2. Messung von Rechensystemen; Ansätze und Instrumente

Wir haben in Kap. 1 den Begriff Leistungsbewertung kennengelernt als Vorgang der Beurteilung der "Güte" der Bearbeitung einer Rechenlast durch ein Rechensystem; konkret verwendete Gütekriterien und zugeordnete Maße waren von den Typen Effektivität und Effizienz; einer der aufgezeigten Wege zur Feststellung von Leistung bestand aus der Beobachtung des Betriebs des zu beurteilenden Rechensystems.

Messung und Bewertung

Beobachtungen zur Feststellung quantitativer Eigenschaften heißen allgemein **Messungen**. Der in Kap. 1 schon verwendete Begriff **Messung und Bewertung** soll ausdrücken, daß wir uns vorbehalten, direkt vornehmbare Beobachtungen (also: Messungen) einer Beurteilung zuzuführen - eben Werte von (zu definierenden!) Leistungsmaßen aus Messungen abzuleiten. Der Vorgang der Messung und Bewertung wurde in Kap. 1 im Rahmen der Methodik der Objekt-Experimente eingeführt. Über diese Methodik werden wir in Kap. 3 ausführlicher zu sprechen haben. Im vorliegenden Kapitel soll unser Interesse den unabdingbaren Grundlagen der Objektexperimente gelten, und zwar werden wir

Lernziele

- in Abschn. 2.1 die während des Betriebs eines Rechensystems beobachtbaren Größen und die prinzipiellen Ansätze zu ihrer Beobachtung, nämlich sampling und eventing, kennenlernen;

- in Abschn. 2.2 die Instrumente, die für eine Beobachtung des Rechenbetriebs zur Verfügung stehen, nämlich Hardware- und Software-Monitore, in ihren Prinzipien zu verstehen versuchen.

Einschränkung

Insbesondere der letzte Punkt wird (leider) sehr schematisch zu behandeln sein. Eine konkretere Schilderung dieser Thematik wäre nur nach detailliertem Studium konkreter Hardware- und System-Software-Strukturen möglich und würde den vorliegenden Rahmen sprengen.

2.1 Meßprinzipien

Messung dynamisches System

Bei der Beobachtung des Betriebs eines RS handelt es sich offensichtlich um die Beobachtung eines Vorgangs, der eine gewisse Zeit andauert: Wir beobachten immer während eines gewissen zeitlichen **Meßintervalls** MI := $\{t;t_a \leq t \leq t_e\}$ $\mathbf{R}^+$ mit den Anfangs- und End-Zeitpunkten t_a,t_e ($t_e > t_a$), und mit der Länge / Dauer $T := t_e - t_a$. Es ist sicher vernünftig anzunehmen, daß zu willkürlichem Zeitpunkt t MI feststellbar ist, "wie" das RS zu diesem Zeitpunkt "ist", daß also der **Zustand** des RS zum Zeitpunkt t beobachtbar ist. Zu einem anderen Zeitpunkt $t' \neq t$ wird i.allg. ein anderer Zustand beobachtet werden: Ein RS ist ein **dynamisches System**, ein System, das seinen Zustand über der Zeit verändert.

RS-Zustand

Was hat man sich unter dem Zustand eines RS vorzustellen? Nun, es ist für unsere Zwecke hinreichend, dabei den Zustand / Inhalt / Wert aller speichernden Komponenten eines RS vor Augen zu haben: den Zustand aller Register, Zähler, Statusbits, den der Speicherzellen des Arbeitsspeichers, den Zustand der Bitpositionen (bzw. der Felder von Bitpositionen) auf externen Speichermedien (Platten, Bändern, ...) u.s.f. Bei dieser Vorstellung fällt Ihnen sicher unmittelbar auf, daß in unserem hierarchisch, mehrschichtig strukturierten RS die Zustandskomponenten verschiedener Schichten unterschiedlich sein können. Einerseits werden Zustandskomponenten unterer Schichten in höheren Schichten ggf. schlicht unbekannt (und daher auch prinzipiell nicht zugreifbar) sein, wie etwa der in der Firmware- / Microcode-Schicht manipulierte Zähler eines sequentiell arbeitenden Multiplikationswerkes oberhalb der Instruktionsebene nicht bekannt ist. Andererseits werden Zustandskomponenten höherer Schichten in tieferen Schichten häufig zwar im Prinzip zugreifbar, aber nicht interpretierbar sein (und daher auch nicht wirklich "existieren"), wie etwa das in der Betriebssystemschicht manipulierte Feld einer "Job Control Table" (JCT), das die Fortsetzungsadresse eines unterbrochenen Jobs enthält, unterhalb der Betriebssystemschicht zwar als Inhalt einer Speicherzelle / eines Registers sichtbar ist ("Bitmuster"), aber eben in seiner Bedeutung / Interpretation unbekannt bleibt.

Zustandsänderungen

Wir bemerkten bereits, daß ein RS als dynamisches System seinen Zustand über der Zeit ändert. Wie geschehen diese Zustandsänderungen? Nun, wenn wir bedenken, daß wir RS-Zustand mit Zustand aller speichernden Komponenten gleichgesetzt haben, und daß jede dieser Komponenten je für sich (konstruktiv so festgelegt) einen diskreten, endlichen Zustandsraum aufweist (die Werte 0 und 1 bei einem Statusbit, alle n-stelligen Bitmuster bei einer n-stelligen Speicherzelle, ...), dann ergibt sich für den (Gesamt-)RS-Zustand ebenfalls ein diskreter, endlicher Zustandsraum. Zustandsänderungen (entlang der Zeit) in einem diskreten Zustandsraum können aber nur sprunghaft, "spontan" stattfinden, so daß eine Darstellung des Zustands über der Zeit (eine sog. **Zustandstrajektorie**) prinzipiell von "treppenförmigem" Aussehen sein muß (vgl. Abb. 2.1.1).

zustandsdiskretes System

Die Treppenform der Zustandstrajektorie war oben rein definitorisch aus der Diskretheit des Zustandsraums abgeleitet worden. Ist das eine realitätskonforme Annahme? Der Gegensatz zu spontanen, zeitlosen Zustandsänderungen wären Zustandsänderungen, die in endlicher Zeit, von einem Anfangszustand aus, einen Endzustand erreichen. Dabei müßte es (wenn unsere Annahme nicht realitätskonform sein sollte) ein Kontinuum von Zwischenzuständen geben (die Zustandsdiskretheit müßte also verletzt sein). Bei einer physikalischen Betrachtung von speichernden RS-Komponenten stellen wir fest, daß es solch ein Kontinuum von Zwischenzuständen zwar u.U. durchaus gibt (ein Flip-Flop als Registerelement "während" des Umschaltens von 0 nach 1, ein Punkt einerren

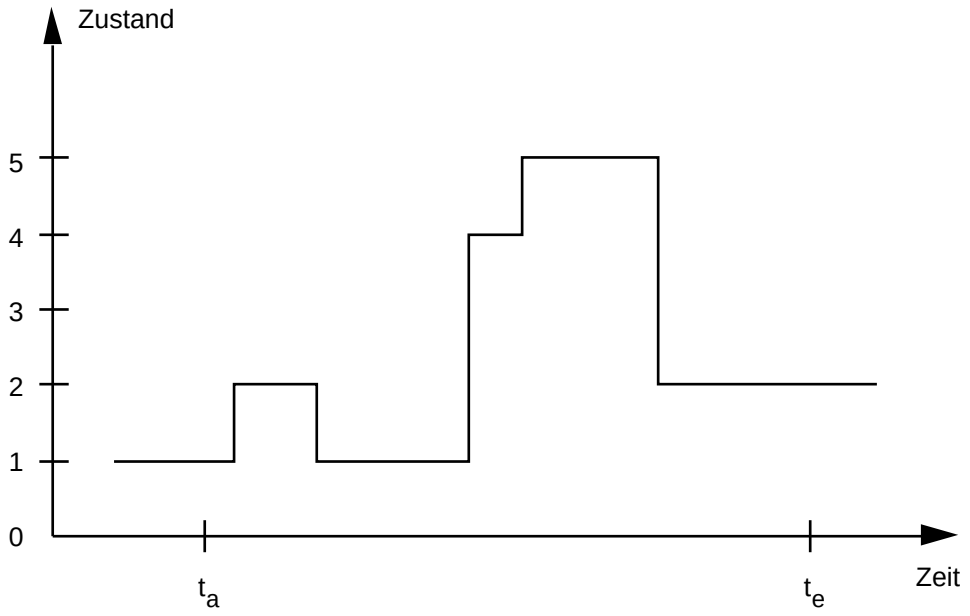

Abbildung 2.1.1: RS-Zustandstrajektorie;
Zustandsraum zur Vereinfachung: {0,1,2,...,n},
d.h. alle Zustände eindimensional "durchnumeriert".

magnetisierbaren Oberfläche "während" des Ummagnetisierens), daß aber konstruktiv alle notwendigen Vorkehrungen getroffen sind, um diese Zwischenzustände einer Beobachtung zu entziehen: Denken Sie etwa an das Prinzip der Taktung in Schaltwerken, das eigens dazu angewandt wird, um Beobachtungen während des Schaltintervalls (also während eines Unsicherheitsintervalls, in dem Zwischenzustände auftreten können) zu verhindern; oder an die Tatsache, daß konstruktiv einfach keine Möglichkeit vorgesehen ist, einen in Ummagnetisierung befindlichen Punkt einer magnetisierbaren Oberfläche (bezüglich eventueller Zwischenzustände) zu beobachten. Die Annahme der Zustandsdiskretheit, und damit der Zeitlosigkeit von Zustandsübergängen (im Rahmen unserer Beobachtungsfähigkeiten), ist also durchaus realitätskonform.

spontane Zustands-änderungen

Jetzt könnten obige detaillierte Überlegungen uns veranlassen, an der Vorstellung von "Zeit" Präzisierungen vorzunehmen, entspricht doch dem Prinzip der Taktung eine diskrete (und keine kontinuierliche) Zeitachse, eine diskrete Menge von Zeitpunkten, zu denen Beobachtungen vornehmbar sind. Diesem Hinweis müssen wir aber, nach nochmaligem Nachdenken, nicht nachgehen: Insgesamt wird ein RS, auch wenn es getaktete Komponenten enthält, i.allg. ein asynchrones System sein; unterschiedliche Prozessoren (CPU's, Kanäle) werden wechselseitig asynchron arbeiten und auch asynchron zur elektromagnetisch "bewegten" E / A-Peripherie (Platten, Bänder, ...). Eine kontinuierliche Zeitachse, also nicht eine diskrete Menge von Zeitpunkten, zu denen Zustandswechsel vorkommen können, ist adäquat: Ein RS sollte als zustandsdiskretes, zeitkontinuierliches System auffassbar sein. Um aber (trotz aller begrüßenswerten Abstraktion) ganz ehrlich zu bleiben: Es verbleiben eine Reihe von Vorgängen, die doch zustandskontinuierlich ablaufen: Die Änderung der Rotationsposition rotierender Speichermedien (Platten, Floppies), die Änderung der Lese-/Schreib-Position linear bewegter Speichermedien (Bänder, Kassetten), die räumliche Position eines "Pakets" auf einem Ethernet-Kabel, ... Eine Verfolgung dieser Zustandskomponenten ist mit der Vorstellung eines zustandsdiskreten Systems nicht verträglich -

solche Zustandskomponenten schließen wir damit als "für uns uninteressant" aus unseren Überlegungen aus.

Beispiel: "busy-bit"

Ein Beispiel für eine potentiell interessante, einfache Zustandkomponente und für Möglichkeiten ihrer Beobachtung: Ein zentraler Prozessor (eine CPU) kann zu bestimmtem Zeitpunkt tätig / arbeitend / "busy" sein oder untätig / ohne Arbeit / "idle". Nehmen wir an, dieser Aspekt des Zustands des betrachteten Prozessors sei an einem hardwaremäßig realisierten "Statusbit" (einem Flip-Flop, einer Position eines Status-Registers) ablesbar - eine durchaus realitätskonforme Annahme. Eine Trajektorie dieses "CPU-busy-bits" könnte folgendes Aussehen haben:

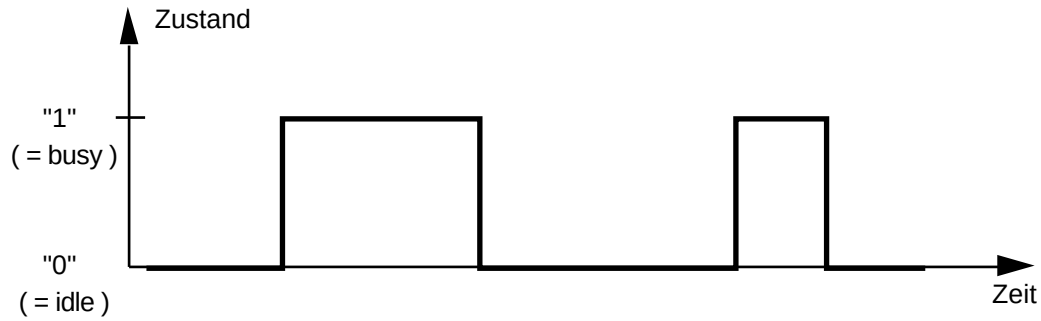

Abbildung 2.1.2: Trajektorie CPU-busy-bit

sampling

Unsere Vorstellungswelt war auf der Annahme aufgebaut, wir könnten zu beliebigem t MI eine Zustandsbeobachtung vornehmen. Um eine Vorstellung vom Verlauf einer Trajektorie zu gewinnen, müssen wir demnach "immer wieder" (während des Meßintervalls MI) beobachten. Dieser prinzipielle Ansatz zur Vornahme von Beobachtungen / Messungen trägt die Bezeichnung **sampling**. Er wird i.allg. durchgeführt, indem ein "Abtastintervall" Δ festgelegt und zeitlich äquidistant beobachtet wird; Δ ist also der zeitliche Abstand je zweier, unmittelbar aufeinander folgender Meßzeitpunkte. (Als Nebenbemerkung: Äquidistanz von Meßzeitpunkten ist, auf der Basis theoretischer Überlegungen, nicht zwingend, dennoch aber allgemein üblich.) In unserem Beispiel könnte ein sampling-Vorgang wie folgt aussehen:

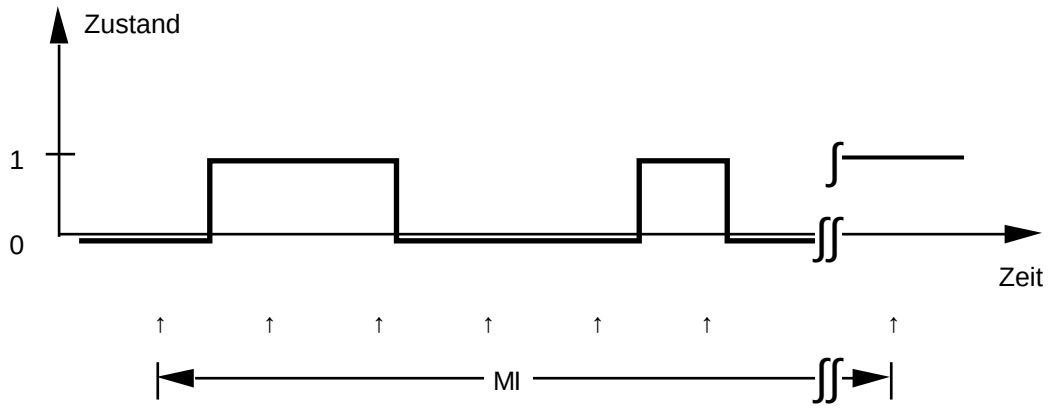

Abbildung 2.1.3: Sampling-Beobachtung CPU-busy-bit

Beobachtungen

Sampling liefert demnach eine Folge von Beobachtungen, deren volle Informa-

tion durch eine Folge von Paaren (Zeit, Zustand) beschrieben ist, im Beispiel also (wenn wir willkürlich den Beginn von MI mit Zeitpunkt "0" gleichsetzen):

(2.1.4) (0,0),(Δ,1),(2Δ,1),(3Δ,0),(4Δ,0),(5Δ,0),(6Δ,1),(7Δ,0),…,(nΔ,1)

Das Meßintervall hat dabei eine Länge T=nΔ; Anfang und Ende des Meßintervalls sind durch je eine Messung festgelegt; es liegen n+1 Messungen vor.

Verdichtungen

Nun ist unser direktes Interesse an der potentiell sehr langen Beobachtungsfolge, die der sampling-Vorgang liefert, höchstwahrscheinlich sehr gering. Wir werden an "gröberen" Charakterisierungen der Zustandstrajektorie interessiert sein - z.B. an der Feststellung der (relativen) Häufigkeit des Auftretens von busy- bzw. idle-Beobachtungen als Hinweis auf die Auslastung (relative busy-Zeit) der CPU. Messung **und** Bewertung: Es gilt, Werte von Leistungs- (hier Effizienz-)Maßen aus direkten Messungen abzuleiten. Die relativen Häufigkeiten, rH(…), lassen sich natürlich einfach bestimmen (wir verwenden im folgenden das Zeichen "#" für "Zahl von"):

(2.1.5) rH(busy) = #1/(n+1)
rH(idle) = #0/(n+1)

Aufzeichnung und Zählung

Ein unmittelbarer Hinweis auf die Methodik des Messens verdient hier festgehalten zu werden: Je nach angestrebtem Bewertungsmaß muß nicht die volle Information (vgl.(2.1.4)) einer Messung festgehalten/aufgezeichnet werden; vielmehr kann es sich anbieten, Messungen unmittelbar zu "verdichten" (vgl. (2.1.5), wo die Zahl der 1-Abtastwerte und der 0-Abtastwerte als - sehr viel ökonomischeres - Meßergebnis hinreicht und daher statt einer **Aufzeichnung** eine verdichtende **Zählung** vorgenommen werden kann).

Testfrage 2.1.6: Können Sie im obigen Beispiel auch noch auf die Zählung "#0" verzichten, also mit "#1" sowohl rH(busy) als auch rH(idle) bestimmen? Allgemeine Folgerung aus Ihrer Antwort?

Interpretation von Messungen

Die Bestimmung der relativen Häufigkeiten rH(…) obiger Messung ist für sich völlig in Ordnung. Mögliche Interpretationen dieser Werte sollten aber hinterfragt werden. Ist

(2.1.7a) rH(busy) = relativer zeitlicher Bruchteil
der CPU-busy-Zeit (in MI) ??

(2.1.7b) rH(busy) = Wahrscheinlichkeit, zu beliebigem Zeitpunkt t
die CPU busy zu finden ??

Die Antworten auf diese Fragen sind gar nicht so leicht! Zu (2.1.7a) fällt schnell auf, daß die in Frage gestellte Behauptung bei "sehr hoher" Abtastfrequenz (=1/Δ) so ungefähr in Ordnung gehen sollte - aber was ist bei niedrigerer Abtastfrequenz? Das sehr viel allgemeinere (2.1.7b) (es ist ja gar nicht auf ein bestimmtes MI bezogen) führt ebenso schnell zu der Überlegung, daß bei einer nächsten Messung (ebenfalls mit Abtastintervall Δ und ebenfalls der Länge T=nΔ) wohl ein anderes rH(busy) resultieren würde und, der langen Rede kurzer Sinn, zu der zwingenden Vermutung, daß bei der Auswertung von Messungen stochastische Überlegungen sowie, darauf aufbauend, statistische Methoden wohl eine bedeutsame Rolle spielen müssen; Konkreteres dazu in Kap. 3.

Beurteilung sampling

Trotz der diversen Fragezeichen: Der sampling-Ansatz scheint zur Bestimmung von (absoluten und relativen) Häufigkeiten des Auftretens von Zuständen und damit eventuell auch zur Bestimmung (absoluter und relativer) Zeitanteile der Zustände an gesamten Meßintervallen, u.U. auch in der Interpretation "Wahrscheinlichkeit des Auftretens von Zuständen", geeignet zu sein.

Zeitdauer-Messungen

Andere Informationen über die Zustandstrajektorie sind mit dem sampling-Ansatz deutlich schwerer zu gewinnen. Nehmen wir als weiteres Beispiel an, wir interessierten uns für die Länge / Dauer von Tätigkeits- bzw. Untätigkeits-Perioden unserer CPU (engl. "busy periods" bzw. "idle periods"; in Abb. 2.1.2 / 2.1.3 sind je zwei volle busy periods eingezeichnet). Wir sehen sofort, daß der Einfluß der Abtastfrequenz auf unsere diesbezüglichen Möglichkeiten sehr stark ist; nur bei relativ hoher Abtastfrequenz können wir ja einigermaßen sichergehen, daß aufeinanderfolgende "1"-Beobachtungen aus genau einer busy period stammen, aufeinanderfolgende "0"-Beobachtungen aus genau einer idle period, und selbst bei hoher Abtastfrequenz müssen wir ohne weitere Kenntnisse davon ausgehen, daß uns die eine oder andere busy period / idle period verlorengeht.

Testfrage 2.1.8: Gesetzt den Fall, wir wüßten, daß eine busy period / idle period nicht kürzer andauern kann als eine Zeit d. Wie hoch müßte die Abtastfrequenz dann mindestens sein, um wirklich jede busy period / idle period zu bemerken?

Einfluß Abtastfrequenz

Bei niedrigerer und besonders bei (im Vergleich zum beobachteten Vorgang) niedriger Abtastfrequenz geht uns offensichtlich die Folgerungsmöglichkeit von der Zahl aufeinander folgender 1- bzw. 0-Beobachtungen auf die Länge von busy bzw. idle periods mehr und mehr und schließlich völlig verloren.

Selbst bei hoher Abtastfrequenz sind wir bei einer ersten Stufe der Beobachtungsverdichtung darauf verwiesen, einer einzelnen busy period / idle period eine Länge von

(2.1.9a) Länge busy period = (#aufeinanderfolgende "1"en)·Δ

(2.1.9b) Länge idle period = (#aufeinanderfolgende "0"en)·Δ

zuzurechnen - mit offensichtlichen Ungenauigkeiten bzgl. Anfangs- und Endzeitpunkt des einzelnen interessierenden busy / idle-Zeitintervalls. Wenn wir diese Ungenauigkeiten im Moment tolerieren, könnte eine volle Aufzeichnung dieser ersten Verdichtungsstufe aus der Beobachtungsfolge

Verdichtungen

(2.1.10) (Länge busy period 1), (Länge idle period 1),
(Länge busy period 2), (Länge idle period 2), …

bestehen; weniger Daten zwar als (2.1.4), aber immer noch relativ viele. Die verschiedenen busy periods / idle periods werden i.allg. unterschiedlich lang sein: Es könnten gröbere Charakterisierungen gefragt sein wie etwa "mittlere Länge einer busy period / idle period". Mittlere Längen lassen sich bestimmen gemäß

(2.1.11) mittlere Länge busy period (dieser Messung!)
$:= (\sum_i (\text{Länge busy period } i)) / (\#\text{busy periods})$

mittlere Länge idle period entsprechend

so daß (bei dieser Interessenlage) auf die volle Aufzeichnung (2.1.10) verzichtet

werden kann und es in ökonomischerer Aufzeichnung (weiter verdichtend!) ausreicht, #1, #0, #busy periods, #idle periods aufzusummieren (Zählungen), woraus in endgültiger "Bewertung" folgt (bitte nachdenken!):

(2.1.12) mittlere Länge busy period
$= (\#1)\cdot\Delta/(\#busy\ periods)$

idle period entsprechend

Unschwer erkennbar: Der Ruf nach statistischer Präzisierung wird deutlicher.

Da nun der sampling-Ansatz zur Bestimmung von Zeitdauern und damit zu einer vollständigen Verfolgung des sequentiellen Verlaufs von Trajektorien beileibe nicht als ideal bezeichnet werden kann, die Frage nach "anderen" Meßansätzen: Für die Beobachtung der Dauern von (z.B.) busy/idle periods wäre es ideal, wenn wir die Zeitpunkte des Umklappens 0->1, 1->0 direkt "bemerken" könnten, also (allgemein) Zeitpunkt und Art aller Zustandsänderungen.Wir möchten also Zeitpunkt und Art von (neue Sprechweise:) **Ereignissen** bemerken. Was genau ein Ereignis ist/sein kann, bleibt dabei zunächst undefiniert außer der (schon angesprochenen) Festlegung, daß ein Ereignis zu bestimmtem **Zeitpunkt** stattfindet (Zeit nach wie vor kontinuierlich) und daß es von gewisser Art/von gewissem **Typ** ist (endliche Menge möglicher Ereignistypen). Genauere Definitionen können nur im Hinblick auf aktuelle Meßbedürfnisse vorgenommen werden, und auch da verbleiben Freiheiten, die nach Zweckmäßigkeit auszufüllen sind.

Ereignisse

Konkreter, im Rahmen unseres Beispiels: Wir wollen Anfangs- und End-Zeitpunkte von busy und idle periods entdecken. Wir könnten definieren, daß wir es mit zwei Ereignistypen zu tun haben, nämlich ↑ := (Wechsel 0->1) und ↓ := (Wechsel 1->0), und daraufhin anstreben, genau diese Ereignisse (nämlich die Tatsache eines Zustandswechsels) zu erkennen. Wir könnten aber auch bedenken, daß die Zustandsänderungen nicht "von sich aus" geschehen, sondern ihre Ursachen haben: Die CPU-Zuteilung an Prozesse erfolgt durch einen "Dispatcher", eine bestimmte Routine des Betriebssystems; der Dispatcher arbeitet gemäß festgelegter Strategie, "weiß", welche Prozesse rechenbereit, durch E/A-Aktivitäten oder Warten auf Synchronisation blockiert, noch nicht rechenbereit aber startbereit, überhaupt beendet, … sind; der Dispatcher verursacht in Verfolgung seiner Strategie die CPU-Zuteilung und den CPU-Entzug; Ereignisse (Erinnerung: in einer höheren Schicht) sind hier von den Typen CPU-Zuteilung und CPU-Entzug - diese zu entdecken, ist für unsere Zwecke gleichbedeutend mit der zuvor vorgeschlagenen Entdeckung von ↑ und ↓.

Allgemeiner: Die Freiheit der Definition von Ereignissen ist i.allg. immer dadurch gegeben, daß wir zu entdeckende Zustandswechsel selbst als Ereignisse auffassen, oder daß wir Geschehnisse, die solche Zustandswechsel auslösen, zu Ereignissen erklären.

Freiheiten der Ereignis-Definition

Der Meßansatz des Entdeckens von Ereignissen nennt sich **eventing** (event = Ereignis); es ist ein zum zuvor besprochenen sampling-Ansatz alternativer Ansatz.

eventing

Versuchen wir mit diesem eventing-Ansatz zu arbeiten, wieder anhand unseres Beispiels: Eine volle Aufzeichnung aller gelieferten Information besteht aus einer Folge von Paaren (Ereigniszeitpunkt, Ereignistyp). Bezüglich Abb. 2.1.3 hätten wir also vorliegen (Ereigniszeitpunkte t_i einfach mit steigendem Index von 1 an

durchnumeriert):

(2.1.13) $(t_1,\uparrow),(t_2,\downarrow),(t_3,\uparrow),(t_4,\downarrow),\ldots$

Verdichtungen

Verdichtende Aufzeichnungen bieten sich auch hier je nach Bewertungsziel an:

- Bei Interesse an allen Längen von busy/idle periods die Folgen von

(2.1.14) t_{i+1}-t_i mit Ereignis $\downarrow$ zur Zeit t_{i+1}
t_{i+1}-t_i mit Ereignis $\uparrow$ zur Zeit t_{i+1}

- Bei Interesse an mittleren Längen von busy/idle periods die Aufsummierung aller ↑-Ereigniszeitpunkte, ↓-Ereigniszeitpunkte und die Zählung der #↑ und #↓ (bei letztlicher Auswertung Anpassungen zu Anfang und Ende des Gesamt-Meßintervalls MI beachten!). Und auch hier entsteht unmittelbar das Bedürfnis hinsichtlich statistischer Absicherung.

Vergleiche Meßansätze

Im Vergleich der beiden Meßansätze sampling und eventing können wir feststellen, daß wir mit dem eventing-Ansatz die Feststellung von Zeitdauern und die (u.U. detaillierte) Verfolgung von Zustandstrajektorien gut im Griff haben - hier hatte der sampling-Ansatz ja Schwierigkeiten. Hat nun, umgekehrt, der eventing-Ansatz Schwierigkeiten, wo der sampling-Ansatz erfolgreich erschien? Erinnern wir uns, es ging dabei um die Feststellung der Häufigkeiten des Einnehmens von Zuständen und daran angelehnter Fragen. Eine Probe aufs Exempel: In (2.1.5, 2.1.7a) ging es z.B. um die Ermittlung von rH(busy). Das Ergebnis entsprechend Interpretation (2.1.7a) können wir aus eventing-Beobachtungen leicht herleiten. Es ist

(2.1.15) $rH(busy)=(\sum \text{Dauer busy periods})/T$

Ökonomie

Die Übereinstimmung dieses Ergebnisses mit dem gemäß (2.1.5) ermittelten ist eher ein Problem des sampling-Ansatzes (Abtastfrequenz!). Also: Keine prinzipiellen Schwierigkeiten beim eventing-Ansatz hinsichtlich "derlei" Fragen. Wo sich allerdings eine Unterlegenheit des eventing-Ansatzes ergeben kann (und auch tatsächlich ergibt), hängt mit der bisher unerörterten Frage zusammen, ob beim sampling-Ansatz auch niederfrequentes Abtasten zu hinreichend guter Bestimmung der Häufigkeiten des Einnehmens von Zuständen führt. Sollte diese Frage positiv zu beantworten sein (dies wohl nur nach geeigneten stochastischen/statistischen Überlegungen), dann stellt sich der sampling-Ansatz (nur "seltenes" Beobachten nötig) als potentiell ökonomischer dar als der eventing-Ansatz ("häufiges" Beobachten bei jedem Ereignis nötig) - dies allerdings unter Verlust der Information bzgl. Zeitdauern, wie vorne ausgeführt.

Zusammenfassung

Wir sind soweit, ein kleines Resümee ziehen zu können: Wir haben zwei unterschiedliche Meßansätze kennengelernt, sampling und eventing, und uns über deren Vor- und Nachteile Klarheit verschafft. Beiden Ansätzen war gemeinsam, daß je nach Interessenlage, d.h. je nach momentanen Bewertungsgrößen, die (potentiell sehr große) Folge von Beobachtungsdaten zunächst verdichtet werden konnte, bevor sie zur letztendlichen Auswertung aufgezeichnet (gespeichert) wurde. Ergänzen wir hier: Wir könnten statt an einer Aufzeichnung an einer direkten Anzeige (Statuslämpchen, Kontrollausdrucke, … spezielles Farbdisplay) Interesse haben. Bei der letztendlichen Auswertung (erahnten wir, ohne bis jetzt Detailliertes erfahren zu haben) sollten statistische Verfahren eine wesentliche Rolle zu spielen haben. Eine (noch etwas angereicherte) Prinzipskizze eines Me-

ßaufbaus / Meßvorgangs ist in Abb. 2.1.16 wiedergegeben (vgl. auch FeSZ83 der Literaturliste des Kap. 1): *Prinzip Meßaufbau*

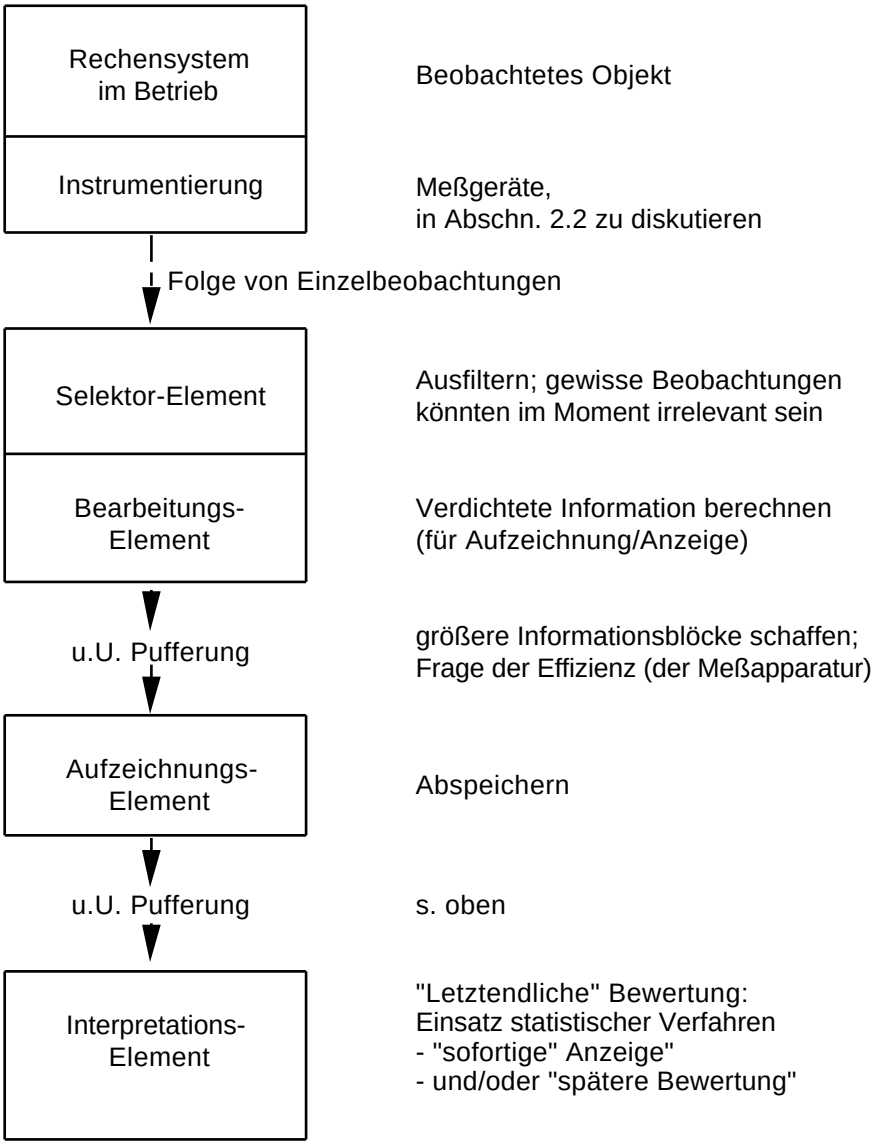

Abbildung 2.1.16: Prinzip-Skizze Meßaufbau / Meßvorgang

2.2 Meßinstrumente

Monitore

Zur Durchführung von Messungen sind selbstverständlich geeignete Meßinstrumente, Meßinstrumentarien erforderlich - wie in Abb. 2.1.16 ohne weitere Erklärung auch eingezeichnet. Die (Meß-)Instrumentierung speziell von RS besteht aus sog. **Monitoren** (engl. monitor = Überwacher). Es gibt zwei unterschiedliche Arten von Monitoren:

- **Hardware-Monitore** sind elektronische Geräte (Hardware): Über Kabel und Sensoren an interne Kontaktpunkte des zu beobachtenden RS angeschlossen, sind sie in der Lage, Spannungsniveaus (gegenüber einem Basisniveau) zu messen; Hardware-Monitore stehen im Prinzip "neben" dem gemessenen Objekt-RS.

- **Software-Monitore** sind Programme (Software); in die Software des zu beobachtenden RS eingebettet, werden sie zu bestimmten Zeitpunkten gestartet und sind dann in der Lage, Inhalte speichernder Elemente zu "messen" (d.h. zu lesen); Software-Monitore sind im Prinzip "in" dem gemessenen Objekt-RS installiert.

Von diesen beiden Monitorarten gibt es Abarten und Mischarten, die zu Ende dieses Abschnitts kurz erklärt werden. Beide Arten von Monitoren sind für beide Typen von Meßansätzen, nämlich sampling und eventing (vgl. Abschn. 2.1), einsetzbar. Unterschiede zwischen den Monitor-Arten und ihre grundlegenden Stärken und Schwächen können wir erst nach detaillierterer Diskussion erörtern.

SW-Monitore

sampling

Beginnen wir mit den Software-Monitoren. Wie könnten wir sampling-Messungen vornehmen? Nun, wir müssen dafür sorgen, daß ein bestimmtes, eigens für diesen Zweck geschriebenes und auf dem RS installiertes "Meß"-Programm in regelmäßigen Abständen (Abtastintervall!) gestartet wird. Mit diesem Startzeitpunkt ist implizit der Zeitpunkt der Messung bekannt. Zur "Zustandsfeststellung" kann das Meßprogramm während seines Ablaufs Inhalte von Speicherzellen und Registern lesen. Je nach Ebene der Beobachtung wird es sich dabei um Zustände eines zu messenden Programms handeln (Speicherzellen, die Werte von Programmvariablen enthalten), oder um Betriebssystemzustände (Speicherzellen, die Tafeln des Betriebssystems enthalten) aber auch um Hardware-Zustände (Statusregister der Hardware, falls diese auf Instruktionsebene sichtbar sind). Das letzte der drei Beispiele weist auf potentielle Schwierigkeiten hin. Zunächst kann nicht jedes Programm auf jedes Speicherelement zugreifen: Die CPU könnte in verschiedenen Modi arbeiten, welche jeweils spezifische Speicherzugriffs- (und Instruktionstyp-)Berechtigungen aufweisen. Um diese Schwierigkeit zu umgehen, legen wir für den Moment fest, daß Meßprogramme im sog. System-Modus arbeiten, also Zugriffsberechtigung bzgl. aller per Software überhaupt lesbaren Speicherelemente besitzen. Aber eben nur auf solche: Speicherelemente, die auf Instruktionsebene nicht sichtbar sind (z.B. Hardware-/Firmware-interne Register) sind mit SW-Monitoren prinzipiell nicht beobachtbar.

eventing

SW-events

Können wir mit Software-Monitoren auch eventing-Messungen anstellen? Ja, ganz allgemein, wenn wir uns bei den festzustellenden Ereignissen auf ganz spezielle, sog. **Software-events**, beschränken. Ein Software-event tritt dann ein, wenn ein bestimmtes festgelegtes Programm über eine bestimmte festgelegte Stelle seines Codes "läuft", wenn also eine genau festgelegte Instruktion zur Ausführung gelangt. An dieser Stelle muß die Ausführung eines Meßprogramms zeitlich eingeschoben werden (wir werden gleich sehen, wie), das seinerseits die

momentane Zeit feststellt und auf Speicherinhalte (Zustandsfeststellung) zugreift. Auch hier aber die Beschränkung auf per Software zugreifbare Speicherelemente.

Beeinflussung durch Messung

In beiden Fällen wird durch die Messung per Software-Monitor der beobachtete Betrieb gestört: Ein Meßprogramm kann nicht umhin, während seines Ablaufs System-Ressourcen zu benutzen; so den Prozessor, der das Meßprogramm ausführt; Speicher zur Notierung der Beobachtungen; zumindest gelegentlich E / A-Peripherie zur Aufzeichnung der Meßresultate. Diese Ressourcen sind dem Normalbetrieb entzogen, teils permanent, teils vorübergehend. Das RS läuft mit (Software-)Meßinstrumentierung anders, als es ohne sie laufen würde. Diese Beeinflussung (der sog. "measuring artefact") ist dem Software-Monitoring inhärent, wir können höchstens versuchen, sie "relativ gering" zu halten.

SW-Monitor, Grundformen

Es gibt drei Grundformen, den Software-Monitor-Code im RS zu installieren und zu aktivieren:

(i) Eigenständige Programme können (ohne Veränderung der "normalen" Objekt-Software) zusätzlich installiert werden; diese müssen dann offensichtlich selbst für ihre Aktivierung zu geeigneten Zeitpunkten sorgen.

(ii) Zu beobachtende Programme der Objekt-Software können vorbereitend derart modifiziert werden, daß sie an bestimmten Punkten ihres Ablaufs spezielle, zuvor installierte Meßprogramme aktivieren.

(iii) Die System-Software, insbesondere die Betriebssystem-Software, kann vorbereitend derart modifiziert werden, daß sie anläßlich der Durchführung bestimmter organisatorischer Aktivitäten Meßwerte erhebt.

Lösung (i)

Vorteile der Lösung (i) liegen bei der Tatsache der minimalen (nämlich wegfallenden) Veränderungsnotwendigkeit der Objekt-Software. Eine reine Anwendung dieser Lösung setzt aber gewisse Hardware-Fähigkeiten voraus: "Irgend jemand" muß das Meßprogramm ja aktivieren; wenn es nicht die Objekt-Software ist (diese fällt, da nicht verändert, dafür aus) und nicht das Meßprogramm selbst (aber wie sollte es, in nicht aktiviertem Zustand, irgend etwas aktivieren, geschweige sich selbst), müssen wohl Hardware-Eigenschaften eingesetzt werden. Diese (notwendigen) Eigenschaften bestehen aus der Existenz von interrupt- (Unterbrechungs-) Mechanismen, darüber hinaus aus der Setzbarkeit von timer- (Uhren-) interrupts von der Instruktionsebene aus.

Interrupts, Timer

Die prinzipiellen Charakteristika einer diesbezüglichen Meßtechnik sind in Abb. 2.2.1 skizziert. Als kurze Erläuterung: Das Meßprogramm, einmal aktiviert, wiederholt laufend (bis es per "Meßbedingung" abgeschaltet wird) die Aktionen:

- Setzen eines Uhren-Interrupts für einen (zukünftigen) Zeitpunkt, den Zeitpunkt der nächsten Beobachtung; offensichtlich kann hier direkt ein sampling-Ansatz realisiert werden (äquidistante Meßzeitpunkte).

- "Warten" auf diesen Meßzeitpunkt, z.B. durch deaktivierenden SVC (synchronen Interrupt), der per diesbezüglicher Interrupt-Behandlungs-Routine den CPU-Dispatcher zur Deaktivierung dieses (des Meß-)Programms nötigt; der zum zukünftigen Meßzeitpunkt eintretende Uhren-Interrupt fordert, per zugeordneter Interrupt-Behandlungs-Routine, den CPU-Dispatcher auf, dieses (das Meß-)Programm wieder zu aktivieren; will man auf das zeitgenaue Eintreffen

dieser Aktivierung bauen, sollte sinnvollerweise der Uhren-Interrupt hohe bzw. sogar höchste Priorität besitzen (Verfügbarkeit priorisierter, "maskierender" Interrupts vorausgesetzt).

Sparsamkeit:

Pufferung

- Zustandsbeobachtung, Ablage Meßdaten; Sparsamkeit dieser Aktionen ist wesentlich, um die Beeinflussung des Normalbetriebs gering zu halten; i.allg. Ablage in einem Puffer, der von einem anderen (dem Meßvorgang zugeordneten) Verdichtungs- und / oder Aufzeichnungsprogramm "bei Gelegenheit" (vorzugsweise, wenn "sonst nichts Wichtiges zu tun ist") geleert wird.

Meßprogramm:

```
"Initialisierung";
LOOP WHILE "Meßbedingung"
   SET_INTERRUPT ("Zeitpunkt");      Existenz war vorausgesetzt
   WAIT;                             z.B. SVC
   "Zustandsbeobachtung";
   "Ablage Meßdaten"
END LOOP
```

Start Messung:

```
"Meßbedingung":=TRUE                 u.U. manuell
ACTIVATE "Meßprogramm"
```

Stop Messung:

```
"Meßbedingung":= FALSE               u.U. manuell
```

Abbildung 2.2.1: Software-Monitor; sampling-Ansatz

Lösung (ii)

Die Lösung (ii) ist Ihnen im Prinzip als eine der üblichen Vorgehensweisen beim Testen von Programmen bekannt. Es gilt, an einer bestimmten Stelle eines (später zu aktivierenden) Programms eine Ausgabeanweisung, jetzt: einen Meßcode, einzubringen.

Zu messendes Programm:

```
.
.     ursprüngliche Anweisungen des Programms
.
"Meßcode"
.
.     ursprüngliche Anweisungen des Programms
.
```

Abbildung 2.2.2: Software-Monitor; (Software-)eventing-Ansatz

Wie schon in der Übersicht erwähnt, wird bei diesem Ansatz ein bestimmtes Ereignis entdeckt, nämlich die Tatsache, daß das beobachtete Programm bei seiner Ausführung die Stelle des eingebrachten Meßcodes erreicht. Wir haben also den typischen eventing-Ansatz vor uns, ausschließlich beschränkt auf Softwareevents. In der skizzierten einfachen Form wird diese Meßtechnik am häufigsten zur Beobachtung von Anwendungsprogrammen eingesetzt. Der Meßcode selbst hat die Aufgabe (vgl. eventing, Abs. 2.1),

- die Zeit des Ereignisses (also die "momentane" Zeit) festzustellen, wozu es erforderlich ist, eine Uhr lesen zu können (spezieller Systemdienst);

- eine Zustandsbeobachtung vorzunehmen, die sich hier vornehmlich auf Variablenwerte des beobachteten Programms beschränken muß (Anwendungsprogramm, und damit auch eingebrachter Meßcode, laufen ja nicht im System-Modus und sind damit in ihren Speicherzugriffsrechten beschränkt);

- eine Ablage der Informationen:
  - Zeit,
  - Kennzeichnung des Ereignisses (es könnten ja mehrere Meßcodeteile über das Programm verteilt sein)
  - (u.U.) Zustandsinformation

  vorzunehmen.

Erweiterungen, und damit schrittweise Beseitigung der Einschränkungen, können vorsehen,

- statt des direkt eingeschachtelten Meßcodes einen Prozedur / Unterprogramm-Aufruf bzgl. einer Meßroutine einzubringen, die etwa vorhandene, verfügbare Systemdienste des Messungs-Sammelns / Aufbereitens / Aufzeichnens in Anspruch nimmt; (zur Erinnerung: Sparsamkeit, und damit Pufferung, ist hochwillkommen);

- den "Aufruf" der Meßroutinen per speziellem SVC (also synchronem interrupt) zu gestalten, wodurch auch ein Modus-Wechsel und damit eine breitere Berechtigung des Zugriffs auf Speicherelemente ermöglicht wird.

Lösung (iii)

Die Lösung (iii) schließlich unterscheidet sich von Lösung (ii) nicht in der Wahl der Mittel. Auch hier wird an einer bestimmten Stelle eines Programms ein Stück Meßcode eingefügt bzw. von dieser Stelle zu einem Meßprogramm verzweigt und anschließend zurückgekehrt (dem Wesen nach also ein Unterprogrammsprung ausgeführt). Nur ist das modifizierte Programm hier typischerweise eine Betriebssystem (BS)-Routine, und bei dem zu entdeckenden Ereignis und der darauf folgenden Zustandsbeobachtung geht es weniger um diese, durch Meßcode veränderte BS-Routine als vielmehr um die Aufgabe, die sie durchzuführen hat, und den Grund für ihren Aufruf. Betrachten wir z.B. den Scheduler, eine typischerweise vorhandene BS-Routine mit der Aufgabe, neu ankommende Benutzer-Tasks / Prozesse zu initiieren und (nach Abschluß oder wegen sonstiger "Verdrängungs"-Ursachen) wieder zu beenden (in der hier verwendeten Terminologie verwaltet ein Dispatcher nur die CPU(s) selbst, die Aufnahme in die Menge Hauptspeicher / CPU-berechtigter tasks, den "multiprogramming set", fällt in die Aufgabe des Schedulers). Der Scheduler wird also immer aktiv, wenn eine Benutzer-Task startet und wenn sie endet, und dies sind auch die Ereignisse, deren Entdeckung sie per Aufzeichnung als Meßergebnis zu melden hat: Wann welche Task begann (ein Typ von Meßergebnis) und wann welche Task endete (ein anderer Typ von Meßergebnis) - so daß aus der Folge dieser Meßergebnisse sich anschließend z.B. die Verweilzeit im "inneren" System für jede Task ermitteln läßt. Da BS-Routinen häufig arbeiten, ist die Beeinflussung des Betriebs durch Messungen nach Lösung (iii) deutlich spürbar - die Sparsamkeits-Forderung damit hier am stärksten.

Erweiterungen

Im Nachhinein betrachtet, besaß Lösung (i) eine Annehmlichkeit, die Lösungen

(ii,iii) nicht aufweisen: Man konnte die Messung "abschalten" (Sparsamkeit!). Diese Annehmlichkeit können wir aber den Lösungen (ii,iii) auch verleihen. Als erste Möglichkeit eine dynamische Aktivierung von statisch im zu beobachtenden Programm vorhandenen Meßstellen (sog. "traps", "Fallen"):

traps

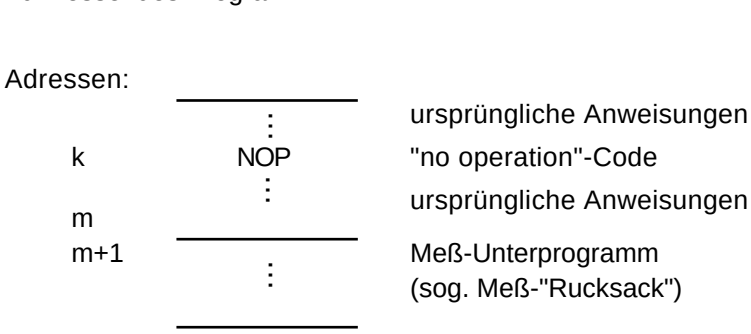

Abbildung 2.2.3; Software-Monitor; (Software-)eventing-Ansatz; statische "trap" mit dynamischer Aktivierung der Messung

hooks

Als zweite, noch elegantere Möglichkeit die dynamische Einbringung von Meßcodeverzweigungen (sog. "hooks", "Haken") in statisch nicht vorbereiteten, zu beobachtenden Code (vgl. Abb. 2.2.4).

Diskussion SW-Monitore

Es wird Ihnen klar geworden sein, daß Software-Monitore, die ja Hardware- und BS-Software-Fähigkeiten weidlich nutzen, sorgfältig auf ein konkretes Hardware-/Software-System abgestimmt sein müssen; Software-Monitore sind daher insbesondere nicht portabel. Auch ist ihr Entwurf und ihre Implementierung kompliziert:

- Eine "totale" Kenntnis von Hardware und System-Software ist erforderlich.
- Die Kontextfragen (wann darf wer unter welchen Bedingungen auf welche Speicher zugreifen) sind schwierig zu lösen.
- Die "overhead"-Fragen (wie "sparsam" muß die Implementierung sein) sind drängend.
- Die Gefahr der funktionalen Beeinflussung (Einbringung von Fehlern) ist eminent.
- Software-Monitore müssen häufig verändert werden (jede neue BS-Version zwingt i.allg. dazu).

Zu messendes Programm:

```
Adressen:
                    ⋮      ursprüngliche Anweisungen
        k           x      ursprüngliche Anweisung
                    ⋮      ursprüngliche Anweisungen

Meß-Unterprogramm:
        l
                    ⋮      eigentlicher Meßcode

        m           NOP
                    RJ     Rücksprung des Meß-UP ("Return Jump")

Meß-Initialisierung(sprogramm):
   Kopiere Inhalt Speicherzelle k (nämlich Operation x) nach Speicherzelle m;
   Setze UP-Sprung nach l in Speicherzelle k

Meß-Beendigung(sprogramm):
   Kopiere Inhalt Speicherzelle m (nämlich Operation x) nach Speicherzelle k;
   (u.U.) NOP nach m
```

Abbildung 2.2.4: Software-Monitor; (Software-)eventing-Ansatz; dynamisch eingebrachter "hook" mit gleichzeitiger Aktivierung der Messung

Für den "gelegentlich" an Messungen Interessierten, der den skizzierten Aufwand nicht treiben kann, als Trost: Zumindest marginale Software-Monitor-Teile sind in jedem größeren BS zu finden: Die normalen "accounting"-Routinen, deren eigentliche Aufgabe die Ermittlung von zu verrechnenden Kosten für die System-Benutzung darstellt, sammeln Daten, die zur Leistungsbeobachtung verwendbar sind: Verbrauchte CPU-Zeit je Task, Start- und Stop-Zeitpunkt, ... Und auch darüber hinaus ist fast jedes größere BS mit "standardmäßig mitlaufenden" Meßroutinen ausgestattet - machen Sie die Probe, und forschen Sie bei den von Ihnen benutzten Rechnern, dem von Ihnen frequentierten Rechenzentrum danach!

Uhren

Der Software-Monitor-Bereich soll nicht abgeschlossen werden, bevor eine der vorausgesetzten Fähigkeiten der Hardware etwas detaillierter betrachtet wurde, nämlich die Zugreifbarkeit von "Uhren" und "Zeit-Interrupts".

Wo vorhanden, besteht eine **Hardware-Uhr** aus einem speziellen, reservierten Speicher (Register oder Hauptspeicherzelle), der als Binärzähler fungiert und seitens eines speziellen Zeitgebers (Oszillators) über äquidistante Impulse fortlaufend hochgezählt wird (spezieller Binäraddierer involviert). Dieser Speicher ist über Software adressierbar, aber aus verständlichen Gründen nur lesbar - der "momentane" Zeitpunkt ist also per Software feststellbar. Konkret findet man z.B. in der IBM / 370-390-Familie eine Uhrenfrequenz von 1 MHz und eine "Breite" der Hardwareuhr von 52 bit (falls Sie eine Überlaufgefahr erahnen: In diesem Falle erst nach 143 Jahren). Neuere Architekturen sind i.allg. noch deutlich "besser". Die angebotene Taktfrequenz bestimmt natürlich die überhaupt mögliche Zeitauflösung ("resolution") von Messungen. Die sog. **Software-Uhr** bietet eine zusätzliche Annehmlichkeit. Oft in mehreren Exemplaren vorhanden, die je ei-

nem (z.B.) Benutzerprozeß zugeordnet sind, wird ein Software-Uhr-Speicher nur während der Zeit hochgezählt, während derer der zugeordnete Prozeß die (bzw: eine) CPU besitzt, also fortschreitet. Auf diese Weise sind Beobachtungen von prozeßbezogenen, "virtuellen" Zeiten möglich.

Timer

Ein software-setzbarer Timer schließlich ist wieder so ein als Binärzähler fungierender Speicher. Er läßt sich (per Software) auf einen "Anfangswert" setzen, entsprechend der Zeit bis zum "Ablaufen" der Uhr, und wird durch den Zeitgeber abwärts / herunter-gezählt. Die Besonderheit des timers besteht im Auslösen eines Interrupts, einer Unterbrechung, bei Erreichung des "0"-Wertes (bzw. konkreter, bei Wechsel des Vorzeichen-bits).

Hardware-Monitore

War schon unsere Betrachtung von Software-Monitoren reichlich kursorisch, so werden wir bei der folgenden Diskussion von Hardware-Monitoren noch kürzer werden. Dies deshalb, weil eine breitere Diskussion so detailliert und produktbezogen sein müßte, daß sie sich hier verbietet.

Wir hatten schon erwähnt, daß wir uns unter einem Hardware-Monitor ein separates Gerät vorzustellen haben, das "neben" dem zu messenden RS steht. Über Kabelverbindungen werden Meßfühler (Sensoren) des Hardware-Monitors an geeignete Kontakte des RS angeklemmt. Die Verbindung ist i.allg. galvanisch aber extrem hochohmig, um eine Beeinträchtigung des RS unter allen Umständen zu vermeiden. Schon aus diesem Prinzip lassen sich drei wesentliche Unterschiede zum Software-Monitor direkt ablesen:

Eigenschaften

- Ein Hardware-Monitor beeinflußt den Betrieb des gemessenen RS nicht.

- Die einem Hardware-Monitor zugängliche Information ist völlig anders als die dem Software-Monitor verfügbare; zum einen können, auf physikalischer Ebene, Effekte beobachtet werden, die per Software nicht zugreifbar sind (das CPU-busy-bit des Abschn. 2.1 könnte auf Instruktionsebene unsichtbar sein, ist aber physikalisch beobachtbar - einen geeigneten zugänglichen Kontakt vorausgesetzt); zum anderen ist die volle Interpretation beobachteter Effekte für den Hardware-Monitor ungleich schwerer, eventuell unmöglich (so ist per Software leicht festzustellen, für welchen Prozeß eine CPU tätig ist - man muß nur in der entsprechenden BS-Tabelle nachsehen, wogegen der für die Tatsache der "busy-bit=1"-Beobachtung verantwortliche Prozeß für den Hardware-Monitor praktisch nicht identifizierbar ist).

- Ein Hardware-Monitor ist sicher zur Beobachtung verschiedener RS einsetzbar und damit "portabler"; Voraussetzung für seinen Einsatz ist allerdings ein viel engerer Kontakt zum jeweiligen RS-Hersteller, von dem allein eine "Meßpunkt-Bibliothek" stammen kann mit Informationen über
  - die Kontakte, an denen eine Meßverbindung (bzgl. Beeinträchtigung der Funktionsfähigkeit) tolerierbar ist;
  - die Bedeutung der an diesen Kontakten beobachtbaren Spannungspegel.

festverdrahteter Aufbau

Es gibt Hardware-Monitore von sehr einfachem bis sehr komplexem Aufbau (entsprechend können sie auch sehr teuer sein - bis zu einigen Hunderttausend DM). Ein festverdrahteter (hard-wired) Hardware-Monitor weist im Prinzip einen Aufbau gemäß Abb. 2.2.5 auf. Über Meßfühler werden Spannungspegel des gemessenen RS weitergegeben, anschließend mittels logischer Verknüpfungen (AND / OR-Gatter) untereinander und evtl. zusätzlich mit einem Uhr-Takt verknüpft (ein Hardware-Monitor hat regelmäßig seine eigene Uhr); entstehende

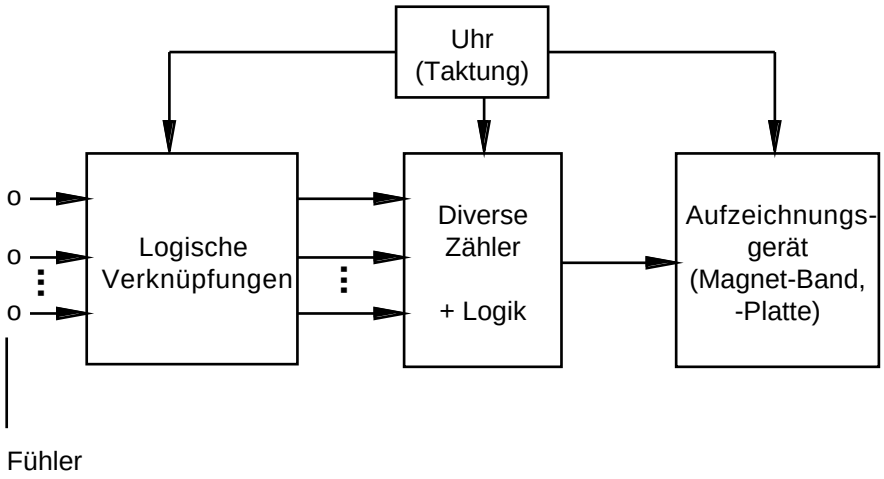

Abbildung 2.2.5: Hardware-Monitor; Prinzipskizze festverdrahteten Aufbaus

Impulse (Spannungspegel & Takt) werden ggf. gezählt; Spannungspegel und Zählerstände werden samt Zeitpunkt (aus: Taktzählung) aufgezeichnet. Die endgültige Auswertung erfolgt "off-line": Das Magnetband / die Magnetplatte mit den Messungen wird auf irgendeinem Universalrechner gelesen, die Messungen werden einem Auswertungsprogramm zugeführt.

Erweiterungen

Zu diesem Grundprinzip gibt es vielfältige und weitreichende Erweiterungen, von einem manuell veränderbaren Steckbrett (das es erlaubt, die logischen Verknüpfungen zu variieren) bis hin zu Mehrprozessor-Hardware-Monitoren, auf denen selbständige Programme laufen, welche Verknüpfungen / Zählungen per Software setzen / verändern / abfragen können, Auswertungsprogramme starten können, beliebige Aufzeichnungen auf Monitor-eigenen Displays, Druckern, Platten, Bändern vornehmen können, ...

Rolle von HW-Monitoren

Noch einige generelle Bemerkungen: Ein Hardware-Monitor sollte in seinem Technologie-Stand dem zu messenden RS ebenbürtig (besser: überlegen) sein; konkreter: Seine Auflösungsfähigkeit (minimal möglicher Meßabstand) sollte besser sein als der / die Basis-Takt(e) des beobachteten Geräts - der Grund ist offensichtlich. Ein Hardware-Monitor "veraltet" also. Des weiteren ist zu bemerken, daß im Zuge stärkerer Hardware-Integration mehr und mehr potentiell interessante Meßpunkte des zu beobachtenden RS "verschwinden": Es gibt keine diesbezüglichen Kontakte mehr, an die Sensoren angeklemmt werden könnten. Dies führt zum einen zu Bemühungen beim Rechner-Entwurf, abfühlbare Meßpunkte speziell für Messungen vorzusehen, zum anderen aber insgesamt zu einer sinkenden Rolle von Hardware-Monitoren bei der Messung von RS. Letzteres zumindest in Bezug auf die Rechner selbst, wohingegen der Einsatz von Hardware-Monitoren zur Messung datenübertragender Komponenten mit steigender Vernetzung stetig auf dem Vormarsch ist (hier gibt es ja weder die Auflösungs- noch die Kontaktpunkt-Schwierigkeiten).

Abschließend kurz etwas zu Abarten / Mischarten von Monitor-Typen:

Firmware-Monitore

Firmware-Monitore sind "Software-Monitore", die auf der Mikroprogrammebene aufsetzen: Hat man ein mikroprogrammierbares RS vor sich, dann kann man die Prinzipien der Software-Monitore in Mikroprogrammen realisieren. Die Vorteile: Beeinflussungen des Objektbetriebs sinken wegen der kürzeren Dauer der eingesetzten Mikro-Operationen; die Beobachtungsmöglichkeiten steigen - jetzt "sehen" wir ja nicht nur die Speicher der Instruktionsebene, sondern auch die der

Mikrocode-Ebene.

Hybrid-Monitore

Hybrid-Monitore schließlich versuchen die Vorteile von einerseits Software-, andererseits Hardware-Monitoren zu vereinen. Die eigentliche Messung und Abtastung wird dem (ohne Beeinflussungen arbeitenden) Hardware-Monitor auferlegt; die Schwierigkeiten, die der Hardware-Monitor bei der Interpretation von Meßwerten hat, werden dadurch beseitigt, daß Software-Monitor-artige Programme (geringen Ausführungsaufwands) genau die fehlende Information in fest vereinbarten Registern / Speicherzellen ablegen, von wo der Hardware-Monitor sie sich "abholen" kann.

Testfrage 2.2.6: Gesetzt den Fall, das BS beschränkt die Zahl aktiver (d.h. im "inneren" System befindlicher) Tasks auf maximal 8 (Multiprogramming-Grad ≤ 8) und vergibt für jede aktive Task eine der "Nummern" 0 bis 7. Zu Meßzwecken hinterlegt der Dispatcher die Nummer der gerade CPU-aktiven Task in einem spezifischen Register. Was ist nötig, um einem Hardware-Monitor (als Teil dieses Hybrid-Monitor-Aufbaus) diese Information verfügbar zu machen?

Zusammenfassung

Fassen wir zusammen: Wir haben die zur Beobachtung / Messung des Betriebs von RS verfügbaren Instrumente in Form von Monitoren der Software- / Hardware- / Firmware- / Hybrid-Typen kennengelernt. Die Einbringung der Software-Meßprogramme wurde in verschiedenen Alternativen diskutiert; als prinzipielle Nachteile der Software-Monitore erkannten wir die Beeinflussung des zu beobachtenden Betriebs, die Beschränkung auf Schichten oberhalb der Instruktionsebene, die Schwierigkeiten bei Implementierung und Wartung, die Tatsache der Nicht-Portabilität. Prinzipien des Arbeitens von Hardware-Monitoren wurden kurz vorgestellt. Als Nachteile wurden identifiziert (des einen Nachteile sind hier gleichzeitig des anderen Vorteile) die Anschaffungskosten, die Schwierigkeiten der Meßinterpretation, die Abhängigkeit von Technologie- und Architekturfragen.

Bezüglich Literatur sei auf das Literaturverzeichnis zu Kap. 1 verwiesen, insbesondere auf Stim76, Svob76, Ferr78, FeSZ83. Besondere Aufmerksamkeit auf die Schwierigkeiten der Messung in parallelen Architekturen legt u.a.

Klar95 Messung und Modellierung paralleler und verteilter Rechensysteme; Teubner 1995

Lösungshinweise zu den Testfragen

2.1.6: ***2.1.6***

Aber sicher doch:
Es ist ja #0 + #1 = n+1
daher auch rH(idle) = 1 - rH(busy)

Und als Prinzip: Wo strenge funktionale Abhängigkeiten zwischen Meßwerten vorliegen, sollten diese auch (Sparsamkeit!) zugunsten des zu leistenden Meßaufwands eingesetzt werden. Alternativ dazu können Abhängigkeiten (bei dennoch voller, insgesamt also redundanter Messung) zur gegenseitigen Überprüfung von Meßresultaten herangezogen werden.

2.1.8: ***2.1.8***

$\Delta < d$

2.2.6: ***2.2.6***

Als Festkommazahl dargestellt, belegt die Task-Nummer die niedrigstwertigen bits des Registers. Drei Sensoren müssen also angebracht (an jedem dieser "bits" einer) und zum HW-Monitor verbunden werden, und der HW-Monitor muß die Interpretation dieser drei Signale als Teil einer ganzen Zahl vornehmen können.

Leerseite