Betriebssysteme, Prof. Dr. Odej Kao, Zusammenfassung von Florian Schoppmann

Zusammenfassung Betriebssysteme

1. Einführung

1.1 Aufgabenbereiche eines Betriebssystems

Grobe Aufteilung:

- Bereitstellung von Hilfsmitteln für Benutzerprogramme
- Vernachlässigung der genauen Benutzerkenntnis von HWEigenschaften und spezieller SW-Komponenten, wie z.B. Gerätetreiber
- Koordination und Vergabe der zur Verfügung stehenden Betriebsmittel an mehrere, gleichzeitig arbeitende Benutzer

Einzelaufgaben: Unterbrechungsverarbeitung, Verteilung (Prozessumschaltung), Betriebsmittelverwaltung, Programmallokation, Dateiverwaltung, Auftragsteuerung (Scheduling), Zuverlässigkeit

1.2 Geschichte der Betriebssysteme

Schlüsseltechniken:

- Multiprogramming (mehrere Applikationen im Speicher), Hardwarespeicherschutz
- Time Sharing (1962, MIT)
- Spooling (Simultaneous Peripheral Operation On Line)
- Virtualisierung

1.3 Unterschiedliche Arten von Betriebssystemen

Eigenschaft	*Netzwerk-BS*	*Verteiltes BS*	*Multiprozessor*
Verhalten wie eine virtuelle CPU?	Nein	Ja	Ja
Gleiches BS für alle Knoten?	Nein	Ja	Ja
Anzahl Kopien des BS	n	n	1
Realisierung der Kommunikation?	Gemeinsame Dateien	Nachrichten	Gemeinsamer Speicher
Gemeinsame Netzwerkprotokolle	Ja	Ja	Nein
Nur eine Prozesswarteschlange?	Nein	Nein	Ja

Verteilte Betriebssysteme: Netzwerk von Rechnern, bleiben dem Benutzer verborgen, Betriebssystem bestimmt, wo Programme ausgeführt werden und wo die benötigten Daten liegen, Benutzer hat den Eindruck einer einheitlichen Computerressource

Anwendungsgebiete: PCs, Echtzeit-Systeme, eingebettete Systeme und PDAs, Chipkarten

1.4 Strukturen der Betriebssysteme

Monolithische Systeme Alle Funktionen werden zu einem Objektcode kompiliert, jede Funktion kann jede andere Funktion aufrufen

Geschichtete Systeme Verallgemeinerung des Strukturmodells für monolithische BS, nur Funktionen der nächsthöheren Schicht aufrufbar

Virtuelle Maschinen Tatsächliche Ausführung von Systemaufrufen auf der realen HW durch Monitor auf HW-Ebene

Exokern Jeder Benutzer bekommt eine Kopie des darunter liegenden Computers, aber mit jeweils einem (statisch festgelegten) Teil der Ressourcen

Client-Server-Systeme mit Mikrokern Kommunikation zwischen Funktionen im Kern und Benutzerraum nach Client-Server-Modell durch Nachrichten

Mechanismus: Wie wird eine Aufgabe *prinzipiell* gelöst?

Policy: Was wird dabei *konkret* realisiert?
Wünschenswert: Genereller Mechanismus, so dass eine Veränderung der Policies durch Anpassung von Parametern umgesetzt werden kann

1.5 Fallstudie: Aufbau von Windows NT/2000

verwendete Modelle:

- (halbwegs) Client/Server Modell für die Bereitstellung verschiedener Betriebssystemumgebungen und Anwendungsprogramme
- Objektmodell für Resourcen (Zugriff über Handles)
- Symmetrisches Multiprocessingmit gemeinsamem Speicherzugriff

1.6 Fallstudie: Aufbau von Linux

- Schichtenbasiertes System
- Monolithischer Kern mit Modulen
- Kernaufgaben: Prozessverwaltung, Speicherverwaltung, Dateisystem, Ein-und Ausgabe, Systemaufrufschnittstelle, ...

2. Prozesse

2.1 Prozesse als zentrales Konzept

Definiert durch Adressraum, Programm, „Aktivitätsträger“ (bspw. Thread)

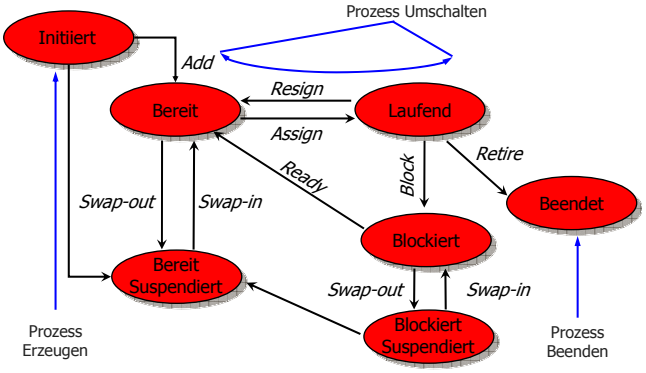
Dispatcher: BS-Komponente zur Verwaltung des realen Prozessors, verdrängt Prozesse und lädt neuer
Zusammenarbeit mit dem Scheduler, der für die Auswahl des nächsten Prozesses zuständig ist

Prozesskontrollblock: (PCB), Datenstruktur, enthält: PID, Zustandsinformationen (Registerwerte, Befehlszähler, Programmstatuswort), Kontrollinformationen (laufend, blockiert, ..., Eltern-Kind-Relationen, ...)

Effiziente Organisation: Aufspaltung in Teil- (verkettete) Listen mit identischen Attributen (z. B. laufend, blockiert)

2.2 Ausführung von Prozessen

Multiprogrammierung: Annahme: Ein Prozess verbringt Anteil p seiner Zeit mit Warten auf E/A-Operationen.
Wkt., dass n Prozesse gleichzeitig am Warten auf E/A-Operationen sind
Wenn Ausnutzung der CPU $1 - p^n$, dann ist n der Grad der Multiprogrammierung



2.3 Grundoperationen mit Prozessen

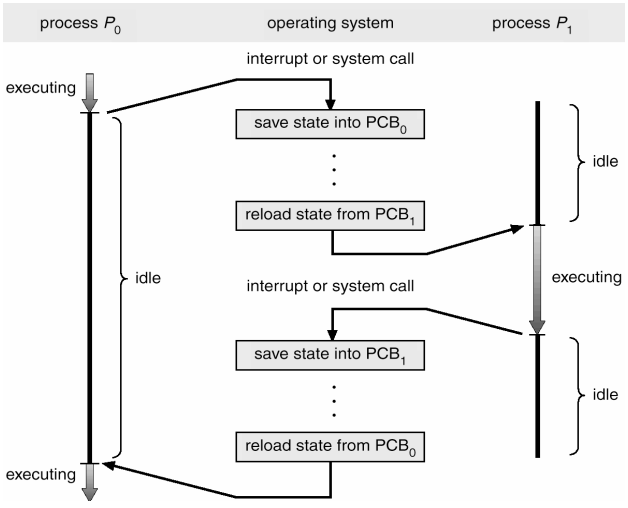
Prozesserzeugung unter UNIX: Systemaufruf ist `fork()`, Kindprozess ruft `execve()` auf, um das Speicherabbild zu wechseln
˜ unter Windows : Systemaufruf ist `CreateProcess()`

Vaterprozess und Kindprozesse bilden unter UNIX eine Familie, d. h. Signale werden an alle Prozesse in der Familie verteilt und jeder Prozess entscheidet über Annahme und Verwertung

Prozessumschaltung: Sicherung des alten Kontextes, Aktualisierung des PCBs und Einordnung in Warteschlange, Laden des Kontextes des neuen Prozesses, Aktualisierung des PCBs und der Speicherinformationen, Aktualisierung des Kontextes durch Anpassung aller Verknüpfungen

Umschalttechniken: direkter Sprung (Einprogrammierung im Quellcode, z. B. bei Echtzeitsystemen), automatisches Umschalten

Leerlaufprozess zur einfachen behandlung des Falls, in dem alle Prozesse im ready-Zustand sind



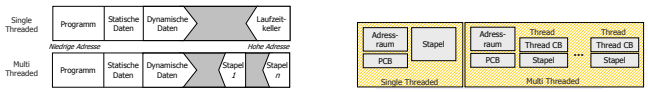
Systemmodus ↔ Benutzermodus, Speicherung im Programmstatuswort

2.4 Thread-Modell

Thread: Keine vollständige Prozesstabelle, selber virtuelle und reale Adressraum wie der Prozess, entspricht einem separaten Kontrollfluss

Threadtabelle: enthält: separaten Befehlszähler, eigenen Code-und Datenteil, Register und Stapel (vollständige Verknüpfungsumgebung), Zustand

Übersicht Prozessmodelle:



Vorteile durch Threads: Besseres Antwortverhalten (insb. bei GUIs), effiziente Betriebsmittelteilung, Thread- günstiger als Prozesserzeugung, Parallelität

Kernel-Level-Threads ↔ User-Level-Threads

Umschaltung bei User-Level-Threads: ohne Prozesskontextwechsel, keine Leerung des Caches, effizient, eigenes Scheduling ⟶ Anpassbar an spezielle Scheduling-Bedürfnisse
Bei Anforderung eines Betriebssmittels:

if Thread muss blockiert werden **then**
  Speicherung der Daten in der Threadtabelle
  if Anderer Thread ist ausführbar **then**
    Auswahl dieses Threads
  else
    Blockierender Systemaufruf
  end if
else
  BM wird zur Verfügung gestellt
end if

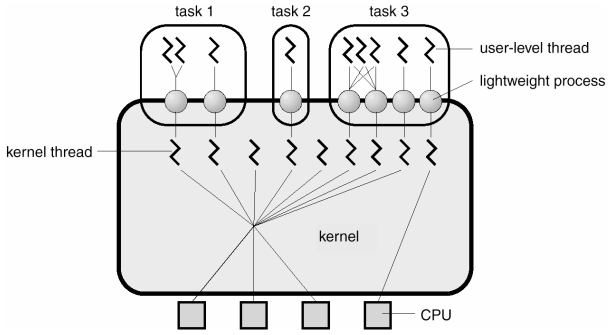
Nachteile von User-level-Threads: Bei Seitenfehler wird gesamter Prozess blockiert

Hybride Threadmodelle: UL-Threads benötigen mind. eines KL-Thread als Träger, Rechenzeit wird einzelnen Prozessen zugeordnet
many-to-one (nicht geeignet für Parallelrechner), one-to-one (Aufwand bei Threaderzeugung), many-to-many (flexibel, Kompromiss)

2.5 Beispiel für Threadbibliotheken: Pthreads

```
pthread_create
pthread_exit
pthread_join
pthread_cancel
pthread_detach
pthread_setcancel
pthread_mutex_init
pthread_mutex_lock
pthread_mutex_trylock
pthread_mutex_unlock
pthread_mutex_destroy
```

Beispiel für Realisierung in Solaris:



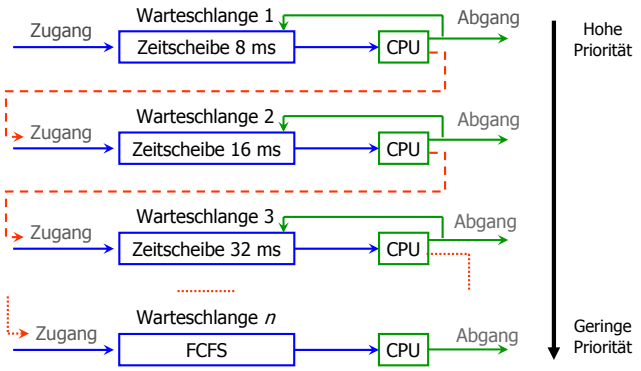
2.6 Zuteilungsverfahren

Typen von Zuteilungsverfahren: Long-term, z. B. Start eines neuen Prozesses (wie viele Prozesse sollen nebeneinander laufen?), Medium-term, z. B. Erstzuweisung von Hauptspeicher (welcher Prozess wird als nächstes ein-/ausgelagert?), Short-term, z. B. Zustandswechsel des Prozesses (Verwaltung von Ereignissen wie Timer, Systemaufrufen)

Schlüsselfaktoren für Scheduling: Länge der CPU-Nutzung in einem Zyklus, Verwaltungsoverheadfür E/A z. B. bei Festplatten unabhängig von der Größe der gelesenen Daten, Schere zwischen Geschwindigkeiten von E/A-Geräten und Prozessorgeschwindigkeit engeht immer weiter auf (⟶ Scheduling von E/A-lastigen Prozessen entscheidend für die Effizienz des Gesamtsystems)

Scheduling-Verfahren: FIFO, LCFS-PR, SJN, SRTN, PRIO-NP, Round-Robin (siehe KMS)

Multilevel-Feedback-Scheduling: dynamische Anpassung der Zeitscheibenlänge, Stufenweise Prioritätenreduktion für lang laufende Prozesse



2.7 Prozesse und Scheduling in Linux

Grundprinzipien: Threads über Biliotheken (ohne Kernelunterstützung), PCBs in doppelt verketteter Liste + Hash-Liste von PID ⟶ PCB

Prozesszustände:
`TASK_RUNNING`, `TASK_INTERRUPTIBLE` (durch `usleep()`, `TASK_UNINTERRUPTIBLE` (durch E/A-Aufruf), `TASK_ZOMBIE` (Prozess beendet, Vaterprozess muss Ergebnis noch abrufen), `TASK_STOPPED` (Prozess wurde angehalten, z. B. durch `kill -STOP` *PID*)

Multilevel-Feedback-Scheduling: RR/FIFO für Echtzeitprozesse, auf PRIO-P Strategie basierendes Verfahren für „normale“ Prozesse

2.8 Prozesse und Scheduling in Windows

Hierarchie: Jobs → Processes → Threads → Fibers (kooperative „lightweight“ Threads, deren Scheduling nicht vom Betriebssystem vorgenommen wird)

System-Priorität für Threads ergibt sich aus Win32-Prozesspriorität und Win32-Threadpriorität

Scheduling: Prioritätenliste mit 32 Prioritäten, RR für jede Prioritätsstufe
Verbesserungen: Prioritätserhöhung nach Beendigung einer E/A-Operation, temporäre Deaktivierung des Schedulings, wenn Systemthread in kritischem Bereich

3. Speicherverwaltung

3.1 Grundlegendes

Dynamische Speicherallokation, Zugriff auf einen fehlenden Inhalt löst Interrupt aus

3.2 Seitenersetzungsstrategien

Nachschubstrategien: Demand Paging, Pre-Paging

Verdrängungsstrategien (Replacement Policies): Probleme bei Realisierung durch aufwendige Operationen

Working Set: Menge $W_i(t, \tau)$ von Seiten, auf die ein Prozess i in den letzten τ Einheiten ab dem Zeitpunkt t zugegriffen hat.

Realisierung: Jeder Eintrag in der Seitentabelle enthält mind. Referenzbi, Modifikationsbit und ungefähre Zeite des letzten Zugriffs

WSClock-Algorithmus: Modifikation des Clock-Algorithmus

3.3 Speicherverwaltung unter Linux

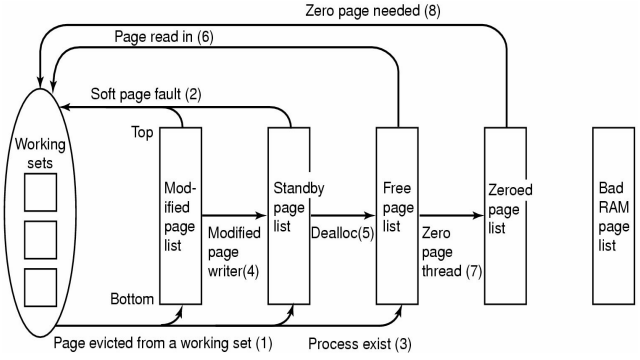
Virtuelle Speicherbereiche: Grafikspeicher oder (memory-mapped) Dateien werden in den Adressraum eingeblendet

Gemeinsam genutzter Speicher mit copy-on-write

Speicherbereinigung: Aufruf von `kswapd` alle 1 s oder bei Bedarf. Beachtung minimaler Verweildauer im Speicher zur Vermeidung von Thrashing

3.4 Speicherverwaltung unter Windows

Arbeitet mit Working-Sets:



3.5 Leistungsaspekte des virtuellen Speichers

WSClock bietet den besten Kompromiss zwischen effizienter Implementierung und guter Leistung

Modellierung des Seitentausches (3-34): **Was soll das denn?**

Thrashing: Überlastphänomen, ähnlich Stau im Straßenverkehr. Der Koordinationsaufwand wächst überproportional an.
Zur Verhinderung des Thrashing muss der Multiprogramminggrad $n_{\max}$ begrenzt werden:

indirekte Strategie Für jeden Prozess wird individuell eine sinnvolle Kachelanzahl festgestelt, und der Multiprogramminggrad begrenzt

direkte Strategie Anhand von Messungen der globalen Seitentauschaktivität wird ein optimaler Wert für $n_{\max}$ berechnet, bspw. max. Prozessanzahl so bestimmen, dass die durchschnittliche Zeit zwischen Seitenfehlern nicht größer ist als die Seitentransferzeit

Paging Deamon: Technik zur Sicherung eines ausreichenden Vorrats an freien Kacheln für schnelle Reaktion auf weitere Speicheranforderungen

4. Ein- und Ausgabe

4.1 Klassifikation und Grundaufgaben

Klassifikation:

Zeichenorientierte Geräte: Nicht adressierbar, keine Suchoperation. Beispiele: Tastur, Maus, Netzwerkkarten

Blockorientierte Geräte: Ein Block kann unabhängig von anderen Blöcken gelesen oder geschrieben werden. Beispiele: Festplatten, CD-ROMs, ...

Abstrakte Geräte zur Repräsentation in der Systemsoftware: Device Control Block, Mindestmenge von Funktionen (open/close, read/write), Gerätetreiber

Grundaufgaben: Steuern des Gerätes (z. B. Lesekopf positionieren), Datentransport von CPU/Speicher zum Gerät und umgekehrt

Transportauftrag: Positionierung der zu übertragenden Daten, Spezifikation der Parameter (Richtung, Länge der Daten), Rückmeldung: Status-/Fehlersignale

4.2 Schlüsselkonzepte von E/A-Software

Anforderungen: Geräteunabhängigkeit, Einheitliches Benennungsschema, Fehlerbehandlung möglichst nah an der Hardware (z. B. automatische Fehlerkorrektur bei CD-ROMs)

Konzepte: Syncrhone ↔ asynchrone Übertragungen, Pufferung, gemeinsame/exklusive Nutzung

Schichtenaufbau: Hardware → Unterbrechungsbehandlung → Gerätetreiber: Direkte Ansteuerung der spezifischen Geräte → Geräteunabhängige E/A-Systemsoftware → E/A-Software in Anwendungen: Systemaufrufe für die E/A

Unterbrechungsbehandlung:

1. Sicherung aller Register
2. Erzeugen des Kontextes für die Unterbrechungsroutine
3. Initialisierung von MMU, TLB, Speichertabellen, ... und Anlegen neuer Speicherbereiche
4. Interruptcontroller informieren und Behandlung weiterer Unterbrechungen veranlassen
5. Kopieren der gesicherten Register in die Prozesstabelle
6. Aufruf der Unterbrechungsroutine (Informationen aus den E/A-Geräten auslesen und notwendige Operationen durchführen)
7. Auswahl des nächsten auszuführenden Prozesses
8. Initialisierung/Wiederherstellung des Kontextes, Laden der Prozessregister
9. Verarbeitung mit neuem Prozess fortsetzen

4.3 Gerätetreiber

Treiber als Betriebssystem-Bestandteil/Modul ↔ Benutzerprozess (Geräteserver)

Code ist reentrant, wenn er von mehreren Programmen gleichzeitig verwendet werden kann

4.4 Geräteunabhängige E/A-Software

Pufferung: Keine Pufferung, Pufferung im Benutzerraum (Nachteil: Sperrung gegen Auslagerung nötig), Pufferung im Kernraum (Nachteil: Sicherung gegen Überlauf nötig, Benachrichtigung an Benutzerprozess), Wechselpuffer

Prozesskoordination (z. B. für CD-Brenner)

Spooling: Simultaneous Peripheral Operations Online

4.5 Beispiel für blockorientierteGeräte: Festplatten

Bedienzeit als Summe von: Armpositionierzeit, Latenzzeit (Drehwartezeit), Schreib/Lesezeit (Tatsächliche Ausführungszeit), Transferzeit

Low-Level- (Anlegen von Spuren, Sektoren und Lücken) ↔ High-Level-Formatierung (Anlegen eines leeren Dateisystems für jede Partition)

RAIDs: (Redundant Array of Inexepensive/Independent Disks) Level:

0. Virtuelle Festplatte wird in Strips unterteilt, Parallelität beim Lesen/Schreiben eines sich über mehrere Strips erstreckenden Blocks, keine Redundanz
1. Alle Platten doppelt vorhanden
2. Aufteilung von Wörtern oder Bytes
3. Wie Level 2, jedoch Speicherung der Parity-Daten auf einer Platte
4. Wie Level 2 und 3, jedoch mit (größeren) Strips, Parity-Platte als Flaschenhals beim Schreiben
5. Ähnlich Level 4, bei Verteilung der Parity-Strips über alle Platten zur Eliminierung des Falschenhals

4.6 Strategien für Plattentreiber

Random Abarbeitung in zufälliger Reihenfolge
FIFS First Come First Serve
Prioritäten Abarbeitung gemäß Prioritäten
SSTF Shortest Seek Time First (Armpositionsierungszeit), Minimierung der Anzahl der Armbewegungen, Bei stark belasteten Platten tendiert der Arm zu den mittleren Sektoren
SCAN Fahrstuhlstratgie, Lokalität von Anfragen nicht ausreichend berücksichtigt
SCAN-C Zyklische SCAN-Strategie, da nach jedem Durchlauf Rücksprung auf Zylinder 0

4.7 Effizienzbetrachtungen

FCFS: Im Mittel Überfahren von $\frac{1}{3}$ der Spuren
SCAN, SCAN-C, SSTF: Vorteile in Hochlastsituationen, im Grenzfall Sprung um nur eine Spur
SCAN und SSTF benachteiligen die Randspuren
SSTF: im Mittel die beste Leistung, Verhungern von Aufträgen möglich
Vorgehensweise Treiber blockieren nach Auftrag / Deblockieren nach Unterbrechung „Fertig“ führt zu langen Zeiten zwischen zwei Aufträgen (Totzeit)
Totzeitreduzierung (Latency Hiding): Überlappung der Aktivitäten „Durchführung des aktuellen Auftrags“ und „Vorbereitung des nächsten Auftrags“
Wie genau soll das funktionieren?
Plattencache: unabhängig vom Betriebssystem-Cache

5. Dateisysteme

5.1 Organisation von Dateisystemen und Dateiattribute

Eigenschaften per Definition: Persistente Speicherung großer Informationsmengen. Abstraktion der Details für die Datenablage, Gleichzeitiger Zugriff auf die Daten durch mehrere Prozesse möglich, Schutzmechanismus für den Datenzugriff

Typische Strukturierung von Dateien: Folge von Bytes (Interpretation beim Einlesen durch Anwendung), Folge von Datensätzen, Hierarchisch

5.2 Zugriffsarten

Sequentiell, wahlfrei (random access), Index-basiert

Indexstrukturen: B-Bäume, Index-Sequential Access Method, invertierte Liste

5.3 Grundlegende Operationen für Datei- und Verzeichnisverwaltung

selbsterklärend

5.4 Layout eines Dateisystems

Sektor 0 einer Festplatte: Master Boot Record zur Lokalisierung der Partitionstabelle und/oder aktiven Partition

Zwei-Phasen-Realisierung: 1. Abbildung Dateien → nummerierte Blöcke fester Länge 2. Abbildung Datenblöcke → Geometrie des Speichermediums

5.5 Zuordnung Dateien → Datenblöcke

Einfachste Möglichkeit: Zusammenhängende Abbildung. Jedoch mit Nachteil: Fragmentierung

Belegung durch verkettete Listen:

interne Verkettung Jeder Block enthält Zeiger auf nächsten Block. Nachteil: Nur sequentieller Zugriff möglich, wenig robust: Ein Dateiverlust bei Zerstörung nur eines Blockes

externe Verkettung Externe Allokationstabelle (File Allocation Table, FAT)

Indexblöcke Jeder Datei wird ein Indexblock zugeordnet, für große Dateien mehrstufige Indexblöcke. Vorteile: Tabelle zur Speicherung der verketteten Liste aller Dateiblöcke wächst linear mit Festplattengröße, Arbeitsspeicherbedarf beim Zugriff ist ausschließlich proportional zur maximalen Anzahl gleichzeitig geöffneter Dateien

Optimierungen: Trennung des Dateinamens (von variabler Länge) von den übrigen Datei-/I-Node-Attributen

5.6 Verwaltung des Plattenspeichers

Verwaltung freier Blöcke: Verkettete Listen (sinnvoll: Nutzung der freien Blöcke zur Speicherung) ↔ Bitmaps

5.7 Leistungsaspekte: Caching, Backup, Konsistenz, ...

Caches: Write Through, Careful write: Schreiben nach einer vorgegebenen Maximalverzögerung, Lazy write: Alles wir vorerst im Cache behalten

Optimierungen: Vorauslesen von Blöcken, virtuelle Festplatten (RAM-Disks), Optimierung der Blockanordnung

Konsistenzüberprüfung: Abgleich freier Blöcke vs. belegter Blöcke, Block in Freiliste vermisst/doppelt vorhanden, kritisch: Block kommt in mehreren Dateien vor

5.8 Log-basierte Dateisysteme

Motivation: CPU-Geschwindigkeit wächst schneller als Festplattenzugriffszeiten. Caches werden wichtiger und damit die Behandlung von Konsistenzproblemen

Optimierung: Schreiben eines temporären Logs, das später erst aufgeräumt wird, d. h. die tatsächlichen Änderungen an Dateien vorgenommen werden

5.9 NTFS Dateisystem

Grundeigenschaften: Clustergröße unabhängig von Datenträgergröße, Alle Daten (auch Meta-Daten) als Dateien, Verschlüsselung, Komprimierung, mehrere Datenströme

6. Sicherheit

6.1 Grundlagen der Sicherheit

Sicherheitsanforderungen: Datenvertraulichkeit, -integrität, Systemverfügbarkeit, Datenschutz

Verwendung von Kryptografie: Symmetrische ↔ Asymmetrische Verschlüsselung, Digitale Signaturen, Hashing mit Einwegfunktionen

6.2 Benutzer-Authentifizierung

Authentifizierung durch: „Geheimnis“ des Benutzers, z. B. Passwort, biometrische Daten, Zusätzliche Hardware (z. B. Chipkarten, herklömmlicher Schlüssel)

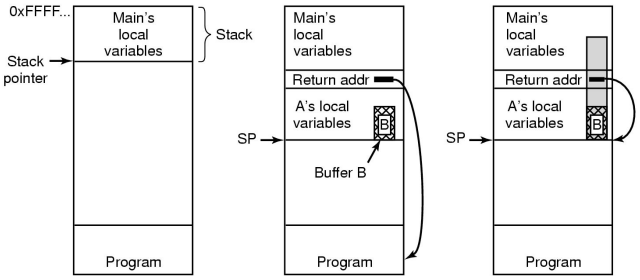
Passwortsicherheit in UNIX: Passwortdatei mit je einer Zeile mit Attributen und verschlüsseltem Passwort je Benutzer. Aktuelle Version: Separate Speicherung der verschlüselten Passwörter in Shadow-Datei

Verbesserung der Passwortsicherheit: Verzögerungsstrategien, Verwendung schwer zu ermittelnder Passwörter, Einmal-Passwörter, Challenge-Response-Verfahren (Erweiterung davon: Smart-Cards mit kleiner CPU)

6.3 Angriffe von innerhalb des Systems

Login-Spoofing: Programm bildet Passwort-Eingabebilschirm exakt nach. Abhilfe: Bspw. Tastenkomnination, die nicht vom Benutzerprogramm abgefangen werden kann

Buffer-Overflow: Überschreibung der Rücksprungsadresse durch überlange Eingaben



6.4 Angriffe von außerhalb des Systems

nicht klausurrelevant

6.5 Schutzmechanismen

Dömane: Menge von Paaren (Objekt, Rechte), jedes Paar spezifiziert ein eindeutig identifiziertes Objekt (HW, Prozesse, Dateien, Semaphore, ...) und die darauf ausführbaren Operationen
Ein Prozess läuft zu jedem Zeitpunkt in einer einzigen Schutzdomäne

Domänenkonzept bei UNIX: Festlegung durch Benutzer-ID und Gruppen-ID

Klassische Realisierung des Domänenkonzepts durch Speicherung in Matrix:

	Object										
Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter2	Domain1	Domain2	Domain3
1	Read	Read Write								Enter	
2			Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Effizientere Speicherung, da in Matrix zu viele leere Felder vorhanden: Nur die Spalten (Access Control Lists – ACLs) oder der Zeilen (Capabilities)

Capabilities: (C-Listen)

6.6 Firewalls

bekannt