BET – VL Zusammenfassung

Inhalt

Danksagung

Diese Danksagung gilt einem Menschen, der nicht nur ein geschätzter Studienkollege ist, sondern auch zu einem engen Freund geworden ist. Lieber Anel diese Rede ist für dich.

Als ich mich entschied, dieses Studium aufzunehmen, war mir nicht bewusst, dass ich neben einer Menge Wissen und Herausforderungen auch eine wertvolle Freundschaft gewinnen würde. Von unserem ersten Tag an hast du mich mit deiner Offenheit, deinem Humor und deiner unglaublichen Hilfsbereitschaft beeindruckt. Du warst stets bereit, mir und anderen zu helfen, egal ob es sich um eine knifflige Aufgabe oder einen persönlichen Rat handelte.

In den unzähligen Stunden, die wir zusammen in der Bibliothek verbracht haben, hast du mich nicht nur durch dein Wissen unterstützt, sondern auch durch deine positive Einstellung und deinen unerschütterlichen Optimismus motiviert. Deine Fähigkeit, selbst in stressigen Zeiten Ruhe zu bewahren und einen klaren Kopf zu behalten, hat mich oft beeindruckt und inspiriert.

Du warst es, der mich in den schwierigsten Momenten daran erinnert hat, warum wir dieses Studium begonnen haben und dass es sich lohnt, für unsere Ziele zu kämpfen. Du hast mir gezeigt, dass man mit Zusammenarbeit und gegenseitiger Unterstützung jede Herausforderung meistern kann. Deine Kameradschaft hat diese Jahre so viel erträglicher und freudiger gemacht.

Es sind nicht nur die fachlichen Diskussionen und die gemeinsame Arbeit an Projekten, die unsere Zeit geprägt haben. Es sind auch die gemeinsamen Pausen, die unzähligen Tassen Kaffee und die vielen Gespräche über das Leben, die uns zusammengeschweißt haben. Du hast mir nicht nur in akademischen Belangen geholfen, sondern auch gezeigt, was wahre Freundschaft bedeutet.

Heute möchte ich dir von Herzen danken. Danke für deine Geduld, deinen unermüdlichen Einsatz und dafür, dass du immer ein offenes Ohr hattest. Danke, dass du mich inspiriert und unterstützt hast, und dass du mir gezeigt hast, dass man gemeinsam mehr erreichen kann. Du bist ein großartiger Kollege und ein noch großartigerer Freund.

Ich freue mich darauf, dass wir auch in Zukunft gemeinsam neue Herausforderungen angehen und Erfolge feiern können. Möge unsere Freundschaft weiterhin wachsen und uns beide bereichern.

Lieber Anel, vielen Dank für alles, was du getan hast und für die Person, die du bist. Ohne dich wäre diese Zeit nicht dieselbe gewesen. Auch wenn du bei dieser Zusammenfassung nicht beteiligt warst bin ich dir dankbar.

Mit herzlichen Grüßen und tiefstem Dank,

Sepp

Allgemeine Grundlagen

Definition

Was ist ein Betriebssystem?

- = Betriebsmittelverwalter
    - Administriert die Ressourcen
    - Entscheidet über die gerechte Verteilung der Betriebsmittel
- = Kontrollprogramm
    - Kontrolliert die Ausführung von Programmen
    - Kontrolliert die Fehlerbehandlung
    - Kontrolliert die Auswirkung von Fehlern
    - Kontrolliert die Verwendung des Computers

Aufgaben

Aufgaben eines Betriebssystems

- Vereinfachung der Maschine und ihrer Komplexität für User
- Bereitstellen einer Benutzerschnittstelle
    - Z.B. Shell
- Verwaltung der Ressourcen
    - Prozessoren
    - RAM und Hintergrundspeicher
    - Schnittstellen und Geräte
    - Rechenzeit
- Bereitstellung und Schutz der Ressourcen
- Koordination von Prozessen
- Bereitstellung einheitlichen Schnittstellen und Tools
- Schutz des Systems
    - Zugriffsschutz
- Fehlerbehandlung
    - „free“ von malloc z.B.

Klassifikation

Klassifikation nach Betriebsart

- Stapelverarbeitungs-BS (batch processing)
    - Abarbeitung im Stapelbetrieb
- Dialogbetrieb-BS (dialog processing)
    - Steuerung des Rechners im Dialog mit User (Maus, Tastatur,...)
- Netzwerk-BS (network processing)
    - Vernetzung des Systems, Client-Server-Betrieb

- Realzeit-BS (realtime processing)
    - Schwerpunkt auf Reaktions- und Verarbeitungszeit
- Universelle BS
    - Erfüllt mehrere oben genannte Kriterien (z.B. Windows, Unix)

Klassifikation nach gleichzeitig angemeldeten Usern

- Single User System
    - Nur 1 eingeloggter User zu einem Zeitpunkt
- Multi User System
    - Mehrere User teilen sich die Ressourcen des Systems

Klassifikation nach gleichzeitiger Prozessierung

- Single tasking System
    - Nur 1 Prozess zu einem Zeitpunkt, mehrere werden hintereinander ausgeführt
- Multi tasking System
    - Wenn mehrere CPUs verfügbar sind werden mehrere Programme gleichzeitig ausgeführt
    - Wenn nur 1 CPU verfügbar ist, werden sie zeitlich verschachtelt ausgeführt

Klassifikation nach Anzahl der Prozessoren

- Single processor system
    - 1 Hauptprozessor (CPU) -> Spezialprozessoren (GPU) werden nicht mitgezählt
- Multi processor system
    - Mehrere Universalprozessoren gleichzeitig
    - Entweder auf jedem Prozessor 1 Programm oder jedes Programm kann jedem Prozessor zugewiesen werden

Klassifikation nach dem Zeitverhalten

- Realtime OS
    - Garantiert eine bestimmte Reaktionszeit
    - Harte Echtzeitsysteme -> Zeiteinhaltung unter allen Umständen
    - Weiche Echtzeitsystem -> gewisse Toleranzen erlaubt

Klassifikation nach Leistung und Einsatzbereich

- Server
    - Kommuniziert mit anderen Systemen (Client-Server-Betrieb)
    - Hohe Zuverlässigkeit
    - Z.B. Mailserver

- Großrechner
    - Komplexes, umfangreiches und leistungsfähiges System
    - Hohe Zuverlässigkeit und I/O Leistung
- Embedded System
    - Kleinere Systeme mit speziellen Anforderungen

Komponenten

Komponenten eines Computersystems

Prozessoren

- Gehirn des Computers (holt Befehle aus dem Speicher und führt sie aus)
- Interne Register zum temporären Ablegen von Parametern

Hauptspeicher

- Speicher zur Ablage von Daten -> Möglichst groß, schnell und billig
- Unterteilung nach Zugriffsart (ROM vs RAM) oder Zugriffsgeschwindigkeit

Ein-/Ausgabegerät

- Controller
    - Setzt Befehle des Betriebssystems in Gerätebefehle um
- Gerät
    - Führt Gerätebefehle aus

I/O Controller

- Datentransfer für I/O:
- Busy Waiting
    - CPU ist blockiert, bis die Arbeit erledigt ist
- Interrupt Bearbeitung
    - Jeweiliges Gerät ist blockiert aber nicht das System -> Betriebssystem kann sich anderer Arbeit widmen
- Direct Memory Access (DMA)
    - Spezieller Chip übernimmt Datenfluss -> CPU kann sich um andere Aufgaben kümmern

Bussysteme

- Parallele Bussysteme
    - Daten werden parallel angelegt (z.B. Datenbus)
- Serielle Bussysteme
    - Daten werden nacheinander auf der Leitung übertragen (vom Rechner zu I/O Komponenten

Komponenten eines Betriebssystems

Kernel

- Verwaltet Hardware, Programme zum Start des Betriebssystems und Konfig des Systems

Boot Loader

- Durch Firmware (BIOS) geladen und ausgeführt
- Startet OS
- Multistage loader (mehrstufiger Bootvorgang)
- Chain loader (hintereinander ausführen von mehreren Bootloadern)

Device Driver

- Programm, das eingebaute Hardware steuert

System services (daemons)

- Hintergrundprogramme, um bestimmte Aufgaben zu erledigen

Programm libraries

- Hilfsmodule und -funktionen

Utilities

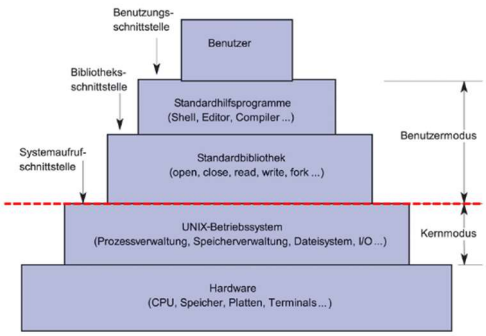
- Hilfsprogramme zur Verwaltung des Systems

Architektur

Architektur eines Betriebssystem

Strukturierung erfolgt in Schichten

- Unterste Schicht verwaltet die realen Betriebsmittel des Rechners -> BIOS
- Mit jeder Schicht steigt die Benutzerfreundlichkeit

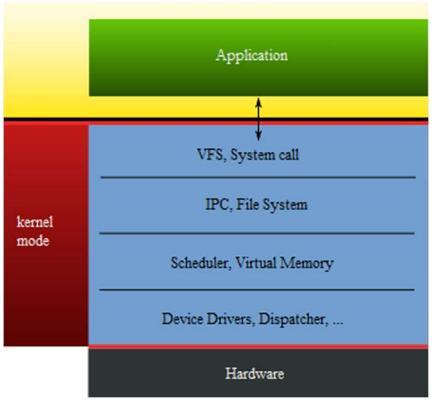


Architekturanforderungen

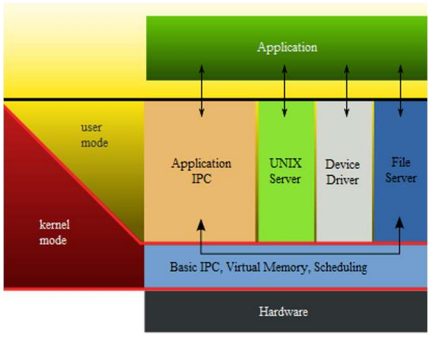
- Multi User
    - Muss Schutz der einzelnen Ressourcen gewährleisten
    - Ungewollter Zugriff auf die Ressource einer anderen Applikation muss ausgeschlossen werden

Betriebssystemstrukturen

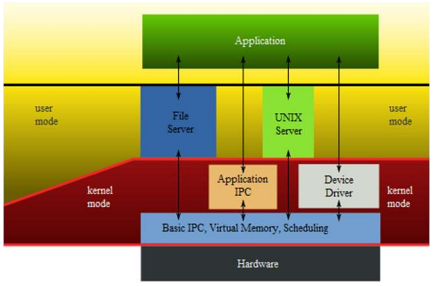
- Monolithischer Kernel
    - Minimale Struktur im Kernel
    - Modularisierung nur als Funktion implementiert
    - Nachteile: keine Abgrenzung der Funktionen untereinander -> 1 Funktion stürzt ab = Kernel stürzt ab
    - Vorteile: einfach und performant, wenig Kommunikation und Overhead bei Funktionsaufruf
    - Bsp: Unix, Linux



- Mikrokernel
    - Minimalkernel
    - Betriebssystemkomponenten sind als eigenständige Prozesse ausgelagert
    - Nachteile: mehr Overhead, geringere Geschwindigkeit, Synchronisationsaufwand, Treiber oft aufwendig
    - Vorteile: Komponenten separiert, Treiber im User Mode, Kernel klein, weniger Seiteneffekte
    - Bsp: Minix, Mach



- Hybridkernel (Makrokernel)
    - Mischung aus 1 und 2
    - Mikrokernel, bei dem bestimmte Komponenten aus Performancegründen doch in den Kernel übernommen werden
    - Versucht die Vorteil von 1 und 2 zu vereinen
    - Bsp: Windows, Mac OS



Privilegien und Systemaufrufe

Privilegierung über Ringe

- Moderne OS schränke Prozesse ein
- Ring = bestimmte Privilegierungsstufe
- Prozesse in einem bestimmten Ring erhalten
    - Eingeschränkten Befehlssatz der CPU

    - Eingeschränkten Zugriff auf die Hardware
    - Prozesse im Ring 0 sind im Kernel Mode alle anderen im User Mode
    - Nur Kernel Mode Prozesse erhalten uneingeschränkten Zugriff auf Hardware
    - Prozesse im User Mode können keine anderen Prozesse beeinflussen

Ringe, Kernel und Systemaufrufe

- Kernroutinen laufen in Ring 0
- Unprivilegierte Prozesse können nur über bestimmte Gates auf darunterliegende Ringe zugreifen. Gates können nur über System Calls überwunden werden

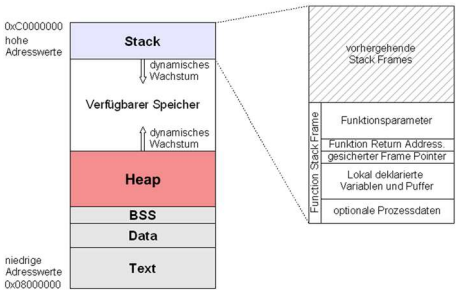
System Calls

- Aufrufe von Betriebssystemfunktionen aus Applikationen heraus
- Bei Syscall muss vom User-Mode in den Kernel-Mode geschalten werden
- Dieser Wechsel erfordert einige CPU-Zeit und ein spezielles Vorgehen mit Interrupts
    - Parameter werden auf Stack der CPU abgelegt
    - Softwareinterrupt wird ausgelöst
    - CPU unterbricht die laufende Funktion und springt zum Interrupt
    - Arbeitet ISR ab und kehrt dann wieder zurück

Speichermanagement

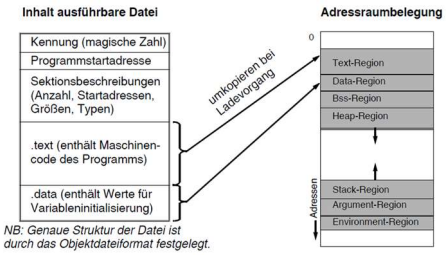
Prozessspeicher, Speicherhierarchie und Belegungsstrategien

Prozessspeicher: Segmente und Stack



- Textsegment
  - Instruktionen in Maschinencode
  - Code in C
- Datensegment
  - Beschreibungen von globalen Variablen
  - Initiale Werte der Variablen
- Bss-Segment (Block started by symbol)
  - Speicher für nicht initialisierte globale Variablen
- Prozess-Stack für Zugriff im User-Mode
  - Funktionsaufrufe -> Parameter
  - „normale“ Programmbearbeitung im User Mode
- Prozess-Stack für Zugriff durch Kernel
  - Nur vom Kernel benutzt
- Mit malloc werden Daten im Heap gespeichert, mit free entfernt
- Int arr[] wird z.B. in den Stack gespeichert

Prozessspeicher: Sektionen



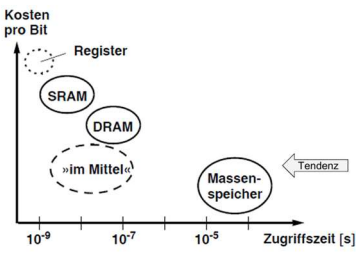
NB: Genaue Struktur der Datei ist durch das Objektdateiformat festgelegt.

Prozessspeicher: Relokation

- EXE liegt auf Speichermedium
- Beim Laden muss OS Bereiche für Code, Daten, Heap und Stack reservieren
- Relokation bei Programmstart notwendig = Anpassung der Zugriffsadressen im Code

Speicherhierarchie

- Forderungen an den Speicher
  - Minimale Zugriffszeit
  - Minimale Kosten/Bit
  - Persistenz



Rechnerspeicher – Eigenschaften

- Primärspeicher
    - Kurzzeitige Ablage der Daten während des Betriebs
    - Direkt adressierbar
    - Realisierung als RAM oder ROM
- Sekundärspeicher
    - Längerfristige Ablage der Daten
    - Indirekt adressierbar
    - Realisierung als feste Platten oder Bänder

Cache = Pufferspeicher

- Cache-Speicher in der Software
    - Von Betriebssystem für Plattenspeicher
    - Für Datenbanken
- Cache-Speicher in der Hardware
    - Lokaler Pufferspeicher in Peripherie (z.B. Festplatte, Netzwerkkarte)
    - Pufferspeicher zwischen CPU und Hauptspeicher
    - Pufferspeicher in MMU

Cache zwischen CPU und Hauptspeicher

- Zweck des Chaches
    - Schnellstmöglicher Speicherzugriff für CPU
    - Sinnvoll, wenn CPU schneller als Hauptspeicherzugriff
- Realisierung
    - Prozessorintern oder -extern
    - Ein- oder mehrstufig
- Grundprinzip
    - Cache puffert Teilbereiche des Hauptspeichers
    - Enthält oft benötigte Teile des Hauptspeichers
    - Adresse + Speicherinformation im Cache abgelegt
    - Effizient bei wiederholtem Zugriff auf gleiche Adressen

Cache Memory: Prinzip

- Cache puffert Teilbereiche des Hauptspeichers in Cache-Zeilen (legt Adresse und Speicherinformation im Cache ab)

Cache Memory: Leistung

- Einfluss auf Cache-Leistung:
- Größe des Cache-Speichers
    - Falls zu klein -> öfter Nachladen
- Größe der Cache-Zeilen
    - Beeinflusst Verwaltungsaufwand

- Organisationsform
    - Wann wird eine Zeile geladen? Im besten Fall bevor die CPU darauf zugreifen will
- Programmstruktur
    - Geschicktes Organisieren der Daten und des Programmflusses

Cache Memory: Leistung -> Probleme

- Vorhersage auf welche Adresse der Prozess zugreifen wird
- Verwaltung in Zeilen statt Bits -> mehr Datentransfervolumen
- Konsistenzproblem (Schreiben nicht gecached oder mit verzögertem Rückschreiben möglich)
- Adressdecoder muss Peripherieadressen kennen und Cache umgehene

Cache Memory: Leistungsberechnung

- Gibt Wahrscheinlichkeit an mit der CPU, Daten direkt aus dem Cache holen kann
- $h = \frac{N_{hits}}{N_{ref}} = \frac{N_{hits}}{N_{hits}+N_{miss}}$

N_{hits} .... Anzahl der *Hits* bei Zugriff auf den Speicher
N_{ref} .... Gesamtanzahl der CPU Zugriffe auf den Speicher
N_{miss} ... Anzahl der *Misses* bei Zugriff auf Speicher
h .... Trefferrate (*Hit ratio*) beim Speicherzugriff auf Speicher
h = 0 … alle Zugriffe gehen auf den Hauptspeicher
h = 1 … alle Zugriffe gehen in den Cache
in der Praxis: h = 0.95 … 1.0

- Cacheleistung über die Zugriffszeit
- $T_{eff} = h * T_C + (1 - h) * T_m$

T_{eff} .... effektive Zugriffszeit
T_c .... Zugriffszeit des Cache-Speichers (*cache memory*)
T_m .... Zugriffszeit des Hauptspeichers (*main memory*)
h .... Trefferrate beim Speicherzugriff

- Zugriff muss auch bei Miss zweistufig stattfinden
- $T_{avg} = h * T_C + (1 - h) * (T_m + T_C) = T_C + (1 - h) * T_m$

T_{avg} .... mittlere Zugriffszeit
T_c .... Zugriffszeit des Cache-Speichers (*cache memory*)
T_m .... Zugriffszeit des Hauptspeichers (*main memory*)
h .... Trefferrate beim Speicherzugriff (*hit ratio*)

Speicherbereitstellung

- Statische Daten
    - Lebensdauer = gesamte Programmlaufzeit
    - Im Heap (Datensegment)
    - Bsp: globale Arrays, statische Variablen
- Dynamische Daten
    - Lebensdauer = kürzer
    - Im Stack (Bestandteil des Aktivierungsrahmens)
    - Bsp: lokale „automatische“ Variablen
    - Im Heap (allokierter Speicherbereich)
    - Bsp: über malloc allokiert

Heapmanagement

- Dynamische Bereitstellung von Speicher
    - Aus großem Heap werden kleinere Bereiche gemacht
- Heapverwaltung erfolgt durch
    - Laufzeitbibliotheken
    - OS
- Schnittstelle zur Reservierung: malloc/free

Fragmentierung

- Externe Fragmentierung
    - Durch unkoordinierte Reservierung kommt es zu kleinen Lücken
- Interne Fragmentierung
    - Feste Größen führen zu Verschnitt -> ich kriege 1024 obwohl ich nur 1000 angefordert habe

Heapmanagement: Anforderungen

- Flexible Zuordnungsgröße: dem Bedarf entsprechend, möglichst wenig Verschnitt
- Zusammenhängende Bereiche
- Schnellstmögliche Zuordnung
- Maximale Speichernutzung: gleich 1024 anfordern, wenn man bei 1000 sowieso 1024 bekommt
- Adressraumausrichtung: z.B. nur durch 2 oder 4 teilbare Adressen

Speicherbelegungsstrategien

Überblick

- First, Next, Best und Worst Fitzählen zu den sequential fit algorithms
- Freilisten werden verwaltet als: LIFO, FIFO
- Externe Fragmentierung möglichst klein halten und hohe Geschwindigkeit

First Fit

- Erstes ausreichend großes Speicherstück verwenden
- Variable Blockgrößen
- Vorteile:
    - Schnell
- Nachteile:
    - Häufung der kleinen Lücken am Listenanfang

Next Fit

- Finde eine Lücke und beim nächsten mal wird von dort gestartet wo ich letztes mal aufgehört habe
- Variable Blockgrößen

- Vorteile:
    - Schnell
    - Häufung der kleinen Lücken wird vermieden
- Nachteile:
    - Insgesamt größere Fragmentierung -> schlechtere Gesamtperformance

Best Fit

- Suchen, welche Lücke am besten passt
- Variable Blockgrößen
- Vorteile:
    - Effizient bei wiederholten Anforderungen gleicher Größe -> weniger Verschnitt
- Nachteile:
    - Aufwendiges Suchen -> langsamer
    - Sehr kleine Lücken entstehen, die nicht mehr verwendet werden können

Worst Fit

- Suchen nach der größten Lücke
- Variable Blockgrößen
- Vorteile:
    - Kleine unbrauchbare Lücken werden vermiede
- Nachteile:
    - Aufwendiges Suchen -> langsamer
    - Größte Lücken werden systematisch verkleinert

Quick Fit

- Getrennt geführte Freilisten für verschiedene und feste Zuordnungsgrößen
- Wenn ich 4L Topf brauche fange ich bei 5L Töpfen zu suchen an
- Erste freie Lücke einer List wird verwendet
- Feste Blockgrößen
- Vorteile:
    - Sehr schnell
- Nachteile:
    - Rekombination der freien Blöcke aufwendiger -> Freigabe langsamer
    - Größere Interne Fragmentierung (mehr organisatorische Kosten)

Buddy System

- Blockgrößen auf Basis 2^k Byte
- Start mit 2^m (Größe des gesamten Bereichs) -> Block solange teilen, bis minimale Größe erreicht wird
- Blockausnutzung immer zwischen 50 und 100% (50% schlechtester Fall z.B. brauche 16,1kB bekomme aber 32kB)
- Feste Blockgrößen

- Vorteile:
    - Sehr schnell
- Nachteile:
    - Rekombination der freien Blöcke aufwendiger -> Freigabe langsamer
    - Größere Interne Fragmentierung (mehr organisatorische Kosten)

Heapmanagement: Verfahren gegen Fragmentierung

- Kompaktierung / Speicherverdichtung
    - Lückeneliminierung durch Defragmentierung
        - Verschieben belegter Speicherbereiche
        - Probleme:
            - Startadressen ändern sich -> Pointer nicht mehr gültig
            - Zeitaufwendig
        - Lösung:
            - Master-Pointer als Referenz auf Speicher (Mein Pointer zeigt auf Master-Pointer und der zeigt erst auf Adresse)
    - Reservierung
        - In bestimmten Größen, die bestimmten Größenklassen entsprechen
        - Reservierung/Freigabe in bestimmter Reihenfolge
    - Rekombination
        - Zusammenlegung von Speicher bei der Freigabe mit bereits freiem benachbarten Adressraum

Verfahren gegen knappen Speicher

- Overlay-Technik
    - Nur momentan benötigte Programmteile werden in Hauptspeicher geladen
    - Auf Prozedurebene muss Programm schon vorher entsprechend modularisiert werden (vom Entwickler)
    - Anwendung bei Monoprogrammierung
    - Nur mehr bei embedded Systems im Einsatz
- Swapping
    - Ganze Prozesse werden auf spezielle Bereiche der Platte ausgelagert, wenn sie im Moment nicht benutzt werden
- Demand Paging
    - Nur Teile des Codeumfangs werden ausgeführt

Realer Speicher, Virtueller Speicher, Adressumsetzung

Adressräume

- Adressraum (virtueller Speicher)
    - Direkt adressierbarer Speicher in Bytes z.B mit 32 Bit => 2^32 4GB adressierbar
- Speicherraum (realer Speicher)
    - Physischer Hauptspeicher (meist kleiner als der mögliche Adressraum)

Adressumsetzung

- Programmadressen
    - Von der CPU verwendet (sichtbar im Debugger)
- Speicheradressen
    - Auswahl der Speicherstellen über Speicherbus
- Umsetzung erfolgt in der MMU
    - Programmadressen werden mit Tabelle in Speicheradresse umgesetzt

Virtueller Speicher

- Jeder Prozess bezieht eigenen privaten Adressraum
- Prozess hat scheinbar den ganzen Adressraum für sich alleine
- Virtuelle Adressräume sind voreinander geschützt

Virtueller Speicher: Adressumsetzung

- Segmentbasierte Adressumsetzung
    - Segment = zusammenhängender Adressbereich beliebiger Größe
    - Aufteilung der Prozesse in Reihe von Segmenten
    - Segmente haben eindeutige ID für Zugriff auf Umsetzungstabelle
    - Adressierung über Segmentnummer + Relativadresse
    - Vorteile
        - Passgenaue Bereiche
        - Orientiert am Prozessbedürfnis
        - Geringer Platzbedarf für Umsetzungstabelle
    - Nachteile
        - Wahrscheinlich externe Fragmentierung
        - Hauptspeicherverwaltung aufwendig
        - Hoher Zeitaufwand
        - Segmente für Anwendungen sichtbar
- Seitenbasierte Adressumsetzung
    - Seiten = zusammenhängende Speicherblöcke gleicher Größe (Windows und Linux 4KB)

    - Vorteile
        - Einfache Hauptspeicherverwaltung
        - Schnell
        - Transparent für Anwendungen
        - Geringer HW-Aufwand
        - Keine externe Fragmentierung
    - Nachteile
        - Wahrscheinlich interne Fragmentierung
        - Hoher Platzbedarf für Umsetzungstabelle
        - Verwaltung des virtuellen Adressraums notwendig
- Seitenwechselverfahren
    - Logischer Adressraum wird in pages unterteilt und physischer Adressraum in page frame
    - Seiten können auf Hintergrundspeicher ausgelagert werden
    - Dynamisch
    - Stark verbreitet in OS
    - Vorteile
        - Geringer Ladeumfang
        - Weniger Hauptspeicherplatz
        - Kürzere Reaktionszeit
    - Nachteile
        - Zusätzlicher Platzbedarf auf Platte
        - Unerklärlicher Plattenverkehr
        - Nachladezeiten
    - Leistungsoptimierung
        - Genug Hauptspeicher für alle Prozesse vorsehen
        - Minimierung der Seitenfehlerrate und Seitenwechsel
        - Verdrängungs-, Lade- und Entladestrategie optimieren

Prozesse und Threads

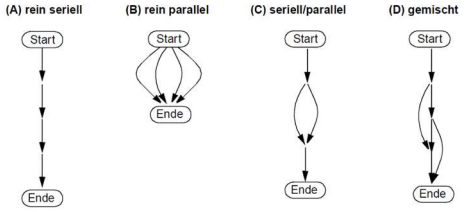
Prozesse, Threads, Zustände und Scheduling

Grundbausteine der Parallelverarbeitung

- Programm = Verfahrensvorschrift
- Process = Programm in Ausführung
- Thread = parallel ablaufende Aktivität innerhalb einer Umgebung (jeder Thread hat eigenen Stack aber gemeinsamen Heap)
- Job oder Session = voneinander unabhängig laufende Anwendungen

Parallelverarbeitung

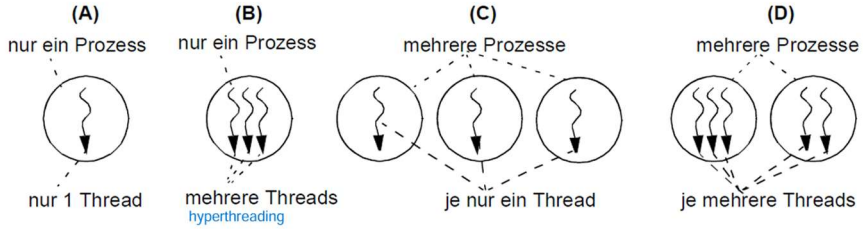
- Wenn verschieden Aktivitäten gleichzeitig stattfinden
- Unterscheidung der Verarbeitungstypen seriell – parallel:



Varianten der Software-Parallelität

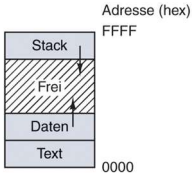
- Prozesse
  - Eigene Ablaufumgebungen für Apps
  - Jeder Prozess teilt sich Adressraum nur mit OS
  - Prozesse voneinander isoliert
- Threads
  - Eigene Ablaufstränge innerhalb Apps
  - Teilen sich Adressraum innerhalb der Apps
  - Keine Isolation untereinander
- Prozesse und Threads sind kombinierbar

Parallelität mit Prozessen und Threads



Prozess = Instanz

- Prozess besitzt 3 Speichersegmente
- 3 Speichersegmente + Statusinformation = Instanz



Prozess und Prozessoren

- Mehrere Prozessoren oder Kernen können mehrere Prozesse parallel ausführen
- Ein Prozessorkern bedient zu einem Zeitpunkt nur 1 Prozess -> kann Prozesse nur „quasiparallel“ ausführen
    - Scheduler teilt dem Prozess die CPU Zeit zu

Prozesse im Mehrprozessbetrieb

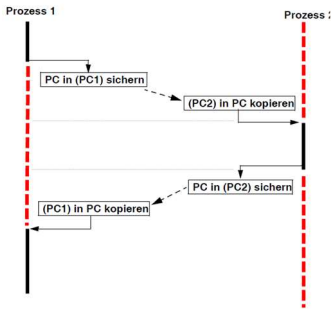
- Jeder Prozess besitzt eigene virtuelle CPU und
- Eigene Instanz mit getrenntem Adressraum und
- Kann eine Priorität zugeordnet werden
- Prozesse können unabhängig oder voneinander abhängig arbeiten => kooperierende Prozesse
- Prozesse können miteinander kommunizieren und
- Andere Prozesse erzeugen

Prozesszustände und Metainformationen

- Bei Prozessumschaltung werden Prozesszustände und Metainformationen in Prozesstabelle gesichert
- Jeder Prozess wird in einen Process Control Block (PCB) geschrieben
- PCB beschreibt
    - PID
    - Process State
    - Program Counter
    - CPU Register
    - CPU Scheduling Information
    - Memory Management Information
    - I/O State Information
    - Account Information

Prozesswechsel

- Ablauf:
- Kontextsicherung aktueller Prozesse
    - Sicherung der CPU Register in PCB1
- Auswahl des nächsten Prozesses
    - CPU Scheduling
- Kontextwiederherstellung des neuen Prozesses
    - Rückkopieren der CPU Register



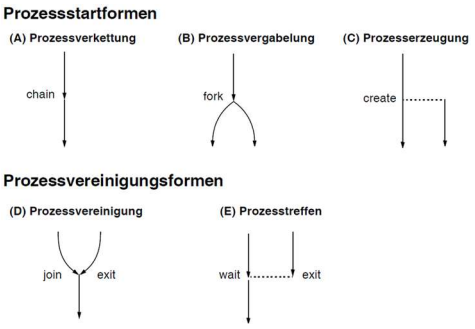
Prozesshierarchien

- Unter Linux besteht bleibender Zusammenhang zwischen Elter- und Kindprozess
- Jeder Prozess hat 1 Elterprozess und
- Kann 0 bis n Kindprozesse haben und
- Kann nicht enterbt werden
- Wird einem Prozess ein Signal gesendet erhalten das auch alle Nachkommen

Prozesserzeugung und -beendigung

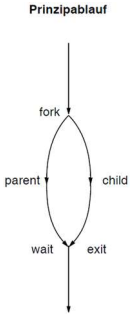
- Erzeugung erfolgt
    - Bei Initialisierung des System
    - Durch anderen Prozess
    - Durch User
    - Durch Batchbetrieb
- Beendigung erfolgt
    - Freiwillig
        - Normales Terminieren
        - Fehler
    - Unfreiwillig
        - Schwerwiegender Fehler
        - Terminierung durch anderen Prozess

Prozessstart und -vereinigung



Vergabelung durch fork()

- Zombie: Child terminiert bevor Parent wartet -> führt keinen Code mehr aus, belegt aber noch Einträge in Prozesstabelle
- Orphan: Parent terminiert vor dem Child -> wird dann vom init-Prozess adoptiert



Threads vs. Prozesse

- Prozesse: Parallele Ausführung von E-Mail Client und Textverarbeitung
- Threads: Texteingabe & Rechtschreibprüfung
- Bei Threads gibt es keinen Schutz der Daten untereinander
- Pro Thread gibt es:
    - Programmzähler (wo im Code bin ich)
    - Register (fürs Rechnen)
    - Stapel
    - Zustand (aktiv oder inaktiv)
- Threadumschaltung schneller als Prozessumschaltung

CPU-Scheduling: Rechenzeitzuteilung

- Process based scheduling
    - Zuteilung nach ganzen Prozessen -> Kern betrachtet Prozess wie single-threaded process
- Thread based scheduling
    - Zuteilung zu einzelnen Threads auf User/Kernel Ebene

CPU-Scheduling: Zuteilungsstrategien

- Nicht verdrängend (non-preemptive) -> Prozess läuft bis:
    - Er auf Ein-/Ausgabedaten wartet (blockiert)
    - Er auf anderen Prozess wartet
    - Er freiwillig auf Prozessor verzichtet
    - Keine Umschaltung bei:
        - Systemuhreninterrupt
        - Anderer Prozess ist ablaufbereit
- Verdrängend (preemptive) -> Prozess läuft bis:
    - Alles wie oben
    - Bei Systemuhreninterrupt, Zeitquantum vorbei
    - Ein wartender Prozess bevorzugt werden soll

Zuteilungsstrategien

- Anforderungen
    - Geringe Durchlaufzeit

    - Geringe Antwortzeit
    - Einhaltung Deadline
- Strategien
    - FCFS
    - SJF
    - SRT
    - RR
    - ML (Multi Level) -> Prozesse haben Priority
    - MFL (Multi Level Feedback) -> Lange Prozesse werden nach und nach heruntergestuft

Prozesszustände

- Running, ready, blocked (wartet auf ein ext. Ereignis)