(12) NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS (PCT) VERÖFFENTLICHTE INTERNATIONALE ANMELDUNG

(19) Weltorganisation für geistiges Eigentum
Internationales Büro

(43) Internationales Veröffentlichungsdatum
4. Mai 2006 (04.05.2006)

PCT

(10) Internationale Veröffentlichungsnummer
WO 2006/045733 A2

(51) Internationale Patentklassifikation:
Nicht klassifiziert

(21) Internationales Aktenzeichen: PCT/EP2005/055390

(22) Internationales Anmeldedatum:
19. Oktober 2005 (19.10.2005)

(25) Einreichungssprache: Deutsch

(26) Veröffentlichungssprache: Deutsch

(30) Angaben zur Priorität:
10 2004 051 967.6
25. Oktober 2004 (25.10.2004) DE

(71) Anmelder *(für alle Bestimmungsstaaten mit Ausnahme von US)*: **ROBERT BOSCH GMBH** [DE/DE]; Postfach 30 02 20, 70442 Stuttgart (DE).

(72) Erfinder; und

(75) Erfinder/Anmelder *(nur für US)*: **WEIBERLE, Reinhard** [DE/DE]; Kalkaeckerstrasse 10, 71665 Vaihingen/Enz (DE). **MUELLER, Bernd** [DE/DE]; Stahler Strasse 38, 70839 Gerlingen (DE). **HARTER, Werner** [DE/DE]; Hummelberg 4, 75428 Illingen (DE). **ANGERBAUER, Ralf** [DE/DE]; Clara-Schumann-Strasse 4, 71701 Schwieberdingen (DE). **KOTTKE, Thomas** [DE/DE]; Leimentalstrasse 13/1, 71139 Ehningen (DE). **COLLANI, Yorck** [DE/DE]; Lisztweg 9, 71717 Beilstein (DE). **GMEHLICH, Rainer** [DE/DE]; Hoehenweg 2, 71254 Ditzingen (DE).

(74) Gemeinsamer Vertreter: **ROBERT BOSCH GMBH**; Postfach 30 02 20, 70442 Stuttgart (DE).

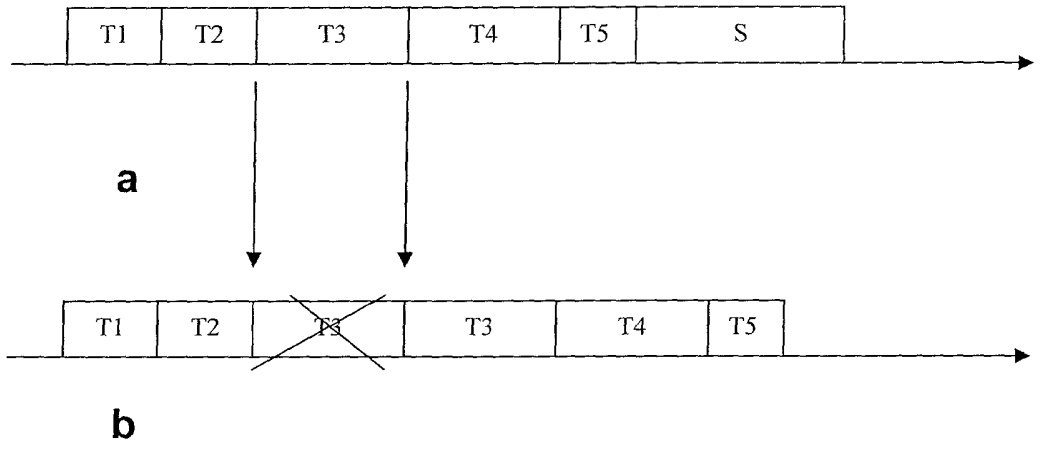
(81) Bestimmungsstaaten *(soweit nicht anders angegeben, für jede verfügbare nationale Schutzrechtsart)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Bestimmungsstaaten *(soweit nicht anders angegeben, für jede verfügbare regionale Schutzrechtsart)*: ARIPO (BW,

[Fortsetzung auf der nächsten Seite]

(54) Title: METHOD, OPERATING SYSTEM AND COMPUTING ELEMENT FOR RUNNING A COMPUTER PROGRAM

(54) Bezeichnung: VERFAHREN, BETRIEBSSYSTEM UND RECHENGERÄT ZUM ABARBEITEN EINES COMPUTERPROGRAMMS



(57) Abstract: The invention relates to a method for running a computer program on a computing element, particularly on a microprocessor. The computer program has a number of program objects, and over the course of the method, errors are detected while running the computer program on the computing element. When an error is detected, at least one program object that has already been run is transferred into a defined state and restarted from this state, and subsequent additional program objects are postponed.

(57) Zusammenfassung: Die Erfindung betrifft ein Verfahren Verfahren zum Abarbeiten eines Computerprogramms auf einem Rechengerät, insbesondere auf einem Mikroprozessor, wobei das Computerprogramm mehrere Programmobjekte umfasst und bei dem Verfahren während der Abarbeitung des Computerprogramms auf dem Rechengerät Fehler detektiert werden, wobei bei einer Detektion eines Fehlers mindestens ein Programmobjekt, das bereits einer Abarbeitung zugeführt wurde, in einen definierten Zustand überführt und aus diesem heraus erneut gestartet wird, und nachfolgende weitere Programmobjekte verschoben werden.

WO 2006/045733 A2

GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), eurasisches (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), europäisches (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Veröffentlicht:

— *ohne internationalen Recherchenbericht und erneut zu veröffentlichen nach Erhalt des Berichts*

Zur Erklärung der Zweibuchstaben-Codes und der anderen Abkürzungen wird auf die Erklärungen ("Guidance Notes on Codes and Abbreviations") am Anfang jeder regulären Ausgabe der PCT-Gazette verwiesen.

Verfahren, Betriebssystem und Rechengerät zum Abarbeiten eines Computerprogramms

Stand der Technik

Die vorliegende Erfindung betrifft ein Verfahren zum Abarbeiten eines Computerprogramms auf einem Rechengerät, insbesondere auf einem Mikroprozessor. Das Computerprogramm umfasst mehrere Programmobjekte. Bei dem Verfahren werden während der Abarbeitung des Computerprogramms auf dem Rechengerät Fehler detektiert.

Die Erfindung betrifft außerdem ein Betriebssystem, das auf einem Rechengerät, insbesondere auf einem Mikroprozessor, ablauffähig ist.

Schließlich betrifft die vorliegende Erfindung auch ein Rechengerät zum Abarbeiten eines Computerprogramms, das mehrere Programmobjekte umfasst. Das Rechengerät weist einen Fehlerentdeckungsmechanismus zur Detektion eines Fehlers während der Abarbeitung des Computerprogramms auf dem Rechengerät auf.

Bei der Abarbeitung eines Computerprogramms auf einem Rechengerät kann es zu sogenannten transienten Fehlern kommen. Da die Strukturen auf den Halbleiterbausteinen (sog. Chips) immer kleiner, die Taktrate der Signale aber immer größer und die Spannungen der Signale immer niedriger werden, treten transiente Fehler immer häufiger auf. Transiente Fehler treten im Gegensatz zu permanenten Fehlern nur temporär auf und verschwinden nach einiger Zeit in der Regel von selbst wieder. Bei transienten Fehlern sind lediglich einzelne Bits falsch, ohne dass das Rechengerät an sich permanent beschädigt ist. Transiente Fehler können verschiedene Ursachen haben, wie beispielsweise elektromagnetische Einflüsse, Alpha-Partikel oder Neutronen.

Bei Kommunikationssystemen liegt schon heute der Schwerpunkt bei der Fehlerbehandlung auf transienten Fehlern. Bei Kommunikationssystemen (z.B. bei dem Controller Area Network, CAN) ist es bekannt, bei der Detektion eines Fehlers die fehlerhaft übermittelten Daten erneut zu senden. Außerdem ist es bekannt, in Kommunikationssystemen einen Fehlerzähler einzusetzen , der bei Detektion eines Fehlers erhöht wird, bei korrektem Senden erniedrigt wird, und ein Senden von Daten verhindert, sobald er einen bestimmten Wert überschreitet.

Bei Rechengeräten zur Abarbeitung von Computerprogrammen erfolgt eine Fehlerbehandlung jedoch im Wesentlichen nur für permanente Fehler. Eine Berücksichtigung transienter Fehler beschränkt sich auf das Inkrementieren und gegebenenfalls Dekrementieren eines Fehlerzählers. Dieser wird in einem Speicher abgelegt und kann off-line, das heißt beispielsweise bei einem als Kraftfahrzeugsteuergerät ausgebildeten Rechengerät während eines Werkstattaufenthalts, als Diagnose- oder Fehlerinformation ausgelesen werden. Erst dann kann auf den Fehler entsprechend reagiert werden.

Die Fehlerbehandlung mittels Fehlerzähler erlaubt also zum einen keine Fehlerbehandlung innerhalb einer insbesondere für sicherheitsrelevante Systeme erforderlichen kurzen Fehlertoleranzzeit und zum anderen auch keine konstruktive Fehlerbehandlung in dem Sinne, dass innerhalb der Fehlertoleranzzeit das Computerprogramm wieder ordnungsgemäß abgearbeitet wird. Statt dessen wird beim Stand der Technik das Computerprogramm nach dem Überschreiten eines bestimmten Wertes des Fehlerzählers in einen Notlaufbetrieb umgeschaltet. Das bedeutet, dass statt des fehlerbehafteten Teils des Computerprogramms ein anderes abgearbeitet wird und die auf diese Weise ermittelten Ersatzwerte für die weitere Berechnung herangezogen werden. Die Ersatzwerte können beispielsweise anhand anderer Größen modelliert werden. Alternativ können die mit dem fehlerbehafteten Teil des Computerprogramms berechneten Ergebnisse als fehlerhaft verworfen und durch für den Notlaufbetrieb vorgesehene Standardwerte für die weitere Berechnung ersetzt werden. Die bekannten Verfahren zur Behandlung eines transienten Fehlers eines auf einem Rechengerät ablaufenden Computerprogramms erlauben somit keinen systematischen, konstruktiven Umgang mit der transienten Natur der meisten Fehler.

Aus dem Stand der Technik ist es außerdem bekannt, bei der Abarbeitung eines Computerprogramms auf einem Rechengerät auftretende transiente Fehler durch einen kompletten Neustart des Rechengeräts zu beheben. Auch diese Lösung kann nicht wirklich befriedigen, da in dem bisherigen Verlauf der Abarbeitung des Computerprogramms gewonnene Größen verloren gehen und das Rechengerät für die Dauer des Neustarts seine bestimmungsgemäße Funktion nicht erfüllen kann. Dies ist insbesondere bei sicherheitsrelevanten Systemen inakzeptabel.

Schließlich ist es als Fehlerbehandlung für transiente Fehler eines auf einem Rechengerät abgearbeiteten Computerprogramms auch bekannt, das Computerprogramm um einige Takte zurückzusetzen und einzelne Maschinenbefehle des Computerprogramms zu wiederholen. Dieses Verfahren wird auch als Micro Roll-Back bezeichnet. Bei dem bekannten Verfahren wird nur um Objekte auf Maschinenebene (Takte, Maschinenbefehle) zurückgesprungen. Das erfordert eine entsprechende Hardwareunterstützung auf Maschinenebene, was mit einem erheblichen Aufwand im Bereich des Rechengeräts verbunden ist. Eine Ausführung des bekannten Verfahrens rein durch Software gesteuert ist nicht möglich.

Die aus dem Stand der Technik bekannten Fehlerbehandlungsmechanismen können auf transiente Fehler, die bei der Abarbeitung eines Computerprogramms auf einem Rechengerät auftreten, nicht in geeigneter Weise reagieren.

Transiente Fehler sind aber gerade in künftigen Technologien sehr häufig. Falls man sie entdeckt z.B. über Dual Core Mechanismen, dann bleibt somit immer noch die Frage einer Fehlerlokalisierung zu beantworten, um das korrekte Resultat zu identifizieren. Dies gilt um so mehr, wenn man das Ziel hat, dass ein transienter Fehler nicht immer einen Neustart des Rechners zur Folge hat. Eine Fehlerlokalisierung ist typischerweise wie beschrieben nur über vergleichsweise aufwändige Verfahren zu erreichen.

Der vorliegenden Erfindung liegt die Aufgabe zugrunde, beim Auftreten von transienten Fehlern beim Ararbeiten eines Computerprogramms in einem Rechnersystem diese derart konstruktiv zu behandeln, dass die volle Funktionsfähigkeit und die Funktionssicherheit des Rechnersystems innerhalb einer möglichst kurzen Fehlertoleranzzeit wieder hergestellt ist.

Zur Lösung dieser Aufgabe wird ausgehend von dem Verfahren der eingangs genannten Art vorgeschlagen, dass bei einer Detektion eines Fehlers mindestens ein Programmobjekt, das bereits einer Abarbeitung zugeführt wurde, in einen definierten Zustand überführt und aus diesem heraus erneut gestartet wird.

Auf Systemebene stellt sich allerdings weiterhin die Frage, wie man ein solches Konzept der Taskwiederholung vernünftig einsetzen kann. Es ist in der Regel nicht der Fall, dass man ein beliebiges fehlerhaftes Task einfach nochmals rechnen kann, da die zusätzlich benötigte Rechenzeit und auch der dafür verwendete Zeitpunkt aus Systemsicht anders verplant sind. Wenn die Auslastung des Prozessors schon nahe 100% ist (und das ist sie in der Regel), dann wird durch eine solche ungeplante Zusatzlast (wie sie eine Taskwiederholung darstellt) eine Überlastung des Systems erzeugt, die typischerweise zu einem Zusammenbruch führen kann. Noch deutlicher wird dies, wenn man zeitgesteuerte Systeme betrachtet (die sich, wie es sich abzeichnet, zumindest teilweise durchsetzen werden). Bei diesen ist eine Deadlineverletzung nicht tolerierbar, ebenso wenig wie bei den meisten anderen harten Echtzeitkonzepten.

Als Konsequenz ergibt sich, dass man also aus Systemsicht die Zusatzlast, die durch eine potenzielle Taskwiederholung entstehen kann, einplanen muss. Wenn man nach jedem Task die Rechenzeit vorhält, die eine Taskwiederholung benötigt, so ist das sicherlich funktionsfähig, allerdings muss man gegenüber einem nicht-fehlerbehandelnden System 100% zusätzliche Performanz bezahlen. Aus Kostensicht ist das nicht akzeptabel.

Es ist daher weiterhin Aufgabe der Erfindung eine optimale Systemstrategie anzugeben, die das zweifache Rechnen eines Tasks nicht immer einplant (und damit einen permanenten und sehr großen Overhead erzeugt) und gleichzeitig die Frage löst, wie sich das mit zeitgesteuerten Ansätzen verbinden lässt.

Vorteile der Erfindung

Es wird somit ein Verfahren, eine Betriebssystem und eine Rechengerät im Rahmen einer erfindungsgemäßen Systemstrategie vorgeschlagen, die es erlauben, das Konzept der Taskwiederholung mit minimalem oder sogar ohne Performanzoverhead einzubinden. Eine solche Systemstrategie ist dabei eine Randbedingung für Schedulingverfahren von Tasks, Aufgaben, Programmen oder Programmteilen, welche im Weiteren insbesondere als Programmobjekt oder Programmobjekte bezeichnet werden.

Das Programmobjekt, das erneut gestartet wird, muss bei der Detektion des Fehlers nicht vollständig abgearbeitet worden sein. Im Sinne der Erfindung können auch solche Programmobjekte beim Auftreten eines Fehlers erneut gestartet werden, die zum Zeitpunkt der Fehlerdetektion noch nicht vollständig abgearbeitet worden sind, mit deren Abarbeitung aber wohl schon begonnen wurde. Erfindungsgemäß wird also beim Auftreten eines transienten oder eines permanenten Fehlers mindestens ein Betriebssystemobjekt erneut ausgeführt. Die Vorteile gegenüber dem Micro Roll-Back liegen insbesondere darin, dass die Wiederholung eines Programmobjekts mit einer sehr geringen Hardwareunterstützung realisiert werden kann. Es ist höchstens zusätzlicher Speicherplatz erforderlich, um einige für die erneute Ausführung des Programmobjekts erforderliche Informationen (z.B. Eingangsgrößen des Programmobjekts) abspeichern zu können. Die eigentliche Verwaltung des erfindungsgemäßen Verfahrens kann durch das Betriebssystem des Rechengeräts ausgeführt werden. Das heißt, das erfindungsgemäße Verfahren ist mit herkömmlichen, handelsüblichen Prozessoren realisierbar, ohne dass es zusätzlicher Hardware bedarf. Selbstverständlich ist es aber auch möglich, das erfindungsgemäße Verfahren mit Hardwareunterstützung zu realisieren.

Günstiger ist es somit im Vergleich zum Stand der Technik, das fehlerhafte Task, also die fehlerhafte Aufgabe bzw. das fehlerhafte Programm oder Programmteil bzw. Programmobjekt oder mindestens Betriebssystemobjekt nochmals zu rechnen. Falls der Fehler ursprünglich transient war, sind bei einer Neuberechnung die beiden Outputs gleich, der Fehler ist also verschwunden und durch die Neuberechnung behandelt.

Vorteilhafter Weise wird somit ein Verfahren zum Abarbeiten eines Computerprogramms auf einem Rechengerät, insbesondere auf einem Mikroprozessor dargestellt, wobei das

Computerprogramm mehrere Programmobjekte umfasst und bei dem Verfahren während der Abarbeitung des Computerprogramms auf dem Rechengerät Fehler detektiert werden, wobei bei einer Detektion eines Fehlers mindestens ein Programmobjekt, das bereits einer Abarbeitung zugeführt wurde, in einen definierten Zustand überführt und aus diesem heraus erneut gestartet wird, und nachfolgende weitere Programmobjekte verschoben werden.

Zweckmäßiger Weise werden die Programmobjekte in wenigstens zwei Klassen unterteilt, wobei bei einer Detektion eines Fehlers Programmobjekte der ersten Klasse wiederholt werden und bei einer Detektion eines Fehlers in einem Programmobjekt der ersten Klasse, das bereits einer Abarbeitung zugeführt wurde, dieses Programmobjekt der ersten Klasse anstelle eines Programmobjektes der zweiten Klasse erneut gestartet wird. Dabei wird die Detektion eines Fehler nur in Klassen durchgeführt, die erneut gestartet werden.

Zweckmäßiger Weise ist für alle Programmobjekte eine Gesamtrechenzeit in einem Durchlauf vorgesehen und die Gesamtrechenzeit wird derart aufgeteilt, dass die Programmobjekte welche bei Detektion eines Fehlers erneut gestartet werden im fehlerfreien Fall höchstens 50% der Gesamtrechenzeit in einem Durchlauf erhalten, wobei die Programmobjekte der unterschiedlichen Klassen abwechselnd abgearbeitet werden und das fehlerhafte Programmobjekt der ersten Klasse anstelle des direkt nachfolgenden Programmobjektes der zweiten Klasse abgearbeitet bzw. erneut gestartet wird.

Die Fehlerdetektion selbst kann nach einem beliebigen Verfahren erfolgen. Denkbar ist der Einsatz jeder Art von Fehlerentdeckungsmechanismus, der Fehler während der Abarbeitung des Computerprogramms (sog. concurrent checking) detektieren kann. Beispielsweise bei einer Dual Core Architektur ist der ganze Rechnerkern zweifach ausgebildet. Wenn die Rechnerkerne in einem Lockstep-Modus betrieben werden, kann für jede Instruktion verglichen werden, ob beide Rechnerkerne die gleichen Ergebnisse liefern. Ein Unterschied der Ergebnisse lässt dann mit Sicherheit auf einen Fehler schließen. Dieser Fehlerentdeckungsmechanismus entdeckt also Fehler bei der Abarbeitung der Programmobjekte in Echtzeit. Entsprechendes gilt auch für fehlerentdeckende Codes, die in der Prozessorarchitektur durchgängig verwendet werden, oder auch für duplizierte Teilkomponenten des Rechengeräts. All diesen

Fehlerentdeckungsmechanismen ist gemeinsam, dass sie transiente Fehler sehr schnell entdecken und ein Fehlersignal liefern, wenn ein Fehler detektiert wurde.

Auf ein solches Fehlersignal hin kann ein Fehlerbehandlungsmechanismus angestoßen werden, der das Programmobjekt wiederholt. Falls bei der erneuten Ausführung der gleiche Fehler nochmals auftritt, kann auf einen permanenten Fehler geschlossen werden, oder ein Fehlerzähler erhöht werden, wobei erst bei dessen Überschreiten eines bestimmten Wertes auf einen permanenten Fehler geschlossen wird. Wenn der Fehler dagegen bei der erneuten Ausführung des Programmobjekts nicht mehr auftritt, kann davon ausgegangen werden, dass der Fehler ein transienter Fehler war. Noch während der fehlerfreien erneuten Ausführung des Programmobjekts steht das Computerprogramm wieder für seine bestimmungsgemäße Funktion bereit. Die Verfügbarkeit ist also bereits nach kürzester Zeit wieder hergestellt. Eine Wiederholung mindestens eines Programmobjekts ist somit ein gutes Mittel, transiente Fehler zu behandeln.

Gemäß einer vorteilhaften Weiterbildung der Erfindung wird vorgeschlagen, dass die Programmobjekte als Laufzeitobjekte des Computerprogramms ausgebildet sind (im Folgenden speziell Tasks genannt) und bei der Detektion eines Fehlers mindestens eine Task erneut ausgeführt wird. Bei einer Task handelt es sich um ein typisches Objekt insbesondere auf Betriebssystemebene. Die Wiederholung einer Task kann mit minimalem Aufwand, falls gewünscht sogar rein softwaremäßig gesteuert, realisiert werden.

Gemäß einer bevorzugten Ausführungsform der Erfindung wird vorgeschlagen, dass ein zum Zeitpunkt der Detektion des Fehlers ausgeführtes Programmobjekt erneut gestartet wird. Alternativ oder zusätzlich können aber auch Programmobjekte gestartet und erneut abgearbeitet werden, die zum Zeitpunkt der Detektion des Fehlers bereits vollständig abgearbeitet waren.

Es wird vorgeschlagen, dass während der Abarbeitung der Programmobjekte, insbesondere zu Beginn der Abarbeitung der Programmobjekte, mindestens ein definierter Zustand der Programmobjekte erzeugt und abgespeichert wird. Dies kann bspw. dadurch erfolgen, dass die Werte aller für den Zustand des Programmobjekts relevanten Größen abgespeichert werden.

Des weiteren wird vorgeschlagen, dass zur Fehlerdetektion ein zu dem Rechengerät, auf dem das Computerprogramm mit den mehreren Programmobjekten abgearbeitet wird, redundant arbeitendes weiteres Rechengerät verwendet wird. Selbstverständlich kann auch mehr als ein redundantes Rechengerät zur Fehlerdetektion eingesetzt werden.

Vorteilhafterweise wird das erfindungsgemäße Verfahren in einem Kraftfahrzeug, insbesondere in einem Kraftfahrzeugsteuergerät, verwendet, um trotz unvermeidbarer transienter Fehler beim Abarbeiten eines Computerprogramms eine sichere und zuverlässige Abarbeitung des Computerprogramms zu gewährleisten. Das ist vor allem für die Abarbeitung von Steuer- und/oder Regelungsprogrammen in sicherheitskritischen Anwendungen in einem Kraftfahrzeug von Bedeutung.

Es wird ferner vorgeschlagen, dass auf einen permanenten Fehler geschlossen wird, falls der gleiche Fehler bei der erneuten Ausführung des mindestens einen Programmobjekts erneut auftritt. Es ist auch denkbar, dass erst dann auf einen permanenten Fehler geschlossen wird, wenn der Fehler nach einer vorgebbaren Anzahl an Wiederholungen des Programmobjekts noch immer auftritt. In diesem Fall wird also selbst dann noch auf einen transienten Fehler geschlossen, wenn dieser erst nach einer dritten oder einer noch späteren Wiederholung des Programmobjekts wegfällt. Durch diese Weiterbildung der Erfindung können wichtige Programmobjekte bspw. 3-mal statt nur 2-mal wiederholt werden.

Gemäß einer anderen vorteilhaften Weiterbildung der Erfindung wird vorgeschlagen, dass die Anzahl der Wiederholungen des mindestens einen Programmobjekts auf einen vorgebbaren Wert begrenzt wird. Dadurch wird verhindert, dass bei einem permanenten Fehler das gleiche Programmobjekt beliebig oft wiederholt wird. Die Beschränkung der Anzahl der Wiederholungen des mindestens einen Programmobjekts kann bspw. mittels eines Zählers oder über Zeitschranken erfolgen. Durch die Vorgabe des taskabhängigen Wiederholwerts wird außerdem ermöglicht, wichtige Tasks öfter zu wiederholen als weniger wichtige, und somit wichtigen Tasks öfter bzw. länger die Möglichkeit zu geben, ohne transiente Fehler fehlerfrei abzulaufen, während bei weniger wichtigen Tasks relativ schnell auf einen permanenten Fehler geschlossen und eine andere Systemreaktion eingeleitet wird.

Gemäß einer weiteren bevorzugten Ausführungsform der Erfindung wird vorgeschlagen, dass die Anzahl der Wiederholungen des mindestens einen Programmobjekts auf einen vorgebbaren Wert dynamisch begrenzt wird. Vorteilhafterweise wird die Anzahl der Wiederholungen des mindestens einen Programmobjekts in Abhängigkeit von einer verbleibenden Restzeit für ein Scheduling auf einen vorgebbaren Wert dynamisch begrenzt. Auf diese Weise können bspw. eine erste Task und eine zweite Task durchlaufen, während eine dritte Task mehrmals wiederholt werden kann.

Zur Realisierung des erfindungsgemäßen Verfahrens wird vorgeschlagen, dass während der Abarbeitung des Computerprogramms vor der Ausführung eines Programmobjekts die Werte der zur Ausführung des Programmobjekts notwendigen bzw. den Zustand des Programmobjekts definierenden Größen gespeichert werden. Gemäß dieser Ausführungsform werden also die Größen aller Programmobjekte abgespeichert.

Alternativ wird vorgeschlagen, dass bei einem in einer Periode periodisch abzuarbeitenden Computerprogramm bei einer Detektion eines Fehlers auf ein bestimmtes Programmobjekt an einem vorgebbaren Rücksprungpunkt in der Periode des Computerprogramms zurückgesprungen wird. Gemäß dieser Ausführungsform wird also bei einem Fehler stets an die gleiche Stelle innerhalb der Periode gesprungen. Vorzugsweise werden dann während der Abarbeitung des Computerprogramms nur vor der Ausführung eines Programmobjekts an dem Rücksprungpunkt die Werte von allen für den Zustand des Programmobjekts relevanten Größen gespeichert. Es müssen also nur einmal pro Zyklus oder Periode nur die Werte der relevanten Größen des Programmobjekts an dem Rücksprungpunkt abgespeichert werden. Dadurch kann Zeit für die Speicherung und Speicherplatz eingespart werden.

Bei einer erneuten Ausführung eines Programmobjekts nach der Detektion eines Fehlers werden dann die abgespeicherten Eingangsgrößen aufgerufen und dem erneut auszuführenden Programmobjekt als Eingangsgrößen zur Verfügung gestellt.

Als eine weitere Ausführungsform der Erfindung wird vorgeschlagen, dass zu einem Programmobjekt mehrere Rücksprungpunkte angelegt werden. Beim Auftreten eines Fehlers muss

nicht das gesamte Programmobjekt, sondern nur ein Teil des Programmobjekts erneut abgearbeitet werden. Beim Auftreten eines Fehlers wird einfach auf denjenigen vorangegangenen Rücksprungpunkt gesprungen, bis zu dem die Abarbeitung des Programmobjekts fehlerfrei war. Zum Beispiel kann bei einem fehlerfreien Abarbeiten des Programmobjekts bis zum n-ten Rücksprungpunkt beim Auftreten eines Fehlers zwischen diesem und dem (n+1)-ten Rücksprungpunkt auf den n-ten Rücksprungpunkt zurückgesprungen werden. Das Programmobjekt wird dann ab dem n-ten Rücksprungpunkt erneut abgearbeitet. Damit ist eine Zeitersparnis möglich. Vorzugsweise wird beim Überschreiten eines jeden Rücksprungpunktes während der Abarbeitung des Programmobjekts jeweils mindestens ein definierter Zustand erzeugt und abgespeichert.

Von besonderer Bedeutung ist die Realisierung des erfindungsgemäßen Verfahrens in der Form eines Betriebssystems. Dabei ist das Betriebssystem auf einem Rechengerät, insbesondere auf einem Mikroprozessor, ablauffähig und zur Ausführung des erfindungsgemäßen Verfahrens programmiert, wenn es auf dem Rechengerät abläuft. In diesem Fall wird also die Erfindung durch das Betriebssystem realisiert, so dass dieses Betriebssystem in gleicher Weise die Erfindung darstellt wie das Verfahren, zu dessen Ausführung das Betriebssystem geeignet ist. Das Betriebssystem ist vorzugsweise auf einem Speicherelement abgespeichert und wird zur Abarbeitung an das Rechengerät übermittelt. Als Speicherelement kann insbesondere ein beliebiger Datenträger oder ein elektrisches Speichermedium zur Anwendung kommen, beispielsweise ein Random-Access-Memory (RAM), ein Read-Only-Memory (ROM) oder ein Flash-Memory.

Als eine weitere Lösung der Aufgabe der vorliegenden Erfindung wird ausgehend von dem Rechengerät der eingangs genannten Art vorgeschlagen, dass das Rechengerät einen Fehlerbehandlungsmechanismus aufweist, der bei einer Detektion eines Fehlers durch den Fehlerentdeckungsmechanismus ein erneutes Ausführen mindestens eines Programmobjekts veranlasst.

Gemäß einer vorteilhaften Weiterbildung der Erfindung wird vorgeschlagen, dass der Fehlerbehandlungsmechanismus eine Triggerlogik aufweist, welche bei einer Detektion eines Fehlers das mindestens eine Programmobjekt neu startet.

Gemäß einer bevorzugten Ausführungsform wird vorgeschlagen, dass auf dem Rechengerät ein Echtzeit-Betriebssystem, bspw. OSEK time, abläuft. Schließlich wird vorgeschlagen, dass das Rechengerät einen Mikroprozessor umfasst.

Zeichnungen

Weitere Merkmale, Anwendungsmöglichkeiten und Vorteile der Erfindung ergeben sich aus der nachfolgenden Beschreibung von Ausführungsbeispielen der Erfindung, die in der Zeichnung dargestellt sind. Dabei bilden alle beschriebenen oder dargestellten Merkmale für sich oder in beliebiger Kombination den Gegenstand der Erfindung, unabhängig von ihrer Zusammenfassung in den Patentansprüchen oder deren Rückbeziehung sowie unabhängig von ihrer Formulierung beziehungsweise Darstellung in der Beschreibung, beziehungsweise in der Zeichnung. Es zeigen:

Figur 1 Ein Ablaufdiagramm eines erfindungsgemäßen Verfahrens gemäß seiner bevorzugten Ausführungsform; und

Figur 2 ein erfindungsgemäßes Rechengerät gemäß seiner bevorzugten Ausführungsform in einer schematischen Darstellung, und

Figur 3 bestehend aus den Figuren 3a und 3b einen ersten erfindungsgemäßen Ansatz zur Einbindung der Taskwiederholung, und

Figur 4 bestehend aus den Figuren 4a und 4b einen zweiten erfindungsgemäßen Ansatz zur Einbindung der Taskwiederholung, und

Figur 5 bestehend aus den Figuren 5a und 5b einen dritten erfindungsgemäßen Ansatz zur Einbindung der Taskwiederholung.

Beschreibung der Ausführungsbeispiele

Die vorliegende Erfindung betrifft ein Verfahren zum Abarbeiten eines Computerprogramms auf einem Rechengerät, insbesondere auf einem Mikroprozessor. Das Computerprogramm umfasst mehrere Programmobjekte, die vorzugsweise als Tasks ausgebildet sind. Bei dem Verfahren werden während der Abarbeitung des Computerprogramms auf dem Rechengerät Fehler detektiert. Die detektierten Fehler können transienter (also vorübergehender) oder permanenter Art sein.

Die transienten Fehler können bei der Abarbeitung eines Computerprogramms auf einem Rechengerät auftreten. Da die Strukturen auf den Halbleiterbausteinen (sogenannten Chips) der Rechengeräte immer kleiner, die Taktrate der Signale aber immer größer und die Spannungen der Signale immer niedriger werden, treten bei der Abarbeitung eines Computerprogramms auf einem Rechengerät immer häufiger transiente Fehler auf. Im Gegensatz zu permanenten Fehlern treten sie nur temporär auf und verschwinden nach einiger Zeit in der Regel von selbst wieder. Bei transienten Fehlern sind lediglich einzelne Bits falsch, ohne dass das Rechengerät an sich permanent beschädigt ist. Transiente Fehler können verschiedene Ursachen haben, wie beispielsweise elektromagnetische Einflüsse, Alpha-Partikel oder Neutronen.

Aufgrund der Tatsache, dass transiente Fehler nahezu unvorhersehbar auftreten und deshalb nicht reproduzierbar sind, erfolgt bei aus dem Stand der Technik bekannten Rechengeräten eine Fehlerbehandlung im wesentlichen nur für permanente Fehler. Eine Berücksichtigung transienter Fehler beschränkt sich auf das Inkrementieren und gegebenenfalls Dekrementieren eines Fehlerzählers. Dieser wird in einem Speicher abgelegt und kann off-line, das heißt beispielsweise während eines Werkstattaufenthalts, als Diagnose- oder Fehlerinformation ausgelesen werden. Erst dann kann auf den Fehler entsprechend reagiert werden. Die bekannte Fehlerbehandlung erlaubt also keine Fehlerbehandlung innerhalb einer insbesondere für sicherheitsrelevante Systeme

erforderlichen kurzen Fehlertoleranzzeit und zum anderen auch keine konstruktive Fehlerbehandlung in dem Sinne, dass innerhalb der Fehlertoleranzzeit das Computerprogramm wieder ordnungsgemäß abgearbeitet wird und das Rechengerät seine bestimmungsgemäße Aufgabe erfüllen kann.

Im Gegensatz dazu erlaubt das erfindungsgemäße Verfahren eine Behandlung eines transienten Fehlers eines auf einem Rechengerät ablaufenden Computerprogramms mit einem systematischen, konstruktiven Umgang mit der transienten Natur der meisten Fehler. Ein Ablaufdiagramm des erfindungsgemäßen Verfahrens am Beispiel eines Laufzeitobjekts, einer sogenannten Task, ist in Fig.1 dargestellt. Die Existenz weiterer Tasks beeinflusst den prinzipiellen Ablauf nicht, eine Berücksichtigung entfällt also. So wie gemäß dem in Fig. 1 dargestellten Ablauf eine Task behandelt wird, können erfindungsgemäß also auch mehrere Tasks behandelt werden. Besonders vorteilhaft ist ein parallel arbeitender Fehlerentdeckungsmechanismus (sog. concurrent checking). Dieser ist in einem Ablaufdiagramm so aber nicht darstellbar, er ist an der entsprechenden Stelle als serieller Baustein eingefügt.

Das erfindungsgemäße Verfahren beginnt in einem Funktionsblock 1. In dem Funktionsblock 1 wird mit der Abarbeitung der Task auf dem Rechengerät begonnen; die Task wird aufgerufen. In einem Funktionsblock 2 wird ein Rücksprungpunkt erzeugt. Zu diesem Zweck werden sichere relevante Taskeingangsgrößen, die ausreichen, die Task in eine definierten Zustand für einen Neustart zu versetzen und die Task nochmals zu starten, in einem Speicherelement des Rechengerätes abgespeichert. Vorzugsweise werden alle Eingangsgrößen der Task abgespeichert. In einem Funktionsblock 3 wird dann die Task weiter abgearbeitet. Die Abarbeitung kann entweder zu einem weiteren Rücksprungpunkt oder bis zum Ende der Task erfolgen. Dann wird ein Fehlerentdeckungsmechanismus ausgeführt. Die Fehlerdetektion kann nach einem beliebigen Verfahren erfolgen. Die Fehler werden während der Abarbeitung des Computerprogramms detektiert (sogenanntes concurrent checking). So ist beispielsweise bei einer sogenannten Dual-Core-Architektur der ganze Rechnerkern zweifach ausgebildet. Wenn die Rechnerkerne in einem sogenannten Lockstep-Modus betrieben werden, kann für jede Instruktion verglichen werden, ob beide Rechnerkerne die gleichen Ergebnisse liefern. Ein Unterschied der Ergebnisse lässt dann mit Sicherheit auf einen Fehler schließen. Ein solcher Fehlerentdeckungsmechanismus entdeckt also

Fehler bei der Abarbeitung der Task in Echtzeit. Entsprechendes gilt auch für fehlerentdeckende Codes, die in der Prozessorarchitektur durchgängig verwendet werden oder auch für duplizierte Teilkomponenten des Rechengerätes. Vorzugsweise werden solche Fehlerentdeckungsmechanismen eingesetzt, die transiente Fehler sehr schnell entdecken und ein Fehlersignal liefern, wenn ein Fehler detektiert wurde.

In einem Abfrageblock 4 wird überprüft, ob ein Fehler, also ein transienter oder ein permanenter Fehler, entdeckt wurde. Falls ein Fehler entdeckt wurde, wird in einen weiteren Abfrageblock 7 verzweigt, wo der aktuelle Wert einer Fehlerzählerlogik überprüft wird. Falls der Fehlerzähler einen vorgebbaren Zählerstand noch nicht unterschritten (bei einem dekrementierenden Fehlerzähler) oder überschritten (bei einem inkrementierenden Fehlerzähler) hat, kann die Task, während deren Abarbeitung der Fehler aufgetreten ist, bzw. können eine bestimmte Anzahl an Tasks, die vor dem Auftreten des Fehlers abgearbeitet wurden, noch einmal ausgeführt werden. Falls ein erneutes Starten der Abarbeitung der Task möglich ist, wird in einem Funktionsblock 8 verzweigt, in dem der Status der Fehlerzählerlogik mit der Information, dass ein weiterer Fehler aufgetreten ist, aktualisiert (dekrementiert oder inkrementiert) wird. Von dort wird in einen Funktionsblock 5 verzweigt, in welchem die in dem Funktionsblock 2 abgespeicherten Größen geladen und der Task zum Erzeugen eines definierten Zustandes zu Beginn der Abarbeitung zugeführt werden. Dann wird zu dem Funktionsblock 3 verzweigt, wo die zu wiederholende Task teilweise, das heißt beispielsweise von einem bereits abgearbeiteten Rücksprungpunkt aus, oder aber als ganzes, das heißt die Task wird von Beginn an noch einmal gestartet, noch einmal abgearbeitet wird.

Falls sich in dem Abfrageblock 4 ergibt, dass während der Abarbeitung der Task in dem Funktionsblock 3 kein Fehler aufgetreten ist, wird in einem Funktionsblock 9 verzweigt, in welchem der Status der Fehlerzählerlogik mit der Information aktualisiert wird, dass kein Fehler detektiert worden ist. Von dort aus wird in einen Abfrageblock 11 verzweigt, wo überprüft wird, ob das Computerprogramm zu Ende abgearbeitet ist. Falls dem so ist, wird zum Ende des Computerprogramms in Funktionsblock 6 verzweigt. Anderenfalls wird in einen Funktionsblock 12 verzweigt, wo entsprechend dem aktuellen Taskstatus ein weiterer Rücksprungpunkt erzeugt wird, indem sichere relevante Taskeingangsgrößen, die ausreichen, um die Task nochmals zu

starten, definiert und abgespeichert werden. Von dort aus wird dann wieder zu dem Funktionsblock 3 verzweigt, wo die zu wiederholende Task erneut gestartet und entweder teilweise oder als ganzes noch einmal abgearbeitet wird.

Falls sich in dem Abfrageblock 7 ergibt, dass aufgrund des Standes der Fehlerzählerlogik ein weiterer Versuch zur erneuten Abarbeitung der Task nicht mehr möglich ist, wird in einen Funktionsblock 10 verzweigt. In dem Abfrageblock 7 wird überprüft, ob der Wert der Fehlerzählerlogik für diese Task größer als ein taskabhängiger Wiederholwert ist. Dieser taskabhängige Wiederholwert kann entweder für verschiedene Tasks gleich oder aber individuell für jede Task einzeln vorgegeben werden. Auf diese Weise ist es möglich, dass beispielsweise besonders wichtige Tasks zunächst mehrmals wiederholt werden, bevor ein permanenter Fehler gemeldet wird. Wenn der taskabhängige Wiederholwert als 1 vorgegeben wird, wird die Task nur einmal wiederholt, bevor ein permanenter Fehler detektiert wird. Falls der taskabhängige Wiederholwert auf 2 oder 3 vorgegeben wird, wird die Task 2 oder 3-mal wiederholt, bevor ein permanenter Fehler detektiert wird. Die Task hat in diesem Fall also eine längere Zeit, beziehungsweise mehr Durchläufe zur Verfügung, bis der transiente Fehler nicht mehr auftritt. In dem Funktionsblock 10 wird ein permanenter Fehler detektiert und eine entsprechende Maßnahme eingeleitet. Diese Maßnahme kann beispielsweise darin bestehen, das Computerprogramm in einen Notlaufbetrieb zu überführen oder zunächst nichts zu unternehmen und den Ablauf des Computerprogramms dann zu beenden.

Das erfindungsgemäße Verfahren muss nicht notwendiger Weise alle in Fig. 1 dargestellten und oben erläuterten Funktions- und Abfrageblöcke umfassen. So kann bspw. auf die Blöcke 7 bis 9 verzichtet werden, welche die Fehlerzählerlogik betreffen. Bei der Detektion eines Fehlers würde(n) die erneut zu startende(n) und auszuführende(n) Task(s) so lange wiederholt werden, bis der Fehler nicht mehr auftritt. Ein permanenter Fehler würde nicht detektiert werden, so dass auch Funktionsblock 10 wegfallen könnte. Alternativ kann der taskabhängige Wiederholwert als 1 vorgegeben werden, so dass die Funktionsblöcke 8 und 9 zum Aktualisieren des Fehlerzählers entfallen könnten. Schließlich wäre es auch möglich, auf die Blöcke 11 und 12 zu verzichten, wenn nur eine einzige Task mit einem einzigen Rücksprungpunkt ausgeführt wird.

In Fig. 2 ist ein erfindungsgemäßes Rechengerät zum Abarbeiten eines Computerprogramms gemäß seiner bevorzugten Ausführungsform dargestellt. Das Rechengerät ist in seiner Gesamtheit mit dem Bezugszeichen 20 bezeichnet. Das Rechengerät umfasst ein Speicherelement 21, das beispielsweise als ein elektronischer Speicher, insbesondere als ein Flash-Memory, ausgebildet ist. Außerdem umfasst das Rechengerät 20 einen Mikroprozessor 22, auf dem ein Computerprogramm abgearbeitet werden kann. Das Computerprogramm ist auf dem elektronischen Speichermedium 21 abgespeichert und mit dem Bezugszeichen 23 bezeichnet. Zur Abarbeitung des Computerprogramms auf dem Mikroprozessor 22 wird das Computerprogramm entweder als Ganzes oder abschnittsweise, beispielsweise befehlsweise, über eine Datenverbindung 24 an den Mikroprozessor 22 übertragen. Die Datenverbindung 24 kann als eine oder mehrere Datenleitungen oder als ein Bus-System zur Datenübertragung ausgebildet sein. Auf dem Speichermedium 21 ist außerdem ein Betriebssystem abgespeichert, dass beim Hochfahren des Rechengerätes 20 zumindest teilweise aus dem Speicher 21 an den Mikroprozessor 22 übertragen und dort ausgeführt wird. Das Betriebssystem ist mit dem Bezugszeichen 25 bezeichnet. Es hat die Aufgabe, die Abarbeitung des Computerprogramms 23 auf den Mikroprozessor 22 und an das Rechengerät 20 angeschlossene Peripheriegeräte zu steuern und zu verwalten. Gemäß der vorliegenden Erfindung ist das Betriebssystem 25 in besonderer Weise ausgebildet, so dass es zur Ausführung des erfindungsgemäßen Verfahrens programmiert ist und das erfindungsgemäße Verfahren ausführt, wenn es auf dem Mikroprozessor 22 abläuft. Insbesondere umfasst das Betriebssystem 25 einen Zugang zu einem Fehlerentdeckungsmechanismus zur Detektion eines Fehlers während der Abarbeitung des Computerprogramms 23 auf dem Mikroprozessor 22. Außerdem umfasst das Betriebssystem 25 einen Fehlerbehandlungsmechanismus, der bei einer Detektion eines Fehlers ein erneutes Ausführen mindestens eines Programmobjektes (einer Task) des Computerprogramms 23 veranlasst.

Es wird somit ein Verfahren, eine Betriebssystem und eine Rechengerät im Rahmen einer erfindungsgemäßen Systemstrategie vorgeschlagen, die es erlauben, dieses Konzept der Taskwiederholung mit minimalem oder sogar ohne Performanzoverhead einzubinden.

Es wird dabei eine Systemstrategie zugrunde gelegt, die bei jeweils unterschiedlichen Voraussetzungen den Performanzoverhead und damit die Kosten minimiert. Generell wird dabei vorausgesetzt, dass ein insbesondere bereits beschriebener Fehlerentdeckungsmechanismus zur Verfügung steht, der Fehler beim Ablauf eines Tasks entdecken kann (z.B. ein Dual Core Mechanismus, mit redundanter Verarbeitung). Weiter wird hier speziell auf transiente Fehler eingegangen. Um permanente Fehler abzudecken, ist eine Erweiterung notwendig z.B. ein Fehlerzähler wie bereits erwähnt.

Bei der erfindungsgemäßen Strategie zur Einbindung der Taskwiederholung, wie diese in Figur 3 beschrieben wird, sind einige Voraussetzungen zu beachten:

Es werden mindestens zwei Klassen von Tasks (z.B. kritsche und unkritische Tasks) unterschieden. Dabei werden bereits die Fehlerentdeckungsmechanismen nicht für alle Klassen von Tasks eingesetzt und/oder die Taskwiederholung wird nicht für alle Klassen von Tasks durchgeführt.

Im Beispiel von Figur 3 werden zwei Klassen von Tasks unterschieden, bei denen nur bei einer Klasse eine Taskwiederholung durchgeführt wird und/oder nur bei einer Klasse ein Fehlerentdeckungsmechanismus gestartet wird. Bei einer Unterscheidung in kritische und unkritische Tasks, werden eben nur Fehler in den kritischen Tasks erfindungsgemäß abgefangen, insbesondere wird nur für die erste Klasse der Fehlerentdeckungsmechanismus eingesetzt. Kritische Tasks sind dabei solche bei denen eine korrekte Abarbeitung für die Gesamtfunktion oder grundlegende Funktionalität des Systems erforderlich ist und spätestens zu einem bestimmten Zeitpunkt erfolgt sein muss um diese Funktion zu erhalten. Unkritische Tasks sind solche bei denen die Gesamtsystemfunktion oder auch die grundlegende Funktion entweder nicht betroffen oder nicht wesentlich eingeschränkt wird. Insbesondere wird bei einem System dabei zwischen sicherheitskritischen und sicherheitsunkritischen Funktionen unterschieden.

Transiente Fehler in einem Task der zweiten, also hier unkritischen Klasse 2 können ignoriert werden. Weiter gilt wie ausgeführt, dass ein Task der zweiten Klasse auch mal nicht „drankommen darf“, d.h. dass es aus Systemsicht keine gravierenden Folgen hat, wenn ein Task

dieser zweiten Klasse in einer Taskverarbeitungsrunde nicht aufgerufen wird. Darüber hinaus sollte die gesamte Laufzeit von Tasks der ersten, kritischen Klasse 1 vorzugsweise nicht mehr als einen bestimmten systembedingten Prozentsatz (z.B. 50%) der Gesamtrechenzeit eines Durchlaufs bzw. einer Taskverarbeitungrunde beanspruchen. Dabei sind den kritischen Tasks bei einer Zweiteilung in kritische und unkritische insgesamt höchstens 50% der Gesamtrechenzeit zugewiesen, so dass im worst case wenn alle kritischen Tasks fehlerhaft sind diese erneut gestartet bzw. gerechnet werden können.

Dann ist ein Systemansatz gemäß Figur 3 möglich, bei dem sich Tasks der verschiedenen Klassen abwechseln so, dass der „Nachfolger" S1 (Klasse 2) eines Tasks T1 (Klasse 1) im Zeitplan mindestens ebensoviel Zeit erhält, wie die WCET (worst case execution time) von T1.

Grundidee ist dann, dass bei Auftreten eines transienten Fehlers in T1 an Stelle von S1 nochmals T1 gerechnet wird. Damit ist sichergestellt, dass der Fehler in T1 vor der Berechnung von T2 behoben ist. Fehlererkennung und –behandlung (einschließlich einer sehr wahrscheinlichen Heilung unter Beibehaltung der Fehlertoleranzeigenschaften) finden damit innerhalb der Rechenzeit eines Tasks statt.

In Figur 3 wird dazu zwischen Klasse 1 Tasks T1, T2 und T3 und Klasse 2 Tasks S1, S2 und S3 unterschieden. Bei Fehlerfreier Ausführung ist in Figur 3a die Taskabfolge T1, S1, T2, S2, T3, S3 beispielhaft dargestellt. Bei Auftreten eines Transienten Fehlers während T1 wird gemäß Figur 3b T1 in der S1 Zeit nochmals gerechnet und anschließen T2, S2, T3, S3, so dass der tarnsiente Fehler korrigiert wird.
Ein weiterer Systemansatz zur Einbindung der Taskwiederholung ist in Figur 4, bestehend aus den Figuren 4a und 4b dargestellt. Dabei sollte das System mit kleineren Jittern (in der Größenordnung einer Taskrechenzeit) zurecht kommen. Dies sollte sich vorzugsweise durch Anwendung und Plattformmechanismen (vor allem wohl Betriebssystem) hindurchziehen. In Figur 4a ist dabei ein Durchlauf mit den Tasks T1, T2, T3, T4, T5 und S bei fehlerfreier Ausführung dargestellt.

Wieder muss es mindestens zwei Klassen von Tasks (z.B. wie vorher kritsche und unkritische) geben bei denen hier im Beispiel nur bei einer Klasse eine Taskwiederholung durchgeführt wird und/oder nur bei einer Klasse ein Fehlerentdeckungsmechanismus gestartet wird. Bei einer Unterscheidung in kritische und unkritische Tasks, werden eben nur Fehler in den kritischen Tasks erfindungsgemäß abgefangen, insbesondere wird nur für die erste Klasse der Fehlerentdeckungsmechanismus eingesetzt. Dies gilt wie bei Figur 3.

So können auch hier transiente Fehler in einem Task der zweiten Klasse ignoriert werden. Weiter gilt auch hier, dass ein Task der zweiten Klasse auch mal nicht „drankommen darf", d.h. dass es aus Systemsicht keine gravierenden Folgen hat, wenn ein Task dieser Klasse in einer Runde nicht aufgerufen wird. Schließlich muss es mindestens ein unkritisches Task pro Durchlauf oder Zyklus, also Taskrechenrunde geben, das so lange dauert wie das längste der kritischen Tasks. Diese wenigstens eine unkritische Task kann auch als idle Zeit realisiert werden, was natürlich auch für das Beispiel nach Figur 3 gilt.

Im Beispiel wird nun angenommen, dass es gibt in einem Durchlauf oder Zyklus die Tasks Ti von T1,..bis .,Tn (mit n als natürlicher Zahl, hier 5) gibt, die alle kritisch sind und die im wesentlichen in dieser Reihenfolge abgearbeitet werden sollen. Weiter gibt es noch mindestens ein Task S nach Tn, das unkritisch und länger als jedes einzelne Ti ist. In Figur 4b ist nun in Task T3 ein transienter Fehler aufgetreten. Bei Detektion eines Fehlers in Ti wird das Task Ti einfach sofort wiederholt und der Rest, also T4 und T5 nach hinten geschoben, entsprechend Figur 4b. So lange nur ein transienter Fehler im Durchlauf auftritt fällt keines der kritischen Tasks unter den Tisch und keines wird in der Ausführung um mehr als die maximale Tasklänge nach hinten geschoben. Ob dabei bei mehreren Fehlern kritische Tasks nicht ausgeführt werden hängt somit direkt von der Länge der unkritischen Task bzw. der idle Zeit S ab. Aus Sicherheitsgründen kann auch hier die gesamte Laufzeit von Tasks der ersten, kritischen Klasse 1 so gewählt werden, dass diese vorzugsweise nicht mehr als einen bestimmten systembedingten Prozentsatz (z.B. 50%) der Gesamtrechenzeit eines Durchlaufs bzw. einer Taskverarbeitungrunde beanspruchen. Dabei sind den kritischen Tasks bei einer Zweiteilung in kritische und unkritische auch hier aus sicherheitsaspekten insgesamt höchstens 50% der Gesamtrechenzeit zugewiesen, so dass im worst

case wenn alle kritischen Tasks fehlerhaft sind diese erneut gestartet bzw. gerechnet werden können.

Figur 5 bestehend aus Figur 5a und 5b zeigt einen weiteren Ansatz, bei welchem im wesentlichen übereinstimmende Voraussetzungen gelten. Hier kommt das System mit etwas größeren Jittern (in der Größenordnung eines Durchlaufs) zurecht. Dies muss sich durch Anwendung und Plattformmechanismen, insbesondere das Betriebssystem hindurchziehen. Wieder gibt es mindestens zwei Klassen von Tasks (z.B. kritsche T1-T5 und unkritische S). Auch hier wird nur für die erste Klasse (T1-T5) der Fehlerentdeckungsmechanismus eingesetzt. Transiente Fehler in einem Task der zweiten Klasse können auch hier ignoriert werden. Weiter muss gelten, dass ein Task der zweiten Klasse auch mal nicht „drankommen darf", d.h. dass es aus Systemsicht keine gravierenden Folgen hat, wenn ein Task dieser Klasse in einer Runde nicht aufgerufen wird. Schließlich muss es mindestens ein unkritisches Task pro Durchlauf oder Zyklus geben, das so lange dauert wie das längste der kritischen Tasks (kann auch hier als idle time realisiert werden). (Überlegungen wie oben bei Figur 3 und 4).

Angenommen, es gibt in einem Durchlauf die Tasks T1,...,Tn (alle kritisch), die in diesem Durchlauf abgearbeitet werden sollen. Weiter gebe es noch mindestens ein Task S nach Tn, das unkritisch und länger als jedes einzelne Ti ist. In Figur 5a ist dazu der Ablauf ohne fehlerhafte Task dargestellt mit der Reihenfolge T1, T2, T3, T4, T5 und S für einen Durchlauf.

Bei Detektion eines Fehlers in Ti, wie z.B. entsprechend Figur 5b in T3, wird nun das Ergebnis von Ti (T3) als irrelevant erklärt und im Gegensatz zu den vorherigen Beispielen werden die anderen Tasks (T4, T5) abgearbeitet. Nach dem Tn (T5) erledigt ist, wird Ti (T3) nochmals gerechnet. Voraussetzung ist dabei ein Systemdesign, das es nicht fordert, dass die Ergebnisse von Tasks einer Runde von anderen Tasks der selben Runde noch (zwingend) benötigt werden, da hierbei die Reihenfolge der Tasks nicht eingehalten wird.

Diese Systemansätze werden entsprechend der Ausführungen zu den Figuren 1 und 2 mit der entsprechenden Vorrichtung ausgeführt, so dass bezüglich der Offenbarung ausdrücklich jede Ausführungsform mit jeder anderen erfindungsgemäß kombinierbar ist.

Dadurch ist erfindungsgemäß eine optimale FO-Eigenschaft (FO: Fail Operation(al), Fault Operation(al)) bzgl. transienter Fehler, sogar mit einer sehr kurzen Heilungszeit, die die FO-Eigenschaft selbst wieder heilt erzielbar. Dieser Ansatz ist auch sehr gut in zeitgesteuerten Systemen einsetzbar und dahingehend optimierbar.

Ansprüche

1. Verfahren zum Abarbeiten eines Computerprogramms (23) auf einem Rechengerät (20), insbesondere auf einem Mikroprozessor (22), wobei das Computerprogramm (23) mehrere Programmobjekte umfasst und bei dem Verfahren während der Abarbeitung des Computerprogramms (23) auf dem Rechengerät (20) Fehler detektiert werden, **dadurch gekennzeichnet**, dass bei einer Detektion eines Fehlers mindestens ein Programmobjekt, das bereits einer Abarbeitung zugeführt wurde, in einen definierten Zustand überführt und aus diesem heraus erneut gestartet wird, und nachfolgende weitere Programmobjekte verschoben werden.

2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass die Programmobjekte in wenigstens zwei Klassen unterteilt werden, wobei bei einer Detektion eines Fehlers Programmobjekte der ersten Klasse wiederholt werden und dass bei einer Detektion eines Fehlers in einem Programmobjekt der ersten Klasse, das bereits einer Abarbeitung zugeführt wurde, dieses Programmobjekt erneut gestartet wird und nachfolgende weitere Programmobjekte der ersten Klasse verschoben werden.

3. Verfahren nach Anspruch 2, dadurch gekennzeichnet, dass die Detektion eines Fehlers nur in Klassen durchgeführt wird, die erneut gestartet werden.

4. Verfahren nach Anspruch 2, dadurch gekennzeichnet, dass für alle Programmobjekte eine Gesamtrechenzeit in einem Durchlauf vorgesehen ist und die Gesamtrechenzeit derart aufgeteilt wird, dass die Programmobjekte welche bei Detektion eines Fehlers erneut gestartet werden im fehlerfreien Fall höchstens 50% der Gesamtrechenzeit in einem Durchlauf erhalten.

5. Verfahren nach Anspruch 2, dadurch gekennzeichnet, dass die Programmobjekte der unterschiedlichen Klassen abwechselnd abgearbeitet werden.

6. Verfahren nach Anspruch 2 und 5, dadurch gekennzeichnet, dass das fehlerhafte Programmobjekt der ersten Klasse anstelle des direkt nachfolgenden Programmobjektes der zweiten Klasse abgearbeitet wird.

7. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass die Programmobjekte als Tasks des Computerprogramms (23)ausgebildet sind und bei der Detektion eines Fehlers mindestens eine Task erneut ausgeführt wird.

8. Verfahren nach Anspruch 1 oder 6, dadurch gekennzeichnet, dass ein zum Zeitpunkt der Detektion des Fehlers ausgeführtes Programmobjekt erneut ausgeführt wird.

9. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass während der Abarbeitung der Programmobjekte, insbesondere zu Beginn der Abarbeitung der Programmobjekte, mindestens ein definierter Zustand der Programmobjekte erzeugt und abgespeichert wird.

10. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass das Verfahren in einem Kraftfahrzeug, insbesondere in einem Kraftfahrzeugsteuergerät, verwendet wird.

11. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass während der Abarbeitung des Computerprogramms (23) vor der Ausführung eines Programmobjekts die Werte von zur Abarbeitung des Programmobjekts notwendigen Größen gespeichert werden (3).

12. Verfahren nach Anspruch 1, dadurch gekennzeichnet, dass bei einem in einer Periode periodisch abzuarbeitenden Computerprogramm (23) bei einer Detektion eines Fehlers auf ein bestimmtes Programmobjekt an einem vorgebbaren Rücksprungpunkt in der Periode des Computerprogramms (23) zurückgesprungen wird.

13. Betriebssystem (25), das auf einem Rechengerät (20), insbesondere auf einem Mikroprozessor (22), ablauffähig ist, **dadurch gekennzeichnet**, dass das Betriebssystem (25) zur Ausführung eines Verfahrens nach einem der Ansprüche 1 bis 12 programmiert ist und das erfindungsgemäße Verfahren ausführt, wenn es auf dem Rechengerät (20) abläuft.

14. Rechengerät (20) zum Abarbeiten eines Computerprogramms (23), das mehrere Programmobjekte umfasst, wobei das Rechengerät (20) einen Fehlerentdeckungsmechanismus zur

Detektion eines Fehlers während der Abarbeitung des Computerprogramms (23) auf dem Rechengerät (20) aufweist, **dadurch gekennzeichnet**, dass das Rechengerät (20) einen Fehlerbehandlungsmechanismus aufweist, der bei einer Detektion eines Fehlers durch den Fehlerentdeckungsmechanismus veranlasst, dass mindestens ein Programmobjekt, das bereits einer Abarbeitung zugeführt wurde, in einen definierten Zustand überführt und aus diesem heraus erneut gestartet wird, und nachfolgende weitere Programmobjekte verschoben werden.

15. Rechengerät nach Anspruch 14, dadurch gekennzeichnet, dass der Fehlerbehandlungsmechanismus derart ausgebildet ist, dass die Programmobjekte in wenigstens zwei Klassen unterteilt werden, wobei bei einer Detektion eines Fehlers in einem Programmobjekt der ersten Klasse dieses anstelle eines Programmobjektes der zweiten Klasse erneut gestartet wird.

16. Rechengerät (20) nach Anspruch 14, dadurch gekennzeichnet, dass der Fehlerbehandlungsmechanismus eine Triggerlogik aufweist, welche bei einer Detektion eines Fehlers das mindestens eine Programmobjekt neu startet.

17. Rechengerät (20) nach Anspruch 14, dadurch gekennzeichnet, dass auf dem Rechengerät (20) ein Echtzeit-Betriebssystem (25) abläuft.

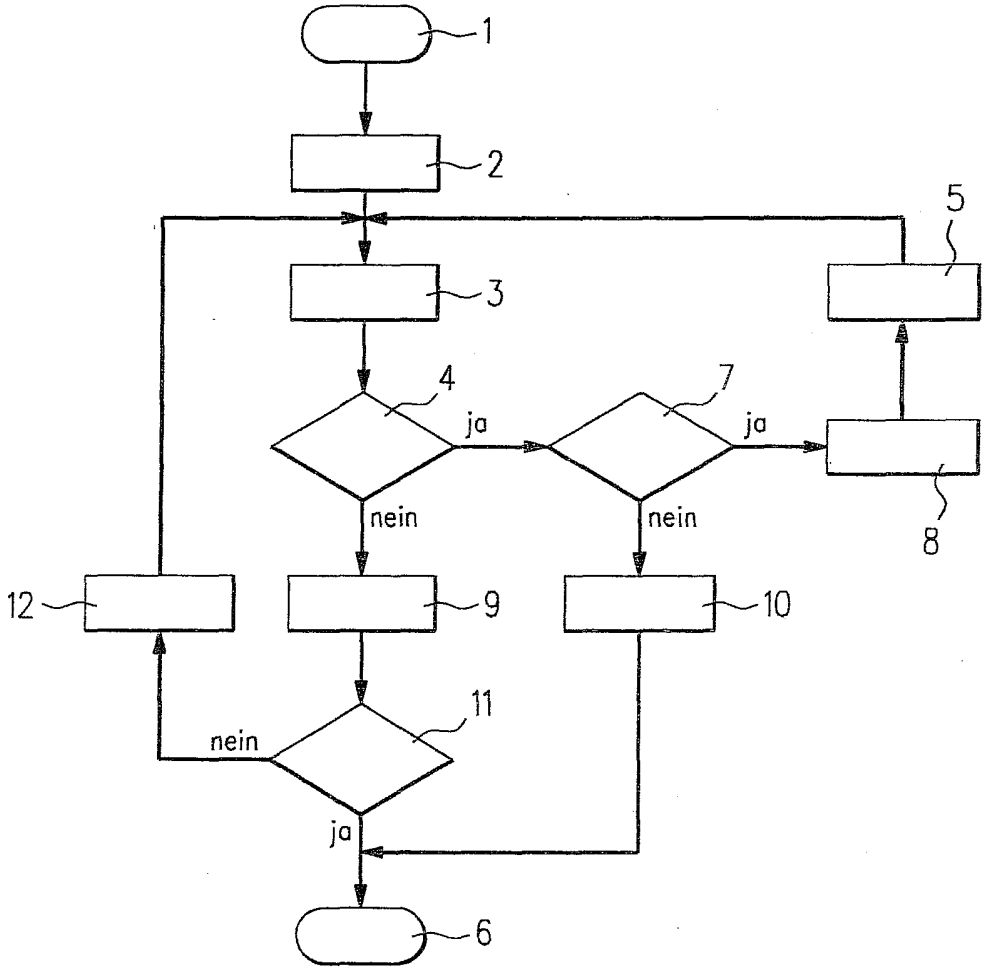


Fig. 1

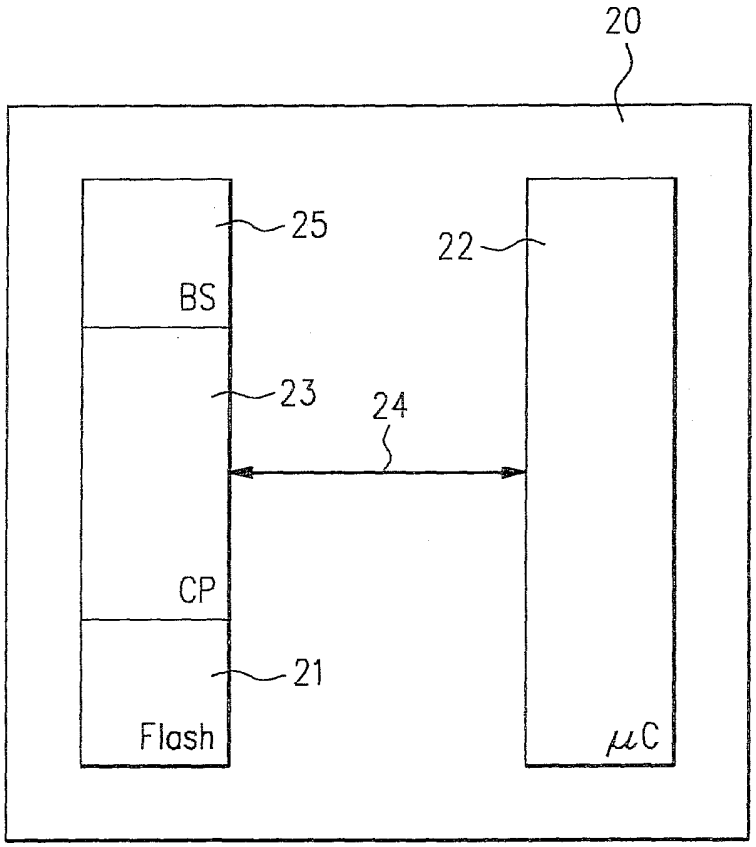


Fig. 2

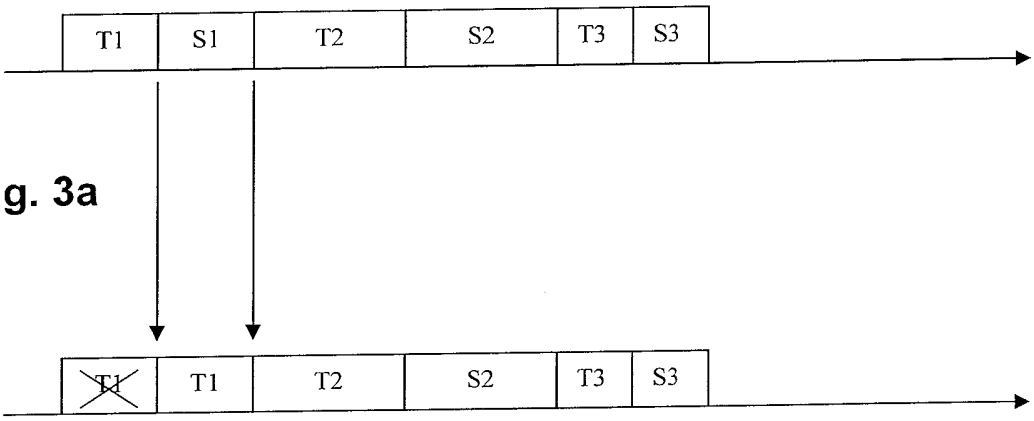


Fig. 3a

Fig. 3b

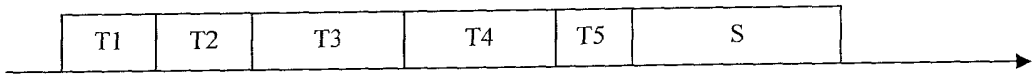


Fig. 4a

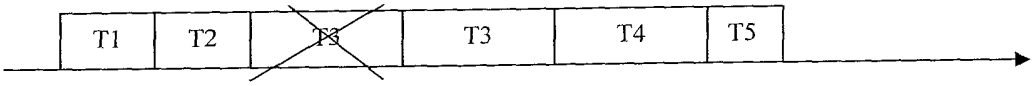


Fig. 4b

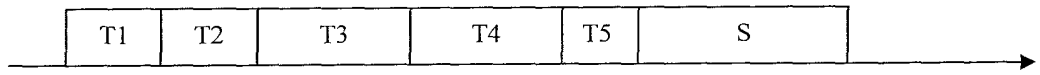


Fig. 5a

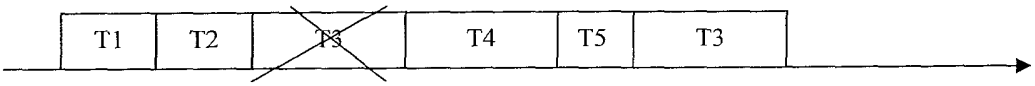


Fig. 5b