•Ziele eines Betriebssystems:

–Anpassung der Benutzerwelt an die Maschinenwelt

–Organisation und Koordination des Betriebsablaufs

–Protokollierung des Betriebsablaufs

•Randbedingungen:

–Die vorhandenen Betriebsmittel sind so einzusetzen, dass Engpässe bzw. Überlastungssituationen vermieden werden.

–Das Betriebssystem soll für die Durchführung seiner Aufgaben einen geringen Rechenaufwand benötigen (Payload versus Overhead).

–Ein Betriebssystem hat robust gegen fehlerhafte Benutzerprogramme zu sein.

–Daten und Programme sollten gegen unerlaubten Zugriff und gelegentliche Ausfälle der Hardware abgesichert werden.

•Dienste für den Betreiber des Systems:

–Erfassung von Verbrauchsdaten („Accounting“)

–Sicherheitsüberwachung: Zugangskontrolle, Zugriffsschutz, Verschlüsselung („Authentification“, „Authorisation“)

–Operationen zur Administration (z.B. Abbruch von Prozessen)

–Testfunktionen (z.B. Entdeckung fehlerhafter Spuren auf Festplatten)

–Funktionen zur Initialisierung des Systems

Ein Betriebssystem stellt eine virtuelle Ablaufumgebung (abstrakte Maschine) bereit, die durch geeignete Abstraktionen die Anwendungsprogramme von der Komplexität der vorhandenen Hardware abschirmt.

•Nachteile der Batchverarbeitung:

–Menschliche (teure) Operateure mit fehleranfälliger „Stapelei“

–Beim Einlesen des nächsten Jobs und beim

Speichern/Drucken der Ausgabe ist CPU blockiert

•Lösung: Speicherpartitionierung

–Jeder Job bekommt eigenen Speicherbereich (Partition)

–Während Wartezeiten werden die Jobs getauscht

–Heap organisiert die Speicherverwaltung

•SimultaneousPeripheralOperation On Line

–Die Multiprogrammierung führt zu besserer Ausnutzung der Maschinen

–Günstigere Magnetband-Spulen (engl. „spools“) als Zwischenspeicher

–Parallel zur Job-Verarbeitung lädt das BS neue Jobs vom Band in den Speicher

–Auch die Ausgabe erfolgt nicht direkt auf das Gerät (Drucker), sondern über schnelle Warteschlangen (Spooler)

•Vorteile des Spoolings:

–Geschwindigkeit

•Immer dann effizient, wenn die Ausgabe oder Eingabe viel schneller ist, als die eigentliche Verarbeitung (z.B. Druckaufträge)

–Mehrfachnutzung

•Prozesse können sich Peripheriegeräte teilen, ohne sich zu blockieren

–Persistenz

•Der Spooler-Speicher muss nicht sofort nach Abarbeitung gelöscht werden und fehlerhafte Arbeiten können wiederholt werden

•VLSI –VeryLarge ScaleIntegration

–Sehr kleine Maschinen mit teilweise völlig neuer Architektur

–Die Computer-Firmen starteten in den 1970er Jahren erste Versuch für neue BS

–Insbesondere Plattform-Unabhängigkeit war ein wichtiges Ziel

•CP/M –Control ProgramforMicrocomputers

–Abstraktionsschichten:

•BIOS –Basic Input Output System von Hersteller

•Alle anderen Schichten Hersteller-unabhängig

–Probleme:

•Beschränkter Support für Festplatten

•Keine Unterordner, etc.

•*Echtzeitverarbeitung*(real-time processing):

–Steuerung von Maschinen in zeitsensitiven Anwendungen

–Reaktionszeiten unterliegen definierten, festen Zeitschranken.

–Das Betriebssystem schaltet zeitschrankengesteuert zwischen Programmen um.

•*Verteilte Verarbeitung* (distributedprocessing):

–Verarbeitung wird auf mehreren Prozessoren verteilt.

–In *eng gekoppelten* Systemen (Multiprozessorsysteme) benutzen die Prozessoren einen gemeinsamen Speicher und Takt.

–In *lose gekoppelten* Systementeilen die Prozessoren weder Speicher noch Takt. Die Prozessoren kommunizieren über explizite Kommunikationsverbindungen.

–Hier spielen beide Komponenten, Betriebssysteme und Netze, eine ausschlaggebende Rolle.

•Wichtige *Register*(Speicher) auf der CPU:

–Programmzähler (InstructionPointer)

•Zeigt die aktuelle Stelle im bearbeiteten Programm

–Stack-Register (Stack Pointer)

•Kann von Programmen als Stack in der CPU verwendet werden

–ProgramStatus Word (PSW) / Status Register / FlagRegister

•Informationen über den Zustand des Prozesses

•Benutzermodus (usermode):

–Der Prozessor bearbeitet Instruktionen von Anwendungsprogrammen.

–Der Befehlssatz des Prozessors und die verfügbaren Register sind auf eine Teilmenge beschränkt (unprivilegedinstructionset).

–Zugriff auf die Hardware ist entsprechend den Möglichkeiten der verwendeten Hardware beschränkt (hardwareprotection).

–Direkte Zugriffe auf andere in Ausführung befindliche Programme sind verboten.

•Betriebssystemmodus (system/kernelmode):

–Der Prozessor arbeitet Instruktionen des Betriebssystems ab.

–Alle Befehle und Register des Prozessors sind verfügbar (privilegedinstructionset).

–Direkter, uneingeschränkter Zugriff auf die gesamte Hardware.

–Manipulation interner Daten laufender Programme möglich.

•Systemaufrufe (systemcalls, supervisorcalls, softwaretraps)

–Synchron zur laufenden Applikation (z.B. count= read(file,buffer,size))

–Parameter werden über Register, den Stack oder einen Parameterblock übergeben

•Hardware Unterbrechungen (hardwareinterrupts)

–Asynchron zur laufenden Applikation (z.B. Empfang einer Nachricht von Netzwerkkarte).

•Hardware Traps

–Synchron zur laufenden Applikation (z.B. Division durch 0)

–Fehler wird vom Betriebssystem an die verursachende Applikation weitergeleitet.

•Software Unterbrechung (softwareinterrupts)

–Asynchron zur laufenden Applikation

•Monolithische Betriebssysteme:

–Das Betriebssystem ist eine Sammlung von Prozeduren und Funktionen ohne Struktur ("The bigmess.", A. Tanenbaum).

–Jede Prozedur im Betriebssystem kann beliebige andere Prozeduren aufrufen.

–Jede Prozedur im Betriebssystem kann beliebige Datenstrukturen ändern.

–Funktionen, die gemeinsam eine Teilfunktion des Betriebssystems realisieren, werden oftmals gruppiert, was aber meist nicht zur strengen Kapselung führt.

•Geschichtete Systeme:

–Betriebssystem besteht aus mehreren Schichten, wobei die Schicht n+1 aufbauend auf den Funktionen der Schichten 1,...,n neue Funktionen bereitstellt.

–Modulare Struktur vereinfacht die Fehlersuche und die Qualitätssicherung.

–Festlegung der Funktionalität der einzelnen Schichten schwierig.

–Strikte Einhaltung der Schichten und strenge Parameterprüfung an den Einstiegspunkten der Schichten erhöht den Overhead.

•Virtuelle Maschinen:

–Trennung der Funktion multi-programmingvon den übrigen Funktionen eines Betriebssystems.

–Emulation der Hardware zu sogenannten virtuellen Maschinen (VM).

–Verschiedene Betriebssysteme können gleichzeitig auf den virtuellen Maschinen laufen.

–Die virtuellen Maschinen werden durch ein Monitorprogramm (virtual machinemonitor, VMM) voneinander getrennt.

•Client/Server Betriebssysteme:

–1. Ziel: Kleinere Betriebssystemkerne (micro kernel) durch die Auslagerung von Teilfunktionen in spezielle Server.

–2. Ziel: Server können im Netzwerk verteilt realisiert werden.

–Systemaufrufe werden vom Betriebssystemkern an den zuständigen Server weitergeleitet.

–Betriebssystemkern reduziert auf Kommunikationsprimitive und elementares multi-programming.

–Die Server selbst können im Benutzermodus laufen.

–Trennung von Mechanismen und Strategien (Policies)

Threads sind parallele Kontrollflüsse, die nicht gegeneinander abgeschottet sind

–laufen innerhalb eines Adressraums, innerhalb eines „echten“ Prozesses

–teilen sich gemeinsame Ressourcen

Kontextwechsel zwischen Threads effizienter als zwischen Prozessen

–kein Wechsel des Adressraums

–kein automatisches Scheduling

–kein Retten und Restaurieren des Kontextes (nur Programmzähler und Register)

Pro Zeiteinheit sind viel mehr Threadwechsel als Prozesswechsel möglich, deshalb auch der Begriff „leichtgewichtige Prozesse“

ABER: Schutzfunktionen getrennter Adressräume ?? Synchronisationsmechanismen werden benötigt

Prozesse werden intern in Warteschlangen (queues) gehalten

–wenn die angeforderten Betriebsmittel (z.B. CPU) nicht zur Verfügung stehen

–oder die erwarteten Ereignisse (z.B. Beendigung eines Prozesses) noch nicht eingetreten sind.

*Readyqueue*–CPU-Warteschlange:

–Menge der im Hauptspeicher geladenen und ausführungsbereiten Prozesse

•*I/O queue*–Gerätewarteschlange:

–Menge der Prozesse, die auf ein I/O-Gerät warten

Der Prozess-Scheduler wählt aus der CPU-Warteschlange den nächsten Prozess aus, der in den Status runningwechseln darf.

Anforderungen an einen Scheduler:

–*Fairness*: Jeder Prozess soll einen fairen Anteil an der CPU bekommen.

–*Effizienz*: Die CPU soll möglichst immer belegt sein.

–*Antwortzeit*: Die Antwortzeit für interaktive Benutzer soll minimal sein.

–*Verweilzeit*: Die Verweildauer von (Batch-) Programmen soll möglichst gering sein.

–*Durchsatz*: Die Anzahl bearbeiteter Prozesse pro Zeitintervall soll maximiert werden.

Alle:

–Fairness: fairer CPU-Anteil für alle Prozesse

–Umsetzung der gewünschten Policy

–Nutzung der vorhandenen Ressourcen

Batch :

–Maximierung des Durchsatzes (Zahl der Jobs/Zeit)

–Minimierung der einzelnen Laufzeiten

–Maximierung der CPU-Nutzung

Interaktiv (Präemptionist hilfreich, Stabilität) :

–Minimierung der Antwortzeit

Echtzeit (Präemption?):

–Einhalten von Deadlines, Vermeidung von Datenverlust

–Vermeidung von Qualitätsverlusten in MM-Systemen

•Batch:

–First-Come First-Served(FCFS)

–ShortestJob First (SJF)

–Shortest-Remaining-Time-Next (SRTN)

•Interaktiv:

–Round-Robin Scheduling

–PriorityScheduling

–ShortestProcessNext

•Echtzeit:

–EarliestDeadline First (EDF)

Prinzip:

–Der Prozess, der einen Prozessor zuerst angefordert hat, bekommt ihn als nächster zugeteilt.

–Eine Unterbrechung laufender Prozesse ist nicht gestattet.

•Eigenschaften:

–Lässt sich sehr einfach implementieren.

–Die mittlere Wartezeit kann unter Umständen sehr hoch werden.

Prinzip:

–Es wird jeweils der Prozess mit der kürzesten Rechenzeit als nächstes gerechnet.

–Es findet keine Unterbrechung laufender Prozesse statt.

–Die Rechenzeiten sind vorab bekannt.

Prinzip :

–Es bekommt jederzeit der Prozess die CPU, der die geringste verbleibende Restrechenzeit hat.

–Neu eintreffende Prozesse, die eine kürzere Rechenzeit haben als der gerade bearbeitete Prozess, verdrängen den aktuellen Prozess aus dem Prozessor (Präemption).

Eigenschaften: –Die mittlere Wartezeit ist minimal.

Problem: –Bei immer neuen eintreffenden kurzen Prozessen kommen lange rechnende Prozesse nie dran: Aushungern (*starvation*)

Prinzip:

–Prozesse werden gemäß ihrer Priorität (Integer) in eine der Warteschlangen eingereiht.

–Die Prozessoren bearbeiten zuerst die Prozesse in der Warteschlange mit der höchsten Priorität (z.B. kleinster Integerwert).

–Wenn diese Warteschlange leer ist, dann wird die Warteschlange mit nächsthöherer Priorität betrachtet usw.

SJF ist Priority Scheduling mit der Priorität = erwartete CPU burstZeit

Eigenschaften:

–Problem : Verhungern –Prozesse mit niedriger Priorität werden nie ausgeführt

–Lösung : Altern (*aging*) –mit zunehmender Wartezeit wird die Priorität der Prozesse erhöht (Integerwert verkleinert).

Deterministisches Scheduling:

–Die Ausführungszeiten der Prozesse sind bekannt.

–Die Prozesse werden zeitlich so angeordnet, dass sich ein gewünschtes Systemverhalten ergibt (z.B. minimale Durchlaufzeit).

ProbabilistischesScheduling:

–Lediglich die Erwartungswerte bzw. die Wahrscheinlichkeitsverteilung der Ankunfts-oder Bedienungszeiten sind bekannt.

–Das Verhalten des jeweiligen Scheduling-Algorithmus wird unter der wahrscheinlichsten Last beschrieben.

–Analyse und Bewertung von probabilistischenScheduling-Algorithmen mit Hilfe der Wahrscheinlichkeitstheorie.

•Hard real-time Systeme:

–Kritische Prozesse müssen garantiert in einem festgelegten Zeitintervall beendet werden.

–Der Scheduler muss die Ausführungszeiten von BS-funktionen garantieren können.

–BS reserviert Betriebsmittel für den schlechtesten Fall.

•Soft real-time Systeme:

–Kritische Prozesse werden immer gegenüber weniger wichtigen bevorzugt, aber es gibt keine festen Garantien.

–Soft real-time Systeme werden zunehmend wichtiger z.B. für MM-Anwendungen

•Prinzip:

–Prozesse werden in einer Warteschlange eingereiht und in FIFO-Ordnung ausgewählt.

–Ein rechnender Prozess B wird vom Betriebssystem nach dem Ablauf einer Zeitscheibe (*time slice*, *quantum*) unterbrochen (a) und am Ende der Warteschlange eingefügt (b)

Prozesse arbeiten oft nicht allein, sondern müssen Informationen austauschen, um eine gemeinsame Aufgabe zu erfüllen.

***Gemeinsame Variablen***: Vor allem in Ein-Prozessor und Multiprozessor-Systemen mit gemeinsamem physikalischem Speicher

***Nachrichtenaustausch***: Vor allem bei verteilten Systemen, also Kommunikation (auch) über Rechnergrenzen hinweg

Durch den Austausch von Nachrichten lassen sich Prozesse (Threads) synchronisieren. Benötigt werden zwei primitive Operationen:

–send(destination, message)Sendet eine Nachricht an einen Empfänger (destination).

–receive(source, message)Empfängt eine Nachricht von einer Quelle (source).

*Blockierende (synchrone) Nachrichtensysteme*

–blockieren den Prozess im send()bzw. receive()Aufruf, wenn

•beim Senden kein Empfänger empfangsbereit ist

•oder beim Empfangen kein aktiver Sender existiert („rendezvous“).

•*Speichernde (asynchrone) Nachrichtensysteme*

–verwalten Nachrichten in speziellen Speicherbereichen (Mailbox, Warteschlange), so dass Prozesse asynchron Nachrichten austauschen können.

Semaphore können zählen und damit z.B. die Nutzung gemeinsamer Betriebsmittel (auch durch mehrere Prozesse) überwachen.

•Semaphore werden durch spezielle Systemaufrufeimplementiert, die die geforderten atomaren Operationen upund downrealisieren.

•Semaphore können in beliebigen Programmiersprachen benutzt werden, da sie letztlich einem Systemaufruf entsprechen.

•Semaphore realisieren ein passives Warten bis zum Eintritt in den kritischen Abschnitt.

•Dazu kann eine einfacher zu implementierende Variante, der sogenannte *Mutex*(von „mutual exclusion“), verwendet werden.

•Dieser kann auch im User Space realisiert werden (Thread-Bibliotheken, wie z.B. PThreads) und hier z.B. für die Unterstützung von Threads ohne Systemaufrufe verwendet werden.

•Programmierfehler bei der Benutzung eines Semaphors können zu Verklemmungen oder inkorrekten Ergebnissen führen.

•Typische Fehler:

–Sprünge aus kritischen Bereichen, ohne das mutex-Semaphor freizugeben.

–Sprünge in kritische Bereiche, ohne das mutex-Semaphor zu setzen.

–Vertauschungen von Semaphoren zum Schutz von kritischen Abschnitten und Semaphoren, die vorhandene Betriebsmittel zählen.

•Alles in allem sind Semaphore eine „low-level“-Lösung, die erhebliche Disziplin vom Programmierer verlangt.

•Ein Monitor ist eine Sammlung von Prozeduren, Variablen und Datenstrukturen, die in einem Modul gekapselt sind.

•Prozesse können die Prozeduren des Monitors aufrufen, aber keine internen Daten ändern.

•Monitore besitzen die Eigenschaft, dass *immer nur genau ein Prozess im Monitor aktiv*sein kann.

•Monitore sind Konstrukte einer Programmiersprache und erfordern daher spezielle Compiler.

•Es ist die Aufgabe des Compilers, Maschinenbefehle zu generieren, die den wechselseitigen Ausschluss im Monitor garantieren.

•Szenario:

–Zwei Prozesse wollen ein Dokument speichern.

–Prozess A reserviert sich zunächst den Sanner, B zunächst den CD-Brenner.

–Beide versuchen dann, die andere Ressource zu reservieren –die schon belegt ist.

–Die entstehende Situation wird als *Deadlock*bezeichnet.

•Definition:

–Eine Menge von Prozessen befindet sich in einer Verklemmung (deadlock), wenn jeder Prozess der Menge auf ein Ereignis wartet, das nur ein anderer Prozess aus der Menge auslösen kann.

•Zwei Typen von Ressourcen

–*Preemptable*:

•Die Ressource kann, ohne Probleme zu verursachen, vorzeitig zurückgegeben werden (z.B. Speicher)

–*Non-preemptable*:

•Ein vorzeitiges Wegnehmen der Ressource von einem Prozess würde zu Problemen führen (z.B. CD, Drucker)

•Zuweisung, Nutzung:

–Ressourcen können vom BS verwaltet werden

•Hier kann es geeignete Strategien geben.

–Ressourcen können auch von der Anwendung selbst verwaltet werden

•Beispiel: gemeinsame Variablen, die von mehreren Threads verwendet werden

•Gerade in solchen Fällen sind Deadlocks häufig.

1.Wechselseitiger Ausschluss (*mutualexclusion*):

–Ein Betriebsmittel ist entweder genau einem Prozess zugeordnet oder es ist verfügbar.

2.Wartebedingung (*hold and wait*):

–Es gibt einen Prozess, der ein Betriebsmittel belegt und auf ein anderes Betriebsmittel wartet, das von einem anderen Prozess belegt wird.

3.Keine Verdrängung (*no preemption*):

–Einem Prozess kann ein Betriebsmittel nicht entzogen werden.

4.Zirkuläres Warten (*circularwait*):

–Es gibt eine Menge {P1, P2,..., Pn} von Prozessen, so dass

•P1auf ein Betriebsmittel wartet, das P2belegt,

•P2wartet auf ein Betriebsmittel, das P3belegt, ...

•… und Pnwartet auf ein Betriebsmittel, das P1belegt.

Ein Betriebsmittel-Zuweisungsgraph ist ein *gerichteter Graph*, der zur Beschreibung von Verklemmungen verwendet wird:

•Betriebsmittel-Zuweisungsgraph ohne Zyklen=> KEINE Verklemmung.

•Betriebsmittel-Zuweisungsgraph mit Zyklus

–von jedem beteiligten Betriebsmittel genau ein Exemplar=> Verklemmung.

–von beteiligten Betriebsmitteln existieren mehrere Exemplare=> Verklemmung möglich.

1.Verhinderung von wechselseitigem Ausschluss:

–Prozesse können gleichzeitig auf alle Betriebsmittel zugreifen

–Beispiel: Drucker

–Problem:

•Nicht alle Betriebsmittel (z.B. Speicher, Einträge in der Prozesstabelle) können zwischen Prozessen geteilt werden. Im Beispiel kann der Plattenplatz zur Ablage von Druckausgaben selbst als Betriebsmittel betrachtet werden, das Auslöser für eine Verklemmung ist.

2.Verhinderung der Wartebedingung:

–Jeder Prozess darf nur dann Betriebsmittel anfordern, wenn er selbst keine anderen Betriebsmittel belegt.

–Alternative 1:

•Jeder Prozess fordert sämtliche Betriebsmittel an, bevor seine Bearbeitung beginnt.

•Problem: Geringe Effizienz, da alle Betriebsmittel eines Prozesses während seiner Ausführung belegt sind. Die meisten Prozesse kennen ihre Betriebsmittelanforderungen nicht, bevor sie starten.

–Alternative 2:

•Ein Prozess muss alle Betriebsmittel abgeben, bevor er weitere anfordern kann.

•Problem: Es ist einem Prozess nicht immer möglich, alle Betriebsmittel freizugeben, bevor er weitere Betriebsmittel belegt (z.B. der Eintrag in der Prozesstabelle). Außerdem kann diese Methode dazu führen, dass die Bearbeitung eines Prozesses unendlich verzögert wird, da ein benötigtes Betriebsmittel immer von einem anderen Prozess belegt ist (starvation).

3.Entzug von zugewiesenen Betriebsmitteln:

–Bereits belegte Betriebsmittel können einem Prozess entzogen werden.

–Probleme:

•Normalerweise besitzen die zugewiesenen Betriebsmittel einen Zustand, der beim Entzug gesichert und später wieder restauriert werden muss. Nicht jedes Betriebsmittel (z.B. Drucker) erlaubt es, den Zustand zu sichern und später wieder zu restaurieren.

–Verwendung:

•Das Verfahren findet häufig Anwendung bei Betriebsmitteln, deren Zustand leicht gesichert werden kann, wie z.B. CPU-Registerinhalte oder Hauptspeicherbereiche.

4.Verhinderung von zirkulärem Warten:

–Es wird eine totale Ordnung auf den vorhandenen Betriebsmitteln definiert.

–Ein neues Betriebsmittel darf nur dann angefordert werden, wenn das Betriebsmittel bezüglich der Ordnung größer ist als alle bereits belegten Betriebsmittel eines Prozesses.

–Mehrere Exemplare eines Betriebsmittels müssen in einer Anfrage angefordert werden.

–Probleme:

•Es ist schwierig, eine totale Ordnung für Betriebsmittel zu definieren, die allen Einsatzmöglichkeiten eines Betriebssystems gerecht wird.

•Benötigt ein Prozess ein Betriebsmittel mit geringerer Ordnung, so müssen zunächst sämtliche Betriebsmittel freigegeben werden.

Ein kritischer Abschnitt (*criticalsection*) eines Programms ist eine Menge von Instruktionen, in der das Ergebnis der Ausführung auf unvorhergesehene Weise variieren kann, wenn Variablen, die auch für andere parallel ablaufende Prozesse oder Threads zugreifbar sind, während der Ausführung verändert werden.

•Diese Werte sind falsch, da ja ein Element eingefügt und eines entfernt wird, der Wert müsste also bei 5 bleiben.

•Die angegebene Lösung erzeugt falsche Ergebnisse, die von der Bearbeitungsreihenfolge der Prozesse abhängen.

•Jede Situation, in der mehrere Prozesse gemeinsame Daten manipulieren, kann zu derartigen Synchronisationsproblemen (raceconditions) führen.

•*Synchronisationsverfahren*garantieren, dass immer nur ein Prozess zu einem bestimmten Zeitpunkt gemeinsam benutzte Daten manipulieren kann.

Ein *Compiler/Assembler*übersetzt symbolische Adressen (Namen von Variablen und Funktionen) zu absoluten oder relativen Speicheradressen.

Ein *Linker*bindet mehrere Objektmodule mit relativen Adressen und die benötigten Bibliotheken zu einem ausführbaren Programm

Adresszuweisung zu drei verschiedenen Zeitpunkten möglich

–*Übersetzungszeit*(CompileTime):

Wenn die Speicheraufteilung bekannt ist, kann absoluter Code erzeugt werden. Z.B. MS-DOS „com“-Dateien oder eingebettete Systeme.

–*Ladezeit*(LoadingTime):

Programm muss reallokierbarsein, wenn der Speicherbereich nicht zum Zeitpunkt der Erstellung (Compile) bekannt ist.

–*Ausführungszeit*(Runtime/ExecutionTime):

Adressbindung erst zur Zeit der Ausführung. Kann während der Ausführung verschoben werden. Benötigt Hardware-Unterstützung.

•Logische Adressen aus Sicht des Programms/Prozesses

•Physikalische Adressen definiert durch Adressbusbreite und physikalischer Speicher Hardwarespeicher

•Virtueller Adressraum ist eine logische Adressierung zur effizienten Nutzung des physikalischen Speichers.

•Umsetzung zwischen diesen Adressen (addresstranslation) über Memory Management Unit (MMU)

Klassische Lösung, die in vielen alten CPUs verwendet wurde:

–Jede CPU bekommt zwei Hardwareregister, das Limit-und das Base-Register

–Programme werden in aufeinanderfolgende Speicherbereiche geladen, ohne Relokation!

–Das Basisregister enthält die physikalische Adresse des ersten Bytes des Programms

–Das Limitregisterenthält die Länge des Programms

•Der komplette Adressraum eines Prozesses wird beim Prozesswechsel auf den Hintergrundspeicher ausgelagert und ein anderer Adressraum eingelagert.

•Ist im Betriebssystem ohne weitere Hardware-Unterstützung zu realisieren.

•Extrem aufwändige Prozesswechsel, da die Zugriffszeiten auf den Hintergrundspeicher im allgemeinen um Größenordnungen langsamer sind als Zugriffe auf den Hauptspeicher.

•Der Hauptspeicher wird vom Betriebssystem in Segmente variabler Länge eingeteilt, die den Prozessen zugewiesen werden. Zur Verwaltung dienen Segmenttabellen.

•Segmente können verschiedene Zugriffsrechte besitzen, zwischen Prozessen geteilt werden oder bei Bedarf wachsen.

•Durch das Entfernen und Einfügen von Segmenten entstehen langfristig kleine unbenutzte Speicherbereiche (*externe Fragmentierung*).

•Durch das Verschieben von Segmenten im Speicher werden viele kleine Löcher

zu einem großen Stück freien Speicherzusammengefügt

•Kompaktifizierungsetzt voraus, dass die Adressen dynamischabgebildet werden.

•Die Suche nach einer optimalen Kompaktifizierungsstrategieist schwierig.

•Kompaktifizierungbenötigt viel CPU-Zeit, wird deshalb ungern eingesetzt.

•Der Memory Manager muss sich belegte Speicherbereiche (a) merken

•Zwei typische Lösungen:

–Bitmaps (b)

•Jedes Bit iden-tifizierteinenkleinen Speicher-bereich, daherfeste Größe.

–Verkettete Listen (c)

•Vereinfachen die dynamische Verwaltung.

•*best fit*: Auswahl des kleinsten Loches, das das Segment aufnehmen kann. Diese Strategie lässt einerseits große Löcher lange bestehen, während sie andererseits eine Vielzahl kleiner und nutzloser Überreste erzeugt.

•*worstfit*: Auswahl des jeweils größten Loches. Dieses Verfahren tendiert dazu, alle Löcher auf etwa die gleiche Länge zu bringen, die dann aber eventuell zu klein zur Aufnahme eines bestimmten Segments sein können.

•*firstfit*: Auswahl des ersten hinreichend großen Loches. Dieses Verfahren liegt in seinem Verhalten zwischen den beiden anderen und ist sehr effizient.

•*nextfit*: Dieses Verfahren ist eine Variation von firstfit. Um zu verhindern, dass sich Löcher einer bestimmten Größe an einer Stelle des Speichers häufen, beginnt jede Suche am Ende der vorherigen Suche. Der Speicher wird also ringförmig durchsucht.

•*buddysystem*: Die Löcher werden in k Listen so einsortiert, dass die i-teListe jeweils Löcher der Länge größer gleich 2ifür i=1,...,k enthält. Dabei können zwei benachbarte Löcher der i-tenListe effizient zu einem Loch der i+1-ten Liste zusammengefügt werden. Umgekehrt kann ein Loch der i-tenListe einfach in zwei Löcher der i-1-ten Liste aufgeteilt werden.

Ist ein Programm größer als der zur Verfügung stehende Hauptspeicher, dann halte immer nur die aktuell notwendigen Teile im Speicher; lade andere Teile bei Bedarf nach

*Zweistufiges*Adressensystem:

–*virtuelle Adressen*, die die Programme benutzen, werden von der *emory Management Unit*in *physikalische Adressen* umgewandelt

–und dann erst an den Speicher gegeben

•Der physikalische Speicher wird in Kacheln (*frames*) fester Größe eingeteilt.

•Der logische Adressraum wird in Seiten (*pages*) gleicher Größe eingeteilt.

•Logische Adressen werdenin eine Seitennummer (*pagenumber*) und eine seitenrelative Adresse (*pageoffset*) aufgeteilt.

•Eine Umsetzungstabelle (*pagetable*) bildet die Seiten auf die verfügbaren Kacheln ab.

•Keine externe Fragmentierung.

–Allerdings wird die letzte Seite eines logischen Adressraums normalerweise nicht vollständig genutzt (*interne Fragmentierung*)

–Und wir benötigen eine Umsetzungstabelle (pagetable)

•Gemeinsamer Speicher lässt sich mit Hilfe von gemeinsamen Seiten realisieren (sharedmemory)

•Speicherschutz wird durch Schutzbits realisiert. Weitere Bits geben an, ob eine Kachel gültig ist (valid) oder ob eine Kachel modifiziert wurde (modified).

•Es muss nicht der gesamte Adressraum eines Prozesses im Speicher sein, um ihn auszuführen. Zugriffe auf gerade nicht geladene Seiten lösen einen Seitenfehler (*pagefault*) aus.

•Das Betriebssystem behandelt Seitenfehler, indem es die benötigte Seite in eine freie Kachel einlagert und den Befehl, der den Seitenfehler auslöste, neu startet.

Zwei große Anforderungen in allen Paging-Systemen:

–Die Umrechnung von virtueller auf physische Adresse muss schnell gehen.

–Bei großem virtuellen Adressraum wird die Seitentabelle groß.

•Die Ladestrategie bestimmt, wann Seiten in den Hauptspeicher geladen werden:

–*Swapping*:Übertragung eines ganzen Adressraums mit einem einzigen Zugriff auf den Hintergrundspeicher.

–*Demand-Paging*:Die benötigten Seiten werden genau dann in den Speicher geladen, wenn auf sie zugegriffen wird.

–*Pre-Paging*:Es werden Seiten geladen, auf die in der Zukunft ein Zugriff erwartet wird. Erfordert Kenntnisse über typische Zugriffsmuster.

–*Page-Clustering*:Gemeinsame Übertragung von mehreren zusammengehörigen Seiten. Ermöglicht die Nutzung großer Seiten auf Hardware, die nur geringe Seitengrößen unterstützt.

•Die Ersetzungsstrategie (*replacementalgorithm*) bestimmt, welche der belegten Kacheln ausgelagert werden, damit wieder freie Kacheln zur Einlagerung von benötigten Seiten vorhanden sind.

•*Lokale Strategien* weisen jedem Prozess eine konstante Anzahl von Seiten zu. Seitenfehler wirken sich daher nur auf den verursachenden Prozess negativ aus.

•Bei *globalen Strategien* wird der gesamte Speicher dynamisch auf alle Prozesse verteilt, um eine effiziente Nutzung des Hauptspeichers zu erreichen.

•Meist werden globale Strategien über alle Prozesse hinweg verwendet.

Andere Idee: stelle jedem Programm mehrere virtuelle Adressräume zur Verfügung,

die es für die jeweiligen Daten verwenden kann Segmentierung des virtuellen Adressraums

•Vorteile der Segmentierung

–Segmente können wachsen, Management durch das BS

–Linken wird wegen der jeweiligen Startadresse 0 stark vereinfacht

–SharedLibraries können leicht realisiert werden

•Methode 1: Trennung zwischen I/O-Register und Arbeitsspeicher

–Der Controller hat speziellen Speicher (I/O ports)

•Jedes Register hat I/O portnumber

•Menger aller Register bildet I/O portspace

•Methode 2: *Memory-Mapped*

–Ein Teil des Hauptspeichers wird für die Geräte reserviert

–Vorteile:

•Einheitlicher Zugriff auf Speicher und Geräte

•Kein Assembler-Code für Registerzugriff notwendig (Treiber in C/C++)

•Einfacherer Schutz (Benutzerprogramme haben keine Adressen im Speicherbereich)

•Schneller: Umkopieren von Status-Infos in Hauptspeicher unnötig

–Nachteil:

•Caching muss selektiv verhindert werden

(Sonst ist Zustand u.U. nicht aktuell)

•Geräte müssen Zugriff auf Speicher abfangen

•Methode 3: *Hybrid*

–Pufferspeicher im Hauptspeicher

–Kontrollregister für I/O-Ports

–Beispiel: Pentium

Dreiverschiedene Verfahren

–*Polling*–Ständiges Abfragen

Nach dem Absetzen eines Befehls an das Gerät wird der Status ständig abgefragt (busywait)

–*Interrupt*-Unterbrechungsroutinen

Interrupts werden von Geräten verwendet, um das Ende eines Vorgangs anzuzeigen.

–*DMA*-DirectMemory Access

I/O kann mittels DMA deutlich beschleunigt werden, da die CPU weniger belastet ist

I/O-Software soll vor allem die Komplexität der Hardware vor dem BS bzw. dem Anwendungsprogrammierer verbergen.

Wichtige Aufgaben:

–Bereitstellen einheitlicher Schnittstellen für Gerätetreiber erleichtert den Einsatz neuer Geräte

–Puffern von verfügbaren Daten

•Vorteil: erhöht die Performance (kein Interrupt pro ankommendes Datum, sondern pro Block)

•Problem: zu langes Liegen von Daten im Puffer, zu viele Kopien zwischen Speicherbereichen: schlechte Performance

–Geräteunabhängige Fehlerbehandlung

–Spezielle Behandlung dedizierter Geräte (CD-ROM) z.B. durch explizites „open“

–Bereitstellung einer einheitlichen Blockgröße

Teil der I/O-Software läuft in User Space:

–Bibliotheken, die mit den Anwendungsprogrammen gelinkt werden

Spooling-Programme

–Integration exklusiv benutzbarer Geräte in ein Multiprogrammiersystem

–Beispiel Drucker:

•Anwenderprogramme drucken in spezielles spooling directory

•Hintergrund-Daemons druckt die Spooler-Dateien, da nur er

Zugriffsberechtigungen auf den Drucker-Controller besitzt.

Lineare Liste von Dateinamen mit Zeiger auf die Datenblöcke.

–Einfach zu programmieren

–Jedoch lange Zeit beim Suchen von Dateien, z.B. bei open()

Hash-Tabelle –Lineare Liste mit vorgeschalteter Hash-Datenstruktur.

–Verringert die Suchzeit innerhalb eines Verzeichnisses

–*Kollision*–Situationen in der zwei oder mehrere Dateien zum gleichen Haschwert führen

–Feste Größe der Hashtabelle

•*Externe Fragmentierung*

–Es ist genügend Speicher verfügbar, um Datei zu erzeugen, aber der Speicherbereich ist nicht zusammenhängend. => Nur ~ 2/3 des Datenspeichers kann verwendet werden

–Kompaktierung: Reduzierung der externen Fragmentierung

•Verschieben der Datenblöcke, so dass wieder ein großer freier Speicherblock entsteht.

•Während der Kompaktierungist Zugriff auf Dateien eingeschränkt.

•Belegt die Ressource, ohne wirklichen Nutzen zu bringen.

•*Interne Fragmentierung* –

–Letzter belegter Block einer Datei normalerweise nicht vollständig genutzt; Mindestgröße einer Datei auf dem Datenträger = 1 Datenblock.

–Speicherbedarf auf dem Datenträger sind immer Vielfache der Datenblockgröße (512B/1k/2k/4k/8k/16k/32k).

–Die Größendifferenz bis zum Vielfachen bleibt ungenutzt.

•Effizienz abhängig von:

–Festplattenbelegung und Algorithmen für Verzeichnisstruktur

–Typen der Daten, die pro Datei in einem Verzeichniseintrag stehen

•Leistungssteigerung

–Caching der Festplatteninhalte –Spezieller Speicherbereich im

Hauptspeicher für oft verwendete Datenblöcke, z.B. FAT

–Sequentieller Dateizugriff kann optimiert werden durch

•free-behind

•read-ahead

*Die* ***Virtualisierung*** *abstrahiert IT-Ressourcen durch das Einfügen*

*einer zusätzlichen Ebene zwischen Anwendung und Hardware. Für den*

*Anwender verhält sich das virtuelle Objekt wie ein dediziertes Hardoder*

*Softwareobjekt* (2)

▶ Virtuelle Nachbildung von Geräten, Betriebssystemen oder

Services

▶ Kein Unterschied für Anwender

▶ Verhalten nach außen wie physische Pendants

▶ Wirt (Host): Wirtsystem stellt Ressourcen bereit

▶ Gast: auf dem Host ausgeführte Instanz

▶ Hypervisor/VMM: erstellt und verwaltet virtuelle Hardware

▶ Sicherheit: volle Kontrolle über Ressourcen

▶ Effizienz: wenige Eingriffe

▶ Wiedergabetreue: identisches Programmverhalten

▶ Para: griechisch nebenbei, entlang

▶ Maschinenähnliche Softwareschnittstelle

▶ VM nicht identisch mit tatsächlicher Hardware

▶ Modifikation des Gastbetriebsystems nötig

▶ Gast-BS kommuniziert mit Hypercalls

▶ Gast-BS ”weiß” von Hypervisor

▶ Vorteil: Durch Hypercalls schnelleres Gesamtsystem

▶ Nachteil: Gast-BS auf Hypervisor zugeschnitten

▶ Unternehmen mit Multicomputern:

▶ Viele Server, viele Rechner, über Netzwerk verbunden

▶ Wg. Lastverteilung, Zuverlässigkeit, IT-Sicherheit

▶ Nachteil: teuer, Verwaltung

▶ Vorteil Einsatz VM:

▶ Partielles Ausfallsystem: Fehler in VM keine Konsequenz aufandere VM

▶ ”Multicomputer” mit geringeren Kosten und einfacher Wartung

▶ Erhöhte Sicherheit durch virtuelle Maschinenisolation

▶ Eine leistungsfähige Plattform günstiger als viele

▶ Nachteil Einsatz VM: ”Alles auf eine Karte”