Die Einbettung eines Datenbanksystems in eine Betriebssystemumgebung

von Theo Härder
Forschungsgruppe Datenverwaltungssysteme I
Forschungsgruppenleiter:

Prof. Dr. H. Wedekind

Nr. DVI 79-2

Juni 1979

Die Einbettung eines Datenbanksystems in eine Betriebssystemumgebung

1. Einleitung

Ein Datenbanksystem (DBS) soll seinen Benutzern eine Vielzahl von komplexen Funktionen zur Definition, Manipulation, Kontrolle und Speicherung von Daten auf einer hohen, möglichst datenunabhängigen Schnittstelle bieten, die vor allem die Problemlösung des Benutzers unterstützen soll. Funktional gesehen handelt es sich hierbei un eine erhebliche Erweiterung und Ausdehnung der Einsatzmöglichkeiten des Datenverwaltungssystems (DVS) eines Betriebssystems (BS), unter dessen Kontrolle das DBS im praktischen Betrieb läuft. Bei der Entwicklung eines DBS sind folglich zwei grundsätzlich verschiedene Vorgehensweisen denkbar:

1. Eine Erweiterung der BS-Funktionen führt zur Integration des DBS.
2. Das DBS stellt eine eigenständige Entwicklung dar.

Da viele Funktionen zur Datenverwaltung (DV)- wenn auch manchmal in rudimentärer Weise - in BS vorhanden sind, erscheint der erste Ansatz erfolgversprechend. Solche Funktionen, die ein BS zu unterstützen hat, sind beispielsweise Verfahren zur Speicherverwaltung, zum Sperren und zur Vergabe von Betriebsmitteln und zur Wiederherstellung von Datenobjekten. Besonderes Gewicht bekommt die Idee der Integration durch die Möglichkeit der Zentralisierung der gesamten Betriebsmittelverteilung und -zuordnung durch eine Instanz im BS. Damit würden viele Probleme, die von konkurrierender Betriebsmittelbenutzung von Prozessen und von Tasks innerhalb von Programmen hervorgerufen werden, einfacher und vor allem effizienter lösbar sein.

Das entscheidende Problem bei diesem Ansatz liegt darin, daß die Verarbeitungs- und Dateneinheiten, die ein DBS zu kontrollieren hat, in sehr großer Zahl und komplexer Struktur auftreten. Kein existierendes BS deckt auch nur annähernd typische Anforderungen von DBS in befriegender Weise. Zum Beispiel sind gewöhnlich Sperren nur auf der Ebene von Dateien zur exklusiven Benutzung oder Wiederherstellungsverfahren nur auf physischer Ebene für vollständige Datenträger (Magnetplatten etc.) verfügbar, während ein DBS sehr kleine Sperrgranulate (Tupel, Attribut) mit ver-

schiedenen Sperrmodi (gemeinsam benutzbar, exklusiv) oder transaktionsbezogene Wiederherstellung von Datenobjekten unterstützen sollte. Eine Anreicherung der DV-Funktionen existierender BS für die Zwecke der Datenbankverwaltung erscheint deshalb nicht gerechtfertigt, weil der Verwaltungsaufwand des BS, der durch das breite Spektrum der anzubietenden Standardfunktionen hervorgerufen wird, wegen der Vielzahl und wegen des hohen Spezialisierungsgrades zusätzlicher DV-Funktionen über die Maßen anwachsen würde.

Eine Alternative scheint in der Entwicklung spezieller Datenbankbetriebssysteme (DBBS) zu liegen, deren Funktionsumfang verglichen zu allgemeingültigen BS drastisch reduziert ist. Sie sind als zugeschnittene Lösungen für den Einsatz großer DB-Anwendungen zu konzipieren und sollen spezielle Datenbanktransaktionen mit einem beträchtlichen Anteil an Datenkommunikation und einer hohen Prozeßwechselrate von Anwendungsprozessen in zufriedenstellender Weise bedienen können. In (Gr78) wird berichtet, daß die Leistungsfähigkeit solcher zugeschnittener Systeme oft um den Faktor 10 besser ist als die eines allgemeingültigen BS. Die Problematik dieses Ansatzes liegt zunächst in der Wartung eines speziellen BS und in der Notwendigkeit der Unterstützung und Weiterentwicklung neuer Funktionen, um den Bedarf neuer Anwendungen und die wachsenden Ansprüche laufender Anwendungen abdecken zu können. Die Anpassung eines solches BS an die technologische Weiterentwicklung und an steigende Ansprüche im Hinblick auf leichtere Bedienbarkeit und wirkungsvollere Benutzbarkeit sind nicht zu unterschätzen. Die Erfahrung zeigt, daß mit dem Umfang von Aufgaben der Funktionsbedarf wächst, so daß im Laufe der Entwicklung allmählich der Standard eines allgemeingültigen BS erreicht wird.

Es ist deshalb eine offene Frage, ob komplexe DB-Anwendungen durch zugeschnittene BS unterstützt werden sollen und wie gleichzeitig der Funktionsumfamg allgemeingültiger BS dem Benutzer zugänglich gemacht werden kann. Die parallele Verfügbarkeit eines DBBS und eines allgemeingültigen BS unter Kontrolle eines Hypervisors (VM/370) , der im Prinzip eine Art Unterbrechnungsbehandler darstellt und dem einen oder anderen BS den realen Rechner zuteilt, ist ein bedenkenswerter Ansatz. Durch die Einführung virtueller Umgebungen für die BS ergeben sich jedoch von vornherein Leistungsverluste. Eine weitere geeignete Einsatzmöglichkeit eines zugeschnittenen DBBS wäre bei Zuordnung eines separaten Rechners für DB-Aufgaben (Back-End-Processor) gegeben, wobei ausschließlich die speziellen DB-Funktionen benötigt werden und somit

Abstract:

A number of difficulties related to an independent position of a database system in an operating system environment consisting of two levels of control are discussed. Particular conflicts concern the isolation and access control of processes and the distribution of control over resources.

Operating system primitives supporting cooperation, synchronisation and sharing of main storage of independent processes are insufficient.

Present operating systems do not distinguish between protection domain and scheduling unit.

As a consequence, database system and application program have to be separated using different address spaces for reasons of data security. An improved concept concerning the position of the database system in relation to the operating system and the application program is proposed. It is assumed that an address space provided by the operating system can be divided into separate protection domains allowing the allocation of application program and database system in the same address space.

Zusammenfassung:

Es werden die Schwierigkeiten diskutiert, die bei der Einbettung selbständiger Datenbanksysteme in die Umgebung heutiger Betriebssysteme entstehen, die sich durch zwei Ebenen der Kontrolle auszeichnen. Besondere Konflikte liegen im Bereich der Isolation und Zugriffskontrolle und in der Aufteilung der Kontrolle über Betriebsmittel. Unzureichend sind die Hilfsmittel zur Kooperation, Synchronisation und gemeinsamen Benutzung von Speicherbereichen durch selbständige Prozesse.

In herkömmlichen Konzepten der DBS-Einbettung sind aus Gründen der Datensicherheit Datenbanksystem und Anwendungsprogramm räumlich getrennt, da heutige Betriebssysteme keine Unterscheidung zwischen Schutzgebiet (protection domain) und Adreßraum (scheduling unit) durchführen. Es wird ein Konzept für die DBS-Einbettung vorgeschlagen, das die Aufteilung des Adreßraumes in verschiedene Schutzgebiete voraussetzt und die Zuordnung von Anwendungsprogramm und Datenbanksystem im selben Adreßraum erlaubt.

vorteilhafterweise mit reduzierten BS-Funktionen integriert werden können.

In den meisten praktischen Fällen ist das DBS eine eigenständige Entwicklung. Es sollen deshalb in größerem Detail die Konflikte und Probleme der Einbettung eines solchen separat entwickelten DBS in eine BS-Umgebung diskutiert werden. Aus der Sicht des BS ist das DBS in diesem Fall ein "Anwendungsprogramm", das wie jedes andere Anwendungsprogramm Betriebsmittel im Rahmen der konkurrierenden Anforderungen zugeteilt bekommt. Es können für die Aufgaben der Datenbankverwaltung eine oder mehrere Aktivierungen des DBS als ein oder mehrere unabhängige Prozesse aus der Sicht des BS ablaufen. Die vielfältigen Aufgaben der Datenverarbeitung von der physischen Ein-/Ausgabe von Datenseiten als Transporteinheiten zur Übertragung der Daten zu externen Speichern bis zur Bereitstellung mächtiger Operatoren für die logischen Datenstrukturen an der Endbenutzerschnittstelle werden durch das Programm "DBS" realisiert. Als Unterstützung durch das BS werden dabei eine Reihe niederer (hardware-naher) Funktionen als Primitive ausgenutzt, was sich vor allem im Hinblick auf Portabilität und Geräteunabhängigkeit des DBS vorteilhaft auswirkt (Hä 78).
Zu diesen Funktionen gehören in erster Linie

- die Verwaltung des virtuellen Speichers
- die Zuordnung und Verwaltung externer Speicher
- die Durchführung der Ein-/Ausgabe physischer Datenblöcke
- die Unterbrechnungsbehandlung asynchroner Ereignisse
- die Prozeßverwaltung und -zuteilung
- die Synchronisation bei der Prozeßkommunikation.

Die Eigenständigkeit des DBS unter herkömmlichen BS schafft eine Reihe von Problemen, die vor allem im Bereich der Isolation und Zugriffskontrolle und in der Betriebsmittelvergabe und -kontrolle liegen. Bevor wir verschiedene Möglichkeiten der Einbettung eines selbständigen DBS in einer BS-Umgebung untersuchen, wollen wir die einzelnen Konfliktsituationen betrachten.

2. Besondere Probleme der BS-Einbettung selbständiger DBS

2.1 Isolation und Zugriffskontrolle

Die Trennung von DBS und BS hat Konsequenzen für die Schutzmechanismen, durch die ein hohes Maß an Datensicherheit gewährleistet werden soll. Dabei hat das BS für eine vollständige Isolierung aller seiner Prozesse untereinander zu sorgen und dem DBS die alleinigen unbeschränkten Zugriffsrechte auf die Speichereinheiten, die der Datenbank zugeordnet

sind, zu garantieren. Das DBS hat durch eine wirkungsvolle Zugriffskontrolle dafür zu sorgen, daß jede Anwendung nur über solche Daten der Datenbank verfügen kann, für die sie eine Zugriffsberechtigung besitzt. Dabei sollte die Vergabe von abgestuften Verfügungsrechten auf Datenobjekte derart möglich sein, daß jede Anwendung mit einer minimalen Menge von Rechten genau auf die Daten zugreifen kann, die sie zur Erfüllung ihrer Aufgabenstellung braucht (least privilege principle). Die Verwaltung der unterschiedlichen Rechte der einzelnen Benutzer für die Objekte der Datenbank läßt sich in einfacher Weise mit Hilfe einer Zugriffsmatrix durchführen.

Die Forderung nach Prozeßisolation wird durch die räumliche Trennung, wie sie durch das Konzept virtueller Adreßräume zum Ausdruck kommt, besonders wirkungsvoll erfüllt. Verschiedene Anwendungsprogramme (AP) haben keine Möglichkeit, sich zu stören, da sie in unterschiedlichen Adreßräumen ablaufen. Die strikte Trennung zwischen AP und DBS ist aus Performance-Gründen nicht erwünscht, da beide Programme zur Durchführung ihrer Aufgaben in kontrollierter Weise zusammenarbeiten müssen. Eine Zusammenlegung von AP und DBS in einen Adreßraum, die eine Kontrollübergabe sehr erleichtern würde, ist jedoch bei der Struktur heutiger virtueller BS nicht möglich. Vom Betriebssystem aus gesehen ist nämlich jeder Prozeß einem virtuellen Adreßraum zugeordnet; der Adreßraum ist Schutzgebiet (protection domain) und Einheit für unabhängige Betriebsmittelzuweisungen (scheduling unit). Zugriffsrechte eines Programmes, die vom BS überwacht werden, sind folglich jeweils an den Adreßraum, in dem das Programm abläuft, gebunden und nicht an das spezielle Programm. Falls mehrere Programme in einem Adreßraum ablaufen und die Rechte aller Programme auf den Adreßraum übertragen werden, findet gewissermaßen ein unkontrollierter Transfer von Rechten statt. Sobald ein Programm in den Adreßraum eines anderen Programmes geladen wird, kann es dessen Zugriffsrechte mißbrauchen (Trojan Horse Problem). Das Schutzbedürfnis der in der Datenbank gespeicherten Daten erzwingt folglich bei der Struktur heutiger BS die räumliche Trennung von AP und DBS.

Unabhängig vom unkontrollierten Transfer von Zugriffsrechten könnte sich ein AP wegen des fehlenden Speicherschutzes durch gezielte Modifikation der Zugriffsmatrix des DBS Zugriffsrechte verschaffen, die von den vorgesehenen Kontrollinstanzen nicht zurückgewiesen werden. Unabhängige Zugriffskontrolle durch das DBS läßt sich also nur garantieren, wenn auch AP und DBS räumlich getrennt in verschiedenen virtuellen Adreß-

räumen ablaufen, wie es in Bild 1 skizziert ist.

Der BS-Adreßraum, der nur in einer physischen Kopie vorhanden ist, wird jeweils an die einzelnen Benutzer-Adreßräume angehängt. Er besitzt gegenüber allen Benutzern Privilegien (Supervisor-Modus) und ist von ihnen isoliert. Dienstleistungen des BS können nur über spezielle Primitive (SVC's) angefordert werden.

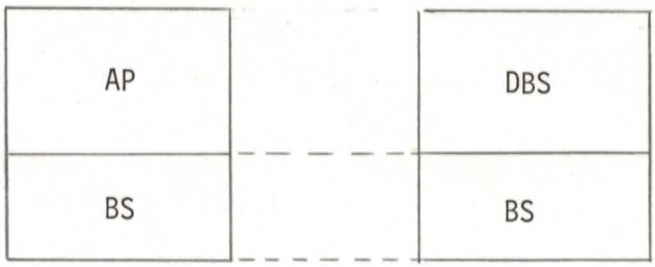


Bild 1: Isolation durch räumliche Trennung von AP und DBS

2.2 Kooperation zwischen verschiedenen Prozessen

Virtuelle BS verbessern zwar die Sicherheit durch strikte Isolation der Prozesse, leisten aber im allgemeinen keine besondere Unterstützung der Kooperation von gleichberechtigten Prozessen in verschiedenen Adreßräumen (execution domains). Als Konsequenz muß bei jeder Kommunikation zwischen AP und DBS das BS eingeschaltet werden. Deshalb verursacht die durch das Schutzbedürfnis erzwungene Lösung hohe Kosten für die Inter-Prozeß-Kommunikation (IPK), die durch Primitive wie SIGNAL und WAIT (GrW79) implementiert werden kann. Jede DBS-Anforderung, die vom AP gewöhnlich synchron durchgeführt wird, hat zwei Prozeßwechsel zur Folge. Der Aufwand für einen Prozeßwechsel, der in der Regel einen Austausch von Daten zwischen zwei Adreßräumen impliziert, läßt sich typischwerweise mit 5000 Instruktionen abschätzen.

In Sonderfällen ergeben sich mit dieser Tatsache sehr ungünstige Verhältnisse zwischen benutzerbezogener Arbeit und Verwaltungsaufwand des Gesamtsystems. Unterstellen wir als Beispiel ein sequentielles Lesen und Ausdrucken von Datensätzen durch AP. Die durch das DBS bereitzustellenden Datensätze seien bereits im Systempuffer, so daß keine physische Eingabe nötig ist. Das Verarbeiten eines Datensatzes erfordert dann folgenden Aufwand (Größenordnung):

Kontrolle über den Rechner	Anzahl der Instruktionen
BS: Prozeßwechsel AP - DBS	5000
DBS: Bereitstellen des Datensatzes	1000
BS: Prozeßwechsel DBS - AP	5000
AP: Verarbeitung des Datensatzes	100

Dieser aufgelistete Aufwand ist als Minimum anzusehen, da zusätzlicher Overhead durch Paging etc. eingeführt werden kann. Untersuchungen des synchronen Aufrufverhaltens von AP und DBS in einem Realspeichersystem ergaben einen Aufwand von etwa 1000 Instruktionen pro Prozeßwechsel, wobei allerdings die Anforderungen an das Sicherheitsbedürfnis des DBS-

Prozesses recht bescheiden waren. Bei direkter Übergabe der Kontrolle zwischen AP und DBS (Unterprogrammaufruf unter Verzicht auf jede Schutzfunktion) im Falle eines eingebundenen DBS (linked-in DBS) würden nur 1100 Instruktionen (etwa 10%) erforderlich sein, da kein Aufwand für den Prozeßwechsel anfällt.

Als Zielvorstellung sollte bei der BS-Entwicklung die Verbesserung der Kooperation zwischen AP und DBS bei gleichzeitiger Einhaltung des Sicherheitsstandards angestrebt werden. Die Einrichtung verschiedener Schutzgebiete (protection domains) in einem Adreßraum (scheduling unit) garantiert einerseits strikte Trennung von DBS und AP, erlaubt aber andererseits eine direkte und sichere Kontrollübergabe zur synchronen Kooperation.

2.3 Aufteilung der Kontrolle über Betriebsmittel

Die Eigenständigkeit eines DBS impliziert weiterhin eine Reihe von Nachteilen, die in der Notwendigkeit der Aufteilung der Kontrolle über die Betriebsmittel auf mehrere unabhängige Systeme begründet liegen. Schon im einfachsten Fall der Zuordnung lassen sich zwei Ebenen unterscheiden:

1. Das BS verwaltet und kontrolliert alle Betriebsmittel des Rechners und teilt sie aus der Sicht des BS unabhängigen und gleichberechtigten Prozessen zu. Solche Prozesse sind das DBS, DBS-Anwender (AP) und sonstige Benutzer.
2. Das DBS kontrolliert und verwaltet bei Mehrfachbenutzung bestimmte Betriebsmittel für die AP.

Die Aufteilung der Kontrolle wird unter bestimmten Umständen in zweifacher Weise fortgesetzt.

3. Ein bestimmtes Anwendungsprogramm des DBS kontrolliert mehrere Tasks (multi-tasking), falls die Möglichkeit asynchroner Verarbeitung gegeben ist. Ein solches Anwendungsprogramm könnte beispielsweise ein TP-Monitor sein.

4. Im Falle eines virtuellen Maschinenkonzeptes (VM/370, Hypervisor) läuft selbst das BS in virtueller Systemumgebung ab und kann nur virtuelle Betriebsmittel verwalten. Die realen Betriebsmittel des Rechners werden durch einen virtuellen Maschinenmonitor (VMM) verwaltet und den einzelnen virtuellen Maschinen zugeteilt.

Im allgemeinen Fall kann also die Kontrolle über die Betriebsmittel auf bis zu 4 Ebenen verteilt sein. Da die Scheduling-Algorithmen auf den verschiedenen Ebenen nicht aufeinander abgestimmt sind, sondern auf spezielle Charakteristika der jeweils unterschiedlichen Aufgabenstellungen hin optimiert sind, ergeben sich neben den beträchtlichen Kosten für den Verwaltungsoverhead zusätzliche Reibungsverluste durch die unterschiedlichen Zuteilungs- und Vergabestrategien.
Ein Musterbeispiel für unkoordinierte Scheduling-Strategien läßt sich an der Systempufferverwaltung des DBS in einer virtuellen BS-Umgebung beobachten (Ef78). Sowohl Programmcode als auch Systempufferbereiche des DBS unterliegen dem BS-Paging. Das DBS versucht durch spezielle , auf Datenbankanwendungen zugeschnittene Seitenersetzungsstrategien wie GCLOCK oder SECOND CHANCE bestimmte Seiten (Zugriffspfadseiten) möglichst lange im Puffer zu halten. Das BS betrachtet die Systempufferseiten als Teil des virtuellen Adreßraumes, den es gewöhnlich durch LRU-Ersetzungsstrategien verwaltet. Da beide Ersetzungsstrategien nichts voneinander wissen, können die Optimierungsversuche des DBS bei der Systempufferverwaltung durch das BS in ihrer Gegenteil verkehrt werden. Folgende Fehlseitenbedingungen können sich in dieser Situation ergeben:

Page Fault: Die angeforderte DB-Seite befindet sich im Systempuffer, ist aber nicht im Hauptspeicher verfügbar.

Database Fault: Die angeforderte DB-Seite befindet sich nicht im Systempuffer. Der für ihre Einlagerung ausgewählte Pufferrahmen ist im Hauptspeicher verfügbar.

Double Page Fault: Die angeforderte DB-Seite befindet sich nicht im Systempuffer. Die zu ersetzende Seite ist nicht im Hauptspeicher verfügbar. Es kommen also ein Page Fault und Database Fault zusammen.

Diese Situation läßt sich weiter verschärfen, wenn das BS in einer virtuellen Maschine (VM) mit virtuellem Adreßraum, virtueller E/A, virtueller Peripherie etc. unter Kontrolle eines VMM läuft. Dann ist der Systempuffer des DBS in einem virtuellen Adreßraum, den das BS zur Verfügung stellt, angeordnet, wobei schließlich alle virtuellen Adreßräume des BS in der durch den VMM geschaffenen virtuellen Umgebung realisiert sind. Es ist leicht einzusehen, daß jede normale Fehlseitenbedingung eines Programmes unter BS einem Double Page Fault entspricht, so daß beim Zusammentreffen dieser Fehlseitenbedingung mit einer DB-Seitenanforderung, die sich nicht im Systempuffer befriedigen läßt, von einem sogenannten Triple Page Fault gesprochen werden kann.
Die Ineffizienz von BS-Operationen in virtueller Umgebung, die vor allem von der redundanten Ausführung von Funktionen durch BS und VMM herrührt, läßt sich eindrucksvoll belegen (MK79). Deshalb wurden als Per-

formance-Optionen Änderungen in solchen BS vorgenommen, die gezielte Annahmen über die virtuelle Umgebung und Kommunikation mit dem darunter ablaufenden Kontrollprogramm (VMM) zulassen. Solche speziellen Mechanismen (Performance Enhancement), die auf die Umgehung, Eliminierung oder Reduzierung von Funktionen, die in einer virtuellen Umgebung redundant oder ineffizient sind, abzielen, sind beispielsweise Handshaking, VM Assist oder REAL-VIRTUAL Modus (SM79).
Während das Zusammenwirken zwischen BS und VMM auf diese Weise optimiert werden kann, bleibt die Betriebsmittelverwaltung zwischen BS und DBS ohne Kontrolle und Unterstützung, da das DBS als bloßes Anwendungsprogramm betrachtet wird.

2.4 Gemeinsame Benutzung von Speicherbereichen

Falls kein gemeinsam benutzbarer Speicherbereich zwischen AP und DBS vorhanden ist, müssen bei jedem DBS-Aufruf Parameter und Ergebnisse über spezielle Bereiche im BS-Adreßraum ausgetauscht werden. Dazu sind jeweils 2 Kopiervorgänge für die zu übergebenen Daten notwendig (MOVE-Modus).

Eine gemeinsame Benutzung (Lesen und Schreiben) von Bereichen im Hauptspeicher würde den Datenaustausch sehr vereinfachen (LOCATE-Modus).

Idealerweise sollten mehrere Prozesse genau soviel Speicher wie nötig gemeinsam benutzen können. Da bei der dynamischen Adreßumsetzung nur ganze Seiten mehreren Adreßräumen zugeordnet werden können, ist die kleinste Einheit für eine gemeinsame Benutzung eine Seite, was für paarweisen Datenaustausch zwischen AP und DBS ausreicht. Durch Zuordnung einer separaten Seite für jedes Paar AP -DBS bliebe die Isolation zwischen AP und DBS und den AP untereinander erhalten. Aus technischen Gründen ist jedoch in vielen BS nur die Zuordnung von gemeinsam benutzbaren Segmenten (Seitentabelle ≙ 16 Seiten) als kleinste Einheit möglich. Die Zuordnung disjunkter Segmente für jedes Paar APi - DBS würde sehr schnell den gesamten verfügbaren Adreßraum des DBS verbrauchen. Als Konsequenz wird deshalb häufig das gleiche Segment von mehreren AP geteilt. Diese Maßnahme reduziert die Datensicherheit, da alle zwischen den verschiedenen APi und DBS ausgetauschten Daten allen beteiligten APi zugänglich werden und potentiell verfälscht werden können.

Die Möglichkeit, Schreib-/Lese-Segmente gemeinsam benutzbar zu machen, bringt auch für die Implementierung und den Einsatz des DBS Vorteile. Durch die gemeinsame Benutzung von Segmenten können dann mehrere Aktivie-

rungen des DBS in verschiedenen Adreßräumen als unabhängige Prozesse (vom BS her gesehen) reentrant ablaufen, um n AP-Prozesse zu bedienen. Der Systempuffer (SP) und die globalen Systemtabellen (GST) wie Kontrollblöcke für Sperren und Warteschlangen und zentrale Tabellen für Verwaltungsinformation müssen in diesem Fall von unabhängigen DBS-Prozessen gemeinsam benutzt werden, was die Lösung des "Leser- und Schreiber-Problems" erfordert. Zugriffe auf diese globalen DBS-Tabellen müssen synchronisiert werden, damit keine Änderungen verloren gehen oder inkonsistente Daten gelesen werden.

2.5 Synchronisation mehrerer DBS-Prozesse

Ein Synchronisierung über Primitive des BS erscheint vor allem wegen der großen Häufigkeit der Synchronisierungsereignisse, der extrem langen Pfade durch das BS und der geringen Tauglichkeit der Synchronisierungsprimitive für DB-Anforderungen völlig unrealistisch. Deshalb muß das Synchronisierungsproblem durch geeignete Protokolle und Konventionen zwischen den einzelnen DBS-Prozessen direkt gelöst werden. Die Hardware des Rechnersystems stellt dazu in der Regel geeignete atomare Instruktionen wie "Test and Set" oder "Compare and Swap" zur Verfügung, um Semaphore und die dazugehörigen Warteschlangen zu implementieren. Die Synchronisation zwischen den gleichberechtigten DBS-Prozessen erfordert folgenden Aufwand, da jeder Zugriff auf globale Daten durch Setzen eines Semaphors (Sperre) zu schützen ist (GrW 79):

1. Falls die benötigte Datenstruktur nicht belegt ist, fallen praktisch keine Kosten an. Das Setzen und Freigeben der Sperre benötigt 10 Instruktionen.

2. Falls ein gesetzter Semaphor vorgefunden wird, hat der anfragende Prozeß zu warten. Zwei verschiedene Formen des Wartens können unterschieden werden:
   - Aktives Warten (busy wait, spin lock): Der Prozeß fragt ständig an, ob er zugreifen kann. Auf diese Weise kann er lange Zeiten mit IDLE-Operationen verbringen.
   - Deaktivierung (interrupt wait, suspend lock): Der wartende Prozeß läßt sich in den Ruhezustand versetzen, bis er durch ein Signal eines anderen Prozesses erfährt, daß er an der Reihe ist (FIFO-Disziplin). In diesem Fall ist die Interaktion mit dem BS -beispielsweise über SIGNAL/WAIT - erforderlich, so daß der Aufwand für das Senden und Empfangen eines Signals mit 5000 Instruktionen abgeschätzt werden kann.

Vor allem wegen der Garantie für faire Betriebsmittelvergabe, aber auch wegen des vergleichsweise "geringen" Verbrauchs an Rechenzeit ist das Prinzip der Deaktivierung bei der Synchronisation vorzuziehen.

2.6. Vorzeitiger Entzug des Rechnerkerns (preemptive scheduling)

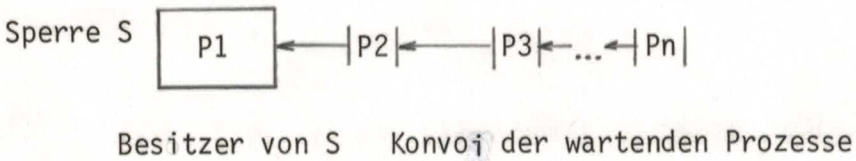
Das skizzierte Synchronisationsverfahren zwischen gleichberechtigten Prozessen ist auch sicher, wenn vom BS vorzeitiger Entzug des Recherkerns praktiziert wird. Es können sich jedoch Quasi-Deadlock-Zustände zwischen den DBS-Prozessen einstellen, die durch die Synchronisationshäufigkeit zusammen mit der FIFO-Zuteilungsdisziplin an speziellen DB-Tabellen bedingt sind. In (BGMP79) wird diese Erscheinung als Konvoi-Phänomen bezeichnet.

Einige Systemressourcen wie Systempuffer und Systemlog zeichnen sich durch sehr hohes Verkehrsaufkommen aus. Sie sind durch Sperren (High Traffic Locks) einem Prozeß P i exklusiv zuzuordnen, werden typischerweise nur sehr kurze Zeit benötigt (Sperrdauer ≈ 50 Instruktionen), werden aber gewöhnlich nach einem Ausführungsintervall von etwa 1000 Instruktionen vom gleichen P i wieder angefordert.

Angenommen P1 hält sich während 5% seiner Ausführung in einer kritischen Zone auf und blockiert durch eine Sperre S alle anderen DBS-Prozesse. Falls P1 in der kritischen Phase in den Wartezustand geht (Zeitscheibenende, Paging-E/A), passiert folgendes:

- Alle anderen DBS-Prozesse werden der Reihe nach durch den BS-Scheduler aktiviert und fordern nach kurzer Zeit die Sperre S an.
- Jeder Prozeß findet S belegt und läßt sich in einer Warteschlange vor S deaktivieren.
- Die gesamte DB-Verarbeitung ist blockiert.

Es ergibt sich folgende Situation S1:



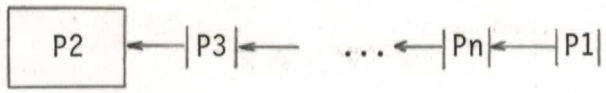
Wegen der Zuteilungsdisziplin FIFO und dem hohen Verkehrsaufkommen von S sind nach erneuter Aktivierung von P1 folgende Aktionen zu erwarten:

- P1 gibt S frei, teilt S P2 zu und veranlaßt die Aktivierung von P2 durch das BS.

- Nach einem Ausführungsintervall von 1000 Instruktionen benötigt P1 jedoch wieder S.
- Da S belegt ist, läßt sich P1 in der Warteschlange vor S deaktivieren.

Nun ist folgende Situation S2 entstanden.

Sperre S



Besitzer von S Konvoi der wartenden Prozesse

Wegen des hohen Verkehrsaufkommens bildet ein einmal entstandener Konvoi ein stabiles Phänomen. Die n DBS-Prozesse werden dadurch unter Einführung sehr langer Wartezeiten serialisiert. Wegen des extrem ungünstigen Verhältnisses von BS-Interaktionen (SIGNAL/WAIT) zu DB-Verarbeitung entsteht eine Art "thrashing"-Situation vor S, die nach außen hin als Quasi-Deadlock in Erscheinung tritt. Falls ein DBS-Aufruf m verschiedene Zugriffe auf S erfordert, muß sich der ausführende DBS-Prozeß Pi zur Erledigung der Benutzeranforderung m Mal durch das BS deaktivieren lassen. Nach m-maliger Durchwanderung der Warteschlange bekommt der Benutzer die Ausführung seines Auftrages gemeldet.

Zur Verhinderung des Konvoi-Phänomens sollten DBS-Prozesse in kritischen Phasen nicht unterbrechbar sein. Bei der unterstellten Trennung von DBS und BS läßt sich dieses Ziel nicht erreichen, da der BS-Scheduler (preemptive scheduling) keine Sonderbehandlung von DBS-Prozessen durchführt. Deshalb muß die Lösung des Konvoi-Problems im Synchronisierungsverfahren der DBS-Prozesse gesucht werden. Zunächst kann die Wahrscheinlichkeit seines Auftretens durch Minimierung der Sperrdauer reduziert werden. Ein Prozeß, der einen "High Traffic Lock" besitzt, sollte nie auf eine andere Sperre warten, niemals einen E/A-Vorgang anstossen oder einen Page Fault provozieren, d.h. nur auf häufig benutzte Seiten zugreifen. Falls sich dennoch ein Konvoi gebildet hat, wird als Lösung in (BGMP79) ein Abweichen von der FIFO-Zuteilungsdisziplin, die eine faire Betriebsmittelvergabe garantiert, vorgeschlagen. In der Situation S1 hat P1 nach seiner erneuten Aktivierung folgende Maßnahmen durchzuführen:

- P1 entfernt alle am Konvoi beteiligten Prozesse aus der Warteschlange von S.
- Die Sperre S wird durch P1 als frei gekennzeichnet.
- P1 veranlaßt die Aktivierung aller Konvoi-Mitglieder durch das BS (SIGNAL).

Nach einem Ausführungsintervall von 1000 Instruktionen wird jetzt P1 S wahrscheinlich frei vorfinden. Wegen der geringen Wahrscheinlichkeit des Auftretens von Konvois bei der unterstellten Disziplin der DBS-Prozesse (nur bei Zeitscheibenende in einer kritischen Phase) finden auch alle anderen Pi nach ihrer Aktivierung S frei vor.

3. Herkömmliche Konzepte der DBS-Einbettung

3.1 Zuordnung eines DBS-Prozesses

Das Schutzbedürfnis von DB-Anwendungen erzwingt bei der Struktur heutiger BS die räumliche Trennung von AP und DBS. Das einfachste Zuordnungsprinzip ist in Bild 2 skizziert. Es wurde dabei der Idealfall unterstellt, daß individuelle Adreßräume zugeschnittene Speicherbereiche DAi für den Datenaustausch gemeinsam benutzen können. In praktischen Einsatzfällen liegen mehrere DAi in einem Segment, das mehrere APi und das DBS gemeinsam benutzen. Systempuffer und globale Systemtabellen GST bleiben von allen Benutzern des Systems völlig isoliert. Die Kommunikation zwischen APi und DBS erfolgt über Kommunikations- und Synchronisationsprimitive des BS (SIGNAL/WAIT).

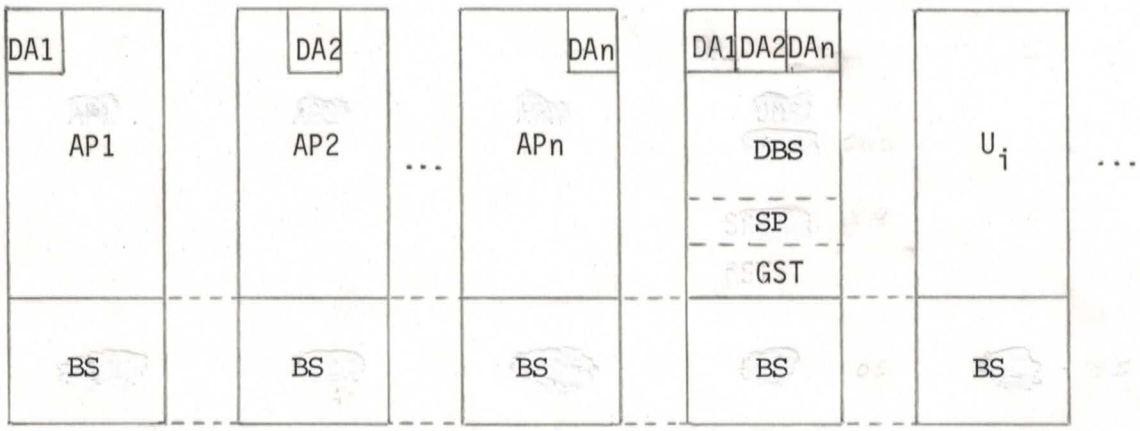


Bild 2: Zuordnung von n AP-Prozessen zu einem DBS-Prozeß

Falls nur ein DBS-Prozeß n AP-Prozesse bedienen soll, muß er sein eigenes multi-tasking durchführen, was auf eine beträchtliche Erhöhung der Komplexität des DBS hinausläuft. Im Betrieb stellt er sozusagen den Engpass bei der DB-Verarbeitung dar. Dieses Konzept ist deshalb nur gangbar, wenn der DBS-Prozeß durch hohe Prioritätszuweisung bei der Zeitscheibenvergabe durch das BS bevorzugt werden kann. Da die gesamte DB-Verarbeitung ruht, wenn der DBS-Prozeß blockiert ist, muß er zwingend asynchrone DB-Ein-/Ausgabe durchführen. Im Falle einer Fehlseitenbedingung dieses Prozesses stoppt jedoch der DB-Betrieb. Die Synchronisation der Zugriffe auf die DB, die er für die verschiedenen APi ausführt, ist vergleichsweise einfach, da er als

einzige Instanz die DB verwaltet. Sperren für Datenobjekte, die gewöhnlich als spezielle Kontrollblöcke implementiert sind, dienen dem DBS-Prozeß quasi als Merkhilfe, welchem APi ein Datenobjekt gerade zugeordnet ist. Da er sein eigenes multi-tasking durchführt, kann er den Task für ein APi deaktivieren, falls er durch eine Sperre eines anderen blockiert wird. Da in einem solchen Fall keinerlei Interaktion mit dem BS notwendig ist, bleibt die Aktivierung und Deaktivierung von Tasks eine billige Operation.

3.2 Symmetrische Zuordnung von DBS-Prozessen

Die Möglichkeit der Paarbildung von AP- und DBS-Prozessen führt auf eine 2n-Adreßraum-Lösung. Sie ist in Bild 3 skizziert.

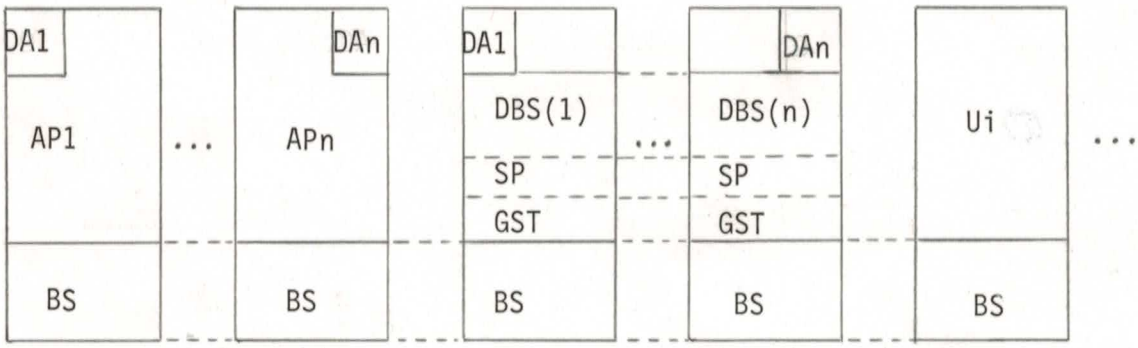


Bild 3: Symmetrische Zuordnung von DBS-Prozessen zu AP-Prozessen

Die Vorteile dieses Konzeptes liegen bei der DBS-Implementierung vor allem darin, daß es im wesentlichen als Einbenutzer-Programm ohne die Notwendigkeit des multi-tasking entworfen werden kann. Dadurch, daß im Betrieb mehrere Aktivierungen des DBS in verschiedenen Adreßräumen über gemeinsame Segmentbenutzung reentrant ablaufen, garantiert dieses Prinzip eine "gleichmäßigere" Verteilung der Last auf die einzelnen DBS-Prozesse, setzt außerdem keine besonderen Prioritätssteuerungen durch das BS voraus und verhindert die totale Blockierung der DB-Aktivitäten durch Paging. Neben der Kommunikation AP-DBS ist in diesem Konzept auch die Synchronisation DBS-DBS notwendig, die im Wartefall mit Hilfe des BS gelöst wird. Da DBS-Code, SP und GST physisch nur einmal vorhanden sind, ist für DBS-Prozesse ein gutes Pagingverhalten zu erwarten. Dadurch, daß bei synchroner Verarbeitung von AP, was als Regelfall angenommen werden soll, bei der obigen Konfiguration immer nur einer von zwei zusammengehörigen Prozessen aktiv ist, wird durch das symmetrische Zuordnungskonzept gewissermaßen eine Verschwendung von Adreßräumen eingeführt. Die 2n-Adreßraum-Lösung begrenzt außerdem unter Berücksichtigung der maximalen Anzahl von Prozessen in heutigen BS die mögliche Parallelität von DB-Anwendungen sehr stark.

3.3 Asymmetrische Zuordnung von DBS-Prozessen

Das Argument der synchronen Verarbeitung führt zu der Überlegung, n AP-Prozessen m DBS-Prozesse (n>m) zuzuordnen. Da jetzt die Zuordnung AP-DBS situationsbedingt ist, muß quasi eine zentrale Instanz die Vergabe der DBS-Prozesse regeln. Es ist ein DBS-Verwalterprozeß notwendig, der die Verteilung der ankommenden Aufträge vornimmt. Eine solche Art Monitor-Prozeß ist auch im Falle der symmetriechen Zuordnung notwendig, da sich die normalen DBS-Prozesse für die globalen Systemverwaltungsaufgaben wie Behebung von Engpässen, Durchführung von Sicherungspunkten, Initialisierung des Wiederanlaufs etc. nicht zuständig fühlen.

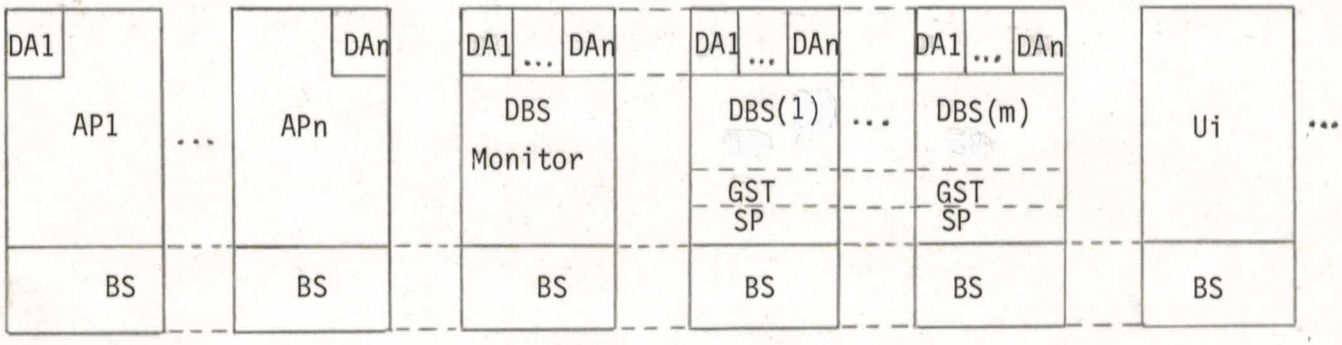


Bild 4: Asymmetrische Zuordnung von n AP-Prozessen zu m DBS-Prozessen

Dem Vorteil der Reduzierung von Adreßräumen steht bei diesem Konzept der Nachteil erhöhter Kommunikation gegenüber. Ein DBS-Aufruf, der über den DBS-Monitor verteilt wird, erfordert bei symmetrischer Rückkehr über den Monitor-Prozeß 4 Prozeßwechsel. Da die Rückkehr vom DBS-Prozeß zum AP-Prozeß direkt erfolgen kann, bleiben pro Benutzeranforderung die 3 Prozeßwechsel APi → DBS-Monitor → DBS (j) → APi übrig, was einen Kommunikationsaufwand von 15000 Instruktionen ausmacht.

4. Idealkonzept der DBS-Einbettung

Ziel der DBS-Einbettung in eine BS-Umgebung sollte die Minimierung des BS-Interaktion bei der Kommunikation AP-DBS und bei der Synchronisation von DBS-Prozessen unter Einhaltung des geforderten Schutzbedürfnisses sein. Bei Zuweisung von AP und DBS in verschiedene Adreßräume (scheduling units) wird die totale Isolation garantiert, die Kooperation jedoch sehr aufwendig, da sich die aus der Sicht der BS unabhängigen und asynchronen Prozesse durch allgemeine Kommunikationsprotokolle mit BS-Unterstützung verständigen müssen. Die Aufteilung eines Adreßraumes in verschiedene Schutzgebiete erlaubt die Zuordnung von AP und DBS im selben Adreßraum, wobei beide als unabhängige, hinreichend isolierte Tasks eines Prozesses ablaufen. Damit kann das Schutzbedürfnis von DB-Anwendungen ohne die Kosten und Verzögerungen, die durch Prozeßwechsel

bei der Kooperation auftreten, befriedigt werden. Durch gemeinsame Benutzung eines Schreib-/Lese-Segmentes zwischen AP und DBS wird der Datenaustausch denkbar einfach. Die Übergabe der Kontrolle kann über eine synchrone Schnittstelle ohne BS-Interaktion erfolgen.

Die Ausnutzung gemeinsamer Segmente in verschiedenen Adreßräumen gestattet für jedes AP jeweils die Aktivierung eines DBS-Programmes, das wegen seiner Reentrant-Fähigkeit nur in einer physischen Kopie geladen zu sein braucht. Da jede DBS-Aktivierung nur ein AP zu bedienen hat, kann das DBS im wesentlichen als Einbenutzer-Programm entworfen sein. Aus Konsistenzgründen sind beim Zugriff auf gemeinsam benutzte Daten Sperren zu setzen, die als Kontrollblöcke im gemeinsamen Speicherbereich abzulegen sind. Durch bestimmte Konventionen und Zugriffsprotokolle können sich die einzelnen Aktivierungen des DBS bei der Bearbeitung dieser Daten selbständig synchronisieren. Synchronisierungshilfe durch das BS wird außer beim hoffentlich seltenen Auftreten des Konvoi-Phänomens nur noch erforderlich, wenn auf die Freigabe von Sperren (low traffic locks) (BGMP 79) für Datenobjekte der DB zu warten ist. In diesem Fall treten unvorhersehbar lange Wartezeiten für die anfordernde DBS-Aktivierung auf, so daß sich der zugehörige AP-DBS-Prozeß sinnvollerweise schlafen legt, bis ihm die Freigabe der Sperre signalisiert wird.

Die Zuordnung von AP-DBS-Prozessen ist in Bild 5 skizziert.

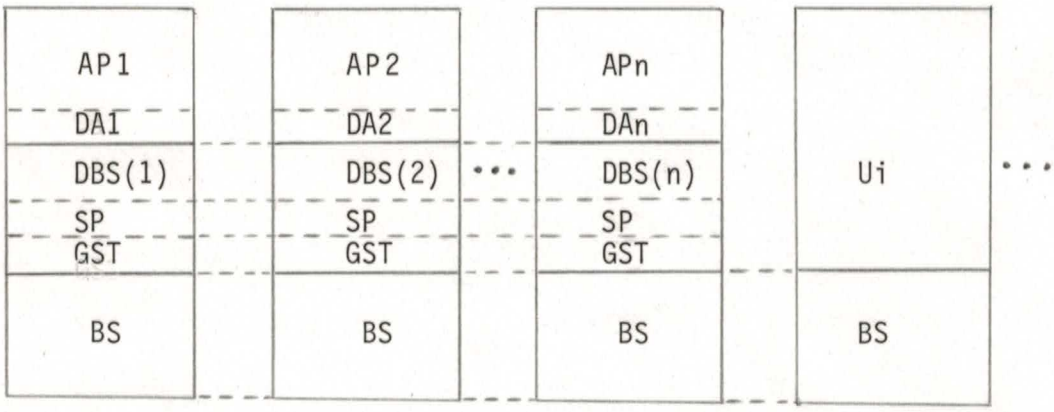


Bild 5: Symmetrische Zuordnung synchroner AP-DBS Prozesse.

Bei diesem Zuordnungskonzept sind keine Annahmen über die Prioritätsvergabe des BS-Schedulings nötig. Für den DBS-Code und die globalen Systemtabellen ist wegen der Vielfach-Benutzung ein gutes Paging-Verhalten zu erwarten. Eine Fehlseitenbedingung führt lediglich zur Zuteilung des Rechnerkerns an einen anderen Prozeß und nicht zum Stillstand der DB-Verarbeitung.

Ein Implementierungvorschlag für dieses Konzept wird in (At 79) unter dem Namen VCS (Virtual Control Storage) für VM/370 diskutiert. Als wesentliches Ergebnis einer Testimplementierung ist festzuhalten, daß bei hinreichend starker Iso-

lierung die Kontrollübergabe an der synchronen Schnittstelle zwischen AP- und DBS-Tasks im Vergleich zur Kommunikation zwischen asynchronen Prozessen etwa um den Faktor 10 reduziert wurde.

Literaturverzeichnis

(At 79) Attanasio, C.R.: Virtual Control Storage - Security Measures in VM/370, in: IBM Systems Journal, Vol. 18, No.1, 1979, S. 93 - 110.

(BGMP79) Blasgen, M., Gray, J., Mitoma, M., Price, T.: The Convoy Phenomenon, in: ACM Operating Systems Review, Vol.13, No.2, April 1979, S. 20-25.

(Ef 78) Effelsberg, W. Die Optimierung der Verwaltung des Systempuffers in Datenbanksystemen, Forschungsbericht DV I 78-3, FG Datenverwaltungssysteme I, TH Darmstadt, November 1978.

(Gr 78) Gray, J.N.: Notes on Data Base Operating Systems, in: Lecture Notes in Computer Science 60, Advanced Course on Operating Systems, Springer-Verlag Berlin, 1978, S. 393-481.

(GrW 79) Gray, J.N., Watson, V.: A Shared Segment and Inter-Process Communication Facility for VM/370, IBM Research Report, RJ 2450, Jan 15,1979, San Jose, Calif.

(Hä 78) Härder,T.: Implementierung von Datenbanksystemen, Carl-Hanser-Verlag, 1978

(MK 79) Mac Kinnon, R.A.: The Changing Virtual Machine Environment: Interfaces to Real Hardware, Virtual Hardware, and other Virtua Machines, in: IBM Systems Journal, Vol. 18, No.1, 1979, S. 18-46.

(SM 79) Seawright, L.A., MacKinnon, R.A.: VM/370 - a Study of Multiplicity and Usefulness, in: IBM Systems Journal, Vol. 18, No. 1, 1979, S. 4-17.