Betriebssystemkonzepte für Massiv Parallele Systeme

Wolfgang Schröder-Preikschat
GMD FIRST an der Technischen Universität Berlin
Hardenbergplatz 2, W-1000 Berlin 12
schroeder-preikschat@kmx.gmd.dbp.de

1. Februar 1991

Vorwort

Benötigen massiv parallele Rechnerarchitekturen überhaupt spezielle Betriebssysteme? Spätestens wenn die Rechenleistung eines neu konstruierten Rechners nicht den Erwartungen entspricht, werden solche oder ähnliche Fragen immer wieder aufgeworfen. Der vorliegende Artikel motiviert die Notwendigkeit spezieller, paralleler Betriebssysteme für auf verteiltem Speicher basierender massiv paralleler Rechnerarchitekturen. Am Beispiel von PEACE werden Betriebssystemkonzepte vorgestellt, die eine anwendungsorientierte Betriebsmittelverwaltung solcher Architekturen unterstützen.

1 Einleitung

Die Ursache für die im Vorwort genannte Infragestellung ist im wesentlichen darin zu sehen, daß oftmals die Haltung vertreten wird, massiv parallele Rechner seien klassischen *shared memory* Parallelrechnern durchaus vergleichbar, weshalb sich insbesondere die üblichen Betriebssystemkonzepte direkt übertragen ließen. Mehr noch, da massive Parallelität ein Wesensmerkmal der Anwendungsprogramme sei - denn ohne dies wäre zweifellos die Forderung nach unterstützenden Hardwarestrukturen nicht begründet -, bestände keine Notwendigkeit an speziellen Betriebssystemfunktionen. Die Anwendungsprogramme selbst würden schon für alles sorgen. Im übrigen umfasse das Betriebssystem sowieso nur einen Bruchteil der gesamten, auf dem Parallelrechner ablaufenden Software. Schon allein deshalb sei es von zweitrangiger Bedeutung.

Die Realität sieht jedoch gänzlich anders aus. Dies haben alle nennenswerten Rechnerentwicklungen der letzten Jahre gezeigt, so auch SUPRENUM [5]. Anforderungen massiv paralleler Systeme derart zu definieren, daß sie sich lediglich durch innovative Anwendungsprogramme, Übersetzer und Rechnerarchitekturen auszeichnen, aber mit herkömmlichen oder überhaupt keinen Betriebssystemfunktionen auskommen, mag zwar wünschenswert sein, ist und bleibt jedoch reine Utopie. Eine Synergie von allen Bereichen ist notwendig. Daß hier insbesondere im Softwarebereich noch ein erheblicher Mangel besteht, den es

unbedingt abzuschaffen gilt, zeigen verschiedene nationale und internationale Förderprogramme [3]. Der Bedarf an neuen Betriebssystemstrukturen gerade für massiv parallele Systeme ist eindeutig gegeben.

2 Massiv Parallele Systeme

Unter innovativen, massiv parallelen Systemen werden Rechnerarchitekturen verstanden, die aus mehreren tausend autonomer Prozessoren bestehen können. Das Wesensmerkmal dieser Systeme ist ihre *distributed memory* Organisation, die die Grundlage dafür ist, daß der Parallelitätsgrad durch weitere Prozessoren nahezu beliebig ausgebaut werden kann. Die Prozessoren sind über ein Hochgeschwindigkeitsnetzwerk miteinander verbunden. Sie besitzen nur direkten Zugriff auf ihren lokalen Speicher. Globaler Speicherzugriff, auf die jeweils lokalen Speicher anderer Prozessoren, erfolgt ausschließlich durch Nachrichtenkommunikation. Gefordert sind objekt-basierte Programmierdisziplinen.

Eine parallele Anwendung ist demzufolge in autonome Prozesse zu unterteilen, die dann über das Prozessornetzwerk in geeigneter Topographie zu verteilen sind. Jeder Prozeß bearbeitet nur ein Teilproblem und synchronisiert sich (implizit oder explizit) mit anderen Prozessen, um Zwischenergebnisse auszutauschen oder das Endergebnis aufzubereiten und zu produzieren.

Es ist leicht erkennbar, daß die Programmierung massiv paralleler Rechner in neue Dimensionen vordringt [7]. Dies gilt eben nicht nur für Anwendungen, sondern gerade auch für jene Systemfunktionen, die die Betriebsmittelverwaltung des Rechners repäsentieren. Eine Vielzahl von Prozessoren ist zu überwachen und ein Mindestmaß an Verfügbarkeit ist zu gewährleisten. Kommunikation ist wenigstens dahingehend zu unterstützen, daß den Anwendungen die Feinheiten (und Unwegbarkeiten) des Netzwerkzugangs verborgen bleiben, nicht nur aus Portabilitätsgründen. Mögliche Lücken zwischen der im Netzwerk physikalisch gegebenen und der von Anwendungen geforderten Datentransfersicherheit sind zu überbrücken - die Lücken sind da, insbesondere auch bezogen auf die unterschiedlichen Kommunikationssemantiken [1].

Anwendungen müßen ortstransparent ablaufen können, damit Lastausgleich überhaupt ermöglicht wird. Dies wiederum erfordert Zugriffstransparenz [11] z.B. auf Dateien, Ein-/Ausgabegeräte und Systemdienste, aber auch auf die Anwendungsprozesse selbst. Das initiale Hochfahren aller Prozessoren und das netzwerkweite Laden von Prozessen ist zu ermöglichen. Hier stellt sich u.a. das Problem, globale Systemabläufe zu synchronisieren und die minimale Funktionsfähigkeit des Systems zu überprüfen. Zentralisierte Ansätze sind aus naheliegenden Gründen zu vermeiden, damit die Inbetriebnahme des Rechners nicht zum Alptraum wird. Schließlich werden auch traditionelle Betriebssystemdienste wie Prozeß- und Speicherverwaltung erwartet - bei weitem jedoch der kleinste Teil aller Systemfunktionen.

Massiv parallele Systeme ohne Betriebssystem betreiben zu wollen, würde zwangsläufig dazu führen, daß Anwendungsprogrammierer selbst ihr eigenes Betriebssystem schreiben müßten, bevor sie sich ihrer eigentlichen Problemlösung zuwenden könnten. Diese im übertragenen Sinne steinzeitliche Vorgehensweise

entspräche sicherlich nicht dem, was heutzutage unter anwendungsorientierter Programmierung verstanden wird. Sie wäre lediglich Anlaß dazu, den Begriff und das Arbeitsgebiet Betriebssystem nach mehr als drei Jahrzehnten erneut zu erfinden [13].

3 Konzepte Paralleler Betriebssysteme

Mit aller Deutlichkeit ist zu bejahen: massiv parallele Rechnerarchitekturen benötigen sehr wohl spezielle Betriebssysteme! Dennoch bleibt die eingangs diskutierte Infragestellung prägend für die Systementwicklung. Sie einfach zu ignorieren, weil es grundsätzlich unmöglich ist, die Arbeitsfähigkeit eines Rechners vollständig ohne Betriebssystem garantieren zu wollen, käme einer Selbstisolation gleich. Vielmehr ist dafür Sorge zu tragen, daß das Betriebssystem der Infragestellung dahingehend gerecht wird, durch Struktur und Dienstleistungen anwendungsbezogen für den Benutzer konfigurierbar zu sein. Um Akzeptanz zu erzielen, muß dies für den Benutzer ersichtlich und gegebenenfalls auch nachvollziehbar sein.

Betriebssystemkonstruktion ist heute in erster Linie ein Entwurfs- und Strukturproblem, und stellt längst keine grundlegende algorithmische Unbekannte mehr dar. Daran haben auch die relativ jungen Gebiete verteilter und massiv paralleler Systeme nichts wesentliches geändert. Im Gegenteil, sie haben diese Situation bedeutend verschärft und dadurch die Entwurfs-/Strukturproblematik nur noch weiter erhärtet. Aufgrund ihrer Komplexität gilt dies in besonderem Maße für massiv parallele Systeme. Obgleich sie verteilte Systeme darstellen, sind jedoch verteilte Betriebssysteme [18] allein nicht der Schlüssel zum Erfolg.

Leichtgewichtige Systemstrukturen und, vor allen Dingen, -ausprägungen sind gefordert. Welche Konzepte dafür geeignet sind und wie sie dann zur Konstruktion des verteilten und parallelen Betriebssystems PEACE [16] umgesetzt worden sind, ist Inhalt der nachfolgenden Abschnitte. Daß damit nicht nur massiv parallele Systeme arbeitsfähig im Sinne der auf ihnen ablaufenden Anwendungen werden, sondern auch die Konstruktion beliebiger (verteilter) Betriebssysteme unterstützt wird, ist eine mehr als positive (und auch nicht ungewollte) Nebenerscheinung. Erfahrungen mit der SUPRENUM Entwicklung sowie die neuen, im wesentlichen auch architektonischen Anforderungen für MANNA [6] haben hier wesentlichen Einfluß geliefert.

4 Der Familienbegriff

Die PEACE Struktur ist geprägt durch das Konzept von Programmfamilien [12]. Dieses Konzept geht davon aus, eine *minimale Basis* von Systemfunktionen zu finden, die dann durch *minimale Erweiterungen* schrittweise zu einem vollständigen Systemkomplex komplettiert wird. Ausschließlich Mechanismen und keine Strategien zur Verfügung zu stellen, ist Wesensmerkmal der minimalen Basis. Damit ist der Grundstein für eine anwendungsorientierte Betriebssystemstruktur gelegt, da keine einschränkenden Entwurfsentscheidungen vorweg

genommen werden.

Durch minimale Erweiterungen erfolgt die funktionale Anreicherung der minimalen Basis, wodurch über mehrere Iterationsstufen hinweg immer neue (mehr oder weniger anwendungsorientierte) Basismoduln geschaffen werden. Dieses von unten ablaufende Konstruktionsverfahren beim Systementwurf wird von oben, letztlich durch die Anwendungen, kontrolliert. Neue Basismoduln werden dann in den Baukasten (die Familie) mit aufgenommen, wenn die Gegebenheiten dies erfordern.

Alle derart identifizierten PEACE Moduln sind *abstrakte Datentypen* [10]. Funktional eng zusammenhängende Moduln, die in ihrer Gesamtheit typische Betriebssystemfunktionen implementieren, bilden die Bausteine der PEACE Betriebssystemfamilie. Typische Bausteine sind etwa Prozeß-, Speicher- und Dateiverwaltung. Neue Betriebssystemausprägungen werden dadurch geschaffen, indem neue Bausteine hinzukommen. Dabei tritt oft die Situation ein, daß ein bereits bestehender Baustein mit einer alternativen, internen Repräsentation (bzw. Implementierung) wiederholt vorhanden ist. Je nach Anforderungsprofil der jeweiligen Anwendung, kommt die passende Bausteinrepräsentation zur Geltung. Beispiel hierfür wäre etwa eine Speicherverwaltung, die mit und ohne Hardwareunterstützung auskommt, in beiden Fällen aber dieselben Dienstschnittstellen zur Verfügung stellt.

5 Eine Familie Paralleler Betriebssysteme

Soweit die Theorie. Wie sieht jedoch die Praxis aus? In PEACE werden, bis auf eine einzige Ausnahme, alle Bausteine als aktive Objekte aufgefaßt. Das einzige passive Objekt ist der Nukleus - dazu später mehr. Ein aktives Objekt ist als Team leichtgewichtiger Prozesse implementiert, d.h. es kapselt mehrere Kontrollflüsse innerhalb eines Adreßraumes [4]. Interaktionen zwischen den aktiven Objekten sind zugriffstransparent und erfolgen ausschließlich durch Fernaufrufe. PEACE ist somit streng objekt-basiert und gestattet dadurch vom Ansatz her eine beliebige Verteilung der Bausteine über die massiv parallele Rechnerarchitektur.

Bei Prozessoren, die zwischen privilegierten und nicht-privilegierten Arbeitsmodus unterscheiden, ist festzulegen, welcher Arbeitsmodus für die Bausteine jeweils in Frage kommt. Es ist Trugschluß, anzunehmen, ein Betriebssystem muß vollständig im privilegierten Arbeitsmodus des Prozessors laufen. Nur die wenigsten Komponenten sind tatsächlich davon betroffen. Können diese aufgrund einer ungenügenden Systemstruktur jedoch nicht entflechtet werden, sind somit notgedrungenerweise alle Komponenten und damit das gesamte Betriebssystem dem privilegierten Arbeitsmodus unterworfen.

5.1 Die Triade

In PEACE gilt die Maxime, die Anzahl der Bausteine des privilegierten Arbeitsmodus auf ein absolutes Minimum zu reduzieren. Nur dadurch kann auch der Aufwand prozessorgebundener Systemsoftware minimiert und die Hochfahrproblematik massiv paralleler Systeme erheblich verringert werden. Hunderte oder

tausende von Prozessoren initial mit Systemsoftware zu laden, ist ohnehin ein nicht triviales Unterfangen und wird dadurch gerade nicht vereinfacht, daß pro Prozessor bereits eine Vielzahl von Bausteinen zu laden wäre. Eine hochmodulare Betriebssystemstruktur ist gefordert (Abbildung 1).

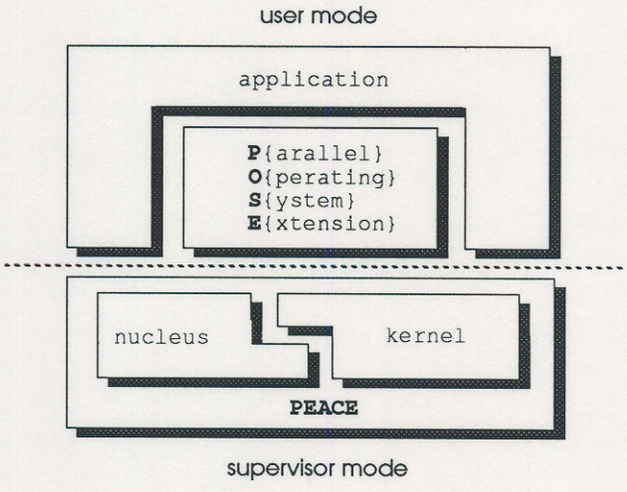


Abbildung 1: PEACE Organisation

Anwendungen besitzen direkten Zugang zu den drei Hauptkomponenten *System* [1], *Kern* und *Nukleus*. Durch die Entflechtung der Komponenten ist ein Flaschenhals bei der Dienstinanspruchnahme nicht gegeben. Die Systembausteine sind alle dem nicht-privilegierten Arbeitsmodus (*user mode*) zugeordnet und zum überwiegenden Teil ortsunabhängig. Sie können damit frei verteilt werden. Kern- und Nukleusbausteine sind potentiell dem privilegierten Arbeitsmodus (*supervisor mode*) zugeordnet und ortsabhängig, d.h., sie stellen prozessorgebundene Bausteine dar. Der Nukleus ist die minimale Basis zur Ausführung von Anwendungsprogrammen. Kern- und Systemkomplex repräsentieren minimale Erweiterungen, wobei ersterer zusätzliche (meist prozessorabhängige) Mechanismen und letzterer Strategien definiert.

Das Resultat ist kein monolithisches, sondern ein funktional dediziertes Betriebssystem. Die Verteilung der Bausteine erfolgt problem- und anwendungsorientiert. In Abhängigkeit von der gegebenen Hardwareorganisation erfolgt die problemorientierte Bausteinabbildung. Nicht jeder Prozessor ist daher mit denselben Bausteinen ausgestattet - wo kein Plattenanschluß verfügbar ist, wird auch kein Plattentreiber benötigt. Eine vergleichbare Situation, die anwendungsorientierte Bausteinabbildung, ist durch unterschiedliche Anwendungsprofile gegeben. Anwendungen, die etwa auf isolierte Prozeßadreßräume verzichten können, erfordern keinen *memory management unit* (MMU) Treiberbaustein. Damit wird ihr Ablauf auch auf Rechnerarchitekturen unterstützt,

[1] POSE, für *p*arallel *o*perating *s*ystem *e*xtension.

die keine dedizierte Hardware zum Adreßraumschutz anbieten.

5.2 System

Der Systemkomplex (d.h., POSE) stellt anwendungsorientierte Dienste zur Verfügung. Hierbei handelt es sich konkret um Symbol-, Objekt-, Prozeß-, Speicher-, Ein-/Ausgabe-, Dateiverwaltung, Ausnahmebehandlung, Laden und Netzwerkkopplung. Mit Ausnahme der beiden zuerst genannten Bausteine kapseln allen anderen typische Betriebssystemfunktionen.

Symbol- und Objektverwaltung sind zwei der wichtigsten PEACE Bausteinkomplexe überhaupt. Sie ermöglichen u.a. die Zugriffstransparenz durch Fernaufrufe. Alle aktiven Objekte, bzw. die Schnittstellenfunktionen, und ihre aktuelle Lokalität sind in einem hierarchisch strukturierten und verteilten Namensraum erfaßt [14].

Der Namensraum wächst und schrumpft dynamisch in seiner Größe. Neu erzeugte aktive Objekte machen sich innerhalb einer bestimmten (anwendungsorientierten) Domäne namentlich bekannt. Namen terminierender aktiver Objekte werden anschließend wieder invalidiert. Anwendungen selbst erhalten private Namensdomänen, womit eine logische Isolation erzielt wird. Namenskonflikte, d.h., die Bezeichnung unterschiedlicher aktiver Objekte mit demselben Namen, werden dadurch vermieden.

Die Bindung aktiver Objekte ist dynamisch. Sie wird erst dann durchgeführt, wenn das Objekt erstmalig durch einen Fernaufruf benutzt werden soll. Hier ist der Ansatzpunkt für die dynamische Rekonfiguration aktiver Objekte in PEACE gegeben [15]. Ist das bezeichnete Objekt nicht namentlich bekannt, wird es auf Anforderung erzeugt und nachgeladen. Die Entscheidung, wohin das Objekt zu plazieren ist, erfolgt entweder willkürlich (zwecks dynamischen Lastausgleichs) oder richtet sich nach objektgebundenen Attributen.

Der gesamte Systemkomplex verändert sich somit in Abhängigkeit von der jeweiligen Anwendung. Das wiederum bedeutet, daß ein massiv paralleles System nur mit den zum dynamischen Nachladen notwendigen PEACE Bausteinen initial vorgeladen werden muß. Um das Nachladen selbst zu aktivieren, ist jedoch zuerst eine Anwendung zu laden und ablaufen zu lassen. Dazu wird im Extremfall lediglich ein freier (bisher unbenutzter) Prozessor benötigt, der dann wenigstens mit dem Nukleus und einem Anwendungsprozeß zu laden wäre. Da auch die Dienstfunktionen der Bausteine zum dynamischen Nachladen über Fernaufrufe in Anspruch genommen werden, müssen diese Funktionen lediglich erreichbar, nicht aber auf jedem Prozessor angesiedelt sein. Der Anwendungsprozeß bewirkt schließlich, daß die von ihm benötigen Systemdienste (Bausteine) automatisch nachgezogen werden. Langwierige Hochfahrzeiten entfallen und werden auf ein abolutes Minimum reduziert.

Es sei an dieser Stelle schließlich noch angemerkt, daß die dynamische Rekonfiguration nicht nur auf den Systemkomplex beschränkt bleiben muß. Ebenso würde verfahren, wenn die Anwendungen aus aktiven Objekten mit ausgezeichneten Schnittstellenfunktionen beständen und damit Fernaufrufe für die Interaktionen der aktiven Anwendungsobjekte zum Einsatz kämen.

5.3 Kern

Der Kernkomplex ist für die Erzeugung und Zerstörung von Prozeßobjekten verantwortlich. Darüberhinaus stellt er Dienste zur Verfügung, um Prozeßobjekte (Kontrollflüsse) mit Adreßräumen zu verknüpfen und Ausführungsdomänen (zum *Scheduling*) zuzuordnen. Er läßt das resultierende Prozeßmodell jedoch vollständig offen und implementiert lediglich eine abstrakte PEACE Basismaschine für den Systemkomplex und die Anwendungen.

Der zweite große Aufgabenbereich des Kerns ist die Geräteverwaltung. Nach außen sind Geräte als aktive Objekte sichtbar, d.h., der Gerätetreiber ist als leichtgewichtiger Kernprozeß realisiert. Wie im Falle des Systemkomplexes sind sämtliche Kerndienste nur über Fernaufrufe aktivierbar. Der Dateiverwaltung, z.B., bleibt damit verborgen, wo tatsächlich die von ihr benutzten Platten liegen. Eine vollständige Geräteabstraktion ist erzielt. Ob Geräte physikalisch oder nur virtuell vorhanden sind, bestimmt letztlich das Innenleben des jeweiligen aktiven Objektes.

Als dritter Aufgabenbereich des Kerns ist die Verknüpfung aktiver Objekte mit Unterbrechungen (*Traps*, *Interrupts*) zu nennen. Synchrone oder asynchrone Unterbrechungen werden in Nachrichten umgewandelt und dem jeweils zugeordneten aktiven Objekt zugestellt. Die Unterbrechungsbehandlung muß somit nicht vor Ort stattfinden, was z.B. dem *Debugging* paralleler, verteilter Programme sehr dienlich ist. Unterbrechungen, die keine Behandlung erfahren, werden abgefangen und als Ausnahmesituationen angezeigt. Der hierfür zuständige PEACE Baustein ist dem Systemkomplex zugeordnet und nimmt Unterbrechungen an, die von mehreren Prozessoren (vermöge ihres jeweiligen Kerns) signalisiert worden sind. Falls angefordert, verteilt er die Unterbrechungsnachricht an andere aktive Objekte, so auch an Anwendungsprozesse.

5.4 Nukleus

Der einzige, auf jedem Prozessor notwendige PEACE Baustein ist der Nukleus, der nur für die netzwerkweite Kommunikation und die lokale Prozeßausführung verantwortlich ist. Der Nukleus ist darüberhinaus der einzige Baustein, dem kein eigener Kontrollfluß zugeordnet ist - er ist das einzige passive Objekt. Seine Dienstleistungen werden im herkömmlichen Sinne prozedural und immer lokal in Anspruch genommen. Hierzu kommen zwei gleichberechtigte Verfahren zum Einsatz, die zwischen adreßraumlokaler und adreßraumübergreifender Aktivierung unterscheiden.

Die adreßraumlokale Aktivierung erfolgt immer dann, wenn der Nukleus als Bibliothek dem umfaßenden Programm (aktiven Objekt) direkt eingebunden wurde. Dies ist der Fall für dedizierte Anwendungen, deren Prozesse in eins-zu-eins Beziehung auf die Prozessoren abbildbar sind. Auf dem jeweiligen Prozessor wird dann für die gesamte Laufzeit der Anwendung lediglich ein Prozeß ausgeführt. Dies bedeutet somit Einprogrammbetrieb für den betreffenden Prozessor.

Die adreßraumübergreifende Nukleusaktivierung entspricht dem üblichen Verfahren von Systemaufrufen in traditionellen Betriebssystemen. Sie kommt

immer dann zum Einsatz, wenn der betreffende Prozessor im Mehrprogrammbetrieb genutzt werden soll. Dem Nukleus ist dabei ein separater Adreßraum zugeordnet, der des Kerns, so daß lediglich der Kern den Nukleus adreßraumlokal aktivieren kann.

5.4.1 Funktional dedizierte Ausprägungen

Beide Verfahren resultieren in dedizierte Nukleusausprägungen, die als gleichberechtigte Mitglieder die PEACE Nukleusfamilie begründen (Abbildung 2). Einprogrammbetrieb im Falle der adreßraumlokalen Nukleusaktivierung bedeutet nicht, den mehrprogrammfähigen Nukleus zu nehmen und lediglich einen Prozeß ablaufen zu lassen. Vielmehr existiert hierzu eine spezielle Nukleusrepräsentation. Der Nukleus ist also ein abstrakter Datentyp, der viele interne Ausprägungen hat. In Abhängigkeit von der Anwendung kommt dann der passend zugeschnittene Nukleus zum Einsatz.

Der wesentliche Grund für diese Vorgehensweise ist es, den oftmals enorm harten Effizienzanforderungen massiv paralleler System gerecht zu werden. Genau hier ist einer der Hauptpunkte gegeben, weshalb bekannte verteilte Betriebssysteme (mit Ausnahme von PEACE) nicht für massiv parallele Rechner geeignet sind. Experimente mit PEACE auf SUPRENUM haben gezeigt, daß Einprogrammbetrieb mit einem mehrprogrammfähigen Nukleus bis zu 74 Prozent Leistungsverlust bei der Interprozeßkommunikation mit sich bringen kann [17]. Dies ist besonders kritisch für jene Anwendungen, die mit der Anzahl der zur Verfügung stehenden Prozessoren skalierbar sind und demzufolge keinen Mehrprogrammbetrieb benötigen. Um hier den verschiedenen Anwendungsprofilen auf der untersten Ebene des Systems gerecht zu werden, ist eine Nukleusfamilie passiver Objekte definiert worden.

Verschiedene Anforderungen (d.h., Funktionalitäten) implizieren unterschiedliche Nukleusrepräsentationen. Im Einprogrammbetrieb etwa wird unterschieden, ob ein oder mehrere Kontrollflüsse innerhalb desselben Adreßraumes zu unterstützen sind. Im Falle eines einzigen Kontrollflusses, ist der Nukleus nichts weiter als ein Kommunikationsmodul (*communication*). Soll Nebenläufigkeit möglich sein, verarbeitet der Nukleus zusätzlich leichtgewichtige Prozesse (*threading*). Für Mehrprogrammbetrieb wird die Ablaufkontrolle schwergewichtiger Prozesse, d.h., Teams leichtgewichtiger Prozesse, benötigt. Dies führt zu einer weiteren Nukleusrepräsentation (*scheduling*).

Adreßraumübergreifende Nukleusaktivierung bedeutet, daß Nukleus und Kern geschützt worden sind (*isolation*), was aber nicht für die Prozesse gelten muß. Speicherschutz allgemein ist Thema einer weiteren, ausgezeichneten Repräsentation (*protection*). Um letztlich der Forderung nach Mehrbenutzerbetrieb gerecht zu werden, muß der Nukleus auch bei der netzwerkweiten Kommunikation die Integrität der Prozesse gewährleisten. Eine weitere Ausprägung ergibt sich (*security*).

Gemäß der Stammbaumstruktur erhöht sich die Nukleusfunktionalität von links nach rechts und von oben nach unten. In gleicher Weise verringert sich aber auch die Kommunikationseffizienz. Die oben genannten 74 Prozent drücken in etwa die Diskrepanz zwischen adreßraumlokaler Nukleusausprägung für leicht-

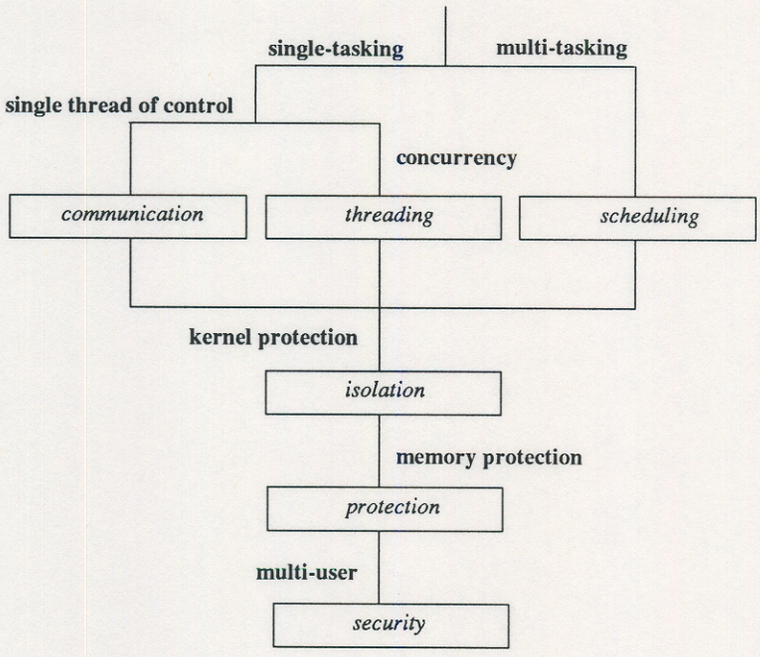


Abbildung 2: Stammbaum des PEACE Nukleus

gewichtige Prozesse (*threading*) und der Ausprägung für schwergewichtige Prozesse mit vollständigem Speicherschutz (*protection*) aus. Es ist offensichtlich nicht angeraten, nur eine Lösung anzubieten. Anwendungsorientiert bedeutet gerade auch, die beste Lösung von vielen auswählen zu können. Diesem Zweck dient die Nukleusfamilie.

5.4.2 Problemorientierte Struktur

Die interne Organisation des Nukleus muß dem Familienkonzept entsprechen, damit neue Nukleusausprägungen auch ohne großen Aufwand geschaffen werden können. Hierzu ist eine Struktur bestehend aus drei Bausteinen festgelegt worden (Abbildung 3).

Der Baustein NICE (*network independent communication executive*) ist für die lokale Prozeßausführung verantwortlich und treibt das systemweite Internukleusprotokoll zur entfernten Prozeßzustandskontrolle. Mit COSY (*communication system*) wird eine Abstraktion von den gegebenen Netzwerkeigenschaften erreicht. Es kapselt das dem Netzwerk zugeschnittene Datentransferprotokoll. Die Abstraktion von der tatsächlichen, physikalischen Netzwerkschnittstelle leistet CLUB (*club of network driver*). Dieser Baustein beherbergt den Gerätetreiber und bindet den Nukleus an die gegebene Netzwerkhardware. Der obere Einstieg in den Nukleus erfolgt potentiell adreßraumübergreifend und dann über Traps, wohingegen der untere Einstieg potentiell über Interrupts stattfindet. Im Falle eines Interrupts, z.B. nach dem physikalischen Nachrichtenempfang,

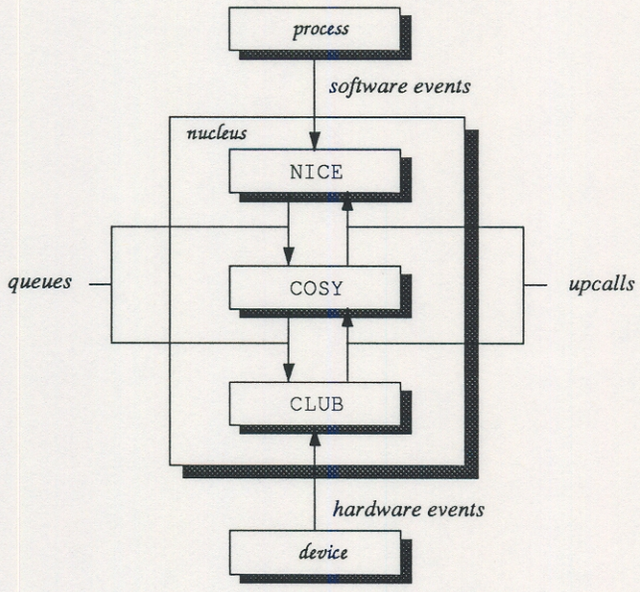


Abbildung 3: Interne Nukleusorganisation

findet die Interaktion mit den höheren Ebenen über *Upcalls* [2] statt.

Verschiedene Ausprägungen des Nukleus bezogen auf die Nukleusfamilie werden durch entsprechende NICE Bausteine realisiert. Jedes Familienmitglied basiert dann vermöge von COSY auf ein abstraktes Kommunikationssystem und ist damit selbst unabhängig vom jeweils zum Einsatz gekommenen Kommunikationsprotokoll. Der gesamte Nukleus schließlich geht von einer abstrakten physikalischen Netzwerkschnittstelle aus, die durch CLUB zur Verfügung gestellt wird. Daß das Netzwerkgerät wiederum virtuell sein kann, zeigt die Tatsache, den Nukleus als Gastebene oberhalb herkömmlicher Betriebssysteme ablaufen lassen zu können.

6 Ausblick

Der PEACE Ansatz geht weit darüber hinaus, was international an aktueller Diskussion bzgl. Mikrokern eines verteilten Betriebssystems stattfindet. Das, was z.B. in Mach und Chorus als Mikrokern verstanden wird, umfaßt in PEACE immerhin den komplexesten Nukleus (*security*), den gesamten Kern und die Bausteine des Systemkomplexes zur Prozeß-, Speicherverwaltung und Ausnahmebehandlung. Dieser Mikrokern läuft dann auf jedem Prozessor eines verteilten Systems. Anwendungen unter PEACE können jedoch bereits nur durch den Nukleus (egal welcher Ausprägung) auf jedem Prozessor unterstüzt werden.

PEACE durchläuft gegenwärtig eine Metamorphose vom prozeßorientierten zum dynamisch rekonfigurierbaren und objekt-orientierten System, was sich u.a. darin niederschlägt, eine schrittweise Neuimplementierung in C++ durch-

zuführen. So konzentrieren sich die aktuellen Arbeitsinhalte für MANNA auf die Realisierung der Objektverwaltung und der Nukleusfamilie. Der Fernaufrufmechanismus ist einer Überarbeitung unterworfen und eine neue, objektorientierte Schnittstellendefinitionssprache inklusive Fernaufrufgenerator auf Basis von C++ wird implementiert. Viel stärker als bei SUPRENUM wird darauf Wert gelegt, hauptsächlich Basismechanismen zur Verfügung zu stellen und damit die Konstruktion beliebiger Systeme, insbesondere auch Betriebssysteme, zu ermöglichen.

Der Bereich virtuell gemeinsamer Speicher (*virtually shared memory*) nimmt eine dominierende Stellung der gegenwärtigen PEACE Weiterentwicklungen für MANNA ein [8]. Darüberhinaus werden spezielle Laufzeitsystemmodelle und die dazu passenden Betriebssystembausteine für massiv parallele Programmstrukturen entwickelt. All diese Arbeiten verfolgen das Ziel, die Qualität (massiv) paralleler Rechensysteme durch Softwaremaßnahmen zu verbessern. Unter dieser Prämisse werden auch Verfahren zum dynamischen Lastausgleich und zur Fehlertoleranz in PEACE integriert. Die Zukunft sieht hier u.a vor, mobile, persistente Objekte zu verwalten [9].

Literatur

[1] P. M. Behr, W. K. Giloi, W. Schröder, "Synchronous versus Asynchronous Communication in High Performance Multicomputer Systems", IFIP Working Conference 5, Stanford, August 22-26, 1988

[2] D. D. Clark, "The Structuring of Systems Using Upcalls", *ACM Operating Systems Review*, 19, 5, pp. 171-180, 1985

[3] ESPRIT IWG, Workprogramme 1991, DG XIII/A-1, 24th October 1990

[4] C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages", *Artificial Intelligence* 8, pp. 323-364, 1977

[5] W. K. Giloi, "The SUPRENUM Architecture", CONPAR 88, Manchester, UK, Cambridge University Press, pp. 10-17, 1989

[6] W. K. Giloi, "Entwurf und Erprobung einer universellen, massiv-parallelen Rechnerarchitektur für nicht numerische Anwendungen", BMFT Antrag auf Förderung eines Forschungsvorhabens, GMD FIRST, Berlin, FRG, 1989

[7] W. K. Giloi, W. Schröder-Preikschat, "Programming Models for Massively Parallel Systems", International Symposium on New Information Processing Technologies '91, Tokyo, Japan, March 13-14, 1991

[8] W. K. Giloi, C. Hastedt, F. Schön, W. Schröder-Preikschat, "A Distributed Implementation of Shared Virtual Memory with Strong and Weak Coherence", 2nd European Distributed Memory Conference, Munich, FRG, April 22-24, 1991

[9] J. Lennon, "Give PEACE a Chance", The Plastic Ono Band - Live PEACE in Toronto, Apple Records, December, 1969

[10] B. H. Liskov, S. Zilles, "Programming with Abstract Data Types", *SIGPLAN Notices*, 9, 4, 1974

[11] B. J. Nelson, "Remote Procedure Call", Carnegie-Mellon University, Report CMU-CS-81-119, 1982

[12] D. L. Parnas, "Designing Software for Ease of Extension and Contraction", *IEEE Transaction on Software Engineering*, Vol. SE-5, No. 2, pp. 128-137, 1979

[13] J. L. Peterson, A. Silberschatz, "Operating System Concepts (Second Edition)", *Addison-Wesley Publishing Company*, ISBN 0-201-06079-5, 1985

[14] M. Sander, H. Schmidt, W. Schröder-Preikschat, "Naming in the PEACE Distributed Operating System", International Workshop on "Communication Networks and Distributed Operating Systems within the Space Environment", pp. 293-302, ESTEC, Noordwijk, The Netherlands, October 24-26, 1989

[15] H. Schmidt, "Making PEACE a Dynamic Alterable System", 2nd European Distributed Memory Conference, Munich, FRG, April 22-24, 1991

[16] W. Schröder, "The Distributed PEACE Operating System and its Suitability for MIMD Message Passing Architectures", CONPAR 88, Manchester, UK, Cambridge University Press, pp. 27-34, 1989

[17] W. Schröder-Preikschat, "Overcoming the Startup Time Problem in Distributed Memory Architectures", 24th Hawaii International Conference on System Sciences, Vol. 1, pp. 551-559, Kauai, Hawaii, USA, January 8-11, 1991

[18] A. S. Tanenbaum, R. van Renesse, "Distributed Operating Systems", *ACM Computing Surveys*, Vol. 17, No. 4 (December), 1985