Auftrags-Warteschlangen/ V() beim Eintragen, P() beim Bearbeiten
Erfolgt über Objekte des BSs: ***Message Exchange, Mailbox:*** *Mailbox* enthält zwei Warteschlangen:/ für noch nicht abgeholte Nachrichten/ für Prozesse, die auf Nachrichten warten/ *Mailbox* bietet Methoden zum/ Eintragen von Nachrichten (*send*)/ Nachricht wird in Warteschlange eingereiht/ falls wartende Prozesse vorhanden: wecke den ersten/ Abholen von Nachrichten (*receive*)/ falls keine Nachricht in der Nachrichtenwarteschlange:/ Prozeß wird blockiert/ Prozeß wird in Prozeßwarteschlange eingereiht

Arten von Netzwerken: Nach geographischer Ausdehnung/ **LAN**: *Local Area Network*/ innerhalb eines Gebäudekomplexes, z.B. Ethernet/ **MAN**: *Metropolitan Area Network*/ **WAN**: *Wide Area Network*/ länder- bzw. weltumspannend, z.B. Internet/ Einsatz jeweils unterschiedlicher Technologien
Arten von Netzwerken ...: Nach Art der ¨ Ubertragung bzw. Vermittlung:
Leitungsvermittlung (*circuit switching*)/ für zwei Kommunikationspartner wird eine dedizierte Verbindung hergestellt/ Beispiel: Telefon (früher!)
Paketvermittlung (*packet switching*): Daten werden in Pakete zerteilt, Pakete werden auf beliebigem Weg zum Empfänger befördert/ Beispiel: Post/ In Rechnernetzen fast ausschließlich Paketvermittlung/ bessere Nutzung / Auslastung der ¨ Ubertragungswege

Arten von Netzwerken ... Nach der Struktur der Verbindungen
Punkt-zu-Punkt-Netze: eine Leitung verbindet genau zwei Netzwerk-Knoten (z.B. Rechner)/ spezielle Netzwerk-Knoten (IMP, Router, Switch) zur Weitervermittlung der Daten
Broadcast-Netze: Daten werden über einen Bus gleichzeitig an alle angeschlossenen Teilnehmer gesendet/ z.B.: klassisches Ethernet/ heute meist Punkt-zu-Punkt-Netze im Einsatz:/ höherer Durchsatz, bessere Sicherheit
Teilaufgaben bei der Kommunikation zwischen Rechnern: Bestimmung eines Weges vom Sender zum Empfänger (**Routing**)/ Aufteilen der Daten in Pakete (z.B. wegen notwendiger Zwischenspeicherung), Zusammenbau beim Empfänger (in der richtigen Reihenfolge!)/ Fehlerbehandlung: was ist, wenn ein Paket verlorengeht?/ Quittierung der Pakete/ nach Ablauf einer bestimmten Zeit: Sendung wiederholen/ Jetzt: Behandlung mehrfacher Kopien des Pakets nötig! **Flußkontrolle** (Empfänger an Sender: bitte langsamer!)

Schicht 1: Bit ¨ ubertragungsschicht (*physical layer*): Ubertragung einzelner .roher. Bits/ Elektrische Spezi_kation/ Medium: Kabel, Glasfaser, drahtlos (Funk, Infrarot, ...)?/ Spannungspegel, Lichtwellenlänge, Frequenzen/ Zeitverhalten/ Codierung und Modulationsverfahren Mechanische Spezi_kation/ Form / Art der Stecker und Kabel/ Anzahl der Anschlüsse, ...
Schicht 2: Sicherungsschicht (*data link layer*): Fehlerfreie Datenübertragung/ Zerteilung der Daten in Stücke (Frames), typ. _ 100 Bytes/ Frame durch *Header* und *Trailer* begrenzt/ *Trailer* enthält Redundanzbits (z.B. Prüfsumme) zur Fehlererkennung bzw. –korrektur/ Verlust ganzer Frames) erneute ¨ Ubertragung/ Behandlung von Duplikaten/ Flußkontrolle/ Zugriffskontrolle (*MAC, media access control*)/ nur bei Broadcast-Netzen erforderlich/ verhindert, daß zwei Einheiten gleichzeitig senden/ Beispiel: CSMA/CD beim Ethernet
Schicht 3: Vermittlungsschicht (*network layer*): Sicherungsschicht behandelt nur Kommunikation zwischen direkt verbundenen Netzwerk Knoten/ Vermittlungsschicht stellt Ende-zu-Ende Verbindungen zwischen beliebigen Rechnern im Netzwerk bereit/ Hauptaufgabe: **Routing** = Bestimmung eines Weges zwischen Sender und Empfänger/ statisch, nur aufgrund der Verbindungstopologie/ dynamisch, z.B. lastabhängig/ weitere Aufgaben: Behandlung von Stauungen, Abrechnung/ Beispiel: IP-Protokoll im Internet
Schicht 4: Transportschicht (*transport layer*): Stellt Verbindungen zwischen Endpunkten (Prozessen) auf Rechnern bereit/ Benennungsmechanismus für die zu kontaktierenden Prozesse/ Auf- und Abbau von Verbindungen/ Multiplexen von Verbindugen/ Stellt verbindungsorientierte Dienste bereit/ Kommunikationspartner erhalten den Eindruck einer Leitungsvermittlung/ selbst wenn untere Schichten paketorientiert arbeiten!/ Beispiel: TCP-Protokoll im Internet
Schicht 5: Sitzungsschicht (*session layer*)/ Dienste zur Verwaltung von Sitzungen, z.B./ Dialogsteuerung (.wer darf wann senden?.)/ atomare Aktionen (.alles oder gar nichts.)/ Synchronisierung (z.B. Weiterführung eines unterbrochenen Transfers)/ meist nicht implementiert!
Schicht 6: Darstellungsschicht (*presentation layer*): Kennt als erste Schicht die Semantik der übertragenen Daten/ Konvertiert Datenformate und –darstellung/ Auch: Kompression, Verschlüsselung, ...
Schicht 7: Anwendungsschicht (*application layer*): Spezialisierte Dienste und Protokolle für verschiedene Anwendungsbereiche/Beispiele: HTTP (*Hypertext transport protocol*)/ zur ¨ Ubertragung von WWW-Seiten/ SMTP (*Simple mail transport protocol*)/ zum Austausch von Emails/ CIFS / NFS/ Protokolle für Netzwerkdateisysteme/ SSH (*Secure shell*)/sicheres Protokoll zur Nutzung entfernter Rechner