Matthias Stümpfle

Ober sticht Unter

Prioritätenvergabe in Echtzeit-Systemen

In Multitasking-Betriebssystemen nutzen mehrere Prozesse die Ressourcen einer Maschine. Die Vergabe von Betriebsmitteln erfolgt dabei durch sog. „Schedules" – Zeitpläne, nach denen die Nutzungsrechte auf die Prozesse aufgeteilt werden. Dieser Beitrag stellt verschiedene Strategien und konkrete „Rezepte" für die Prioritätenvergabe vor und erläutert die Konsequenzen an kurzen Beispielen.

Die meisten heute erhältlichen Echtzeitbetriebssysteme unterstützen das Multitasking, die quasiparallele Abarbeitung von mehreren Rechenaufgaben. Eine wichtige Rolle bei der Entwicklung eines Softwaresystems spielt dabei die Vergabe von Prioritäten an die einzelnen Tasks. Anwendungen reichen von Telekommunikationsanlagen bis hin zu Steuergerätenetzen in Fahrzeugen.

Ziel ist in allen Fällen die optimale Nutzung von Systemressourcen, im Fall der Prioritätenvergabe vor allem der Rechenzeit oder allgemein der Bearbeitungszeit durch den Prozessor. Die Bearbeitungszeit wird den Prozessen aufgrund ihrer Anforderungen vom Betriebssystem zugeteilt. Dies erfolgt mit der Hilfe von sog. „Schedules", den Zeitplänen, mit denen die Betriebsmittel verplant werden. Zusätzlich spielen oft Echtzeitanforderungen eine wesentliche Rolle, d.h., man erwartet von den betrachteten Systemen eine Reaktion innerhalb vorgegebener Zeitschranken.

Ein Schedule kann z.B. eine FIFO-Strategie (first in, first out) verfolgen. Dazu werden alle Anforderungen der Prozesse in eine Warteschlange geschrieben. Die älteste Anforderung in der Warteschlange wird dann als nächste bedient. Mit dieser Strategie werden alle Prozesse gleich behandelt: Man hat eine faire Vergabe der Systemressourcen. Diese Strategie bietet sich dann an, wenn alle Prozesse von gleicher Wichtigkeit sind.

Sind die abzuarbeitenden Prozesse allerdings unterschiedlich wichtig, z.B. weil mit einigen Prozessen sicherheitskritische Funktionen erledigt werden sollen, dann reichen solche einfachen Methoden nicht aus. Hier wird die oben angesprochene Fairneß durch die Vergabe unterschiedlicher Prioritäten aufgehoben. Man gibt wichtigeren Prozessen höhere Prioritäten. Die Wichtigkeit kann dabei aus verschiedenen Randbedingungen resultieren, läßt sich aber in den meisten Fällen auf das Einhalten von zeitlichen Anforderungen zurückführen.

Unterbrechbare und nicht unterbrechbare Systeme

Prioritätensysteme können nach mehreren Merkmalen unterschieden werden. Zunächst kann man zwischen unterbrechenden und nicht unterbrechenden Prioritäten unterscheiden. Während bei nicht unterbrechenden Prioritäten laufende Prozesse nicht von höher priorisierten Anforderungen unterbrochen werden können (z.B. ist der CAN-Bus ein derartiges System), kann bei unterbrechenden

Prioritäten eine niederpriore Anforderung durch eine höherpriore Anforderung unterbrochen werden (*Bild 1*).

Eine Mischform aus unterbrechenden und nicht unterbrechenden Prioritäten sind die sog. Unterbrechungsdistanz-Prioritäten [4], bei denen für jede Priorität eine „Distanz" angegeben werden kann, außerhalb der sie andere Anforderungen mit den entsprechenden Prioritäten unterbricht. Die beiden oben vorgestellten Fälle sind dabei als Sonderfall enthalten: Bei der Unterbrechungsdistanz 1 wird jede andere Priorität unterbrochen, man hat also ein unterbrechendes System. Bei der Distanz N – wobei N die niedrigste im System vorkommende Priorität ist – hat man ein nicht unterbrechendes System, da alle im System vorkommenden Prioritäten innerhalb dieser Distanz liegen.

Prioritätszuteilung vor oder nach der „Geburt" des Prozesses

Weiter lassen sich noch zwei grundlegende Ausprägungen unterscheiden: Systeme mit statischen und solche mit dynamischen Prioritäten. Bei den ersteren erhalten die Prozesse ihre Prioritäten vor Beginn der Prozeßlebenszeit zugeteilt, bei dynamischen Prioritäten können diese zur Laufzeit des Systems zugeteilt und verändert werden, z.B. um sich neuen Lastverhältnissen anzupassen.

Schließlich unterscheidet man noch Systeme mit und ohne Echtzeitbedingungen. Werden keine Echtzeitanforderungen gestellt, kommen oftmals Zeitscheibenverfahren auf „round robin"-Basis zum Einsatz. Im Betriebssystem Unix wird dieses Verfahren eingesetzt, wobei jede Anforderung den Prozessor für eine bestimmte Zeit („time slice") nutzen darf. Ist diese Zeit abgelaufen, dann wird der Prozeß unterbrochen, und die nächste Anforderung in der Warteschlange kommt zur Ausführung. Der unterbrochene Prozeß wird am Ende der Warteschlange eingereiht und bei der nächsten Zuteilung des Prozessors weiterbearbeitet.

Im weiteren Verlauf des Artikels sollen insbesondere die statischen Prioritätensysteme betrachtet werden, wie sie in Automatisierungssystemen vorkommen. Ein kurzer Überblick über dynamische Prioritäten folgt am Ende.

Systeme mit statischen Prioritäten

Die Zuteilung von Betriebsmitteln erfolgt durch „Schedules". Dabei werden die Anforderungen eines

Prozesses P_i als periodische Ereignisse betrachtet, die sich mit dem Ankunftsabstand T_i wiederholen. Mit diesem Modell können auch Anforderungen betrachtet werden, die nur sporadisch auftreten. Solange man einen Mindestabstand zwischen den Anforderungen angeben kann, ist es möglich, eine pessimistische Abschätzung für diese Anforderungen durchzuführen, indem sie ebenfalls als periodisch betrachtet werden.

Echtzeitanforderungen werden durch die Angabe von Fertigstellungsterminen D_i („Deadlines") formuliert, d.h., man gibt eine Zeit an, zu der eine Anforderung fertig bearbeitet sein muß, nachdem sie in das System eingetreten ist.

Auslastung von Betriebsmitteln

Vor der Vergabe der Prioritäten kann zunächst geklärt werden, ob das System prinzipiell lauffähig ist, indem die Last der Betriebsmittel überprüft wird. Das Angebot sollte dabei 100 % nicht überschreiten. Eine einfache Form der Angebotsberechnung läßt sich mit folgender Gleichung [8] durchführen:

$$w = \sum_i \frac{C_i}{T_i} \qquad (1)$$

Dabei werden einfach die Anforderungen der einzelnen Prozesse aufsummiert.

Formelzeichen

P_i	Anforderungen eines Prozesses
T_i	Periodendauer, mit der sich die Anforderungen eines Prozesses wiederholen
C_i	Computation Time (Rechenzeit)
D_i	Deadline (Fertigstellungstermin), zu der die Anforderung eines Prozesses fertig bearbeitet sein muß.
r	Response Time (Antwortzeit)

Prozesse mit großer Wichtigkeit haben in diesem Beitrag eine kleine Prioritätsnummer und umgekehrt. Der Prozeß mit der größten Wichtigkeit hat die Priorität 0.

Scheduling-Algorithmen

Wichtige Vertreter von Scheduling-Algorithmen sind der „Rate-Monotonic"-Schedule [9] und der „Deadline-Monotonic"-Schedule [8] und der „Least-Laxity"-Algorithmus, die jeweils bei bestimmten Anforderungen optimal sind. Welcher der Algorithmen in welchem Fall besser geeignet ist, muß am jeweiligen System untersucht werden.

- Bei Anwendung des *Rate-Monotonic-Schedules* werden genau die Prozesse mit hoher Priorität versehen, die einen kurzen Ankunftsabstand zwischen den einzelnen Anforderungen aufweisen, d.h., hier werden die Prozesse bevorzugt, deren Anforderungen oft nacheinander auftreten. Die Bearbeitungszeit für einen Prozeß wird hier nicht berücksichtigt.
- Beim *Deadline-Monotonic-Schedule* werden die Prioritäten entsprechend ihrer Fertigstellungstermine vergeben: je kürzer die Deadline, desto höher die Priorität.
- Beim *Least-Laxity-Schedule* werden die Prioritäten so vergeben, daß Prozesse, deren Bearbeitungszeit nur wenig kleiner ist als eine vorgegebene Deadline, eine hohe Priorität erhalten. Bei diesem Vorgehen trägt man der Tatsache Rechnung, daß Prozesse, die wenig Zeit zwischen dem Fertigwerden und dem Erreichen der Deadline haben, bevorzugt werden müssen.

Sind die Prioritäten mit Hilfe eines der genannten Algorithmen vergeben, so muß überprüft werden, ob damit ein gültiger, d.h. ausführbarer Schedule entstanden ist („feasibility test").

Antwortzeiten

Ein feststellbares Maß für ein funktionierendes System ist die Antwortzeit einer Anforderung („response time"), deren schlimmster Fall („worst-case") berechnet werden kann. Kann man zeigen, daß die schlechteste Antwortzeit für alle Anforderungen des Systems unter der jeweiligen Deadline einer Anforderung liegt, dann erhält man einen gültigen Schedule.

Zur Berechnung der Antwortzeiten r (response time) kann der „time dilation algorithm", 1984 von Harter [5] entwickelt, herangezogen werden:

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{T_i} \right\rceil C_j \qquad (2)$$

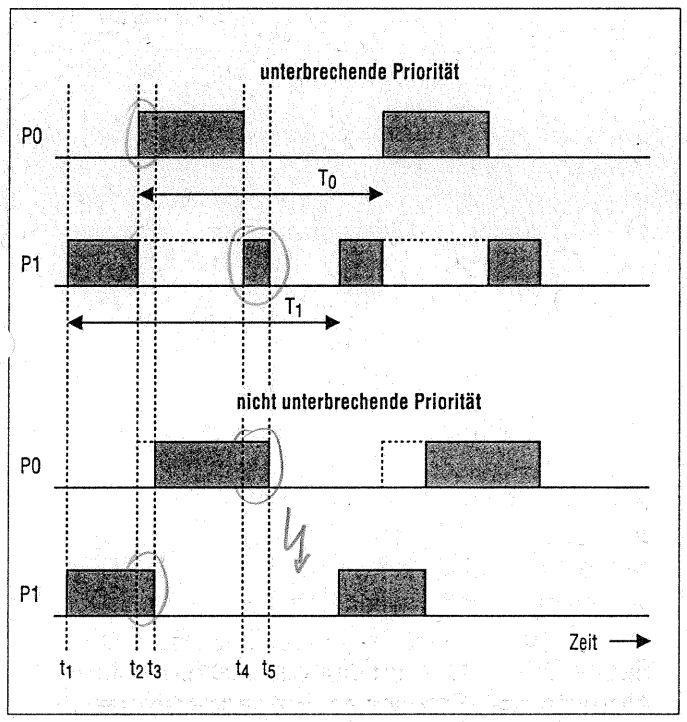

Bild 1. Unterbrechende und nicht unterbrechende Prioritäten. Zum Zeitpunkt t_1 erhält ein Prozeß mit Priorität 1 (P1) das Benutzungsrecht für ein Betriebsmittel. Zum Zeitpunkt t_2 kommt eine Anforderung von Prozeß P0 mit der höheren Priorität 0 in das System. Beim unterbrechenden System (oben) wird die niederpriore Anforderung unterbrochen und erst nach Ablauf von Anforderung 0 fortgesetzt. Ist das System nicht unterbrechbar, läuft Anforderung 1 bis zu ihrem Ende weiter. Anforderung 0 muß deshalb die Zeit t_3–t_2 warten, bis sie bearbeitet wird.

In diesem Algorithmus werden alle möglichen Unterbrechungen eines Prozesses i durch höher priorisierte Prozesse hp(i) berücksichtigt. Man erhält damit eine worstcase-Abschätzung für die Antwortzeit. Der Ceiling-Operator $\lceil\ \rceil$ liefert dabei die kleinste ganze Zahl, die größer oder gleich dem Argument des Operators ist.

Das Problem der Gleichung besteht zunächst darin, daß der gesuchte Ausdruck auch auf der rechten Seite der Gleichung steht. Man kann diese Probleme allerdings iterativ lösen [1], da die rechte Seite der Gleichung in r monoton steigt und nicht divergiert:

$$r_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i^n}{T_j} \right\rceil C_j \qquad (3)$$

Wählt man als Anfangswert $r_i^0 = 0$, so läßt sich der Ausdruck leicht berechnen.

Die Algorithmen und ihre Auswirkungen

Anhand der in *Tabelle 1* enthaltenen Prozesse P1 bis P3, die auf einem gemeinsamen Prozessor ablaufen, werden die Grundlagen für die Planung eines Prioritätensystems gezeigt. Dabei wird besonderes Augenmerk auf die Auswirkung der einzelnen Algorithmen gelegt. Bei den zu bearbeitenden Prozessen erzeugen P1 und P2 zyklische Anforderungen mit vorgegebenen Ankunftsabständen. P3 tritt sporadisch auf, allerdings höchstens alle 100 ms. Diese Anforderung muß dann allerdings innerhalb von 40 ms bearbeitet werden. Das System soll harte Echtzeitanforderungen erfüllen, d.h., das Überschreiten einer Deadline ist in keinem Fall zulässig.

Overhead, der durch das Betriebssystem erzeugt wird (Scheduleraufruf, Zeit für Prozeßwechsel etc.) soll aus Gründen der Vereinfachung nicht betrachtet werden. Zur Vereinfachung wird außerdem angenommen, daß die Deadlines D der Prozesse kleiner oder höchstens gleich sind wie die Ankunftsabstände T.

Zunächst kann die Arbeitslast des Systems nach Gl. (1) bestimmt werden. Wählt man für t die größte im System vorkommende Periode T_{max}, so folgt in diesem Beispiel:

$$w = \frac{10ms}{30ms} + \frac{10ms}{70ms} + \frac{20ms}{100ms} = \frac{71}{105} \qquad (4)$$

Das Angebot ist somit kleiner als 1, und deshalb ist ein gültiger Schedule prinzipiell möglich.

Der „Rate-Monotonic"-Algorithmus

Bei der Vergabe der Prioritäten nach dem Rate-Monotonic-Algo-

rithmus bilden die Ankunftsabstände die Grundlage für die Vergabe der Prioritäten. *Tabelle 2* zeigt das Ergebnis.

Prozeß	Computation Time C in ms	Periode T in ms	Deadline D in ms
P1	10	30	10
P2	10	70	70
P3	20	100	40

Tabelle 1. Prozeßmenge eines Systems. P1 und P2 treten zyklisch auf, P3 nur sporadisch höchstens alle 100 ms. Allerdings muß P3 dann innerhalb von 40 ms bearbeitet werden.

In diesem Beispiel ist P1 aufgrund des kleinsten Ankunftsabstandes der Anforderungen am höchsten priorisiert, gefolgt von P2 und P3. Berechnet man nun die Antwortzeiten mit Hilfe von Gleichung (3), so ergeben sich die folgenden (auch aus Bild 2 nachvollziehbaren) worst-case-Antwortzeiten r_i:

$$P1 = 10ms;\ P2 = 20ms;\ P3 = 50ms \quad (5)$$

Für P1 muß keine besondere Berechnung erfolgen, da dieser Prozeß immer dann zur Abarbeitung kommt, wenn eine entsprechende Anforderung dieses Prozesses ansteht; es werden dann alle anderen laufenden Anforderungen unterbrochen. Die Berechnung der Antwortzeiten für die beiden anderen Prozesse ergibt, daß P2 seine Deadline von 70 ms problemlos halten kann, P3 allerdings seine Deadline um 10 ms überschreitet. Dies kann in einem „harten" Echtzeitsystem, in dem keine Deadline überschritten werden darf, nicht toleriert werden. Obwohl dieser Algorithmus in der Literatur oft als optimal bezeichnet wird, konnte im vorliegenden Beispiel kein gültiger Schedule ermittelt werden. Das liegt daran, daß die Voraussetzung, daß die Deadlines gleich der Periodendauer sein sollen, nicht erfüllt werden konnte. Diese Voraussetzung muß allerdings nicht notwendigerweise erfüllt sein.

Der „Least-Laxity"-Algorithmus

Vergibt man die Prioritäten nach dem Least-Laxity-Algorithmus, so erhält man im Beispiel die Prioritäten und Worst-case-Antwortzeiten nach *Tabelle 3.* P1 erhält wiederum die höchste Priorität, diesmal allerdings gefolgt von P3 und dann P2. Bei dieser Vergabestrategie können alle Prozesse ihre Deadline halten, man erhält also einen gültigen Schedule.

Ähnliche Algorithmen für Systeme mit nicht unterbrechenden Prioritäten

Soll ein System entwickelt werden, das nicht unterbrechbar ist, dann können ähnliche Algorithmen angewendet werden. Ein typischer Vertreter dieser nicht unterbrechbaren Systeme ist das CAN-System, bzw. die Botschaften, die über das Controler Area Network ausgetauscht werden. Jede Botschaft erhält eine Priorität und kann erst auf den Bus gelangen, wenn eine vorher verschickte Botschaft jedweder Priorität den Bus nicht mehr belegt. Bei der anschließenden Vergabe des Busses werden wieder alle wartenden Botschaften berücksichtigt.

Prozeß	T_i in ms	resultierende Priorität
P1	30	0
P2	70	1
P3	100	2

Tabelle 2. Prioritätenvergabe nach dem Rate-Monotonic-Algorithmus. P1 ist aufgrund des kleinsten Ankunftsabstandes der Anforderungen am höchsten priorisiert.

Die Vergabe der Prioritäten erfolgt – wie gezeigt – entweder über den Rate-Monotonic- oder den Least-Laxity-Algorithmus. Danach kann wiederum, diesmal mit Hilfe eines erweiterten „Time Dilation"-Algorithmus, die Antwortzeit r berechnet werden.

$$r_i^{n+1} = C_i + B_{max\,i} + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i - C_i}{T_j} \right\rceil C_j \quad (6)$$

$$\text{mit } B_{maxi} = \max_{\forall k \in lp(i)}(C_k)$$

Die Erweiterung berücksichtigt dabei, daß eine Anforderung i in diesem Fall auch durch eine niederpriore Anforderung $k \in lp(i)$ blockiert werden kann, die zum Zeitpunkt ihres Auftretens bereits bearbeitet wird. Dabei bezeichnet lp(i) die Menge der Anforderungen, die eine geringere Priorität haben als i. Man addiert also einfach die maximale Bearbeitungszeit B aus

Prozeß	$(D_i - C_i)$ in ms	resultierende Priorität	worst-case response time r_i
P1	0	0	10 ms
P2	60	2	50 ms
P3	20	1	30 ms

Tabelle 3. Prioritätenvergabe und maximale Antwortzeiten nach dem Least-Laxity-Algorithmus. Bei dieser Strategie können alle Prozesse ihre Deadlines halten.

der Menge aller niederprioren Anforderungen hinzu. Außerdem verkürzt sich die Zeit, in der die Anforderung i durch höher priore Anforderungen gestört wird, gegenüber einem unterbrechenden System um die Bearbeitungszeit C_i: Wird die Anforderung bearbeitet, dann kann sie nicht mehr unterbrochen werden.

Vereinfachend wird hier angenommen, daß keine Anforderungen ankommen dürfen, bevor nicht vorhergehende Anforderungen derselben Priorität vollständig bearbeitet wurden. Außerdem muß der Ceiling-Ausdruck bei der ersten Iteration zu 1 angenommen werden.

Versucht man das oben vorgestellte Beispiel umzusetzen (*Tabelle 4*), erkennt man sofort, daß auch bei Vergabe nach dem Least-Laxi-

Prozeß-Botschaft	Priorität	$B_{max,i}$	worst-case response time r_i
P1	0	20	30 ms
P2	2	-	40 ms
P3	1	10	40 ms

Tabelle 4. Antwortzeiten in einem CAN-System. Weder P1 noch P2 können ihre Deadlines einhalten.

ty-Verfahren die Worst-case-Antwortzeiten von P1 und P2 über den jeweiligen Deadlines liegen. Für P1 läßt sich die Situation leicht nachvollziehen: P1 möchte z.B. zum Zeitpunkt t = 0 starten. Allerdings belegt P3 das Betriebsmittel (den Bus) minimal vorher zur Zeit $t = 0^-$, so daß P1 20 ms warten muß, bis an die Reihe kommt.

Eine Lösung für diesen Fall besteht darin, die Bearbeitungszeiten C_i der Anforderungen kürzer zu machen. Dies kann hier z.B. durch eine Vergrößerung der Busgeschwindigkeit erreicht werden. Setzt man für die Worst-case-Antwortzeit die Deadline ein, kann man leicht die Busgeschwindigkeit berechnen, die man für ein funktionierendes System braucht. Geht man dabei von einem Beibehalten der Bearbeitungszeitverhältnisse

Prozeß-Botschaft	worst-case computation time	worst-case response time
P1	3,33 ms	10 ms
P2	3,33 ms	13,33 ms
P3	6,66 ms	13,33 ms

Tabelle 5. Antwortzeiten in einem modifizierten CAN-System. Der Bus muß hier mit einer dreimal höheren Geschwindigkeit betrieben werden, um den Anforderungen zu genügen.

aus, d.h. $C_1 = C_2 = C_3/2$ so ergeben sich die Werte nach *Tabelle 5*. Das Ergebnis wird durch die Anforderung P1 bestimmt, da sie ihre Deadline überschreitet. Der Bus

Dipl.-Ing. Matthias Stümpfle arbeitet als wissenschaftlicher Mitarbeiter am Institut für Nachrichtenvermittlung und Datenverarbeitung der Universität Stuttgart. Die Schwerpunkte seiner Arbeit liegen zum einen im objektorientierten Softwareentwurf und zum anderen in der Anwendung dieser Methoden zur Planung von Echtzeitsystemen speziell im automotiven Bereich. Er ist erreichbar unter stuempfle@und.uni-stuttgart.de.

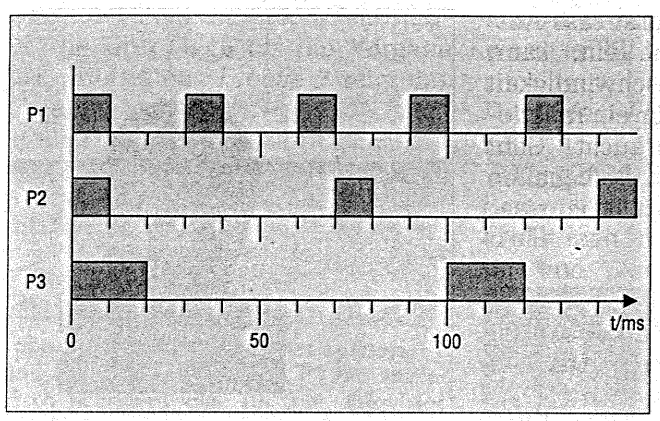

Bild 2. Darstellung des Beispielszenarios aus den Anforderungen der drei Prozesse von Tabelle 1, ohne daß deren Einflußnahme aufeinander dargestellt wird. Dabei wird der Fall betrachtet, daß alle Prozesse eine Anforderung zum Zeitpunkt Null initiieren. Diesen Zeitpunkt bezeichnet man als „kritischen Moment" (critical instant).

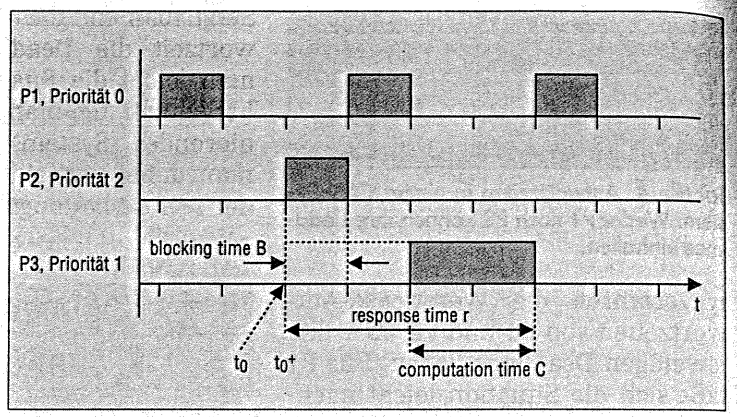

Bild 3. Beispielszenario für ein nicht unterbrechendes System. P1 steht in der Priorität vor P3 und P3 vor P2. Gezeigt wird der „worst case" für die Anforderung von P1. Diese kommt zur Zeit t_{0+} in das System. Kurze Zeit vorher, bei t_0, wurde eine Anforderung von P2 aktiviert, die zu einer Blockierung von P3 führt, obwohl P3 eine höhere Priorität hat als P2. Nach der Bearbeitung von P2 kann P3 immer noch nicht ausgeführt werden, da zunächst die höher priore Anforderung von P1 bearbeitet wird.

müßte also mit einer dreimal höheren Geschwindigkeit betrieben werden, um den Worst-case-Anforderungen zu genügen.

Systeme mit dynamischen Prioritäten

Der Vorteil von dynamischen Prioritäten liegt darin, daß diese zur Laufzeit des Systems berechnet werden und eine Anpassung an aktuelle Systemparameter erlauben wie z.B. der Last auf dem Prozessor. Nachteil dieser Methode ist der erhöhte Aufwand zur Bestimmung der nächsten Anforderung, da der Scheduler mit der entsprechenden Funktionalität ausgestattet werden muß.

Ein verwendeter Algorithmus zur dynamischen Vergabe von Prioritäten ist der „Earliest Deadline First"-Algorithmus, bei dem der Scheduler zu den Zeiten, an denen er aufgerufen wird, überprüft, welche Anforderung in der Warteschlange ihrer Deadline am nächsten gekommen ist. Diese Anforderung wird dann ausgeführt. Ein weiterer Algorithmus, der „Least-Laxity-First"-Algorithmus, bringt die Anforderung zum Ablaufen, deren Abstand zwischen Restrechenzeit und Deadline („laxity") am kleinsten ist.

Ein Problem bei der Vergabe der dynamischen Prioritäten ist die Festlegung, wer die Prioritäten vergeben darf. Diese Systeme können deshalb in solche mit zentraler und solche mit dezentraler Vergabe der Prioritäten unterteilt werden. Im ersten Fall muß eine Instanz existieren, die die Prioritäten systemweit vergibt, und im zweiten Fall müssen die Prozessoren normalerweise ein Protokoll untereinander abwickeln, über das die Prioritäten ausgehandelt werden können. Fragen der dynamischen Vergabe von Prioritäten werden u.a. in [4] diskutiert.

Die Literatur zu Prioritätensystemen und insbesondere über die Scheduling-Theorie ist sehr umfangreich. Eine gute Übersicht über die Fragen des Scheduling mit statischen Prioritäten liefert der Artikel „Fixed Priority Preemptive Scheduling: An Historical Perspective" [2]. Eine grundlegende Arbeit auf dem Gebiet des Scheduling stammt von Conway, Maxwell und Miller: „Scheduling Theory" [3]. *jk*

Literatur

[1] *Audsley, N.C.; Burns, A.; Richardson:* Hard Real-Time Scheduling: The deadline Monotonic Approach. Proc. of 8. IEEE Workshop on Real-Time Operating Systems, USA, 1991.

[2] *Audsley, N. C.; Burns, A. et al.:* Fixed Priority Pre-emptive Scheduling: An Historical Perspective. Real-Time Systems, Vol 8, Kluwer Academic Publ., 1995.

[3] *Conway, R. W.; Maxwell, W. L.; Miller, L. W.:* Theory of Scheduling. Addison-Wesley, 1967.

[4] *Herzog, U.:* Optimal Scheduling Strategies for Real-Time Computers. IBM Jour. o. Research and Dev. Vol. 19, 1975.

[5] *Harter, P. K.:* Response Times in level-structured systems. Techn. Report, Univ. of Colorado, Boulder, 1991.

[6] *Kern, R.:* Prozeßauswahl und Ablaufplanung in Echtzeit-Systemen. *Elektronik* 1992, Heft 14.

[7] *Klein, M. H.; Lehoczky, J. P.; Rajkumar, R.:* Rate Monotonic Analysis for Real Time Industrial Computing. IEEE Trans. on Computers, Jan. 1994.

[8] *Leung, J.; Whitehead, J.:* On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Performance Evaluation, April 1982.

[9] *Liu, C. L.; Layland, J. W.:* Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. JACM, Jan. 1973.