Dynamisches Binden in Multics

Daniel Götz
Friedrich-Alexander-Universität Erlangen
Martensstraße 1
Erlangen, Deutschland
daniel.dg.goetz@fau.de

ABSTRACT

Das Betriebssystem Multics wurde Mitte der 1960er Jahre entwickelt, um viele der Probleme im Zeitalter der aufkommenden Computernutzung zu lösen. Der Fokus lag vor allem darauf, ein System zu entwickeln, das vielen Nutzern gleichzeitig eine zuverlässige und schnelle Abarbeitung ihrer Programme bieten konnte. Die Struktur und Gestaltung war aber auch dafür ausgelegt, das Programmieren selbst zu erleichtern.
Das dynamische Binden war eine Kernkomponente von Multics, die es in dieser Form vorher in keinem anderen Betriebssystem gab. Die vielen Vorteile von dynamischem Binden, beispielsweise das Schonen von Ressourcen und die Möglichkeit verschiedene Versionen einer Bibliothek zu nutzen, sollten dafür sorgen, dass dieses Konzept bis heute in den meistgenutzten Betriebssystemen überdauert hat.
In dieser Seminararbeit werden die Voraussetzungen hardware- und softwaretechnischer Natur sowie der Mechanismus des dynamischen Bindeprozesses in Multics erklärt. Anschließend wird unter verschiedenen Gesichtspunkten evaluiert, warum das dynamische Binden für Multics so wertvoll war, aber auch Nachteile aufgezeigt.

CCS Concepts

•**Operating systems → Dynamic binding;**

Keywords

AKSS; Dynamisches Binden; Multics

1. EINLEITUNG

Das Betriebssystem Multics (**Mult**iplexed **I**nformation and **C**omputing **S**ervice) wurde vor der – für die Informatik gigantischen – Zeitspanne von über 50 Jahren entwickelt, das letzte kommerziell betriebene System schon 2000 ausgemustert. Trotzdem gibt es viele der damals entwickelten Konzepte noch in heutigen Betriebssystemen, allen voran in Unix[7] als geistigem Nachfolger, aber auch in proprietären Systemen wie Windows[8]. Eine dieser wegweisenden Funktionen, die bis heute intensiv verwendet wird, ist das dynamische Binden (*dynamic binding, lazy binding*) von Programmcode.
Um die Ansätze und Vorgehensweise der Multics-Entwickler um den MIT Projektleiter Fernando Corbató im Hinblick auf das dynamische Binden zu verstehen, muss man sich vergegenwärtigen, dass die Nutzung von Computern oft lange Wartezeiten erforderte und die Programmierung unter einem Mangel an Hardware und Softwareplattformen litt. Zum einen war es damals noch nicht üblich, dass Personen ihren eigenen Rechner besitzen – auch Multics war ein Großrechnerbetriebssystem. Daraus resultiert natürlich ebenfalls, dass der Rechner von vielen Personen gleichzeitig benutzt werden würde. Zum anderen war die Verbreitung von Funktionalität in Form von Programmen oder Bibliotheken nicht so einfach wie heute im globalisierten Internetzeitalter. Wollte man in seinem Programm eine Wurzel berechnen, hatte man die Möglichkeit sich selbst ein Programm dafür zu schreiben oder falls es vorhandenen Code gab, diesen statisch ins eigene Programm zu binden, d.h. effektiv eine Kopie des Bibliothekscodes in den selbstgeschriebenen vorzunehmen. Nimmt man diese Aspekte nun zusammen, dann folgt daraus die Konsequenz, dass geschriebene Programme unter Umständen deutlich größer, komplexer und damit auch fehleranfälliger waren als sie sein mussten. Daraus resultierte natürlich auch durch Eingabe und Verarbeitung eines Programms ein höherer Bedarf an teurer Rechenzeit.
Multics sollte diese Probleme allerdings zumindest teilweise lösen – durch dynamisches Binden. Im Gegensatz zum statischen Binden war es hier nicht mehr nötig, den Code in sein Programm hineinzukopieren. Die mehrfache Nutzung von Code über mehrere Programme ermöglichte es also, die Größe von Programmen sinken zu lassen. Dies stellte nicht nur eine Erleichterung der Komplexität dar, sondern auch der monetären Situation in Anbetracht der damaligen Preise für Hauptspeicher[1]. Weiterhin konnten theoretisch alle Benutzer diese Bibliotheken (und den Code anderer, falls gestattet) in ihren Programmen referenzieren, was die geteilte Nutzung erleichterte[3]. Theoretisch konnte man sogar zur Laufzeit Bibliotheken ersetzen, um sie durch neuere oder fehlerfreie Versionen zu ersetzen[1, 13].
Umgesetzt wurde dieses Konzept nun mithilfe einer Kombination von Hardware (initial der GE 645, teilweise wohl auch schon auf der GE 635 möglich) und Software, genauer gesagt dem sogenannten dynamischen Binder. Im folgenden Kapitel wird das Zusammenspiel dieser beiden Komponenten nun genau erläutert.

2. KONZEPT DES DYNAMISCHEN BINDENS IN MULTICS

Im folgenden Kapitel werden die Mechanismen des dynamischen Bindens in Multics genauer beleuchtet und erläutert. Angefangen mit den technischen Voraussetzungen, werden in Unterkapiteln zuerst das benötigte Befehlsformat so-

[1]Noch 1973 kosteten 12 Kilobyte Speicher etwa 4.700$.
Quelle: http://www.jcmit.com/memoryprice.htm

wie die Technik der Adressbildung für die sogenannte "generalisierte Adresse" erklärt. Daran schließt sich ein Unterkapitel über den eigentlichen Bindevorgang mithilfe von ITS-Pointern an.

2.1 Aufbau des Speichers

Um die Funktionalität des dynamischen Bindens verstehen zu können, ist es notwendig, das Speichermodell von Multics (wenigstens auf Segment-Ebene) zu kennen. Die für dieses Seminar-Papier wichtigsten Eigenschaften werden nun kurz erklärt.

Seitennummerierte Segmentierung.

Multics bedient sich zur Strukturierung des Speichers (und auch zur Prozess- und Privilegienisolation) der Segmentierung. Multics-Prozesse sind damit gekennzeichnet dadurch, dass sie aus einer variablen Anzahl von Segmenten bestehen. Pro Prozess (und damit pro Adressraum) kann es maximal 2^{14} Segmente geben, die jeweils aus maximal 2^{18} 36-bit Wörtern bestehen können. Grund für diesen für damalige Verhältnisse großen Adressraum ist die Vermeidung von Namenskonflikten[4]. Die einzelnen Segmente werden innerhalb des Prozesses charakterisiert durch ihre `Segmentnummer`. Außerhalb durch ihren `symbolischen Namen`. Desweiteren gibt es unterschiedliche Segmenttypen: Datensegmente können beispielsweise nicht als Instruktionen gelesen werden, während Prozedursegmente nicht beschrieben werden dürfen (Schreibschutz ist aber natürlich auch für Datensegmente möglich). Prozedursegmente werden als *pure procedures* bezeichnet, das heißt sie werden durch Ausführung nicht verändert. Der Grund dafür ist, dass geteilter Code nicht von den nutzenden Prozeduren geändert werden soll.
Innerhalb der Segmente existiert desweiteren noch eine Seitennummerierung (*Paging*). Seitengrößen können zwischen verschiedenen Segmenten variieren, wobei entweder 64 oder 1024 Wörter pro Seite möglich sind[5]. Für das dynamische Binden spielt die Seitengröße und Seitennummerierung an sich allerdings keine Rolle[10].

Hierarchisches Dateisystem.

Das Dateisystem von Multics entspricht einer hierarchischen Baumstruktur. Die Wurzel ist hierbei ein Spezialtyp einer Datei, der `Directory` oder `Verzeichnis` genannt wird. Dort sind die Pfade zu anderen Dateien, die ebenfalls Verzeichnisse seien können, enthalten. Innerhalb des Verzeichnisses besitzen alle Dateien (und auch die enthaltenen Segmente) mindestens einen symbolischen Namen, der diese eindeutig (auch über alle Prozesse) identifiziert. Dieser kann verwendet werden, um Referenzen eines Benutzers oder Benutzerprozesses aufzulösen, indem in internen Datenstrukturen nach dem symbolischen Namen gesucht wird[5]. Dateien können so ebenfalls geöffnet werden, um deren Segmente in den Speicher einzublenden.

2.2 Adressbildung

Ein Multics-Konzept, welches heute noch aktuell ist, ist die Verwendung einer Abstraktion zur Adressierung von Wörtern. Es wird nicht auf "blankem Speicher" adressiert, sondern stattdessen eine logisch zweistufige bzw. im Befehlsformat zweiteilige "generalisierte Adresse" verwendet. Der erste Teil dieser 18-Bit Adresse ist die Segmentnummer, die das Segment im Prozess eindeutig identifiziert. Der zweite Teil enthält eine Wortnummer, die als ein Versatz (*Offset*) innerhalb des ausgewählten Segments fungiert[5].

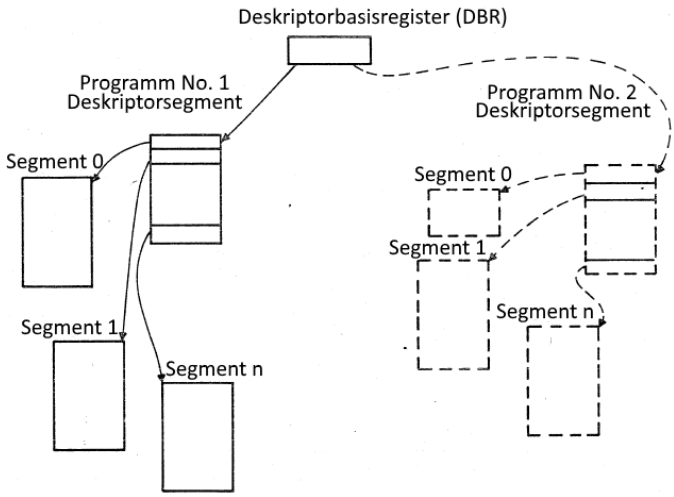

Abbildung 1: Verzeigerung des Deskriptorsegments[5]. Das Deskriptorbasisregister zeigt auf das Deskriptorsegment des gerade aktiven Prozesses, welches die physikalische Speicheradresse der Prozesssegmente enthält. Quelle: GE 645 Manual S.23

Hierbei ist zu beachten, dass sowohl Segment- als auch Wortnummer ortsunabhängige Daten sind, d.h. sie enthalten keine Speicheradressen[5, 12]. Um die richtige Speicheradresse für den Zugriff auszuwählen, ist eine weitere Datenstruktur, das `Deskriptorsegment`, notwendig. Wie in Abbildung 1 zu sehen ist, wird die Segmentnummer als Index im prozesseigenen Deskriptorsegment benutzt. Das Deskriptorsegment enthält wiederum einen Deskriptor für jedes Segment, das der Prozess "kennt" (sh. Kapitel 2.4.3 Auflösung der Verknüpfung und Fehlerbehandlung). Dieser Deskriptor enthält neben der Information über den Ort des Segments auch die Zugriffsrechte darauf. Da das Deskriptorsegment allerdings auch nur ein ansonsten gewöhnliches Segment innerhalb des Prozesses ist, fehlt noch eine Verzeigerung darauf – ein Henne-Ei-Problem, welches durch ein eigens dediziertes Register, das Deskriptorbasisregister (*descriptor base register*, kurz DBR), gelöst wird. Das DBR enthält also die physikalische Speicheradresse des Deskriptorsegments für den aktiven Prozess.

Beim Beispiel des einfachen Abrufens einer Instruktion aus dem aktuell aktiven Prozedursegment reicht es also aus, die Segmentnummer der Prozedur (enthalten im Prozedurbasisregister) als Index im Deskriptorsegment zu nutzen und dann zur im Deskriptor enthaltenen Adresse des Segments den PC als Wortnummer zu konkatenieren.
Zur Nutzung des dynamischen Bindens bei Datenbefehlen oder dem Aufruf einer Subroutine hingegen ist es ebenfalls möglich ein anderes Segment auszuwählen als das im Prozedurbasisregister spezifizierte. Diese Methode wurde "indirect to pair address modifier" genannt. Dazu musste der Prozessor über 4 weitere Doppelregister, die sogenannten `Adressbasisregister` (kurz ABR), verfügen. Es gab die folgenden Register:

- Argumentenliste (*argument pointer/base*)

- Allgemeine Basis (*base pointer/base*)
- Bindungssegment (*linkage pointer/base*)
- Stapelsegment (*stack pointer/base*)

Ob diese Register genutzt wurden, bestimmte sich anhand des sogenannten `Segmentetiketts` (*segment tag*), welches im Befehl mitcodiert wurde (sh. Abschnitt 2.3). Die Verwendung als Einzelregister oder Doppelregister war abhängig von einem Tag im Register.

2.3 Befehlsformat

Das Multics Betriebssystem verwendete 36-Bit Befehle. Wie in Abbildung 2 zu sehen ist, gliederten sich diese Befehle im Wesentlichen in 5 Teile.

Der erste Teil war für die Auswahl eines Adressbasisregisters gedacht.

Der zweite Teil des Befehls enthielt eine 15-Bit Adresse für Argumentreferenzen.

Im dritten Teil befand sich die eigentliche Instruktion bzw. der *opcode*.

Danach folgte die "UIB"-Sektion, welche aus einem ungenutzten Bit, einem Unterbrechungsvermeidungsflag und dem für dynamischen Referenzen elementaren *Basisregisterflag* bestand.

Die letzten 6 Bit des Befehls enthielten das `Befehlsetikett` (*Tag*). Das Befehlsetikett lässt sich noch einmal untergliedern in einen 2 Bit *Tag Modifier* und einen 4 Bit *Tag Descriptor*[5]. Hier konnte der für das dynamische Binden nötige Modus des indirekten Zugriffs gesetzt werden.

Ein Befehl, der dynamisches Binden auslösen bzw. es nutzen könnte, müsste also ein Zugriff über ein Adressbasisregister sein und damit folgende Voraussetzungen erfüllen[12]:

1. Im BR-Feld muss das passende Basisregister/ Basisregisterpaar ausgewählt werden
2. Die Basisregisterflag B muss gesetzt sein
3. In den Tag-Feldern muss der Modus für einen indiziert-indirekten Zugriff spezifiziert sein

An dieser Stelle beginnt die eigentliche Arbeit des dynamischen Binders.

2.4 Dynamisches Binden

Mittels des passenden Befehls ist es nun möglich, einen indirekten Speicherzugriff an eine (noch) unbekannte Speicherstelle zu tätigen. Dabei wird der dynamische Binder von dem oben erwähnten Dateisystemkonzept und dem Mechanismus der Adressbildung in Multics unterstützt. In den folgenden Unterkapiteln wird ausführlich erklärt, wie der dynamische Binder indirekte Zeiger zur Verbindung von Daten- oder Prozedur-Segmenten unterschiedlicher Prozesse nutzt. Angefangen wird dabei mit der Erklärung des Aufbaus dieser indirekten Zeiger, gefolgt von der Verwahrung und Verwaltung derselbigen. Anschließend wird der eigentliche Prozess des dynamischen Bindens erläutert und anhand eines Beispiels visualisiert.

2.4.1 Indirekte Zeiger

Ein indirekter Zeiger (auch genannt Speicherindirektdoppelwort) kann grob beschrieben werden als ein Zeiger auf ein anderes Segment, der aus 2 Speicherworten (72-Bit) besteht. Wie im Unterkapitel zur Adressbildung bereits erwähnt wird, benötigt man zur Identifikation einer Speicherstelle in Multics (aus Sicht des Prozesses) eine Segmentnummer sowie eine Wortnummer. Analog dazu, verfügt der für das dynamische Binden relevante ITS-Zeiger (engl. *Indirect to Segment-Pointer*, kurz ITS-Zeiger) wie in Abbildung 3 zu sehen ist, über ein Segmentnummernfeld in Wort 0 und ein Wortnummern- bzw. Adressfeld in Wort 1. Das Befehlsetikett (die letzten 6 Bit) des Wortes 0 ist hierbei entscheidend für den Verlauf des Bindevorgangs. Es kann entweder die Magic Bytes für ein `ITS` (oktal interpretierte 43) im Falle einer etablierten (*snapped Link*) oder ein `FT2` (oktal interpretierte 41) im Falle einer noch nicht etablierten Verbindung (*unsnapped Link*) zu einem Segment eines anderen Prozesses enthalten.

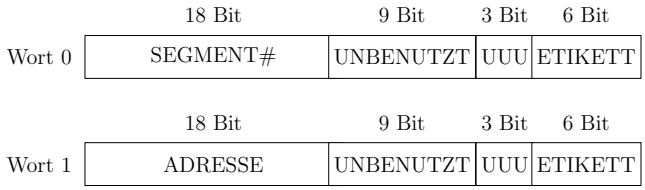

Abbildung 3: Aufbau eines ITS-Zeigers. Wort 0 enthält die Segmentnummer als Index im Deskriptorsegment und ein Etikett (im Bild ITS), das den Etablierungsstatus bestimmt. Wort 1 enthält eine Wortadresse und ein Etikett für den Modus.

Zusätzlich zu den ITS-Zeigern gab es noch die deutlich weniger benutzen ITB-Zeiger, die keine Segmentnummer sondern eine Indirektion über eines der Basisregister spezifizierten (sh. Abschnitt 2.4.6).

Ebenfalls abhängig von der Bindung des prozessfremden Segments war der Inhalt des Doppelworts. Wurde die Verknüpfung bereits etabliert (also ein "ITS" im Etikett), befindet sich tatsächlich eine Segment-/Wortnummer im Doppelwort. Vor dem Verknüpfungsvorgang allerdings steht dort nur der *symbolische* Name des Segments im Verzeichnis (sh. hierarchisches Dateisystem). Der genaue Etablierungsvorgang wird im Kapitel Abschnitt 2.4.3 beschrieben.

2.4.2 Bindungssegment

Da ITS-Zeiger, wie oben erwähnt, Segmentnummern enthalten, ergibt sich daraus die logische Konsequenz, dass ihr Kontext nur für den jeweils aktiven Prozess gilt. Segmentnummern sind also außerhalb nicht nur nicht eindeutig, sondern auch im unteren Zahlenbereich relativ sicher mehrfach belegt[4]. Dies hängt damit zusammen, dass benutzte Segmente für einen Prozess in Multics einfach strikt durchnummeriert werden. Segmente, die später dynamisch hinzugebunden werden (d.h. deren Referenzen aufgelöst werden), bekommen jeweils die nächste freie Segmentnummer im Deskriptorsegment[5]. Grafisch dargestellt ist dies in Abbildung 4. Als logische Konsequenz daraus besitzt jeder Prozess sein eigenes Bindungssegment, welches alle ITS-Zeiger (und damit deren prozesslokale Segmentnummern) enthält. Im Gegensatz zum Bindevorgang zur Laufzeit, wird also bei der Kompilierung (und damit zur Kompilierzeit) einer Objektdatei für Multics bereits eine prototypische Bindungssektion im Bindungssegment, die Einträge für alle im Kernpro-

3 Bit	15 Bit	9 Bit	1Bit	1Bit	1Bit	6 Bit
BR	ADRESSE	OPCODE	U	I	B	ETIKETT

Befehlsformat

Abbildung 2: Befehlsformat von Multics[5]. Entscheidend für die Verwendung des dynamischen Binders sind hier das B-Feld zur Nutzung eines im BR-Feld ausgewählten ABR und das Befehlsetikett am Schluss zur Auswahl des Modus'.

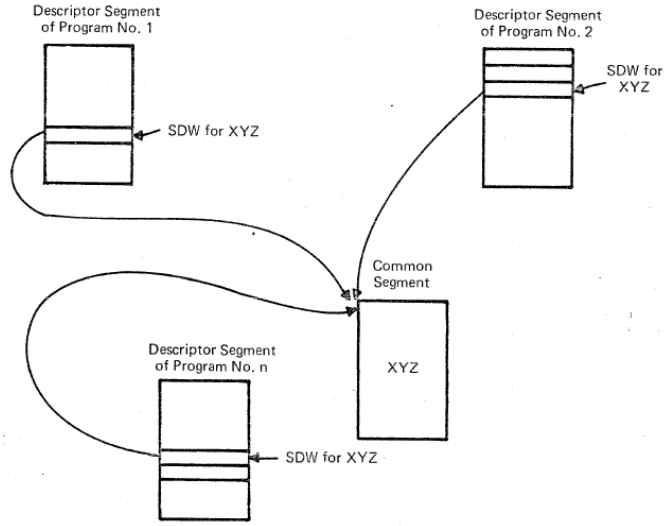

Abbildung 4: Mehrere Programme benutzen ein geteiltes, dynamisch gebundenes Segment. Da das Segment zur Laufzeit an die erste freie Stelle im Deskriptorsegment eines Programms eingetragen wird, besitzt es unterschiedliche Segmentnummern in den Programmen. Quelle: GE-645 Manual S.29

gramm (ohne externe Subroutinen) enthaltenen Verknüpfungen besitzt[2] erstellt.

2.4.3 Bindungsprozess

Nachdem nun die Datenstrukturen und Hardware als Basis des dynamischen Binders erklärt wurden, wird im folgenden der eigentliche Bindevorgang (in Multics bekannt als *snappen*) vorgestellt. Es liegt in der Natur eines dynamischen Binders, dass der eigentliche Bindevorgang im Allgemeinen nur einmal beim ersten Auflösen einer Referenz stattfindet.
Das Ergebnis eines erfolgreichen Bindevorgangs ist ein eine etablierte (*snapped*) Verknüpfung bzw. ITS-Zeiger. Die Vorgänge zur Etablierung eines *unsnapped* ITS-Zeigers können grob in 4 Schritte unterteilt werden:

1. Identifizierung und Ausführung eines Befehls mit indirekter Adressierung
2. Auswahl der geeigneten Verknüpfung im Bindungssegment
3. Versuch der Auflösung des unsnapped Links löst einen Bindefehler (*First-reference-trap*) aus
4. Etablierung der Verknüpfung in der Fehlerbehandlung

Diese Schritte werden in den folgenden Unterkapiteln genauer erläutert.

Identifizierung und Ausführung eines Befehls mit indirekter Adressierung.

Wie im Unterkapitel zum Befehlsformat bereits erklärt, sind hier vor allem das BR-Feld, die Basisregisterflagge und das Etikett von Wichtigkeit. Wird ein Befehl ausgeführt, der sowohl im BR-Feld das Adressbasisregister mit der Adresse des Bindungssegments auswählt als auch die Basisregisterflagge gesetzt hat und indiziert-indirekten Modus spezifiziert, wird eine passende Verknüpfung geladen.

Auswahl der geeigneten Verknüpfung im Bindungssegment.

Die Auswahl der geeigneten Verknüpfung für diesen Maschinenbefehl, der eine externe Referenz enthält, basiert auf der automatischen Erstellung des Bindungssegments und Anpassung des Codes zur Kompilierzeit. Referenziert ein Maschinenbefehl bspw. ein Wort in einem externen Datensegment, dann wird daraus eine unetablierte Verknüpfung im Bindungssegment generiert[11], die statt einer Segment- und Wortnummer Zeiger auf "ACC Strings" mit dem symbolischen Namen von Segment und Eintrittspunkt (*Entrypoint*) enthalten[2]. Ein ACC String entspricht im Wesentlichen einem gewöhnlichen ASCII-String. Der Unterschied besteht darin, dass die 7 Bit ASCII-Zeichen in jeweils 9 Bit kodiert wurden, um der 36 Bit Wortbreite zu entsprechen. Das erste Zeichen eines solchen Strings notiert die Anzahl der Folgezeichen.
Innerhalb des Prozedurcodes wird bei der Kompilierung nur die Nummer des Links im Bindungssegment kodiert. Diese Nummer muss nie geändert werden, da alle bindebedingten Änderungen bei der Etablierung der Verknüpfung im Bindungssegment bzw. im Deskriptorsegment stattfinden, die prozesslokal sind.

Auflösung der Verknüpfung und Fehlerbehandlung.

Ist die passende Verknüpfung im Bindungssegment ausgewählt, führt das FT2-Etikett in diesem unetablierten, indirekten Zeiger zu einem Bindefehler. Die Behandlung des Fehler wird vom Bindelader durchgeführt[12]. Anhand des im ITS-Zeiger enthaltenen ACC Strings, der den symbolischen Namen des adressierten Segments enthält, kann es mit Hilfe des Verzeichnisses eindeutig identifiziert werden[6]. Durch die Benutzung von sogenannten Suchregeln (*search rules*), die vom Nutzer geändert werden konnten, war es möglich, die Verzeichnisse, die nach dem Namen durchsucht wurden, sowie deren Reihenfolge zu beeinflussen. Damit konn-

ten Programmierer faktisch auch die Auswahl des letztendlich gebundenen Segments beeinflussen[13]. Möglich war dies deshalb, weil die erste Suche nach dem symbolischen Segmentnamen immer im sogenannten *Known Segment Table* (kurz KST) stattfindet, der für den Binder die generalisierte Adresse zum symbolischen Namen jedes bereits bekannten Segments enthält. Ist der Name eines zu bindenden Segments in dieser Tabelle nicht enthalten (es existiert also keine etablierte Verknüpfung dazu), wird es mithilfe der Suchregeln im Verzeichnis gesucht. Durch Spezifizierung der Suchregeln konnten also Segmente gleichen Namens gezielt gewählt werden.
Im nächsten Schritt wird das noch unbekannte, aber mittels der Verzeichnissuche gefundene Segment im KST eingetragen und muss nun noch im Adressraum des aktiven Prozesses eingeblendet werden. Dazu ist es nötig, dass das Segment eine Segmentnummer innerhalb des Prozesses erhält und den Deskriptor in das Deskriptorsegment einzutragen[5]. Zur funktionierenden Adressierung über eine generalisierte Adresse fehlt nun noch die Wortnummer zum im ITS-Zeiger per ACC String spezifizierten Eintrittspunkt. Die Wortadressen aller selbst angebotenen, geteilten Variablen (oder bei Prozeduren Eintrittspunkte bzw. *Entrypoints*) werden zur Kompilierzeit in einer für den Binder zugänglichen Definitionssektion der Objektdatei abgelegt. Dazu müssen sie explizit vom Programmierer gekennzeichnet werden[9].
Das damit im Prozessadressraum vorhandene Segment kann jetzt theoretisch über eine generalisierte Adresse benutzt werden. Dazu wird der symbolische Name, der sich im indirekten Zeiger befindet, substituiert mit der generalisierten Adresse, d.h. die Segmentnummer und die aus der Definitionssektion entnommene Wortnummer werden im ITS-Zeiger eingetragen. Um bei nachfolgenden Zugriffen keinen Bindefehler mehr zu erhalten, wird auch das Etikett des indirekten Zeigers von FT2 auf ITS geändert[2]. Zusätzlich ist die Benutzung dieser generalisierten Adresse auch deutlich performanter als die Suche nach dem symbolischen Namen im Verzeichnisbaum[3].

Beispiel.

Um das Konzept des dynamischen Bindens in Multics noch einmal verständlich zu machen, wird es hier visualisiert.
In Abbildung 5 ist ein typischer Befehl (*Instruction*) mit externer Segmentadresse, also der Benutzung eines Adressbasisregisters, zu sehen. Dieser enthält dann den Index eines ITS-Zeigers im Bindungssegment (hier mit `k` und `k+1` benannt). Die Verknüpfung ist hier noch nicht etabliert, das heißt der Zeiger enthält noch keine generalisierte Adresse und das Segment ist noch nicht im Deskriptorsegment des Prozesses eingetragen. Stattdessen befindet sich in den beiden Worten des indirekten Zeigers ein weiterer Zeiger auf die ACC Strings für den Eintrittspunkt beziehungsweise das Segment. Mithilfe der Suchregeln (es existiert noch kein Eintrag im KST) wird dann das Segment im Verzeichnisbaum gesucht und der Eintrittspunkt aus der Definitionssektion für das Segment extrahiert.
In Abbildung 6 sehen wir denselben Aufbau nach der Etablierung. Im ITS-Zeiger befindet sich nun die generalisierte Adresse des Datensegments und ein ITS-Etikett anstelle des FT2. Mittels des Deskriptorbasisregisters kann das De-

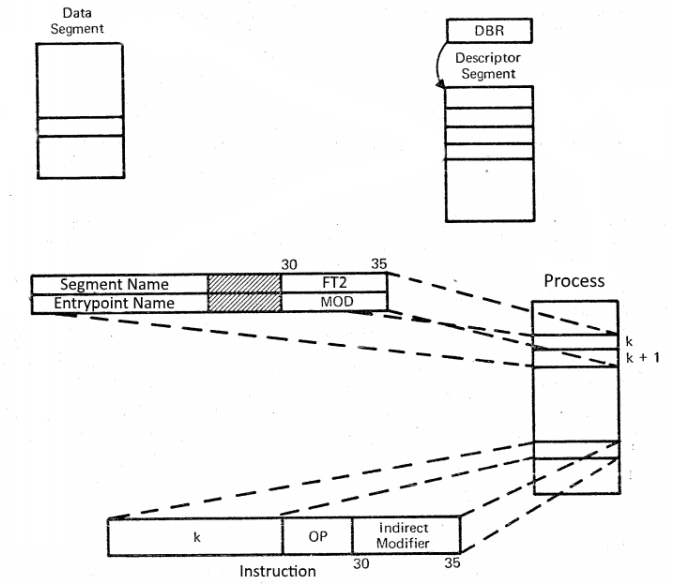

Abbildung 5: Verzeigerung vor dynamischer Bindung eines Segmentes. Der Befehl referenziert einen ITS-Zeiger, der den symbolischen Namen für Segment und Eintrittspunkt eines Datensegments enthält. Angelehnt an GE-645 Manual S.26

skriptorsegment ausfindig gemacht werden, in dem dann die Segmentnummer als Index fungiert. An dieser Stelle des Deskriptorsegments befindet sich der Deskriptor, welcher unter anderem die Speicheradresse des gewünschten Segments enthält. Mit dieser Adresse wird dann die Wortnummer konkateniert und der Operand geholt.

Allerdings ist es nicht nur möglich Datensegmente zu referenzieren, auch Prozedursegmente können über ITS-Zeiger eingebunden werden. Die wesentlichsten Unterschiede zwischen Daten- und Prozedursegmenten beim dynamischen Binden sind die Verwendung von Eintrittspunkten und die Möglichkeit von multiplen Indirektionen.
Prinzipiell funktioniert der Bindungsprozess über Eintrittspunkte analog zum Bindungsprozess für Datensegmente[6, 9]. Statt dem symbolischen Namen eines Datums wird bei der Assemblierung des Codes ein ACC String für den zu benutzenden Entrypoint der Prozedur hinzugefügt, der dann beim Binden über die Definitionssektion der Bibliothek aufgelöst werden kann.

2.4.4 *Multiple Indirektionen*

Außer der Spezifizierung von Eintrittspunkten statt Variablen ist der Hauptunterschied beim Binden von Prozedursegmenten die Möglichkeit von multiplen Indirektionen. Multics gestattete es, Subroutinen theoretisch unbegrenzt weitere Subroutinen aufzurufen. Für die nahtlose Integration von Subroutinen, die der dynamische Binder bieten sollte, war das unerlässlich. Da die Voraussetzung für den Aufruf von jeglichen Sub- oder Subsubroutinen in anderen Segmenten die Vorhandenheit eines indirekten Zeigers im Bindungssegment war, musste dieses auch für die Subroutine vorhanden sein. Im Kapitel zum Bindungssegment wurde bereits

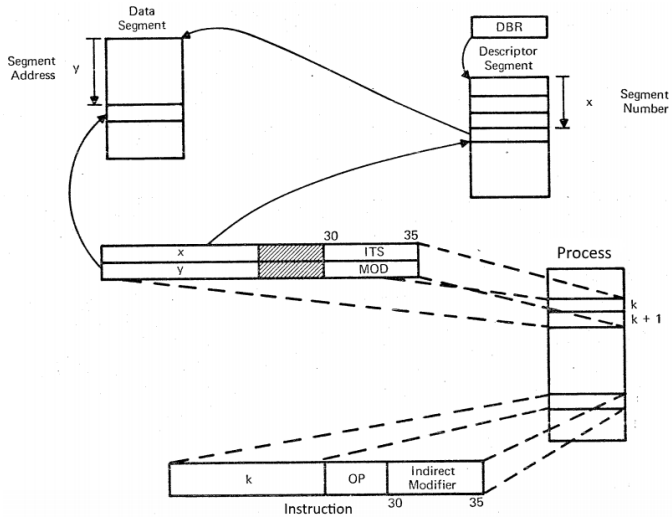

Abbildung 6: Verzeigerung nach dynamischer Bindung eines Segmentes. Der ITS-Zeiger enthält eine Segmentnummer und einen Versatz in das Segment, d.h. eine generalisierte Adresse. Das Datensegment wurde ins Deskriptorsegment eingebunden. Quelle: GE-645 Manual S.26

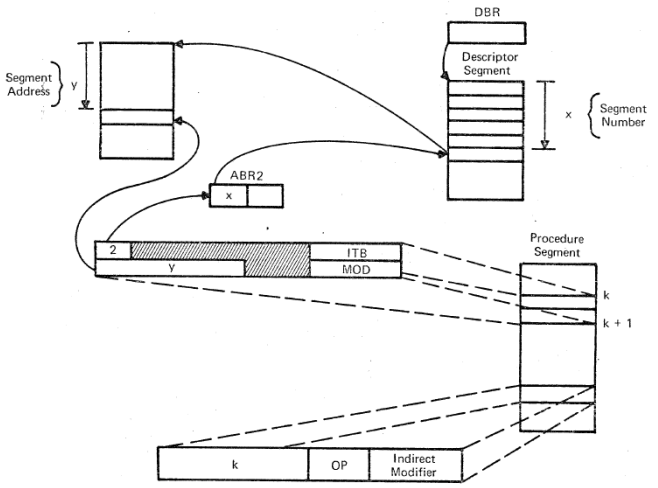

Abbildung 7: Funktionsweise eines ITB-Zeigers. Ein Befehl referenziert einen ITB-Zeiger, der die Nummer eines ABRs enthält. Der Inhalt des ABRs wird als Index im Deskriptorsegment des Programms genutzt. An die Segmentadresse im Deskriptor wird dann die Wortnummer im 2. Wort des ITB-Zeigers konkateniert. Quelle: GE-645 Manual S.27

erwähnt, dass dieses prozessspezifisch ist, was den Zwang zur Folge hatte, auch das Bindungssegment für Subroutinen prozessspezifisch zu halten. In der Folge wurde mit dem Aufruf einer Subroutine (und deren Subroutinen) deren prototypische `Bindungssektion` in das Bindungssegment des aufrufenden Prozesses kopiert[2, 11, 13].
In der Konsequenz hatte dieselbe allgemein geteilte Prozedur `n` verschiedene Bindungssegmente in `n` verschiedenen Prozessen.

2.4.5 Neubinden

Variablen beziehungsweise Code hatten über die Verknüpfung prinzipiell dieselbe Lebensdauer wie ein Prozess. Das heißt, eine Variable, die extern durch Benutzung von einer Prozedur allokiert wurde, existiert solange im Speicher bis alle etablierten Verknüpfungen zu ihr entfernt wurden. Wird aber beispielsweise das Segment, in dem sich die Variable befindet, schon vorher manuell entfernt, werden die etablierten Verknüpfungen im Bindungssegment wieder zurückgesetzt zu unetablierten Verknüpfungen[13].
Damit war es möglich, alle Variablen eines Datensegments auf einmal zurückzusetzen oder aktualisierte Prozeduren neu einzubinden.

2.4.6 ITB-Zeiger

Neben den oben ausführlich beschriebenen ITS-Zeigern gab es auch die weniger genutzten ITB-Zeiger (*Indirect to base*). Der Unterschied bestand im Wesentlichen darin, dass diese indirekten Zeiger keine Segmentnummer enthielten, sondern stattdessen den Index eines Adressbasisregisters. Wie in Abbildung 7 zu sehen ist, war der auslösende Befehl grundsätzlich analog zur Nutzung eines ITB-Zeigers. Dann wurde allerdings der Inhalt des spezifizierten ABR als Index für den gewünschten Deskriptor in der Tabelle des Deskriptorsegments benutzt. Als Folge des Formats fand hier auch keine Ersetzung durch eine generalisierte Adresse im indirekten Zeiger statt, d.h. kein dynamischer Bindungsprozess per se.

3. DISKUSSION

Das übergeordnete Ziel von Multics war die Schaffung eines Computersystems, das es ermöglichen sollte effizienten Mehrbenutzerbetrieb für einen "general purpose" zu gewährleisten. Der Mechanismus des dynamischen Bindens muss also auch in diesem Kontext betrachtet und bewertet werden.
Zur Realisierung des obengenannten Zieles wurde auf mehrere Arten beigetragen, die im Folgenden erwähnt werden.
Zum einen wurde die Belastung des Speichers, welche durch mehrmaliges statisches Einbinden derselben Prozeduren oder Variablen auftritt, stark verringert. In einem Mehrbenutzersystem wie Multics kann dies zu großen Einsparungen führen. Dies hat auch mit der Struktur der segmentierten Programme zu tun – der Ansatz der "reinen" Prozeduren als Kernkomponente des Bindens ermöglichte die Einsparungen erst. Zusätzlich ergab sich durch das dynamische Binden ein mehrfacher Nutzen im Hinblick auf die Komplexität der Programmierung unter Multics. Erstens konnten Funktionalitäten einfach aus Bibliotheken eingebunden werden, da das Binden automatisch geschah und nur der symbolische Name des einzubindenden Segments bekannt sein musste. Zweitens war der Ansatz zur Gestaltung (über das Binden) von Nutzerprogrammen und Systemprogrammen in Multics gleich[3]. Die Konsequenz dieser Punkte war, dass redundanten Programmen effektiv vorgebeugt wurde und die Einstiegshürde zur Systemprogrammierung in Multics sank. Drittens musste der Nutzer weder Wissen zum Platzbedarf noch zur Anzahl eventuell folgender Subroutinen-Aufrufe haben[4]. Zum Vergleich gestattete etwa der IBM 7094 nur 2 Speicherreferenzen nacheinander[9].
Ein weiterer nicht zu vernachlässigender Vorteil war die Automatisierung des Bindens durch das System zur Laufzeit.

Durch das Binden bei Bedarf und die Möglichkeit des erneuten Bindens konnten Bibliotheken leicht durch aktuellere, neukompilierte Versionen ausgetauscht werden, was außer einem erneuten Bindungsprozess keinerlei Nachteile für eine nutzende Anwendung brachte[3]. Außerdem war es mittels der Auswahl von Suchregeln auch möglich, dass verschiedene Nutzer bzw. Prozesse automatisch verschiedene Versionen von Routinen verwendeten[1]. Ausgangspunkt dafür war, dass dynamisch zu bindende Segmente im Verzeichnis mithilfe von durch Nutzer änderbaren Regeln gesucht wurden. So konnten neben den standardmäßig vorhandenen Suchregeln, wie der Suche in der Tabelle bereits verwendeter Segmente, auch eigene Verzeichnisse (z.B. das aktuelle Arbeitsverzeichnis) angegeben werden, die zuerst durchsucht werden konnten[13].

Es gab allerdings auch kritische Stimmen zum Konzept von Multics; Janson beschrieb in einem wissenschaftlichen Papier 1975 eine andere Implementierung für den dynamischen Binder von Multics, da die Standardimplementierung ausnutzbare Sicherheitslücken gehabt hätte. Dort wurde zum einen eine Ausnutzung des Linkers beschrieben, bei der die Initialisierung des Bindungssegments eines aufgerufenen Prozesses eventuell zu einem Overflow führen könnte. Ein weitere Sicherheitslücke enthielt das Umgehen des dynamischen Binders und das Aufrufen einer Prozedur direkt über die Segmentnummer, was zur Folge hätte, dass deren Speicher nicht initialisiert würde. Dies könnte zu einem Programmabbruch führen[6]. Laut Janson war das Problem an der Implementierung, dass der Binder sich im Sicherheitskern befand und schlug eine Implementierung vor, die den Binder aufspalten und den Teilen verschiedene Rechte geben sollte. Die Variante den dynamischen Binder im Userspace unterzubringen, ist so beispielsweise auch in Linux implementiert.
Der Grundgedanke bei der Unterbringung des Binders in Ring 0 war, dass der Binder sich nicht selbst binden muss, da Ring 0 bei der Systeminitialisierung gebunden wird und Verzeichnisse bei der Nutzung der Suchregeln schneller gefunden werden konnten. Außerdem war es so möglich auch Verknüpfungen für Segmente zu binden, die zwar Aufrufe tätigen konnten, aber keine Lese- oder Ausführungsberechtigungen hatten[13].
Ein weiteres Manko war die Notwendigkeit der Vermeidung von Namenskonflikten. In einer Liste, der Referenznamenstabelle (*reference name table*), wurden die in Verknüpfungen enthaltenen symbolischen Referenznamen gesammelt. Das hatte zur Folge, dass solche Referenznamen für Segmente nicht doppelt vorkommen durften[5].
Ebenso war es problematisch, dass es für Datensegmente keine eigenen Bindungssektionen gab. Dadurch war es in Multics nicht möglich, Datenstrukturen, die über mehrere Segmente verteilt waren, zu teilen[11]. Insgesamt kann der dynamische Binder von Multics aufgrund der oben genannten Vorteile meiner Meinung nach allerdings als maßgebende Kernkomponente bei der Implementierung für den effizienten Mehrbenutzerbetrieb betrachtet werden.

4. FAZIT

Das Konzept und die Umsetzung des dynamischen Bindens in Multics konnten zur damaligen Zeit als einzigartige Innovation betrachtet werden. Dass das dynamische Binden noch heute Bestand hat, ist ein starkes Indiz für die Fortschrittlichkeit. Obwohl die Implementierung und Granularität heute teilweise relativ stark von dem abweicht, was in Multics umgesetzt wurde, liegt dies doch vor allem an den Randumständen. Zentraler Grund für den Unterschied hier ist natürlich, dass die seitennummerierte Segmentierung zur Strukturierung des Speichers in der Praxis durch die alleinige Seitennummerierung ersetzt wurde. Damit ist eine Referenzierung von Information auf feingranularer, logischer Segmentbasis heute nicht mehr so umzusetzen.
Die Grundidee des Bindens zur Laufzeit, der "reinen" Prozeduren und der Einfachheit des Teilens von Prozeduren durch Automatisierung blieben allerdings erhalten und elementar.

5. GLOSSAR

ABR Adressbasisregister. Enthält unter anderem die Segmentnummer des Bindungssegments und damit den Ort der ITS-Zeiger

ACC String String, bestehend aus normalen 7 Bit ASCII Zeichen, die allerdings jeweils in 9 Bit kodiert wurden, um die 36 Bit Wortbreite auszufüllen. Das erste Zeichen enthält die Anzahl der Folgezeichen

DBR Deskriptorbasisregister. Enthält die Adresse des Deskriptorsegments

FT2 Englisches Akronym für "*Fault Tag Two*". Mögliche Ausprägung eines ITS-Zeiger-Etiketts. Steht für eine unetablierte Verknüpfung und löst bei Benutzung einen Bindefehler aus

ITS Englisches Akronym für "*Indirect To Segment*. Mögliche Ausprägung eines ITS-Zeiger-Ettikets. Steht für eine etablierte Verknüpfung

ITS-Zeiger Speicherdoppelwort, das als Verknüpfung dient und im etablierten Status die generalisierte Adresse eines Datums oder einer Prozedur enthält

KST "Known Segment Table". Dient für den Binder zur Umsetzung des symbolischen Namens eines Segments zu dessen Segmentnummer. Der KST wird bei einem ausgelösten Bindefehler stets zuerst durchsucht und enthält Einträge für alle bereits bekannten, also gebundenen, Segmente

pure/reine Prozedur Schreibgeschützte Prozedur, die Referenzindirektion besitzt und nicht selbstmodifizierend ist

6. REFERENCES

[1] https://en.wikipedia.org/wiki/Multics.
[2] Website. http://www.multicians.org, besucht am 29.11.15.

[3] F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the multics system. In *Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I*, AFIPS '65 (Fall, part I), pages 185–196, New York, NY, USA, 1965. ACM.

[4] R. C. Daley and J. B. Dennis. Virtual memory, processes, and sharing in multics. volume 11, pages 306–312, New York, NY, USA, 1968. ACM.

[5] General Electric. GE-645 system manual, 1968.

[6] P. A. Janson. Dynamic linking and environment initialization in a multi-domain process. *SIGOPS Oper. Syst. Rev.*, 9(5):43–50, 1975.

[7] J. Levine. Linkers and loaders manuscript. Website. http://www.iecc.com/linker/linker10.html, besucht am 28.11.15.

[8] Microsoft Corporation. Microsoft developer network system services. https://msdn.microsoft.com/en-us/library/windows/desktop/ms681914(v=vs.85).aspx, besucht am 28.11.15.

[9] E. I. Organick. The multics system: An examination of its structure. 1972.

[10] J. Rabenstein. Paging in multics. 2015.

[11] J. Rosenberg and J. L. Keedy. Security and persistence. In *Proceedings of the International Workshop on Computer Architectures to Support Security and Persistence of Information*, Workshops in Computing, pages 31–47, Bremen, West Germany, 1990. Springer London.

[12] W. Schröder-Preikschat. Vorlesung betriebssystemtechnik. 2013.

[13] M. Weaver. The multics runtime environment, 1987.