Rechnerorganisation

Übersetzung:

1. Schritt: Präprozessor
– Kommentare werden entfernt
– Bearbeitung von Include und Macros
2. Schritt: Compilieren
– C wird in Assembler übersetzt
3. Schritt: Assemblieren
– Maschienencode wird erstellt
4. Schritt: Binden
– Ausführbare Datei wird erstellt
– statisch und dynamisch möglich (betrifft Bib-Funktionen)

Grundbegriffe:

– Programm: Folge von Anweisungen
– Prozess: In Ausführung befindliches Programm
  • Kann mehrere Programme ausführen
– Kompilierer: Wandelt Quellsprache in Zielsprache um
– Interpreter: Führt Programm direkt aus

Laden eines Programms:

– Statistisch gebundene Programme:
  • Von Ebene 5 auf Ebene 3
  • Läd beim Compilieren alle Komponenten in den Speicher
  • Alle Adressen werden zum Bindezeitpunkt aufgelöst
– Dynamisch gebundene Programme:
  • Von Ebene 5 auf Ebene 1
  • Läd zur Laufzeit benötigte Komponenten
  • Adressen werden beim Programmstart aufgelöst
  • Auflösung von Adressbezüge beim Übersetzen möglich

Mehrebenenmaschine:

– 5. Problemorientierte Programmierebene
– 4. Assembliersprache
– 3. Maschinenprogrammebene
– 2. Befehlssatzebene
– 1. Mikroarchitekturebene
– 0. digitale Logikebene
– Hinweise:
  • Schichten 3-5 repräsentieren virtuelle Maschine
  • Schichten 0-2 repräsentieren reale Maschine
  • Schicht 4 Logisch existent aber unwichtig geworden
– Durchlauf:
  • $5 - 4$: Kompilation
  • $4 - 3$: Assemblieren und Bindung
  • $3 - 2$: Partielle Interpretation
  • $2 - 1$: Interpretation

Programmhierachie:

– Maschinenprogramme enthalten zwei Befehlsformen:
  • Maschinenbefehle der Befehlssatzebene
  • Systemaufrufe an das Betriebssystem
– Maschinenprogramme (MaschProg):
  • ohne Compiler vom Prozessor ausführbar
  • Meist von Compiler generiert
  • Grundlage bilden Hochsprachen und Assembler
– Triumvirat:
  • MaschProg: Anwendungsprogramm + Laufzeitsystem
  • MaschProg: Benutzerebene
  • Betriebssystem
  • Zentraleinheit
  • Ausführplattform: Betriebssystem + Zentraleinheit
  • Ausführplattform: Systemebene

Betriebssystem

– Menge von Programmen der Befehlssatzebene
– Stellt ein nicht sequentielles Programm dar
– Zählt nicht zur Klasse der Maschinenprogramme
– interpretiert die eigenen Programme nur teils partiell
– sollten unterbrechbar sein
– Interpreter - Ausführung von Systemaufrufen:
  • 1. Prozessorstatus unterbrochener Programme sichern
  • 2. Systemaufruf interpretieren
  • Prozessorstatus wiederherstellen

Maschinenprogrammebene:

Zwei Sorten von Befehlen:
– unpriviligierte Befehle:
  • Wird von der CPU direkt ausgeführt
– priviligierte Befehle:
  • Werden vom Betriebssystem ausgeführt
  • explizit als Systemaufrufe codiert
  • implizit als Ausnahmen ausgelöst

Ausführungsablauf bei priviligierten Befehlen:
  • 1. CPU interpretiert Maschinencode befehlsweise
  • 2. Bei Ausnahmen startet das Betriebssystem
  • 3. Interpretiert Programme des BS befehlsweise
  • Folge von Punkt 3:
  • 4. BS interpretiert Maschinencode befehlsweise
  • 5. CPU wird zum weitermachen verleitet

Ausnahmebehandlung:

– Ausnahmebehandlung ist zwingend
– Es gibt zwei Varianten:
– 1. Trap
  • Abfangung für Ausnahmen von interner Ursache
  • Z.b. Bug im Code
  • falsche Adressierungsart oder Rechenoperation
  • Systemaufruf, Adressraumverletzung,unbekanntes Gerät
  • Seitenfehler im Falle lokaler Ersetzungsstrategien
  • synchron, vorhersagbar, reproduzierbar
  • Behandlungsmodelle: Beendigung und Wiederaufname
– 2. Interrupt:
  • Unterbrechung durch Ausnahmen von externer Ursache
  • Ein externer Prozess signalisiert einen Interrupt
  • Z.b. Beendigung einer DMA- bzw. E/A-Operation
  • Z.b. Seitenfehler im Falle globaler Ersetzungsstrategien
  • Unterbrechungsbehandlung muss Nebeneffektfrei sein
  • Aktiver Prozess wird unterbrochen, nicht abgebrochen
  • asynchron, unvorhersagbar, nicht reproduzierbar
  • Behandlungsmodell: Nur Wiederaufnahmemodell

Aktives Warten:
– Prozess frägt beim warten immer wieder nach
– Vergeudet nicht unbedingt CPU-Zeit
– Benötigt keine Unterstützung durch das Betriebssystem

Passives Warten:
– Prozess Schläft, bis er benachrichtigt wird

Betriebsarten

Stapelbetrieb:

Abgesetzter Betrieb:
– Verwendung von Satellitenrechner und Hauptrechner
– Satellitenrechner sendet Eingabe an Hauptrechner
– Hauptrechner verarbeitet die Eingabe
– Hauptrechner schickt Ergebnis an Satellitenrechner
– +: Entlastung durch Spezialrechner

Überlappte Ein-Ausgabe:
– Durch Interrupts gleichzeitige Prozesse
– Speicherdirektzugriff (DMA) statt programmierte IO
– nebenläufige Programmausführung möglich
– Problem: Leerlauf beim Auftragswechsel

Überlappte Auftragsverarbeitung:
– Verarbeitungsstrom auszuführender Programme
– Technik: Prefetching oder Auftragseinplanung
– Probleme: CPU Monopolisierung, IO Leerlauf

Abgesetzte Ein-Ausgabe: Spooling
– Abwechselnd: CPU-Stoß und IO-Stoß
– Entkopplung durch Pufferbereiche

Mehrprogrammbetrieb:
– Multiplexen der CPU
– Mehrere Programme gleichzeitig im Hauptspeicher
– Nutzung von Aktiven/Passiven Warten
– Wichtiges Tool: Adressraumschutz:
  • Vor Laufzeit: Schutz durch Eingrenzung
  • Zur Laufzeit: Schutz durch Segmentierung

Dynamisches Laden:
– Problem: Programm zu groß für den Arbeitsspeicher
– Lösung:
  • Zerteilung des Programms in kleine Teile
  • Das Nachladen ist Programmiert
  • Nachteil: Programmierer manuell einzubauen

Echtzeitbetrieb:

– System muss ständig betriebsbereit sein
– Ergebnisrückgabe innerhalb vorgegebener Zeit
– externe (physikalische) Prozesse definieren Verhalten für nicht termingerechte Ergebnisrückgabe

Terminvorgabe:
– weich:
  • Ergebnis weiterhin nutzbar
  • Ergebnis wird nur mit der Zeit immer wertloser
  • Terminverletzung ist tolerierbar
– fest:
  • Ergebnis ist wertlos
  • Ergebnis wird verworfen
  • Terminverletzung ist tolerierbar
– hart:
  • kein Ergebnis bedeutet großes Problem
  • Terminverletzung ist nicht tolerierbar

Mehrzugangsbetrieb

– Nur sinnvoll mit CPU- und Speicherschutz

Dialogbetrieb:
– mehrere Benutzer gleichzeitig
– Benutzereingaben und Verarbeitung wechseln sich ab
– Zugang über Dialogstation (z.b. Terminal)
– Problem: Monopolisierung der CPU möglich

Dialogorientiertes Monitorsystem (Hintergrundbetrieb):
– Prozesse im Vordergrund starten
– Prozesse im Hintergrund vollziehen
– mehrere Aufgaben werden parallel bearbeitet
– Problem: Hauptspeichergröße

Teilnehmerbetrieb:
– eigene Dialogprozesse werden interaktiv gestartet
– Zeitscheibe um CPU-Monopolisierung vorzubeugen

Teilhaberbetrieb:
– Client-Server-System

Systemmerkmale

Symmetrische Simultanverarbeitung:
– Mehrere Prozesse gekoppelt über gemeinsames Verindung
– Stellt homogenes System dar

Speichergekoppelter Multiprozessor:
– Alle Prozesse nutzen den Hauptspeicher
– Stellt heterogenes System dar
– Assymetrische Architektur:
  • hardware bedingter assymetrischer Betrieb
– Symmetrishce Architektur:
  • Betriebssystem legt Multiprozessorbetrieb fest

Parallelverarbeitung:
– N Prozessoren können:
  • N Programme echt parrallel ausführen
– Jeder dieser Prozessoren kann multiplexen
  • Jeder Prozessor kann einzeln parallelisiert werden

Schutzvorkehrungen:
– Jeden Prozessadressraum in Isolation betreiben
– Prozessen eine Zugriffsbefähigung erteilen
– Objekten eine Zugriffkontrolliste geben
– 1. Methode: Schutz durch selektive Autorisierung
– 2. Methode: Zugriffsrechtmatrix
– 3. Methode: Schutzring names Multics
  • Einzelne Ringabschnitte mit unterschiedlichen Rechten
  • Von außen nach innen mehr Rechte
  • ring fault bei Rechteverletzung
– 4. Methode: Schutzgatterregister:
  • Permanent benötigte Programme werden isoliert
  • Abgrenzung von den Anwendungsprogrammen
  • Schutzgatter kann manuell ausgeschalten werden

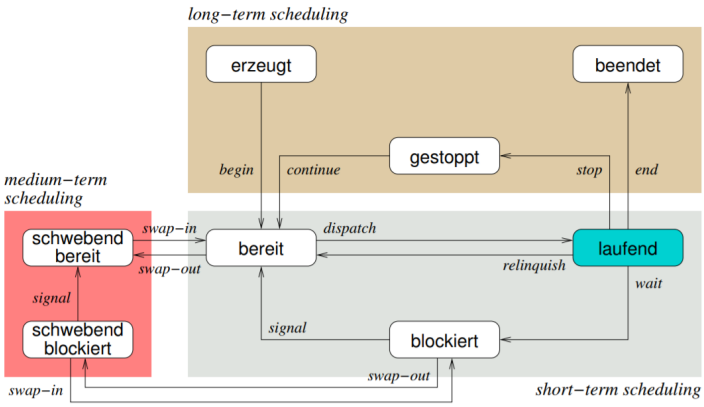
Umlagerung nicht ausführbereiter Programme:
– auch swapping genannt
– schafft Platz im Arbeitsspeicher
– Probleme: Fragmentierung, Verdichtung

Umlagerung ausführbereiter Programme:
– Nicht benötigte Teile werden ausgelagert
– Zugriff auf ausgelagerte Teile unterbricht Prozess
– Intensiver Wechsel zw. aus und einlagern

Prozessverwaltung

Begriffe:
– Einplanung: Reihenfolenbildung von Aufträgen
– Einlastung: Zuteilung der Betriebsmittel

Programmfaden:
– Einplanungseinheit für die Prozessorvergabe ist der Faden
– Lauf- und Wartephasen betreiben einen Rechner stoßartig
– Fäden überdecken passives Warten anderer Fäden



Kurzfristige Planung (short-term scheduling):
– Wichtig für Mehrprozessbetrieb
– bereit: Prozess ist in der Bereitliste
– laufend: Prozess vollzieht seinen CPU-Stoß
– blockiert: Prozess erwartet Betriebsmittelzuteilung

Mittelfristige Planung (medium-term scheduling):
– Anhand der Umlagerung kompletter Programme
– schwebend bereit:
  • Das Exemplar eines Prozesses ist augelagert
  • Die Einlastung des Prozesses ist außer Kraft
– schwebend blockiert:
  • ausgelagerter ereigniserwartender Prozess

Langfristige Planung (long-term scheduling):
– Nutzung von Lastkontrolle
– erzeugt: fertig zur Programmverarbeitung
– gestoppt: erwartet Fortsetzung / Beendigung
– beendet: Prozess erwartet seine Entsorgung

Gütemerkmale:
– Benutzer:
  • Antwort-/Durchlaufzeit, Termine, Vorhersagbarkeit
  • Prozessausführung unabhängig von der Systemlast
– System:
  • Durchsatz, Auslastung, Gerechtigkeit
  • Dringlichkeit, Lastausgleich
  • Gleichmäßige Betriebsauslastung

User-Threads:
– Threads mit nichtpriviligiertem Code
– blockierende syscalls blockieren andere User-Threads
– Schedulingstrategie vom Programmierer definiert
– Können effizient umgeschalten werden
– Nicht für Multiprozessorbetrieb ausgelegt

Kernel-Threads:
– Threads mit priviligiertem Code
– Erzeugung teurer als User-Threads
– blockierende syscalls blockieren keine anderen Threads
– Schedulingstrategie durch Betriebssystem vorgeben
– Umschalten findet immer im Systemkern statt
– Kann für Multiprozessorbetrieb sinnvoll sein

Klassifikation der Prozesseinplanung:
– kooperative Planung (FCFS):
  • für voneinander abhängige Prozesse
  • CPU-Entzug nicht zugunsten anderer Prozesse
  • laufender Prozess gibt CPU nur mit Systemaufruf ab
  • CPU Monopolisierung möglich
– preemptive Planung (RR, VRR, SRTF):
  • für voneinander unabhängige Prozesse
  • CPU-Entzug zugunsten anderer Prozesse möglich
  • ereignisbedingte Verdrängung laufender Prozesse
  • CPU Monopolisierung nicht möglich

– deterministische Planung:
  • Alle Prozesszeiten sind bekannt
  • Einhaltung von Zeitgarantien sichergestellt
  • CPU-Auslastung vorhersagbar
– probabilistische Planung (SPN, SRTF, HRRN):
  • Prozesszeiten unbekannt und nur approximierbar
  • Keine Zeitgarantie möglich
  • CPU-Auslastung nicht vorhersagbar

– statische Planung:
  • Vor Betrieb des Prozesses
  • Keine Planung im laufenden Betrieb
  • Ergebnis der Vorberechnung ist kompletter Ablaufplan
  • Begrenzt auf strikte Echtzeitsysteme
– dynamische Planung:
  • während des Betriebs des Prozesses
  • Stapelsysteme, interaktive Systeme, verteilte Systeme
  • schwache und feste Echtzeitsysteme

– assymetrische Planung:
  • wichtig bei assymetrischen Multiproessorsystemen
  • optional auf symmetrischen Multiprozessorsystemen
  • ungleiche Verteilung mehrer Prozesse auf CPUs
– symmetrische Planung:
  • identische Prozessoren
  • Lastausgleich bei Verteilung von Prozessen auf CPUs

Bekannte Verfahrensweisen:
– FCFS (First Come First Serve):
  • Prozesseinplanung nach Reihenfolge der Ankunft
  • Prozesse mit langen Rechenstößen werden begünstigt
  • Prozesse mit kurzen Rechenstößen werden benachteiligt
  • −: Konvoieffekt (Abwechseln kurze/lange Prozesse)
– RR und VRR: (Round Robin)
  • Verdrängende Variante von FCFS
  • Verringerung der Benachteiligung aus FCFS
  • Prozesse werden nach ihrer Ankuftszeit eingeplant
  • Prozessumplanung in regelmäßigen Zeitabständen
  • Durch Zeitschreibe entsteht CPU-Schutz
  • Weiterhin Problem des Konvoieffekts
  • VRR: Variable Zeitscheibe, nicht voll-verdrängend
  • VRR: Zusätzlich Vorzugsliste für IO
– SPN , HRRN , SRTF: (Shortest Process next)
  • SPN: Schnellster Prozess als nächstes
  • SPN: Prozessverhungern möglich
  • HRRN: Hungerfreies SPN durch Alterungswichtung
  • SRTF: Prozesseinplanung nach Bedienzeit
  • SRTF: unregelmäßige Prozessumplanung
  • SRTF: Verdrängt wird in eine Bereitliste
– MLQ, MLFQ:
  • Kombination der obrigen Strategien
  • Prozesseinplanung nach Typ und Ankuftszeit
  • Unregelmäßige Prozessumplanung
  • Jede Queue hat eigene Einplanungstrategie
  • Gut für kurze Prozesse, schlecht für lange

Prozesssynchronisation

Grundbegriffe:
– Kausalität: Beziehung zw. Ursache und Wirkung
– Gleichzeitige Prozesse:
  • vertikal: time sharing (multiplexen)
  • horizontal: multiprocessing
– Gekoppelte Prozesse: Zugriff auf gleiche Daten
– Sequentialisierung:
  • Koordinierung durch atomare Operationen
– unsicherer Zustand:
  • früher oder später kommt es zu Verklemmung
– Einseitige Synchronisation:
  • Nur ein beteildigter Prozess
  • erfolgt logisch oder bedingt
– Mehrseitige Synchronisation:
  • alle der beteildigten Prozesse betroffen
  • blockierend oder nicht-blockierend möglich

Nebenläufigkeit:
– Keine Abhängigkeit zw. Ereignissen
  • Datenabhängigkeit darf nicht verletzt werden
  • Zeitbedingung anderer darf nicht verletzt werden
– Ursachen:
  • Interrupts / preemptives Scheduling
  • Mono- Multiprozessor-Threads / Unix-Prozess-Signale

Verklemmung:
– Deadlock: gutartig, Prozesse sind blockiert
– Livelock: bösartig, Prozesse zw. laufend und bereit
– notwendige Bedingung:
  • 1. wechselseitiger Ausschluss
  • 2. Nachforderung eines oder mehrerer Betriebsmittel
  • 3. Unentziehbarkeit der zugeteilten Betriebsmittel
– notw. und hinreichende Bedingung:
  • 4. zirkuläres Warten
  • reicht vorzubeugen um Dealocks zu vermeiden
  • kann mittels Virtualisierung realisiert werden

Philosophenproblem:
– 5 Philosophen 5 Gablen
– Philosoph benötigt 2 Gabeln zum Essen
– Alle nehmen gleichzeitig linke Gabel
– Alle Philosophen verhungern

Blockierende Synchronisation:
– Locking Techniken wie Semaphore
– Kritische Abschnitte, vor welchen blockiert wird
– −: Verklemmung, Effizienzverlust

Nicht-blockierende Synchronisation:
– Keine kritischen Abschnitte
– Verwendung von atomaren Operationen
  • Prozessor-Befehle wie Compare-and-Swap
  • nebenläufigkeitsfreie Variablenmodifikation ohne Locks
– −: hohe Komplexität und Verhungern möglich

Semaphoren:
– Krit. Abschnitt mit wechselseitigem Ausschluss sichern
– Eignen sich für ein- und mehrseitige Synchronisation
– Ganzzahlige Variable mit zwei Operationen:
  • P : wait();
  • V : release();
– Operationen sind logisch und physisch unteilbar
– Auf P und V maximal ein Prozess gleichzeitig:
  • Prozesse werden in Warteschlangen gehalten
  • Nichtpassierende Prozesse werden schlafengelegt
– Atomarität wird auf Befehlssatzebene umgesetzt
  • nicht durch "normale" C-Anweisungen umsetzbar
– Mutex: eine Semaphorenspezialisierung
  • Prüft Eigentümerschaft bei Freigabe
  • lässt Freigabe damit bedingt zu
  • für einseitige und mehrseitige Synchro. geeignet

Monitor:
– Monitore sind abstrakte Datentypen
– Monitor Behandelt Parallelitätsprobleme automatisch
– Bei Blockierung muss Prozess den Monitor verlassen
– Konzept:
  • Mehrere Warteschlangen (WS)
  • Prozesse warten immer außerhalb des Monitors
  • MonitorWS: Prozesse warten auf Monitoreintritt
  • EreignisWS:Prozesse warten auf Wartebedingungsende
  • Verwendet bei Blockierung Bedingungsvariablen
  • nicht blockierend: Vorrang Signalgeber
  • blockierend: Vorrang Signalnehmer
– Hansen-Methode (blockierend):
  • Programmier-Primitive: wait() / broadcast()
  • Signalisierung lässt Signalgeber den Monitor verlassen Nachdem er alle Signalnehmer bereit gesetzt hat
– Hoare-Methode (blockierend):
  • Programmier-Primitive: signal()
  • Signal 1: Versetzt laufenden Prozess ins Warten
  • Signal 2: Prozess aus Warteschlange wird fortgesetzt
  • Signalisierung lässt Signalgeber den Monitor verlassen und genau einen Signalnehmer fortsetzen (atomar)
– Mesa-Methode (nicht blockierend):
  • Programmier-Primitive: signal()
  • signal unterbricht laufenden Prozess nicht
  • signal verschiebt Prozess in Monitorwarteschlange
  • Signalisierung lässt Signalgeber im Monitor fortfahren nachdem bereit-setzen eines oder aller Signalnehmer

Betriebsmittelverwaltung

Betriebsmittelklassifikation:
– wiederverwendbare: (begrenzt)
  • Anforderung durch mehrseitige Synchronisation
  • Zusätzliche Unterscheidung: teilbar und unteilbar
  • Werden belegt, benutzt und freigegeben
  • Beispiel: CPU, RAM, GPU
– konsumierbare: (unbegrenzt):
  • Anforderung durch einseitige Synchronisation
  • Werden produziert, empfangen, benutzt und zerstört
  • Beispiel: Signale und Traps

Ziele:
– Durchsetzung der vorgegebenen Betriebsstrategien
– Optimale Realisierung in Bezug auf relevante Kriterien
– Wichtig: Keine Verhungern und keine Verklemmung:

Aufgaben:
– Buchführung über vorhandene Betriebsmittel
– Steuerung der Verarbeitung von Anforderungen
– Betriebsmittelentzug bei fehlerhaften Prozessen

Verfahrensweisen:
– Statisch:
  • Aufgabenbewältigung vor der Laufzeit
  • Risiko: suboptimale Betriebsmittelauslastung
– Dynamisch:
  • Aufgabenbewältigung während der Laufzeit
  • Risiko: Verklemmung abhängiger Prozesse

Speicherverwaltung: Adressräume

Reale Adressen:
– lückenhafter, wirklicher Hauptspeicher
– Die realen Adressen, von der Hardware gegeben
– Vorgegebene Adressen und Adressräume
Logische Adressen:
– lückenloser, wirklicher Hauptspeicher
– Getrennte reale Adressbereiche werden linearisiert
– Zweck:
  • Sicherheit, Virtualisierung, bessere Verwaltung
Virtuelle Adresse:
– lückenloser, scheinbarer Hauptspeicher
– entkoppelt von der Lokalität im Arbeitsspeicher
– Kann größer als der Arbeitsspeicher sein (paging)
– Adresse wird auf den Hauptspeicher abgebildet
– Dafür gibt es dann Abbildungstabellen
– Navigation zum Speicher durch Abbildungstabellen

Eindimensionaler Adressraum:
– Typische Seitengröße bei Seitenadressierung: 4096
– Adressaufbau: Seitennummer p und Offset o (Versatz)
– Zusätzlich wird eine Seitentabelle benötigt
– Adressbildung:
  • Mit p Adresse in Seitentabelle Suchen
  • p in Zahl umrechnen und damit Eintragsindex suchen
  • Bei mehrstufigen: p gleichmäßig Aufteilen
  • o wird so wie es ist weiterhin übernommen
Zweidimensionaler Adressraum:
– Aufteilung des Adressraums in Segmente
– Zweikomponenten Adresse:
  • Segmentname S und Adresse A mit p und o (siehe oben)
– Adressbildung (ohne p und o):
  • Mit Segmentname auf Segmenttabelle zugreifen
  • Tabellen Eintrag mit Adresse verrechnen
– Adressbildung (mit p und o):
  • Mit Segmentname auf Segmenttabelle zugreifen
  • Mit diesen Eintrag auf die Seitentabelle zugreifen

Private Adressräume:
– Illusion eigenes Adressraums für BS und Maschinenprogs.
– Spezialhardware verhindert ausbrechen aus Adressraum
– Sinn: strikte Isolation ganzer Adressräume

Seitentabelle Berechnung bei Seitenadressierung:
– Adressraum : (Speicherseite : Eintrag)
  • Jeder Seite muss adressiert werden
  • Speicherseite : Eintrag – Einträge pro Seite
  • Adressraum : Einträge pro Seite – Ergebnis

Seitendeskriptor:
– Von MMU vorgegebener Verbund von Attributen:
  • seitenausgerichtete reale Adresse
  • Schreibschutzbit / Präsensbit
  • Referenzbit / Modifikationsbit
– Seitendeskriptor des BS in shadow page table
Filedeskriptor:
– Prozesslokale Integerdatei
– Wird von einem Prozess verwendet
– Zugriff auf Dateien, Geräte, Pipes, Sockets
Segmentdeskriptor:
– Von MMU vorgegebener Verbund von Attributen:
  • Basis: Segmentanfang im Arbeitsspeicher
  • Limit: Segmentlänge als Anzahl der Granulate
  • Typ (Text, Daten, Stapel)
  • Zugriffsrechte (lesen, schreiben, ausführen)
  • Expansionsrichtung / Präsensbit

Speicherbereiche

Code-Segment:
– Auszuführenden Programmcode
Data-Segment:
– inizialisierte Globale und statische Variablen
Heap-Segment:
– Von malloc reservierter Speicher
Block-Storage-Segment (BSS):
– nicht inizialisierte globale/statische Variablen
Stack-Segment:
– lokale (Pointer-) Variablen
– Übergebene Argumente
– Platz für Zwischenergebnisse
– Rücksprungadresse / Frame-Pointer

Adressraumschutz durch Eingrenzung:
– Begrenzungsregister legen Adressbereich fest
– Dieser ist im physikalischen Adressraum
– Auf diesen werden Speicherzugriffe beschränkt

Seitennummer und Versatz berechnen:
– Bits der Seitengröße berechnen
– logische Adresse unterteilen:
  • Hintere Bits sind für Seitengröße
– beide Adressen mit 0er vorne Auffüllen

Platzierungsstrategien:

– Bitkarte:
  • für Hohlräume fester Größe
– Lochliste:
  • für Hohlräume variabler Größe

Zuteilungsverfahren:
– Alle folgenden sind Listenbasiert
– Bei allen kann externer Verschnitt auftreten
– Freispeicherverschmelzung: first-fit schneller als worst-fit
– worst-fit:
  • absteigend nach Lochgröße sortiert
  • Größtes passendes Loch wird gesucht
  • +: Suchaufwand ist klein
  • −: Speicherverschnitt ist groß
– best-fit:
  • aufsteigend nach Lochgröße sortiert
  • kleinstes passendes Loch wird gesucht
  • +: Speicherverschnitt ist klein
  • −: Suchaufwand ist groß
– first-fit: (nach Adresse sortiert)
  • Erstes passendes Loch nutzen
  • +: Suchaufwand ist klein
  • −: Speicherverschnitt ist groß
– next-fit: (nach Adresse sortiert)
  • Round-Robin-Variante von first-fit
  • Beginnt Suche beim zuletzt zugeteilten Loch

– Interne Fragmentierung:
  • Speicherblöcke nur teilweise befüllt
  • Ist (durch das Betriebssystem) unvermeidbar
  • Buddy: Entstehung durch das Aufrunden
  • Kein Zugriffsfehler bei Zugriff auf ungenutzen Bereich
– Externe Fragmentierung:
  • Angeforderte Größe zu groß für jedes Loch
  • Dadurch Zerstückelung eines Speicherraums
  • Verringerung durch Speicherblock-Verschmelzung
  • Auflösung durch Kompaktifizierung möglich

Buddie-Verfahren:
– **Datenblock kann vorinitialisiert sein!!!**
– Hier Beschreibung des Binären Buddyverfahrens
– Neuer Speicher wird zu nächster 2er Potenz aufgerundet
– 2^n wird in 2^n großem Block gespeichert!
– Passender Speicher wird itterativ gesucht und verwendet
– Ist kein Speicher vorhanden:
  • Halbieren des ersten größeren Speichers, bis er passt
– Aufeinanderfolgende Buddyadressen um ein Bit different
– Lochlisten Einträge:
  • Links mit Zweierpotenzen sind Blockgrößen
  • Daneben: Adresse/n mit freiem Block jeweiliger Größe
– Bemerkungen:
  • Es gibt keinen externen Verschnitt

1. p1 = Malloc(200) / 2. p2 = Malloc(128) / 3. p3 = Malloc(500) / 4. free(p1) / 5. free(p2)

<table>
<tr><td colspan="4">1024</td></tr>
<tr><td colspan="3">512</td><td>512</td></tr>
<tr><td>p1</td><td colspan="2">256</td><td>512</td></tr>
<tr><td>p1</td><td>p2</td><td>128</td><td>512</td></tr>
<tr><td>p1</td><td>p2</td><td>128</td><td>p3</td></tr>
<tr><td>256</td><td>p2</td><td>128</td><td>p3</td></tr>
<tr><td>256</td><td colspan="2">256</td><td>p3</td></tr>
<tr><td colspan="3">512</td><td>p3</td></tr>
</table>

Ladestrategien:

– Umsetzung durch MMU (Memory Management Unit)
– Der TLB (Tranlation Lookaside Buffer) untersützt MMU
  • spezieller Cache mit Infos vom Seitendeskriptor
  • Speichert Ergebnis: Logisch – physikalisch

Arbeitsweisen:
– Demand-Paging:
  • Läd Adresse erst nach Ansprechen in den RAM
  • Nichtbenutzte Seiten werden ausgelagert
– Anticipatory:
  • Vorausladen bzw. Prefetching
  • Zur Vermeidung von Folgefehler
  • Heuristik liefert Hinweise über zukünftige Zugriffe

Fehler:
– Page/Segment-fault:
  • Speicher/Segment nicht im Hauptspeicher
  • Entstehen z.b. durch ungültige Zeiger
  • Present-Bit des Adressraums ist nicht gesetzt
  • Adresse ist somit vmtl. ausgelagert (Festplatte)
  • MMU löst einen Trap aus
  • Zugriffsfehler: Schwerer Seitenfehler

Ersetzungsstrategien:

Lokale vs. globale Seitenersetzung:
– lokal:
  • Suchraum: Menge residenter Seiten des Prozesses
  • ein Seitenfehler ist vorhersag-/reproduzierbar
– global:
  • Suchraum: Menge alles residenter Seiten aller Prozesse
  • ein Seitenfehler ist unvorhersag-/unreproduzierbar

Lokalitätsprinzip:
Wenn ein Prozess eine Stelle in seinem Adressraum referenziert, dann wird er wahrscheinlich dieselbe Stelle oder eine andere Stelle in direkter Umgebung referenzieren

Seitenflatter:
– Seitenein-/auslagerungen dominiert Systemaktivität
– Sofortiges Einlagern kürzlich ausgelagerter Seiten
– ein mögliches Phänomen der globalen Seitenersetzung
– Verschwindet meist von alleine wieder

Freiseitenpuffer:
– Um schneller einen freien Seitenrahmen zu finden
– FIFO mit Cache für potentiell zu ersetzende Seiten
– Seiten im Cache werden wie bei SSD überschrieben
– Zugriffsfehler auf Seite im Cache:
  • Reklamierung und Neusetzen des Präsensbits

Arbeitsmenge:
Die kleinste Sammlung von Programmtext und -daten, die in einem Hauptspeicher vorliegen muss, damit effiziente Programmausführung zugesichert werden kann.

Strategien:
– FIFO (First in First out):
  • Ersetzt wird zuletzt eingelagerte Seite
  • Implementierung: Verkettete Liste
  • Mehr Seitenrahmen führen zu mehr Seitenfehlern
– LFU (Least frequently used):
  • Ersetzt wird die am seltensten referenzierte Seite
  • Implementierung durch Zähler
– LRU (Least recently used):
  • Ersetzt wird am längsten nicht referenzierte Seite
  • Siehe: Verschiedenes

Dateisysteme:

Kontinuierliche Speicherung:
– Dateispeicherung in aufsteigend nummerierten Blöcken
– Zur Identifizieren: Anfangsblock und Größe
– +: Schnelles Lesen, da keine externe Fragmentierung
– +: Geeignet für Echtzeitsysteme
– +: Freien Speicher finden ist schwierig
– −: Interne Fragmentierung
– −: dynamische Erweiterung sehr aufwändig

Verkettete Speicherung:
– Speicherung von Daten in verketteten Blöcken (Liste)
– +: Dynamische Erweiterung einfach möglich
– −: hohe Fehleranfälligkeit / externe Fragmentierung

Indiziertes Speichern:
– Speicher in gleich große Blöcke aufgeteilt
– Pro Datei ein (evtl. mehrfach verketteter) Inode der auf die Datenblöcke in der richtigen Reihenfolge zeigt.
– Extents: Indexblock zeigt auf Datenblock aber diese Daten sind Pointer auf anderen Speicher

Freispeicherverwaltung:
– Speicherbereich wird in gleich Große Blöcke unterteilt
– Bitvektor speichert pro Block belegt / nicht belegt
– verkettete Liste repräsentiert freie Blöcke
– Besser: Aufeinanderfolgende Blöcke Zusammenfassen
– Besser: Statt Liste: Datenblöcke zeigen auf freien Speicher

Unix Blockstruktur:

Bootblock	Superblock	Inodeliste	Datenblock

– Boot: Infos zum Laden eines initialen Programmes
– Super: Anzahl Blöcke / Inodes und Attribute
– Inode: Dateiart,UserId,rights,größe,linkzähler,hard links
  • Zusätzlich Zeiten: Erstellung, Zugriff, letzte Änderung

Journaling-File-Systems:
– Änderungen als Transaktionen dokumentiert im Log-File
– Log wird vor der Transaktion beschrieben
– Inkonsistenz wird beim Booten durchs Log-File vermieden
– −: Undo nach erkanntem Transaktionsproblem
– Optimiert: Checkpoint (Platte in konsistentem Zustand)
  • Log-File-Einträge bis checkpoint löschen

Raid 0: Gestreifte Platte (Paritätsfrei):
– Daten werden über mehrere Platten gespeichert
– +: Schnelles Lesen und schreiben
– −: Systemausfall bei Plattenausfall / keine Fehlertoleranz

Raid 1: Gespiegelte Platten (Paritätsfrei)
– Zwei oder n Platten mit gleichen Daten
– Verknüpfung von Raid 0 und Raid 1 möglich
– +: n-1 Platten können ausfallen
– +: Schnelles Lesen (Nicht Schreiben)
– −: Hoher Speicherbedarf

Raid 4: Paritätsplatte:
– Daten werden über mehrere Platten verteilt gespeichert
– Eine Paritätsplatte P speichert Paritätsinformationen
– +: Eine Platte kann ausfallen und schnelles Lesen
– +: mind. 3 Platten, beliebig viele mehr möglich
– −: P hoch belastet, da bei jedem Schreiben einbezogen

Raid 5: Verstreuter Paritätsblock:
– Paritätsblöcke werden über alle Platten verteilt
– Vor und Nachteile wie Raid 4
– +: Last der Paritätsplatte wird verteilt
– Raid 6: zwei Paritätsblöcke pro Platte

Verschiedenes:

Hacking:

– strcpy() und strcat() als Sicherheitslücken

Least Recently Used:

– Periodische Unterbrechung – Hintergrundrauschen
– Altersstruktur:
  • Benötigt in Software implementiertes Schieberegister
  • Zusätzlich Zeitgeber für periodische Unterbrechung
  • Referenzbitprüfungs nach jedem Zeitintervall
  • Ersetzt: Global älteste Seite im Aging Register

tick	Ref.	aging register
		00000000
n	1	10000000
n+1	0	01000000
n+2	1	10100000

– Zweite Chance:
  • Basiert auf FIFO, um ein billiges LRU nachzubauen
  • Nutzt Zeitintervall zur periodischen Unterbrechung
  • Ersetzung der Ersten Seite bei der Referenzbit 0 ist
  • Wenn alle Referenzbits 1: FIFO Prinzip nutzen!
– Dritte Chance:
  • Übernimmt alle Merkmale von Second Chance
  • Zusätzliche Nutzung des Modifikationsbits
  • (0,0): unbenutzt = beste Wahl
  • (1,0): beschrieben = Naja
  • (0,1): gelesen = Naja
  • (1,1): kürzlich beschrieben = Schlecht
  • Zwei Umläufe für jede eingelagerte Seite

Wertetabelle:

Hex	Dez	Binär	Name	Name	2er	Dez	Pot
0	0	0000	KiB	KibiB	2^{10}	2	2^1
1	1	0001	MiB	MibiB	2^{20}	4	2^2
2	2	0010	GiB	GibiB	2^{30}	8	2^3
3	3	0011	TiB	TebiB	2^{40}	16	2^4
4	4	0100	PiB	PebiB	2^{50}	32	2^5
5	5	0101				64	2^6
6	6	0110				128	2^7
7	7	0111				256	2^8
8	8	1000				512	2^9
9	9	1001				1024	2^{10}
A	10	1010	Name	10er	2er	2048	2^{11}
B	11	1011	kB	10^3	2^{10}	4096	2^{12}
C	12	1100	MB	10^6	2^{20}	8192	2^{13}
D	13	1101	GB	10^9	2^{30}	16384	2^{14}
E	14	1110	TB	10^12	2^{40}	32768	2^{15}
F	15	1111	PB	10^15	2^{50}	65536	2^{16}

Gewichtsklasse von Prozessen:

– Schwergewichtiger Prozess:
  • Eigener Adressraum und Ausführungsfaden
– Leichtgewichtiger Prozess:
  • teilt sich Adressraum mit anderes Prozessen
– Federgewichtiger Prozess:
  • weder eigener Adressraum noch Ausführungsfaden
  • wird nicht vom Kernel verwaltet

Kleinigkeiten zum Einfügen:

Inode-Aufgaben:

Gegeben:
/Desktop/Inode:

71	d...	3	...	4096	...	.
66	d...	9	...	4096	...	..
58	d...	2	...	4096	...	deeper
27	l...	1	...	53	...	file.txt – deeper/file.txt
93	-...	1	...	25	...	prog.c

/Desktop/Inode/deeper:

58	d...	2	...	4096	...	.
71	d...	3	...	4096	...	..
94	-...	1	...	6	...	file.txt

Lösung:
– Inode mit: Ordner, Datei, Verknüpfung
– Ordner: Inhalt + st_ino + . + ..
– Datei: Inhalt unbekannt: ???
– Verknüpfung: Verknüpfte Datei
– Alle Inode (st_ino) Nummer müssen abgearbeitet werden
– Jede Inode Nummer nur einmal

Inode			Inhalt Datenblöcke	
st_ino:	71		71 .	66 ..
st_nlink:	3	⟶	58 deeper	27 file.txt
st_size:	4096		93 prog.c	

st_ino:	66		66 .	71 Inode
st_nlink:	9	⟶		
st_size:	4096			

st_ino:	58		58 .	71 ..
st_nlink:	2	⟶	94 file.txt	
st_size:	4096			

st_ino:	27		
st_nlink:	1	⟶	deeper/file.txt
st_size:	53		

st_ino:	93		???	
st_nlink:	1	⟶		
st_size:	25			

st_ino:	94		???	
st_nlink:	1	⟶		
st_size:	6			

Bemerkung:
– Root Knoten zeigt mit . und .. auf sich selbst
– Inode auf Datei wird getrennt von Datei gespeichert

UNIX-UFS-Dateisystem:

– Pfadnamen ohne '/' am Anfang:
  • Werden relativ zum aktuellen Verzeichnis interpretiert

– Hierarchisch organisierter Namensraum:
  • Die Absteigende Baumstruktur bei Dateisystemen
  • Kontext (Verzeichnis) als flachen Namensraum
  • Gleiche Namen in unterschiedlichem Kontext möglich
  • Pro Verzeichnis mehrmals gleicher Name Nicht möglich

Dateisystem:
– Symbolic-Links:
  • Können existieren, obwohl Ziel bereits gelöscht
  • Kann auf Verzeichnisse verweisen
– Hard-Link:
  • Pro Reguläre Datei mind. einer im selben Dateisystem
  • Pro Verzeichnis mind. 2 Hard-Links
  • Können nur auf Dateien verweisen
  • Auf Datei: Anlegen nur im selbem Dateisystem

Prozesskontrollblock:
– Als Tabelle vorstellbar mit folgenden Einträgen:
  • Zustande des Prozesses / Befehlszähler
  • CPU-Register / Stack-Pointer
  • Zustande der geöffneten Dateien / aktuelles Verzeichnis
  • Verwaltungsinformationen / UID Eigentümer