Script **generated by TTT**

Title: Grundlagen_Betriebssysteme (23.11.2015)

Date: Mon Nov 23 13:45:21 CET 2015

Duration: 89:58 min

Pages: 20

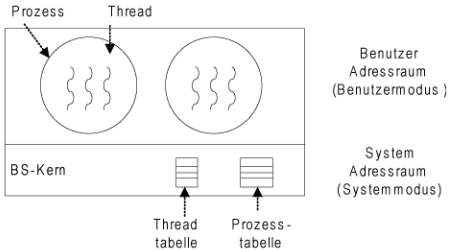
Prozessverwaltung

Dieser Abschnitt behandelt das Prozesskonzept, Datenstrukturen zur Beschreibung des aktuellen Prozesszustandes sowie Dienste zum Aufruf von Systemfunktionen.

Prozesse repräsentieren eine Aktivität; sie haben ein Programm, Eingaben, Ausgaben und einen Zustand.

im System-Adressraum

Neben den Prozessen werden im BS-Kern auch alle Threads verwaltet.



Thread-Tabelle speichert Informationen (Register, Zustand, etc.) über Threads.

Prozessorzuteilung im BS-Kern erfolgt an Threads.

Der Systemaufruf eines Threads blockiert nicht die anderen Threads des Prozesses.

Realisierung von Threads

Es existieren zwei grundlegende Ansätze, Threads in einem Rechensystem zu realisieren: im ***Benutzer-Adressraum*** (Benutzermodus) oder im ***System-Adressraum*** (Systemmodus).

Prozess- und Prozessorverwaltung

Ein Prozess ist der Ablauf eines Programms in einem Rechensystem. Dieser Ablauf ist eine Verwaltungseinheit im jeweiligen Betriebssystem.

Fragestellungen

Dieser Abschnitt gibt eine kurze Einführung in eine der wichtigen Verwaltungsaufgaben eines Betriebssystems:

Verwaltung von Prozessen.

Verwaltung des Prozessors, d.h. Zuteilung der CPU an rechenbereite Prozesse (Scheduling).

Unterbrechungskonzept.

Prozessverwaltung

Prozessorverwaltung

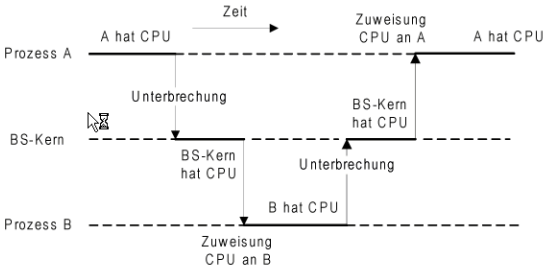
Unterbrechungskonzept

Prozessorverwaltung

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.



Unterscheidung zwischen Prozess-Scheduling und Thread-Scheduling.

Kriterien

Scheduling-Strategien

Beispiel Unix Scheduling

Thread Scheduling

Mehrschichtiges Scheduling

Echtzeit Scheduling

Scheduling-Strategien

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

Kriterien

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

Kriterien der Betriebsarten

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.

Kriterien

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.

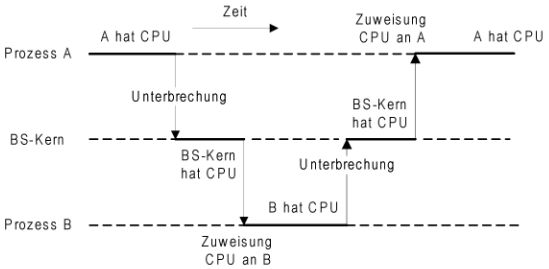
Kriterien der Betriebsarten

Prozessorverwaltung

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.



Unterscheidung zwischen Prozess-Scheduling und Thread-Scheduling.

Kriterien
Scheduling-Strategien
Beispiel Unix Scheduling
Thread Scheduling
Mehrschichtiges Scheduling
Echtzeit Scheduling

Scheduling-Strategien

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

Zeitscheibenstrategie

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.

Scheduling-Strategien

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

Windows XP Scheduling

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

Scheduling-Strategien

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

Shortest-Jobs-First

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 SFJ : 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 Targeteam

Beispiel Unix Scheduling

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 ≤ p_nice ≤ 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 Targeteam

Thread Scheduling

Die Prozessorzuteilung von Threads hängt von deren Art der Realisierung ab.

User-Threads

Realisierung der Threads im Benutzeradressraum ⇒ Kern hat keine Kenntnis bzgl. der Threads. BS-Scheduler wählt nur Prozess aus. Laufzeitsystem des Prozesses wählt rechenwilligen Thread des Prozesses aus; es kann ein beliebiges Scheduling-Verfahren für Prozesse verwendet werden.

Java Virtual Machines verwenden unterbrechendes Prioritäten-Scheduling für Threads;

10 ist die höchste und 1 die niedrigste Priorität;

Kernel-Threads

Realisierung der Threads im Systemadressraum ⇒ BS-Scheduler wählt den nächsten auszuführenden Thread aus.

a) Ist ausgewählter Thread demselben Prozess zugeordnet wie der vorher rechnende Thread ⇒ geringer Kontextwechsel.

b) Ist ausgewählter Thread nicht demselben Prozess zugeordnet wie der vorher rechnende Thread ⇒ aufwendiger Kontextwechsel.

Generated by Targeteam

Mehrschichtiges Scheduling

Einlagern eines rechenwilligen Prozesses von der Platte in den Arbeitsspeicher ist aufwendig; deshalb Mehr-Schichten Scheduling.

Short-Term-Scheduler (CPU Scheduler)

Auswahl eines geeigneten Prozesses aus der Ready-Queue; wird häufig aufgerufen; Verfahren siehe oben.

Long-Term-Scheduler

Auswahl rechenwilliger neuer Aufträge (meist Jobs aus dem Hintergrundbetrieb (batch)) und Einlagerung in den Arbeitsspeicher; Einfügen der Prozesse in die Ready-Queue.

Graphische Darstellung

Generated by Targeteam