1 Einleitung

Das Überwachen und Beobachten von *Software*-Systemen kann aus vielerlei Gründen sinnvoll sein. Durch das Verfolgen von Ereignissen lässt sich das Systemverhalten analysieren, das Systemverständnis verbessern, kritische Situationen im Betrieb identifizieren, die Einhaltung von vertraglich vereinbarten Rahmenbedingungen (*service level agreement*) überprüfen und Fehler beziehungsweise Fehlverhalten der *Software* diagnostizieren.

Der Paradigmenwechsel bei der Architektur von Prozessoren hin zu Mehrkernprozessoren (*multi-core processor*) verlangt von *Software* ein erhöhtes Maß an Nebenläufigkeit, um die verfügbaren parallelen Rechenressourcen effizient zu nutzen. Betriebssysteme sind ein Beispiel für *Software*, die hochgradig asynchron und damit in hohem Maße nebenläufig ist. Wird in einer solchen *Software* das Konzept des gemeinsamen Hauptspeichers (*shared memory*) zur Kommunikation zwischen einzelnen Aktivitäten verwendet, stellen Zugriffe auf diese nicht-skalaren Daten *kritische Abschnitte* dar, die geeignet geschützt werden müssen.

Für das Überwachen und Beobachten des Systems ist häufig ein Zugriff auf nicht-skalare Daten im gemeinsamen Speicher notwendig, die den Zustand des Systems beschreiben. Ohne eine Beachtung des Protokolls, das zur Synchronisation der gemeinsamen Daten verwendet wird, besteht die Gefahr, dass ein solches Überwachungswerkzeug Daten ausliest, die inkonsistent sind und keinen tatsächlichen Zustand des Systems repräsentieren. Mit dem Vorhandensein vieler paralleler Ausführungseinheiten nimmt die Gefahr von Wettlaufsituationen zwischen Betriebssystem und Beobachtungswerkzeug und damit die Gefahr des Auslesens von inkonsistenten Daten zu.

Sobald ein Werkzeug Zugriff auf Zustandsinformationen einer Anwendung ermöglicht, konkurriert das Beobachtungswerkzeug mit dem beobachteten System um den Zugriff auf die Zustandsinformationen. Werden diese Zustandsinformationen zur Laufzeit der Anwendung durch die Anwendung modifiziert, wovon in der Regel auszugehen ist, entsteht eine Wettlaufsituation zwischen Beobachtungswerkzeug und Anwendung in den kritischen Abschnitten der gemeinsam genutzten Daten. Wird die Wettlaufsituation nicht hinreichend synchronisiert, zum Beispiel durch wechselseitigen Ausschluss, kann man keinerlei Aussagen über die Güte der beobachteten Zustandsinformationen treffen, da nicht ersichtlich ist, ob während des Zugriffs des Beobachtungswerkzeugs eine Aktualisierungsoperation im Gange war und das Beobachtungswerkzeug nur einen eventuell ungültigen, inkonsistenten Zwischenwert gelesen hat.

Insbesondere dann, wenn Rückschlüsse auf die Ausführungsgeschichte gezogen werden sollen, ist es von enormer Bedeutung, dass die Daten, die das Beobachtungswerkzeug liefert konsistent sind. Ein

ausgelesenes Datum, das in einer unkoordinierten Wettlaufsituation ausgelesen wurde, darf somit *nicht* für die Analyse des Systems berücksichtigt werden. Dieses Problem verschärft sich, wenn die zu beobachtende Anwendung selbst eine nebenläufig Anwendung ist, d.h. mehrere Aktivitäten gemeinsam den Zustand des Systems verändern. Damit erhöht sich auch die Wahrscheinlichkeit der Zugriffe, die in einem möglichen Konflikt zueinander stehen. In Anbetracht der im vorherigen Abschnitt geführten Diskussion ist es wahrscheinlich, dass verstärkt mit parallelen Anwendungen zu rechnen ist, was die Brisanz des Problems erhöht.

Um diese Probleme zu vermeiden, wird in der vorliegenden Arbeit der *KStruct*-Ansatz vorgestellt, der es einem dynamischen, nicht-integrierten Beobachtungswerkzeug erlaubt, auf den Zustand eines Betriebssystems unter Berücksichtigung des Synchronisationsprotokolls zuzugreifen. Dazu wird mit Hilfe der Sprache *KStruct Access*, einer Erweiterung der Programmiersprache C, das Synchronisationsprotokoll des Betriebssystems definiert, welches als Vertrag zwischen Betriebssystem und Überwachungswerkzeug dient. Die Annotationen werden von einem Compiler in eine Komponente übersetzt, die von Beobachtungswerkzeugen für den Zugriff auf den gemeinsamen Speicher verwendet werden kann, um Daten unter Einhaltung des Protokolls auszulesen.

Um den Annotationsprozess gerade für bestehende Softwaresysteme zu unterstützen, wird zusätzlich *KStruct Advice* vorgestellt, womit sich ein Betriebssystem hinsichtlich des verwendeten Synchronisationsprotokolls analysieren lässt. Dazu wird ein formales Modell einer Sperre definiert, welches die Grundlage für eine statische Datenflussanalyse zur Berechnung der Zustände aller Sperren darstellt. Der Zustand einer Sperre wird schließlich mit Zugriffen auf Datenstrukturen im gemeinsamen Speicher korreliert und so ein Synchronisationsprotokoll abgeleitet. Die Ergebnisses des Ansatzes werden im Kontext des *Windows Research Kernels* betrachtet, diskutiert und evaluiert.

1.1 Beitrag dieser Arbeit

Der Beitrag dieser Arbeit lautet wie folgt:

1. Der Autor schlägt eine *Spracherweiterung* – KStruct Access – für die Programmiersprache C vor, welche es ermöglicht, Annahmen über das Synchronisationsverhalten des Programms explizit zu machen. Insofern kann ein Entwickler damit zum Ausdruck bringen, welche Synchronisationsmechanismen kritische Abschnitte welcher Datenstrukturen bzw. Teildatenstrukturen schützen.

2. Der Autor schlägt ein *Rahmenwerk* – KStruct – vor, auf dessen Basis die Implementierung von sowohl integrierten als auch externen Beobachtungswerkzeugen ermöglicht wird, die in Bezug auf das Synchronisationsverhalten mit dem zu beobachtenden System kooperieren. Des Weiteren werden zwei Beispielanwendung aufgezeigt, um die Praktikabilität des Ansatzes zu untermauern.

3. Der Autor schlägt ein *Analysewerkzeug* – KStruct Advice – vor, mit dessen Hilfe eine gegebene Quellcodebasis auf verwendete Synchronisationsmechanismen und deren Einsatz zur Sicherstellung von wechselseitigem Ausschluss analysiert werden kann. Damit soll erreicht werden, dass auch bestehende *Software*-Systeme in den KStruct-Ansatz integriert werden können, ohne dass von Hand eine aufwendige Anpassung vorgenommen werden muss.

Neben diesem eigentlichen Kern der Arbeit sind im Laufe des Promotionsvorhabens eine Vielzahl weiterer Arbeiten entstanden, in die Teilergebnisse der hier vorliegenden Arbeit einflossen:

- Konferenzpublikationen [Hentschel u. a. 2011; Passing u. a. 2009b; Schmidt u. a. 2008a, 2009; Schmidt und Polze 2010; Schmidt u. a. 2010; Schmidt und Schöbel 2007; Schmidt u. a. 2008b]
- Technische Berichte [Schmidt 2007, 2008]
- Betreute Masterarbeiten [Passing 2008; Sobania 2009]
- Angemeldetes Patent [Passing u. a. 2009a]

Teile der Arbeiten mit dem *Windows Research Kernel* (WRK) beschäftigen sich mit der Entwurf von Experimenten im Rahmen der Lehrveranstaltung „Betriebssysteme“ am Hasso-Plattner-Institut. Die Experimente umfassen dabei Labore zum Vermessen bestimmter Aspekte des Betriebssystemkerns sowie praktische Programmieraufgaben direkt am Kern des WRK. Ein Teil dieser Experimente wurde während zweier Praktikumsaufenthalte des Autors bei Microsoft in Redmond entworfen und implementiert. Neben einer Konferenzpublikation [Schmidt u. a. 2010] berichtete der Autor in Form von eingeladenen Vorträgen auf dem „ACM Symposium on Computer Science Education“ in Chattanooga, TN, auf dem Workshop „Cloud Futures“ 2010 in Redmond, WA, sowie dem „Windows Core Workshop“ in Peking, China, über gewonnene Erkenntnisse. Des Weiteren betreute der Autor den Übungsbetrieb der Lehrveranstaltung „Server-Betriebssysteme“ sowie die Seminare

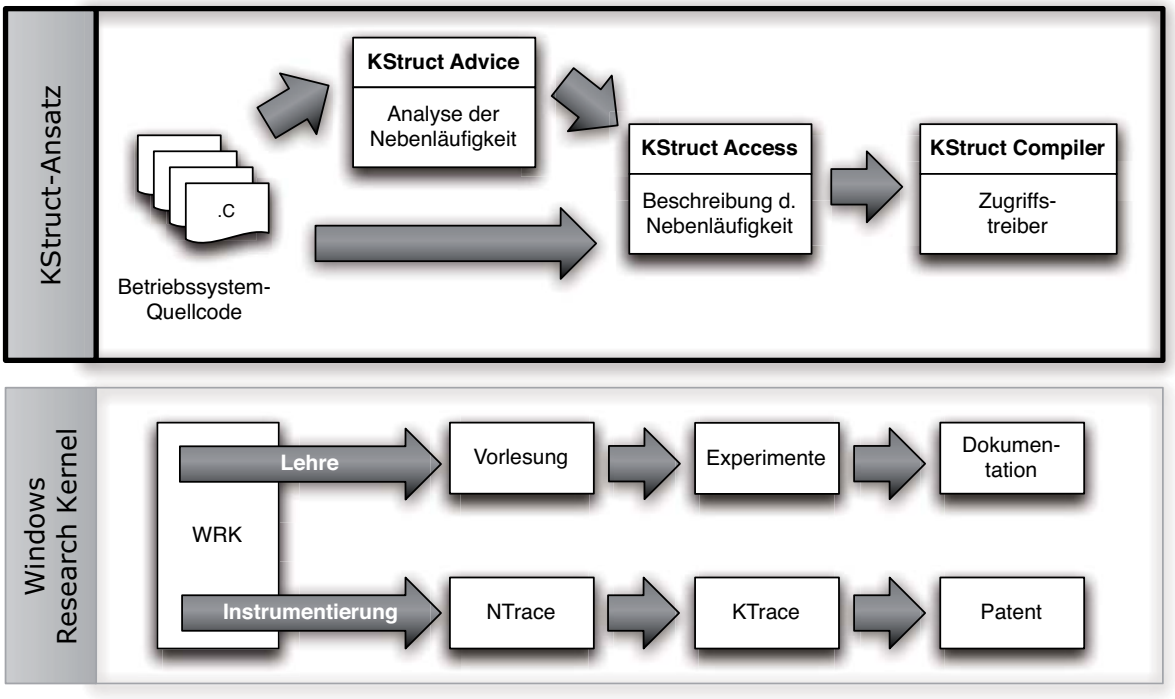

Abbildung 1.1: Struktur der Arbeit

„Quellcode-Analyse“ und „Komponenten im Einsatz“ für die Ausbildung von Master-Studenten am Hasso-Plattner-Institut.

Im Rahmen seiner Tätigkeit als Mitglied des Forschungskollegs „Service-oriented Systems Engineering“ war der Autor verantwortlich für die Organisation des Symposiums „Future Trends in Service-oriented Computing“, das im Jahr 2011 bereits zum 6. Mal ausgerichtet wird. Darüber hinaus berichtete der Autor über Teile dieser Arbeit bei Vorträgen bei SAP Labs sowie HP Labs in Palo Alto, USA, und im Rahmen gemeinsamer Workshops mit Doktoranden des Forschungskollegs an der Universität Kapstadt, Südafrika, und dem Technion in Haifa, Israel.

1.2 Struktur der Arbeit

Der Inhalt der Arbeit gliedert sich in zwei große Teile: das Hauptthema der Arbeit, der *KStruct-Ansatz*, sowie weitere entstandene Arbeiten rund um den Windows Research Kernel. Die Gliederung der beiden Teile ist schematisch in Abbildung 1.1 dargestellt.

Der KStruct-Ansatz ist der zentrale Beitrag der Arbeit. Ausgehend von einem existierenden *Software*-system wird darin untersucht, wie in diesem System (1) Nebenläufigkeit beschrieben und (2)

automatisiert analysiert werden kann. Als Beispiel für ein hochgradig nebenläufiges System wird hier stellvertretend für viele Betriebssysteme der Windows Research Kernel verwendet.

KStruct besteht aus drei einzelnen Komponenten, die in den folgenden Kapitel näher vorgestellt werden: Mit Hilfe von KStruct Access können Nebenläufigkeitsbedingungen direkt im Quellcode des Systems spezifiziert werden. Dafür wird in Kapitel 3 eine Spracherweiterung der Programmiersprache C vorgeschlagen und evaluiert.

Gerade im Umfeld von existierender *Software* ist es nicht einfach, Nebenläufigkeitsbedingungen nachträglich zu spezifizieren. Eine Umfrage während eines Praktikums bei Microsoft in Redmond unter Entwicklern des Windows-Betriebssystemkerns bestätigte diese Vermutung. Aus diesem Grund stellt der Autor *KStruct Advice* in Kapitel 5 vor, das eine gegebene Quellcodebasis mittels statischer Datenflussanalyse auf Nebenläufigkeitsbedingungen hin untersucht. Es wurde vom Autor ein Prototyp implementiert und auf den Windows Research Kernel angewendet. Das Kapitel schließt mit einer Auswertung der Analyseergebnisse.

Schließlich wir der mit KStruct Access annotierte Quellcode vom *KStruct-Compiler*, der in Kapitel 4 vorgestellt wird, in einen Zugriffstreiber übersetzt, mit dessen Hilfe Zustandsdaten eines Betriebssystems konsistent ausgelesen werden können. Darüber hinaus wird im Rahmen der weiteren entstandenen Arbeiten erörtert, wie dieser Zugriffstreiber von Werkzeugen zur Ablaufverfolgung beziehungsweise zum Überwachen verwendet werden kann.

Der zweite Teil der Arbeit widmet sich Fallstudien und Arbeiten mit dem WRK. Diese Arbeiten lassen sich in zwei Themenkomplexe aufteilen: (1) Die Verwendung des WRK in der Lehre und (2) der WRK als Basis für Instrumentierungswerkzeuge.

Der erste Teil von Kapitel 6 geht auf die Verwendung des WRK in der Betriebssystemlehre ein. Im Rahmen der Tätigkeiten des Autors wurden Inhalte für Betriebssystemvorlesungen erarbeitet und diese in praktischen Programmieraufgaben und Experimenten umgesetzt. Ein Teil der Experimente entstand dabei während eines Praktikums bei Microsoft in Redmond in Zusammenarbeit mit Architekten des Windows-Betriebssystemkerns.

Bevor die Arbeit mit einer Zusammenfassung und einem Ausblick schließt, werden im zweiten Teil von Kapitel 6 zwei Werkzeuge zur dynamischen Instrumentierung auf Basis des WRK vorgestellt. Das Werkzeug *KTrace* erlaubt die Ausführung nutzerspezifischer Skripte bei Auftreten vordefinierter Ereignisse innerhalb des Windows-Kerns. Die Skripte werden dabei mit Hilfe einer virtuellen Maschine ausgeführt, so dass ein Fehlverhalten innerhalb eines Skripts, beispielsweise durch eine Speicherzugriffsverletzung, nicht das umgebende Betriebssystem in Mitleidenschaft zieht.

Das Werkzeug *NTrace* ermöglicht die dynamische Ablaufverfolgung von Funktionsaufrufen. Dazu werden drei Ereignisse unterstützt: Funktionseintritt, Funktionsaustritt und das Auftreten einer Ausnahme. Welche Funktionen zur Laufzeit des Systems instrumentiert werden ist dem Anwender überlassen. Die Instrumentierung erfolgt durch Modifikation des Binärcodes des Systems in einem Thread-sicheren Verfahren, das zu einem Patent angemeldet wurde.

1.3 Hinweise zu Sprache und Notation

Diese Arbeit wurde in deutscher Sprache verfasst. Fachbegriffe aus dem Gebiet der Informatik werden häufig direkt aus dem Englischen entlehnt, ohne eine geeignete deutsche Übersetzung. Im Rahmen dieser Arbeit wurde versucht, diese Begriffe konsistent ins Deutsche zu übersetzen. An Stellen, bei denen eine Übersetzung als nicht sinnvoll erschien, weil kein geeignetes deutsches Äquivalent existiert, wird die englische Bezeichnung übernommen, aber zur besonderen Kennzeichnung *schräg gestellt* formatiert.

Auszüge aus Quellcode werden in einer Schriftart mit festem `Zeichenabstand` dargestellt. Besondere Konzepte, die für die Arbeit von Bedeutung sind, werden *kursiv* hervorgehoben.