Windows NT im industriellen Einsatz

Egmont Woitzel · Frank Golatowski · Dirk Timmermann

Universität Rostock
Fachbereich Elektrotechnik und Informationstechnik
Institut für Angewandte Mikroelektronik und Datentechnik
Richard-Wagner-Straße 31, 18119 Rostock-Warnemünde
Telefon (0381) 4983528
{gol, dtim, woi}@baltic.e-technik.uni-rostock.de

Abstract. Windows NT wird von vielen Seiten als zukünftige Plattform für industrielle Anwendungen favorisiert. In diesem Beitrag werden Ergebnisse verschiedener Untersuchungen und Entwicklungen vorgestellt, die eine realistische Einschätzung seiner Möglichkeiten und Grenzen in diesem Bereich erlauben. Die Untersuchungen gliedern sich in die Evaluierung seiner Echtzeiteigenschaften, der möglichen Übernahme POSIX-konformer Software sowie der Möglichkeiten zur Integration dedizierter Prozeßkopplungseinheiten. Es wird gezeigt, daß Windows NT auf allen drei Gebieten über Potenzen verfügt, die es für die Übernahme von Standardaufgaben qualifizieren.

Einführung

Obwohl das Betriebssystem Windows NT vorrangig für den Einsatz in Büroanwendungen konzipiert wurde, erobert es sich einen stetig wachsenden Marktanteil im Bereich klassischer Automatisierungsanwendungen. Wesentlich für diesen Erfolg war dabei die gegenüber den Vorgängersystemen verbesserte Stabilität des Systems sowie die integrierten Sicherheitsmechanismen. Neben der grafischen Bedienoberfläche, die das System für Bedien- und Beobachtungsaufgaben qualifiziert, erlaubt Windows NT vor allem einen unkomplizierten Übergang von der Prozeßleittechnik zu kommerziell orientierten Systemen der Warenwirtschaft und Auftragsverwaltung beziehungsweise Projektierungs- und Programmiersystemen. Damit wird die Schaffung durchgängiger Softwarelösungen greifbar, mit deren Hilfe die Engineering-Kosten für Automationsprojekte halbiert werden sollen. Vor allem aufgrund des wachsenden Drucks aus dieser Richtung werden aber immer häufiger nicht nur nur reine Visualisierungsaufgaben, sondern auch klassische Steuerungsfunktionen auf der Basis von Windows NT realisiert.

In einem solchen Szenario stellt sich die Frage, ob beziehungsweise in welchem Umfang Windows NT klassische Echtzeitbetriebssysteme ersetzen kann. Dazu ist sowohl die Eignung von Windows NT zur Erfüllung sogenannter harter Echtzeitforderungen zu untersuchen als auch seine Integrationsfähigkeit im Rahmen einer verteilten Gesamtlösung. Während dies für die Anbindung an die höheren Ebenen von Automationssystemen unproblematisch erscheint, bleibt dies in Bezug auf die Übernahme und Entwicklung portabler Echtzeitsoftware auf Basis offener Standards sowie die Ankopplung dedizierter prozeßnaher Gerätesysteme genauer zu untersuchen.

1. Evaluierung des Echtzeitverhaltens von Windows NT

Anwendungen im Bereich der Prozeßautomatisierung unterscheiden sich von typischen Büroanwendungen in erster Linie dadurch, daß die Korrektheit des Systemverhaltens nicht nur von den logischen bzw. numerischen Ergebnissen abhängig ist, sondern auch von dem Zeitpunkt, an dem diese Ergebnisse verfügbar sind. Zur Sicherung des stabilen Betriebs des Gesamtsystems muß das Zeitverhalten des steuernden Prozeßrechners deterministischer Natur sein. Darunter wird in erster Linie die Unterschreitung maximaler Antwortzeiten durch das System verstanden, seltener die Bereitstellung der Antwortdaten zu einem exakt bestimmten Zeitpunkt. Ist dabei eine Überschreitung der Antwortzeit

in keinem Fall zulässig, spricht man von harter Echtzeit. Wird dagegen eine begrenzte Überschreitung der Antwortzeiten toleriert, spricht man von weicher Echtzeit.

Die durch ein Softwaresystem erzielbaren Leistungsdaten hängen dabei stark von denen des als Plattform verwendeten Betriebssystems ab, das die kritischen Funktionen zur Prozeßverwaltung und -synchronisation bereitstellt. Erst auf dieser Basis wird beweisbar, ob ein Prozeßautomatisierungssystem in seiner Gesamtheit über ein deterministisches Gesamtverhalten verfügt.

1.1. Applikationsorientierte Benchmarks zur Evaluierung des Echtzeitverhaltens

Die gegenwärtig zur Leistungsbewertung von Echtzeitbetriebssystemen eingesetzten Methoden lassen sich in die folgenden Kategorien einordnen:

- feinkörnige Benchmarks
- applikationsbasierte und applikationsorientierte Benchmarks
- Simulation

Mit feinkörnigen Benchmarks werden systemnahe Kennzahlen wie zum Beispiel die zur Prozeßumschaltung erforderliche Zeit ermittelt. Die Angabe derartiger Zeitwerte ermöglicht zwar eine grobe Einschätzung der Leistungsfähigkeit eines Betriebssystems, das Verhalten unter bestimmten Lastsituationen kann damit jedoch nicht zufriedenstellend geklärt werden. Eine rein softwarebasierte Messung gestaltet sich darüber hinaus wegen der kurzen Zeiten als schwierig. Auch bei hardwarebasierten Messungen bleibt die Versuchsplanung für die Bestimmung der hier relevanten maximalen Werte sehr schwierig, da auf die jeweilige Implementationstechnik des zu untersuchenden Betriebssystems Rücksicht genommen werden muß.

Applikationsbasierte Benchmarks basieren auf einer Standardanwendung, die auf unterschiedlichen Systemen ausgeführt wird. Mit dieser Evaluierungsmethode werden neben dem reinen Laufzeitverhalten dieser Anwendung auch solche wichtigen Gesichtspunkte wie Portierbarkeit, Wartbarkeit deutlich. Applikationsorientierte Benchmarks verzichten auf diese Aussage. Sie ordnen Anwendungen entsprechend ihrer Zeit- und Leistungsforderungen in unterschiedliche Klassen ein. Die durch die jeweilige Anwendungsklasse generierte Systembelastung wirden synthetisch auf dem Echtzeitsystem ausgeführt. Ein Beispiel für einen solchen Benchmark ist der Hartstone-Uniprocessor-Benchmark [WEI92].

Die Simulation unterscheidet sich von den vorherigen Methoden im wesentlichen darin, daß Simulationssystem und Zielsystem unterschiedlich sind. Das Simulationssystem muß das Zielsystem durch ein geeignetes Modell nachbilden. Auf diese Verfahrensweise muß dann zurückgegriffen werden, wenn zum Zeitpunkt des Systementwurfs das reale Zielsystem noch nicht verfügbar ist.

Für die Evaluierung von Echtzeitsystemen bieten sich in erster Linie synthetische anwendungsorientierte Benchmarks an, da sie sowohl das Verhalten des zu untersuchenden Systems unter Lastbedingungen erlauben als auch hinreichend portabel implementiert werden können.

1.2. Ergebnisse

Die Bestimmung feinkörniger systemnaher Leistungsparameter ist wichtig, aber in einem Echtzeitsystem sollten Einflüsse des Gesamtsystems erfaßt werden. Die zur Evaluierung der Echtzeiteigenschaften von Windows NT angewandte Methode basiert auf dem Modell des Hartstone Uniprocessor Benchmarks. Dieses charakterisiert das Anforderungsprofil eines Echtzeitsystem durch einen Satz von periodischen, aperiodischen und Synchronisations-Prozessen. Der Benchmark besteht aus fünf unterschiedlichen Testserien, zu denen mehrere Experimente gehören. Eine detaillierte Beschreibung der Experimente und der als Basis für seine Definition verwendeten Scheduling-Theorie kann der Literatur entnommen werden. Ausgangspunkt einer Testserie ist jeweils ein Basisprozeßsatz, der durch die Anzahl der Prozesse, die Priorität der Prozesse, die Ankunftsrate und Verteilung der auslösenden Ereignisse, die generierte Arbeitslast, die Ausführungszeit sowie die Endtermine charakterisiert ist.

Im Hartstone-Modell haben periodische Prozesse harte Endtermine, die durch die Periode der Prozesse bestimmt sind. Aperiodische Prozesse können harte oder weiche Endtermine besitzen, wobei erstere als sporadische Prozesse bezeichnet werden. Einen Überblick über die einzelnen Testserien und deren Prozesse gibt Tabelle 1. Innerhalb der Testserien wird jeweils ein bestimmter Parameter soweit erhöht, bis das zu testende System die gesetzten Endtermine

überschreitet und damit den harten Echtzeitanforderungen nicht mehr genügt. Als Parameter dienen entweder die Perioden der Prozesse, die von den Prozessen zu verrichtenden Arbeitslasten oder die Anzahl der beteiligten Prozesse. Die maximale Auslastungsrate U_{max}, bis zu der alle zeitlichen Forderungen eingehalten werden, ist ein Leistungsmaß des unterlagerten Echtzeitsystems.

Testserie	Basisprozeßsatz
PH-Serie 1-4	5 unabhängige periodische Prozesses mit harmonischen Frequenzen und harten Endterminen
PN-Serie 1-4	5 unabhängige periodische Prozesse mit nichtharmonische Frequenzen und harten Endterminen.
AH-Serie 1-6	5 unabhängige periodische Prozesse mit harmonischen Frequenzen 1 unabhängiger sporadischer Prozeß mit hartem Endtermin 1 unabhängiger aperiodischer Prozeß mit weichem Endtermin
SH-Serie 1-5	5 unabhängige periodische Prozesse mit harmonischen Frequenzen 1 Synchronisationsprozeß, mit der sich jeder periodische Prozeß jeweils einmal je Periode synchronisieren muß
SA-Serie 1-4	5 unabhängige periodische Prozesse mit harmonischen Frequenzen 1 unabhängiger sporadischer Prozeß mit hartem Endtermin 1 unabhängiger aperiodischer Prozeß mit weichem Endtermin Synchronisationsprozeß der SH- Serie

Tabelle 1: Testserien des Hartstone Uniprocessor Benchmarks.

Die Ausführung ausgewählter Testserien erfolgte mit dem im eigenen Haus entwickelten Tool EVASCAN. Dieses stellt ein C-basiertes Framework für die automatisierte Ausführung synthetischer Applikationen mit beliebigen Basisprozesßsätzen dar. Zur Erzielung einer hohen zeitlichen Auflösung sowie der unter C nicht standardisierten zeitzyklischen Prozeßaktivierung enthält das Framework einen vergleichsweise einfachen systemspezifischen Gerätetreiber, der eine hohe Portabilität des Tools zwischen verschiedenen Betriebssystemplattformen garantiert, ohne dabei jedoch auf die Ausnutzung systempezifischer Besonderheiten verzichten zu müssen. Das Tool EVASCAN besitzt folgende Charakteristika, die es von verfügbaren, ADA-basierten Implementationen unterscheiden:

- Implementierung *aller* Testserien in C und damit einfache Portierbarkeit auf alle verbreiteten Echtzeitbetriebssysteme.
- durch den Gerätetreiber kann eine höhere Auflösung erfolgen, als diese durch den Systemzeitgeber möglich wäre. Nachteilig ist der damit verbundene Overhead, der aber experimentell ermittelt und entsprechend berücksichtigt werden kann.
- Prozeßsätze können variabel mit Hilfe eines zum Programmpaket gehörenden Testfileeditor definiert werden. Die Ausführung der Experimente kann von einer Kommandozeile oder von einer Workbench gesteuert werden.
- Möglichkeit der Untersuchung beliebiger Prozeßsätzen durch eine Scheduling-Analyse
- Graphische Anzeige der Prozeßabläufe

Ergebnis der Ausführung der einzelnen Hartstone-Serien ist die Ermittlung der maximalen Auslastungsgrenze U_{max}, bis zu der alle Endtermine eingehalten werden. Es erfolgt eine Aufzeichnung von Meßdaten über diesen Punkt hinaus, so daß das Gesamtverhalten des Systems unter Überlastbedingungen beobachten werden. Implementationen des Tools liegen für die Betriebssysteme SORIX, LynxOS und Windows NT vor. Eine genauere Beschreibung der Implementation ist [GOL95] und [GOL96] zu entnehmen.

Zur Beurteilung der Echtzeiteigenschaften von Windows NT soll hier die einfachste Hartstone-Testserie herangezogen werden, wobei ein direkter Vergleich mit LynxOS erfolgt. Dabei sollen hier weniger ein Vergleich der erreichbaren Absolutzeiten im Vordergrund stehen, sondern vielmehr das Eintreten von Forderungsverlusten in Abhängigkeit von der Systemauslastung. Anhand des unterschiedlichen Verhaltens von Windows NT (Abbildung 1) und LynxOS (Abbildung 2) läßt sich sehr schnell die mangelhafte Eignung von Windows NT für die Erfüllung harter Echtzeitforderungen erkennen.

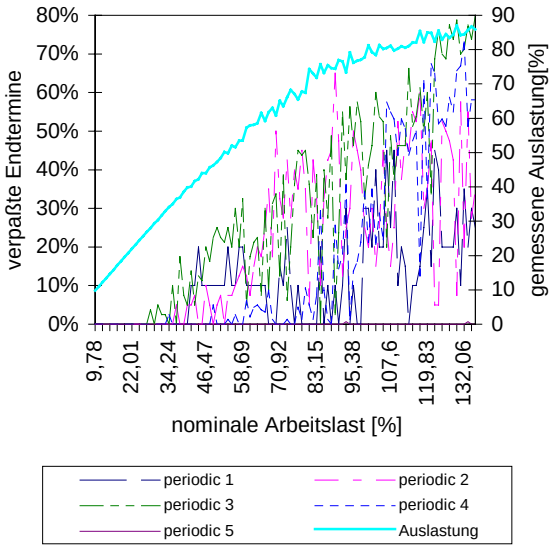

Abbildung 1: Ergebnisse des Versuchs PH-3 unter Windows NT

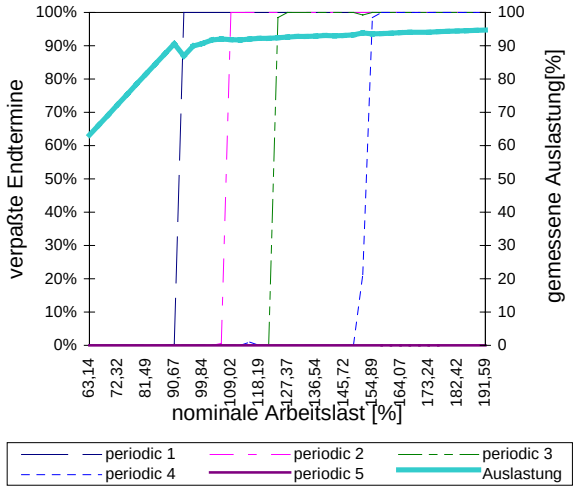

Abbildung 2: Ergebnisse des Versuchs PH-3 unter LynxOS

Während LynxOS erst ab einer relativ hohen Gesamtauslastung beginnt, in umgekehrter Reihenfolge der Priorität Forderungen zu verlieren, sezten unter Windows NT bereits bei einer geringen Systembelastung Forderungenverluste unterschiedlicher Prioritätsklassen auf. Von besonderem Interesse ist jedoch der Echtzeit-Prozeß Periodic5. Das ist der höchstpriore Echtzeit-Prozeß aus dem Prozeßsatz, dessen Last innerhalb der Testserie erhöht wird. Obwohl alle anderen Prozesse (Periodic1-4) bereits bei einer relativ geringen Auslastung ihre harten Zeitbedingungen nicht einhalten können, verpaßt dieser Prozeß auch unter Windows NT keinen Endtermin. Damit verhält sich der höchstpriore Windows-NT-Echtzeitprozeß deterministisch.

Es läßt sich daher feststellen, daß Windows NT erwartungsgemäß nur bedingt für die Erfüllung von Echtzeitaufgaben geeignet ist. Wird nur für einen Prozeß die Einhaltung harter Echtzeitbedingungen gefordert, kann diesem die höchste verfügbare Priorität zugeordnet werden. Sind nur relativ wenige harte Echtzeitforderungen zu berücksichtigen, kann überlegt werden, diese gemeinsam innerhalb des höchstprioren Rechenprozesses zu behandeln. Eine weitere

Alternative stellt für Forderungenströme mit einer Periode oberhalb von etwa 50ms eine starke Überdimensionierung des Prozeßrechnersystems in Bezug auf die Rechenleistung dar, so daß die durch Echtzeitprozesse generierte Systemlast unterhalb von etwa 20% bleibt. Dies ist unter Berücksichtigung der durch moderne Prozessoren zur Verfügung gestellten Rechenleistung eine durchaus ökonomisch sinnvolle Variante. Komplexe Anwendungen mit harten Echtzeitforderungen und hohen Ankunftsraten werden jedoch Domäne der klassischen Echtzeitbetriebssysteme bleiben.

2. Unterstützung offener Programmierschnittstellen

Moderne Echtzeitbetriebssysteme folgen weitestgehend den Standardisierungsbemühungen der POSIX-Gruppe, auf der Basis eines offenen Betriebssysteminterfaces die Portabilität von Anwendungssoftware zwischen unterschiedlichen Plattformen zu gewährleisten. Unter dem Namen POSIX wird dabei eine Familie von Standards zusammengefaßt, die verschiedene Aspekte des Programmierinterfaces von Betriebssystemen behandeln. Von besonderem Interesse sind an dieser Stelle die Standards IEEE 1003.1c und IEEE 1003.4, die Multithreading und Echtzeitverarbeitung standardisieren.

Eine Unterstützung dieser offenen Standards ist für den Einsatz von Windows NT aus zwei grundsätzlichen Erwägungen wünschenswert. Einerseits wäre der Portierungsaufwand für die Übernahme bereits vorhandener Echtzeitsoftware auf Windows NT basierende Systeme bei gleichzeitigem Investitionsschutz für Neuentwicklungen minimal. Andererseits könnten relativ preiswerte PC-Systeme als Entwicklungsplattform für eingebettete Lösungen auf Industrierechnerbasis eingesetzt werden.

2.1. Windows NT Subsysteme

Windows NT besitzt eine modulare Microkernelarchitektur. Über einem relativ kleinen Kernel, der alle notwendigen Basisdienste zur Speicher- und Prozeßverwaltung sowie das I/O-Management bereitstellt, werden alle anderen Services bereits im Rahmen des durch den Kernel vorgegebenen Prozeßmodells implementiert. Dies erlaubt die parallele Bereitstellung unterschiedlicher Programmierschnittstellen (API) für Anwendungsprogramme. Diese werden in der Windows NT Terminologie als Subsysteme bezeichnet. Subsysteme kommunizieren untereinander ausschließlich über Kernel Services, benutzen jedoch dieselben Gerätetreiber. Das Win32-Subsystem nimmt dabei eine Sonderstellung ein, da es für alle anderen Subsystem die Verwaltung der graphischen Ausgaben übernimmt und

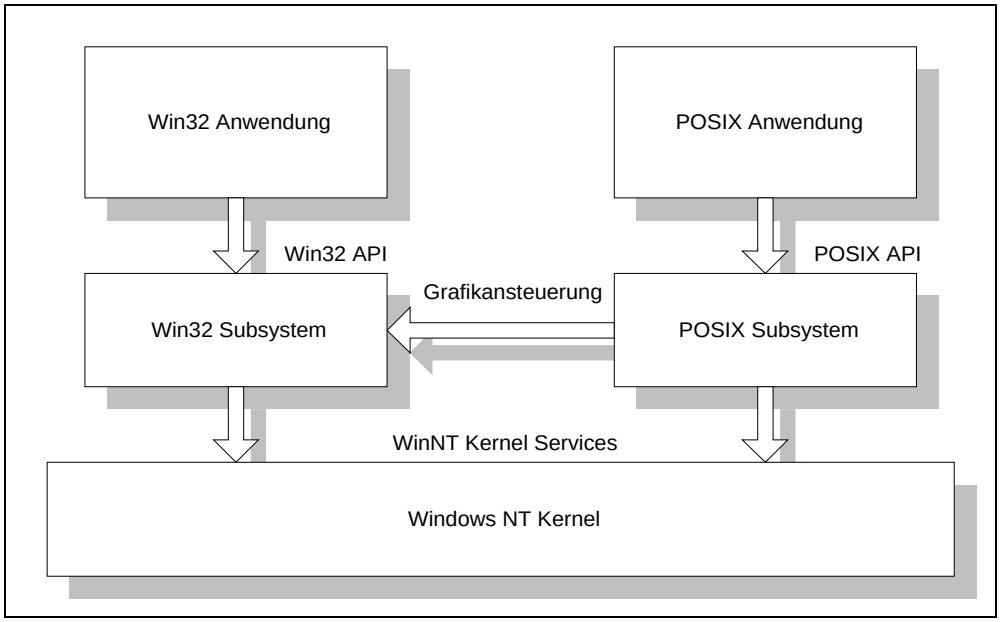

Abbildung 3: Win32- und POSIX-Anwendungen unter Windows NT

koordiniert. Im Lieferumfang von Windows NT ist standardmäßig neben dem Win32-Subsystem ein POSIX 1003.1a-konformes Subsystem enthalten. Abbildung 3 zeigt die prinzipielle Kommunikationsstruktur zwischen Win32- und POSIX-Anwendungen unter Windows NT.

2.2. POSIX-konforme Echtzeitbibliothek

Das POSIX-Subsystem weist einige Einschränkungen auf, die es als Plattform für Echtzeitanwendungen disqualifiziert. Einerseits unterstützt das implementierte POSIX-Subset weder Multithreading noch die Echtzeiterweiterungen. Zudem stehen jedoch auch keine zusätzlichen Dienste zur Verfügung, die einen Zugang zu den Kernel-Services gestatten würden. Das POSIX-Subsystem ist damit nicht erweiterungsfähig. Eine weitere Einschränkung besteht in der ausschließlichen Unterstützung zeichenbasierter Anwendungen. Grafische Anwendungen sind nur innerhalb des Win32-Subsystems realisierbar, das über standardisierte Mittel der Interprozeßkommunikation wie z. B. named pipes erreicht werden kann. Zudem wird das POSIX-Subsystem kaum durch die Programmentwicklungswerkzeuge unterstützt. Die für die Programmentwicklung notwendigen Header-Files sind z. B. nur im Resource-Kit und nicht im Lieferumfang von Visual C++ enthalten.

Für die Implementierung einer POSIX 1003.1c/1003.4 konformen Echtzeitbibliothek erscheinen daher nur zwei Strategien sinnvoll zu sein. Einerseits könnte diese Bibliothek in Form eines neuen Subsystems implementiert werden. Dagegen sprechen jedoch mehrere Faktoren, insbesondere der hohe Implementationsaufwand zusammen mit der vergleichsweise schlechten Dokumentation des für die Implementation eines Subsystems verwendbaren Programmier-interfaces. Zudem wäre eine derartige Implementation gegenüber Versionswechseln des Betriebssystems vermutlich sehr anfällig. Demgegenüber erfordert die Implementierung in Form eines Aufsatzes auf das Win32-Subsystem viel weniger Aufwand. Gleichzeitig wird eine einfache Benutzung der grafischen Oberfläche möglich. Die Struktur einer derartigen Lösung zeigt Abbildung 4.

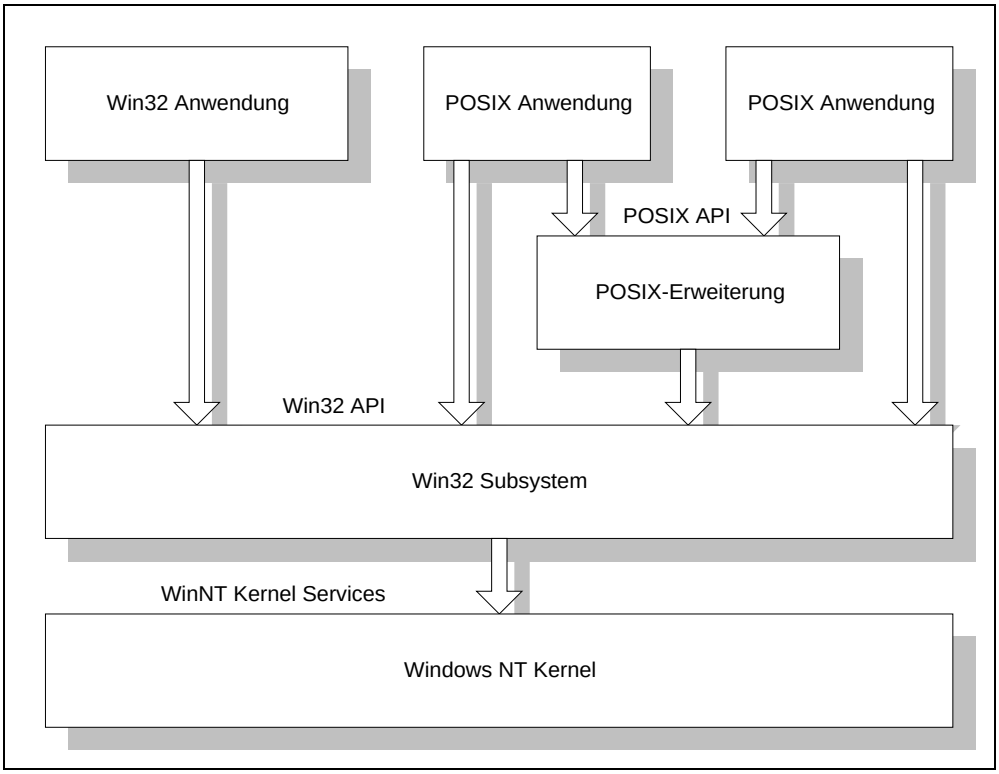

Abbildung 4: POSIX-Erweiterung für das Win32-Subsystem

Anhand einer Prototypimplementierung einer solchen Betriebssystem-Erweiterung [KLU95] konnte nachgewiesen werden, daß die unter dem Win32-Subsystem zur Verfügung stehenden Dienste des Windows NT Kernels zur Implementation POSIX-konformer Services zur Koordinierung konkurrenter und kooperierender Rechenprozesse genügen. Die größten Probleme bereitete dabei die Abbildung des POSIX-Signalkonzepts auf entsprechende

Mechanismen unter Windows NT. Eine einfache Abbildung auf Messages scheidet dabei wegen der ungenügenden und nicht deterministischen Reaktionszeiten aus. Ein sinnvoller Kompromiß wurde in der für den Benutzer transparenten Anmeldung eines Verwaltungsthreads für jeden POSIX-Prozeß gefunden, dem die maximale Priorität zugeteilt wird. Dieser Thread wird nur bei der An- und Abmeldung von Prozessen sowie der Übermittlung von Signalen aktiv und verschlechtert dadurch das Systemverhalten nicht wesentlich. Für das insgesamt erzielbare Zeitverhalten gelten natürlich die obenstehenden Aussagen.

Die Funktionsfähigkeit der als DLL implementierten POSIX-Erweiterung wurde anhand einer Musteranwendung nachgewiesen, die sowohl zeitgebergesteuerte Echtzeitthreads als auch normal priore Threads mit Visualisierungsfunktion kombinierte. Die Echtzeitthreads waren ohne Änderungen am Sourcecode auch unter LynxOS übersetzbar und lauffähig.

3. Verteilte Anwendugen

Die Entwicklung großer Softwaresysteme erfordert eine strenge Modularisierung, da nur so die Wartbarkeit des Systems zu gewährleisten. Der Entwurf der internen Schnittstellen eines Systems wird durch eine Verteilung der einzelnen Rechenprozesse auf mehrere physikalisch getrennte Rechnersysteme zusätzlich kritisch, da nicht mehr nur rein logische Aspekte zu beachten sind, sondern auch der zusätzlich entstehende Kommunikationsaufwand berücksichtigt werden muß. Im Bereich der Büroanwendungen setzen sich dafür Komponentenarchitekturen durch, die als Weiterentwicklung der klassischen Objekttechnologie angesehen werden können.

Für einen Einsatz im Prozeßbereich ist zu untersuchen, inwieweit diese Verfahren geeignet sind, einen zu automatisierenden externen Prozeß abzubilden. Weiterhin ist zu überprüfen, auf welche Weise ressourcenkritische Prozeßkopplungseinheiten in eine solche Architektur einbezogen werden können, ohne selbst ein vollständiges Komponenteninterface implementieren zu müssen.

3.1. Komponentenorientierte Programmierung

Die komponentenorientierte Programmierung stellt eine konsequente Anwendung objektorientierter Ansätze auf den Entwurf und die Implementierung größerer Anwendungssysteme dar. Diese sind durch einen hohen Komplexitätsgrad und eine relativ lange Lebensdauer gekennzeichnet. In einem solchen Umfeld kann die Stabilität eines Systems in der Wartungsphase nur gewährleistet werden, wenn der Austausch von Software-Teilsystemen seiteneffektfrei bleibt. Gerade hier werden Schwachpunkte des klassisch objektorientierten Ansatzes deutlich. Der Einsatz von Vererbung erhöht zwar einerseits die Programmiereffektivität durch Wiederverwendung beträchtlich, jedoch wirken Änderungen im Quellcode von Basisklassen unter Umständen systemweit. In großen Klassenhierarchien sind die Wirkungen derartiger Änderungen auf das Gesamtsystem nur noch schwer abschätzbar.

Ein weiteres Problem stellt die programmiersprachliche Ausrichtung der klassischen Objekttechnologie dar. Objektorientierung wird nur auf Ebene des Quellcodes unterstützt, nicht jedoch zwischen ausführbaren Programmen unter Umsänden unterschiedlicher Hersteller. Diese Makro-Modularisierung ist jedoch für die Realisierung größerer und verteilter Systeme notwendig.

In diese Lücke springen die Interfacetechnologien für Softwarekomponenten, wobei gegenwärtig vor allem die Komponenten-Integrationsmodelle CORBA (Common Object Request Broker Architecture) der Object Management Group und DCOM (Distributed Component Object Model) von Microsoft miteinander konkurrieren. Beide Objektmodelle verfolgen weitestgehend ähnliche Ansätze. Sie erlauben Anwendungen, innerhalb eines verteilten Systems Objekte einer gewünschten Klasse zu finden bzw. zu erzeugen und Dienstleistungen dieses Objekts zu benutzen. Gruppen zusammengehöriger Dienstleistungsfunktionen werden als Interfaces zusammengefaßt. Interfaces erhalten einen global eindeutigen Identifikator. Nach ihrer Freigabe darf die Interfacespezifikation nicht mehr modifiziert werden. Wird eine Weiterentwicklung notwendig, muß ein neues Interface spezifiziert werden. Um innerhalb eines komplexen Systems Interoperabilität zu gewährleisten, können durch ein Objekt mehrere Interfaces, unterstützt werden, so auch solche mit älterenen Interfacespezifikationen.

Das durch DCOM verwendete Objektmodell definiert Interfaces auf binärer Basis. Die Benutzung von Interfaces verursacht im allgemeinen nur geringe Kosten. Je nach Entfernung zwischen Server und Client wird für den Aufruf eines Dienstes entweder ein indirekter Funktionsaufruf (in-process server), ein lokaler Funktionsaufruf (LPC, local server) oder ein entfernter Funktionsaufruf (RPC, remote server) erforderlich, wobei unter Umständen zusätzlich der Transport der Funktionsargumente (marshalling) erfolgen muß.

Auf das Objektmodell setzt das eigentliche Komponenten-Integrationsmodell auf, wobei für CORBA in erster Linie OpenDoc, für DCOM in jedem Fall OLE verwendet wird. Die Integrationsmodelle definieren im wesentlichen Interfaces zur Lösung von Standardaufgaben wie der Objekteinbettung in Dokumente etc. sowie für das Objektmanagement. DCOM und OLE sind im Fall von Windows NT integraler Bestandteil des Betriebssystems, eine Benutzung dieser Technologie für die Kommunikation innerhalb einer Automationsanwendung wäre daher sinnvoll.

3.2. Komponentenorientierte Integration dedizierter Einheiten

Eine einfache Möglichkeit zur Abbildung von Prozeßkopplungseinheiten auf OLE-Objekte stellen ActriveX-Controls dar, da sie sowohl die Steuerung der jeweiligen Einheit durch den Aufruf von Methoden, die Abfrage (Polling) von Zustandsvariablen über Eigenschaften (properties) als auch die asynchrone Benachrichtigung des Clients über Zuständsänderungen (events) unterstützen. Zudem werden Controls durch eine Vielzahl visueller Programmierwerkzeuge wie Visual Basic oder Delphi unterstützt. Eine positive Evaluierung dieses Ansatzes erfolgte im Rahmen einer Modellimplementation zur Einbindung einer Microcontroller-basierten Steuereinheit [WOI96], [RAC96].

Ein gewisser Overhead entsteht dabei allerdings durch die in dem Control zu implementierende grafische Repräsentation. Dies ist in größeren Systemen nicht akzeptabel, da hier die Visualisierung keine einfache Abbildung der Rohdaten des Prozesses darstellt. Daher wurde von einem Konsortium mehrerer großer Automationshersteller eine Interface-Spezifikation für Objekte erarbeitet, die Prozeßgrößen repräsentieren [OPC96]. Diese unter dem Namen "OLE for Process Control" verbreitete Spezifikation wird mittlerweile von einer Vielzahl namhafter Herstellern unterstützt. Dabei geht die Spezifikation grundsätzlich davon aus, daß das Serverobjekt die Kommunikation mit der Peripherie vollständig kapselt, so daß die Clientanwendungen von den verwendeten herstellerspezifischen Kommunikationsprotokollen keine Kenntnis besitzen müssen. In dieser Standardisierung der Datenschnittsellen liegt ein enormes Einsparungspotential für Ingenieurleistungen in der Projektierungs- und Inbetriebnahmephase von Automationssystemen

4. Zusammenfassung

Es wurde gezeigt, daß Windows NT in weiten Grenzen für einen Einsatz in industriellen Anwendungen geeignet ist. Besondere Vorteile bietet es in Bezug auf seine Integrationsfähigkeit zu Anwendungen der kommerziellen Datenverarbeitung im Sinne von CIM-Lösungen sowie bei der Erfüllung von Visualiserungsaufgaben. Darüber hinaus ist Windows NT auch zur Lösung weniger kritischer Echtzeitaufgaben geeignet, für die der Einsatz dedizierter Echtzeitsysteme nicht gerechtfertigt ist. Die durch das Win32-API zur Verfügung gestellten Verfahren zur Koordinierung von Prozessen erlauben die plattformübergreifende Programmierung auf Basis offener Standards, so daß sowohl standardkonforme Programme auf Windows NT portiert werden können, als auch Windows NT als kostengünstige Entwicklungsplattform für offene Echtzeitsysteme eingesetzt werden kann. Schließlich bietet die Integration des DCOM-Objektmodells in Windows NT eine hervorragende Basis für die Schaffung modularer Anwendungssysteme. Gravierende Mängel weist Windows NT dagegen als Betriebssystemplattform für komplexe Anwendungen mit harten Echtzeitforderungen auf. In diesem Bereich werden klassische Echtzeitbetriebssysteme ihren festen Platz behalten. Die Unterstützung der DCOM-Spezifikation wäre eine wünschenswerte Ergänzung für diese Betriebssysteme, die ihre Integration in eine Gesamtlösung vereinfachen würde.

Literatur

[GOL95] Golatowski, F.; Bösel, H.; Paukert, A.:
Implementation eines applikationsorientierten Benchmarks für Echtzeit-UNIX-Betriebessysteme.
Echtzeit '95, Karlsruhe 1995, Konferenzunterlagen S. 62-70

[GOL96] Golatowski, F.; Timmermann, D.:
An evaluation and simulation technique for real-time operating systems.
Embedded computing conference, Paris 1.-2. 10. 1996, Proceedings

[KLU95] Klube, J.:
Möglichkeiten zur POSIX-konformen Echtzeitprogrammierung unter Windows NT.
Diplomarbeit an der Universität Rostock, FB ET, Institut MD, 1995

[OPC96] OPC Taskforce:
OLE for Process Control Standard Version 1.0.
Stand 29. 8. 1996

[RAC96] Rachui, T.:
Remote-Control-Interface unter Verwendung von OLE.
Diplomarbeit an der Universität Rostock, FB ET, Institut MD, 1996

[WEI92] Weidermann, N. H.; Kamenoff, N. I.:
Hartstone Uniprocessor Benchmark: Definitions and Experiments for real-time systems.
Real-Time Systems Journal, Vol. 4, Nr. 4, S. 353-383, Kluwer Academic Publishers, 1992

[WOI96] Woitzel, E.:
Interfacing Remote Software Components using Dynamic Token Threaded Code.
Embedded Microprocessor Systems, IOS Press, Amsterdam, Oxford, Tokyo, Washington 1996