

Script generated by TTT

Title: Grundlagen_Betriebssysteme (28.11.2012)

Date: Wed Nov 28 13:15:42 CET 2012

Duration: 46:18 min

Pages: 11

Eine wesentliche Aufgabe der Prozessorverwaltung besteht darin zu entscheiden, welcher der um den bzw. die Prozessor(en) konkurrierenden Prozesse (bzw. Threads) zu einem Zeitpunkt an den bzw. die Prozessor(en) gebunden wird. Dazu steht die BS-Komponente **Scheduler** zur Verfügung.

Prozessablauf besteht aus einer Sequenz von alternierenden CPU- und E/A-Perioden.

Zeitliche Verschränkung der Prozessbearbeitung bei einer CPU.

Prozess A A hat CPU → Zeit → Zuweisung CPU an A A hat CPU

BS-Kern Unterbrechung BS-Kern hat CPU Unterbrechung

Prozess B Zuweisung CPU an B B hat CPU

Kriterien

- [Scheduling-Strategien](#)
- [Beispiel Unix Scheduling](#)
- [Thread Scheduling](#)
- [Mehrschichtiges Scheduling](#)
- [Echtzeit Scheduling](#)

Generated by Targeteam

Kriterien der Betriebsarten

Der Scheduler wählt aus der Menge der rechenwilligen Prozesse den nächsten auszuführenden Prozess aus. Es existieren unterschiedliche Verfahren die von der jeweiligen Prozessorverwaltungsstrategie abhängen. Mögliche **Leistungskriterien** für ein Schedulingverfahren:

- Fairness.
- Effizienz, Prozessorauslastung.
- Antwortzeit für interaktive Benutzer (Dialogverarbeitung).
- Wartezeit, insbesondere für Batch-Jobs (Stapelverarbeitung).
- Ausführungszeit, d.h. Zeitspanne von Auftragsbeginn bis Auftragsende.
- Abschlusszeit, insbesondere für Realzeitsysteme.
- Durchsatz, Anzahl der Aufträge pro Zeiteinheit.

Kriterien der Betriebsarten

Generated by Targeteam

Die Ziele der Schedulingverfahren hängen von der Umgebung und den Betriebsarten des Rechensystems ab.

Alle Systeme

- Fairness - jeder Prozess bekommt Rechenzeit der CPU.
- Balance - alle Teile des Systems sind ausgelastet.
- Policy Enforcement - Scheduling Policy wird nach außen sichtbar durchgeführt.

Stapelbetrieb

- Durchsatz - maximiere nach Aufträge pro Zeiteinheit.
- Ausführungszeit - minimiere die Zeit von Start bis zur Beendigung.
- CPU-Belegung - belege CPU konstant mit Aufträgen.

Dialogbetrieb - interaktive Systeme

- Antwortzeit - antworte schnellstmöglich auf Anfragen.
- Proportionalität - Erwartungshaltung der Nutzer berücksichtigen.

Echtzeitsysteme

- Abschlusszeit - kein Verlust von Daten.
- Vorhersagbarkeit - Qualitätsverlust bei Multimedia vermeiden.

Generated by Targeteam



Es werden zwischen zwei Klassen unterschieden: **nicht-unterbrechende** (nonpreemptive) und **unterbrechende** Strategien (preemptive).

nicht unterbrechend : Scheduling nur dann möglich, wenn der rechnende Prozess blockiert wird oder wenn er terminiert, d.h. Prozess behält CPU bis er sie selber abgibt.

Beispiel: Microsoft Windows 3.x; unterbrechende Strategien erst ab Windows 95

unterbrechend : Unterbrechung beim Eintreten von speziellen Ereignissen, u.a. Eintreffen eines Prozesses mit höherer Priorität oder Prozess geht in Wartezustand.

[Zeitscheibenstrategie](#)

[Prioritäten](#)

[First-Come First-Served](#)

[Shortest-Jobs-First](#)

Generated by Targeteam



Die **Zeitscheibenstrategie** (Round Robin) ist unterbrechend. Ziel ist die gleichmäßige Verteilung der Rechenzeit auf rechenwillige Prozesse.

Es werden die Prozesse an den Prozessor jeweils für ein festgelegtes Zeitquantum q gebunden und spätestens nach dem Ablauf dieser Zeitspanne wird den Prozessen der Prozessor wieder entzogen.

zyklisches Bedienen der Prozesse (Round Robin).

Ready-Queue (Liste der rechenwilligen Prozesse) als zyklische Warteschlange realisiert.

Wahl des Zeitquantums:

falls q zu klein: viele unproduktive Kontextwechsel.

falls q zu groß: Round Robin wird zu einem reinen FCFS Scheduling (First Come First Served), da die Wahrscheinlichkeit für einen Aufruf eines blockierenden Systemdienst steigt.

Typische Werte für q : 10 bis 100 Millisekunden.

Für $q = 100$ ms gilt bei 1 MIPS Maschine (Million Instructions/Second): ca. 100.000 Instruktionen/ q .

Generated by Targeteam



Diese Strategie ist i.a. unterbrechend. Sie basiert darauf, an die Prozesse **Prioritäten** zu vergeben. Die Prioritätenvergabe kann dabei statisch oder dynamisch sein.

Prioritäten sind i.a. ein festgelegter Zahlenbereich, z.B. 0, ..., 7.

Statische Prioritätenvergabe

jeder Prozess besitzt für die Dauer seiner Existenz eine feste Priorität.

Problem: Gefahr des Verhungerns von Prozessen mit niedriger Priorität

Lösung: Erhöhung der Priorität von lange wartenden Prozessen, d.h. dynamische Prioritäten.

Dynamische Prioritätenvergabe

die Prioritäten der Prozesse können sich dynamisch verändern, d.h. sie werden in gewissen Zeitabständen neu berechnet.

Idee: lange Wartezeiten berücksichtigen (Erhöhen die Priorität).

Prozesse mit großem CPU-Verbrauch sinken in Priorität.

E/A-intensive Prozesse steigen in Priorität (damit E/A-Geräte und CPU parallel genutzt werden).

Zeitscheibenstrategien und Prioritätenvergabe können zu effizienten Verwaltungsstrategien kombiniert werden.

[Windows XP Scheduling](#)

Generated by Targeteam



Windows XP nutzt eine Prioritäten-basierte, unterbrechende Strategie. Thread mit höchster Priorität wird ausgeführt, bis

er terminiert, oder

er seine Zeitscheibe ausgeschöpft hat, oder

eine Blockierung (Systemaufruf, E/A) auftritt.

Es werden Prioritäten von 0 - 31 unterschieden, die in verschiedene Klassen eingeteilt werden

	Echtzeit	hoch	über normal	normal	unter normal	idle
zeit kritisch	31	15	15	15	15	15
höchste	26	15	12	10	8	6
über normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
unter normal	23	12	9	7	5	3
niedrigst	22	11	8	6	4	2
idle	16	1	1	1	1	1



Generated by Targeteam



Dieses nicht-unterbrechende Verfahren (FCFS) teilt einen Prozessor in der Reihenfolge des Auftragseingangs zu. Ready-Queue wird als FIFO-Liste verwaltet; Verfahren einfach zu realisieren.

Ein Kontextwechsel findet nur statt, wenn der Prozess eine blockierende Systemfunktion aufruft oder der Prozess die CPU freiwillig abgibt.

Es kann eine hohe CPU Auslastung erreicht werden.

Problem : Durchschnittliche Wartezeit ist hoch.

Beispiel: Prozesse P1,P2,P3 kommen nahezu gleichzeitig zum Zeitpunkt 0 an;

Dauer ihrer Berechnungszeiten: P1: 24 ms, P2: 3ms, P3: 3ms;

bei Reihenfolge P1, P2, P3: mittlere Wartezeit: $(0 + 24 + 27)/3 = 17$ ms

bei Reihenfolge P2, P3, P1 mittlere Wartezeit $(0+3+6)/3 = 3$ ms

Generated by Targteam



Dieses Verfahren (SJF) führt die Prozessorzuteilung in der Reihenfolge der wachsenden Rechenphasen ("CPU-Burst") zu, d.h. Prozess mit kürzester, nächster Rechenphase erhält Prozessor als nächster.

anwendbar, falls die Dauer der nächsten Rechenphase bis E/A-Befehl, Interrupt etc. bekannt ist.

Beispiel: P1: 6ms, P2: 8ms, P3: 7ms, P4: 3ms

Schedule bei SJF : P4, P1, P3, P2; Wartezeit: $(3+16 +9 +0) /4 = 7$ ms

bei FCFS: 10.25 ms (P1 vor P2 vor P3 vor P4)

Problem: Kenntnis über die Bedienzeiten erforderlich. Für Stapelbetrieb geeignet, da dort Information über Rechenzeiten zur Verfügung stehen (Benutzer geben bei Batch-Jobs Rechenzeit an).

Für interaktive Prozesse wird die Länge der nächsten Rechenphase ("Burst") geschätzt:

$$S_{n+1} = 1/n \sum_{i=1}^n T_i$$

T_i = Ausführungszeit der i-ten Rechenphase.

S_i = Schätzung für die i-te Rechenphase.

S_1 = Schätzung für die erste Rechenphase.

Anpassung der Berechnung

$$S_{n+1} = (1/n) * T_n + (n-1)/n * S_n$$

Generated by Targteam



Beim Unix-Scheduling handelt sich um eine Zeitscheibenstrategie mit dynamischer Prioritätenvergabe. Unix vergibt für seine Prozesse Prioritäten von 0 - 127 (0 ist die höchste Priorität), die in 32 Warteschlangen verwaltet werden.

Zeitscheibenstrategie pro Warteschlange bis Warteschlange leer; dann Scheduling mit nächst niedrigerer Warteschlange.

dynamische Berechnung der Prozesspriorität:

$$(1) u_prio = USER_PRIO + p_cpu/4 + 2 * p_nice$$

p_cpu ist die Prozessornutzung des rechnenden Prozesses und wird alle 10 ms um 1 inkrementiert.

p_nice ist ein vom Benutzer bestimmter Gewichtungsfaktor $(-20 \leq p_nice \leq 20)$.

$USER_PRIO$ ist die Priorität, die dem Prozess beim Start zugeteilt worden ist.

Der Wert von p_cpu wird jede Sekunde angepasst

$$(2) p_cpu = (2 * load)/(2 * load + 1) * p_cpu + p_nice$$

load ist eine Abschätzung der CPU-Auslastung.

neuere Unix Varianten unterstützen Prioritäten von 0 bis 255, unterteilt in

0 - 127: Echtzeitprozesse

128 - 177: Systemdienste

178 - 255: Benutzerprozesse

Linux nutzt eine Kombination von dynamischen Prioritäten und Zeitscheibenverfahren, wobei die Art der Prozesse, Echtzeit oder interaktiv, berücksichtigt wird.

Generated by Targteam



Beispiel Unix Scheduling



Zeitscheibenstrategie pro Warteschlange bis Warteschlange leer; dann Scheduling mit nächst niedrigerer Warteschlange.

dynamische Berechnung der Prozesspriorität:

$$(1) u_prio = USER_PRIO + p_cpu/4 + 2 * p_nice$$

p_cpu ist die Prozessornutzung des rechnenden Prozesses und wird alle 10 ms um 1 inkrementiert.

p_nice ist ein vom Benutzer bestimmter Gewichtungsfaktor $(-20 \leq p_nice \leq 20)$.

$USER_PRIO$ ist die Priorität, die dem Prozess beim Start zugeteilt worden ist.

Der Wert von p_cpu wird jede Sekunde angepasst

$$(2) p_cpu = (2 * load)/(2 * load + 1) * p_cpu + p_nice$$

load ist eine Abschätzung der CPU-Auslastung.

neuere Unix Varianten unterstützen Prioritäten von 0 bis 255, unterteilt in

0 - 127: Echtzeitprozesse

128 - 177: Systemdienste

178 - 255: Benutzerprozesse

Linux nutzt eine Kombination von dynamischen Prioritäten und Zeitscheibenverfahren, wobei die Art der Prozesse, Echtzeit oder interaktiv, berücksichtigt wird.

Linux hat unterschiedlich lange Zeitscheiben, je nach Priorität, z.B. 200ms für die höchste und 10ms für die niedrigste Priorität.

Generated by Targteam