Script generated by TTT

Title: Eingebettete_Systeme (23.10.2012)

Date: Tue Oct 23 14:45:16 CEST 2012

Duration: 67:59 min

Pages: 25

Eingebettete vernetzte Systeme (IN8014)

Teil I: Betriebssysteme und Systemsoftware

- Prof. J. Schlichter
  - Lehrstuhl für Angewandte Informatik / Kooperative Systeme, Fakultät für Informatik, TU München
  - Boltzmannstr. 3, 85748 Garching

Email: schlichter@in.tum.de

Tel.: 089-289 18654

URL: http://www11.informatik.tu-muenchen.de/

Synchronisation und Verklemmungen

In einem allgemeinen Rechensystem finden eine Vielzahl paralleler Abläufe statt. Dieser Abschnitt beschäftigt sich mit den Eigenschaften nebenläufiger Aktionen.

Charakterisierung von Threads

Aus BS-Sicht ist ein Prozess definiert

durch einen Adressraum.

eine darin gespeicherte Handlungsvorschrift in Form eines sequentiellen Programms (Programmcode).

einen oder mehreren Aktivitätsträgern, die dynamisch die Handlungsvorschrift ausführen ⇒ Threads.

Motivation

Ein Thread ist die Abstraktion eines physischen Prozessors; er ist ein Träger einer sequentiellen Aktivität. Gründe für die Einführung von Threads:

mehrere Threads ermöglichen Parallelität innerhalb eines Prozesses unter Nutzung des gemeinsamen Adressraums.

Aufwand für Erzeugen und Löschen von Threads ist geringer als für Prozesse.

Verbesserung der Performanz der Applikationsausführung durch Nutzung mehrerer Threads.

Threads ermöglichen bei einem Multiprozessor-Rechensystem echte Parallelität innerhalb einer Applikation.

Prozess vs. Thread

Prozesse und Threads haben unterschiedliche Zielsetzungen:

Prozesse gruppieren Ressourcen,

Threads sind Aktivitätsträger, die zur Ausführung einer CPU zugeteilt werden.

Threadspezifische Information

Jeder Thread umfasst eine Reihe von Informationen, die seinen aktuellen Zustand charakterisieren:

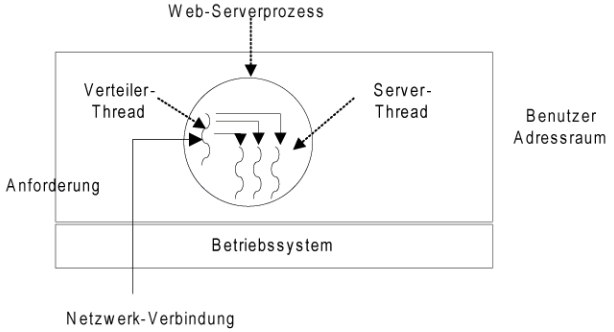
Befehlszähler, aktuelle Registerwerte, Keller, Ablaufzustand des Thread.

Prozessspezifische Information

Die nachfolgende Information/Ressourcen wird von allen Threads eines Prozesses geteilt. Jeder Thread des Prozesses kann sie verwenden bzw. darauf zugreifen.

Adressraum, globale Variable, offene Dateien, Kindprozesse (z.B. erzeugt mit fork), eingetroffene Alarme bzw. Interrupts, Verwaltungsinformation

Ein Prozess kann mehrere Threads umfassen, die unterschiedliche Aufgaben übernehmen. Beispielsweise kann ein Web-Server einen Verteiler-Thread ("dispatcher") und mehrere Server-Threads ("worker-thread") umfassen.

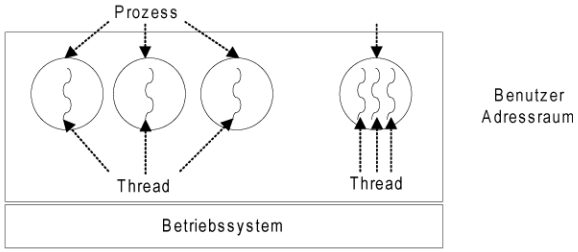


Der Verteiler-Thread dient zur Annahme von Service-Anforderungen und gibt sie weiter an einen der Server-Threads zur Bearbeitung.

Alle Server-Threads nutzen den gemeinsamen Web-Cache.

Thread-Konzept

Threads sind ein BS-Konzept für die Realisierung von nebenläufigen Aktionen in einem Rechensystem.



Charakterisierung von Threads

Synchronisation und Verklemmungen

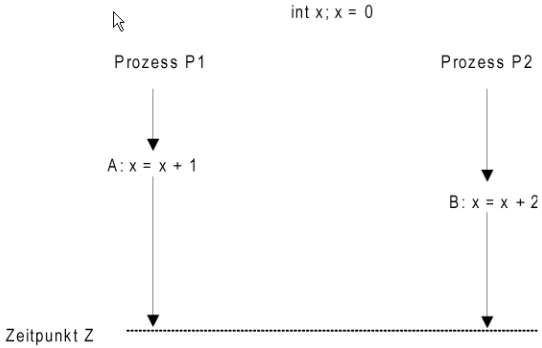
In einem allgemeinen Rechensystem finden eine Vielzahl paralleler Abläufe statt. Dieser Abschnitt beschäftigt sich mit den Eigenschaften nebenläufiger Aktionen.

Thread-Konzept

Synchronisation

Verklemmungen

P1 und P2 sind nebenläufige Prozesse, die in einem Multiprozessorsystem parallel ablaufen. Die Variable x ist gemeinsame Variable. Prozess P1 läuft auf CPU 0 mit Prozess P2 auf CPU 1 gleichzeitig ab.



Mögliche Abläufe

Der Zugriff auf gemeinsame Ressourcen, z.B. auf gemeinsame Variable, muss koordiniert werden. Bei exklusiven Ressourcen wird die Nutzung sequentialisiert.

```
Prozess P1:                Prozess P2:
main () {                  main () {
    .......                    .......
    region x do                region x do
        x = x + 1:                 x = x + 2:
    end region                 end region
    ........                   ........
```

Mögliche Abläufe

Das Ergebnis ist vom zeitlichen Ablauf abhängig. Es sind folgende Fälle möglich:

Fall 1

P1 liest x = 0, erhöht, speichert x = 1;

P2 liest x = 1, erhöht, speichert x = 3; => Wert von x = 3

Fall 2

P2 liest x = 0, erhöht, speichert x = 2;

P1 liest x = 2, erhöht, speichert x = 3; => Wert von x = 3

Fall 3

P1 und P2 lesen x = 0;

P1 erhöht, speichert x = 1;

P2 erhöht, speichert x = 2; => Wert von x = 2

Fall 4

P1 und P2 lesen x = 0;

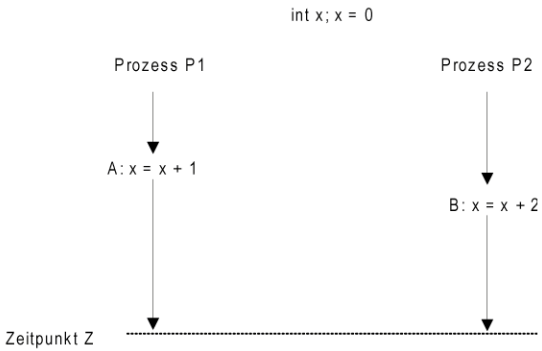
P2 erhöht, speichert x = 2;

P1 erhöht, speichert x = 1; => Wert von x = 1

Verhinderung des Nichtdeterminismus nur dadurch möglich, dass man garantiert, dass die Veränderung von x in den beiden Prozessen unter wechselseitigem Ausschluss (engl. mutual exclusion) erfolgt.

Beispiel: gemeinsame Daten

P1 und P2 sind nebenläufige Prozesse, die in einem Multiprozessorsystem parallel ablaufen. Die Variable x ist gemeinsame Variable. Prozess P1 läuft auf CPU 0 mit Prozess P2 auf CPU 1 gleichzeitig ab.

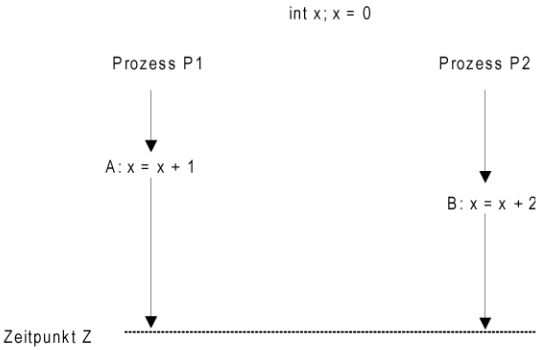


Mögliche Abläufe

Der Zugriff auf gemeinsame Ressourcen, z.B. auf gemeinsame Variable, muss koordiniert werden. Bei exklusiven Ressourcen wird die Nutzung sequentialisiert.

```
Prozess P1:                Prozess P2:
main () {                  main () {
    .......                    .......
    region x do                region x do
        x = x + 1:                 x = x + 2:
    end region                 end region
    ........                   ........
```

Beispiel: gemeinsame Daten



Mögliche Abläufe

Der Zugriff auf gemeinsame Ressourcen, z.B. auf gemeinsame Variable, muss koordiniert werden. Bei exklusiven Ressourcen wird die Nutzung sequentialisiert.

```
Prozess P1:                Prozess P2:
main () {                  main () {
    .......                    .......
    region x do                region x do
        x = x + 1:                 x = x + 2:
    end region                 end region
    ........                   ........
}                          }
```

Generated by Targeteam

Synchronisation

Eine wichtige Systemeigenschaft betrifft die Synchronisation paralleler Ereignisse. Mehrere Prozesse (oder Threads) konkurrieren um eine gemeinsame Ressource (CPU), oder greifen auf gemeinsame Daten zu.

Beispiel: gemeinsame Daten

Synchronisierungskonzepte

Semaphore

Synchronisierung von Java Threads

Anforderungen für wechselseitigen Ausschluss

Folgende Anforderungen sind an eine Realisierung des wechselseitigen Ausschlusses (***w.A*** .) zu stellen:

- Die kritischen Abschnitte der Prozesse sind wechselseitig auszuschließen.
- Eine Realisierung des w.A. darf nicht von einer Annahme über die Reihenfolge, in der die Prozesse ihre kritischen Abschnitte ausführen, abhängen.
- Eine Realisierung des w.A. darf nicht von Annahmen über die Ausführungszeit der Prozesse abhängen.
- Unter der Annahme, dass Prozesse nach endlicher Zeit ihre kritischen Abschnitte wieder verlassen, muss gelten, dass kein Prozess einen anderen Prozess unendlich lange daran hindern darf, seinen kritischen Abschnitt auszuführen.

Jede Realisierung des w.A. benötigt eine Basis, mit festgelegten ***atomaren*** , d.h. nicht teilbaren Operationen.

Synchronisierungskonzepte

Ziel : Einführung wichtiger Realisierungskonzepte zur Synchronisierung paralleler Abläufe. Dazu Konkretisierung des Prozessbegriffs.

Anforderungen für wechselseitigen Ausschluss

Konzepte für wechselseitigen Ausschluss

- **Unterbrechungssperre**
- **Test-and-Set Operationen**
- **Dienste mit passivem Warten**
- **Semaphor-Konzept**
  - Das Semaphor-Konzept ermöglicht die Realisierung des w.A. auf einem höheren Abstraktionslevel als die bereits angesprochenen Hardwareoperationen.
- **Monitor-Konzept**

Unterbrechungssperre

Der Ablauf des Prozesses darf nicht unterbrochen werden.

- In Ein-Prozessorsystemen (1 CPU) kann es ausreichend sein, mit ***Enable*** und ***Disable Interrupt*** Operationen dafür zu sorgen, dass der Prozess, der einen kritischen Abschnitt ausführt, dabei nicht unterbrochen wird.
- Realisierung mit Unterbrechungssperre ist nützlich für den Betriebssystemkern, aber sollte nicht für allgemeine Benutzerprogramme zur Verfügung stehen.

Synchronisierungskonzepte

Ziel : Einführung wichtiger Realisierungskonzepte zur Synchronisierung paralleler Abläufe. Dazu Konkretisierung des Prozessbegriffs.

Test-and-Set Operationen

Test-and-Set Operationen (Test-And-Set-Lock, TSL) erlauben auf Hardware-Ebene (Maschinenbefehl) das Lesen und Schreiben einer Speicherzelle als atomare Operation.

Semantik eines Test-and-Set-Befehls

Die Instruktion liest den Inhalt eines Speicherwortes in ein Register und schreibt einen Wert ungleich Null an die Adresse dieses Wortes. Diese beiden Operationen werden als atomare, unteilbare Sequenz ausgeführt.

Sei a die zu untersuchende Speicherzelle; der Wert 1 in der Speicherzelle kann dahingehend interpretiert werden, dass der Zugang in den kritischen Bereich nicht möglich ist.

Befehl "TSL R, a" entspricht dann

```
Register R = inhalt von a;
a = 1
```

Falls R = 1 ist, ist der kritische Bereich bereits belegt, andernfalls war er frei und er wurde vom ausführenden Prozess belegt.

Problem: wie Atomarität bei Rechensystem mit mehreren CPUs gewährleistet?

Lösung: Die CPU, die die TSL-Instruktion ausführt, blockiert den Speicherbus, um andere CPU's (Multi-Prozessorumgebung) daran zu hindern, auf die Speichereinheit zuzugreifen.

Unterscheidung zwischen

aktivem Warten : Prozess muss periodisch selber prüfen, ob die Voraussetzungen zum Weiterarbeiten erfüllt sind.

passivem Warten : Prozess wird in Warte-Zustand versetzt und aufgeweckt, wenn das Ereignis, auf das er wartet, eingetreten ist.

Methoden oder Dienste, so dass unter Mithilfe des Betriebssystems ein Prozess in den Zustand ***wartend*** übergeführt wird.

beim Eintreten eines passenden "Weckereignisses" wird der Prozess vom Betriebssystem vom Zustand ***wartend*** in den Zustand ***rechenbereit*** übergeführt.

Beispiel: Java wait und notify.

Synchronisierungskonzepte

Ziel : Einführung wichtiger Realisierungskonzepte zur Synchronisierung paralleler Abläufe. Dazu Konkretisierung des Prozessbegriffs.

Anforderungen für wechselseitigen Ausschluss

Konzepte für wechselseitigen Ausschluss

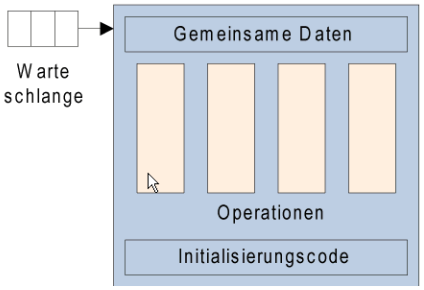
- Unterbrechungssperre
- Test-and-Set Operationen
- Dienste mit passivem Warten
- **Semaphor-Konzept**
  - Das Semaphor-Konzept ermöglicht die Realisierung des w.A. auf einem höheren Abstraktionslevel als die bereits angesprochenen Hardwareoperationen.
- Monitor-Konzept

Generated by Targeteam

Das Monitor-Konzept basiert auf der Idee, die in einem kritischen Bereich bearbeiteten Daten zusammen mit den darauf definierten Zugriffsoperationen in einer sprachlichen Einheit - dem Monitor - zusammenzufassen.

soll Probleme, wie sie bei der Programmierung von Semaphoren auftreten, vermeiden.

zu einem Zeitpunkt höchstens ein Prozess innerhalb des Monitors aktiv.



Definition von Bedingungsvariablen zur Spezifikation anwendungsspezifischer Bedingungen.

Operationen auf Bedingungsvariable `cond`

`cond.wait()`: Prozess wird wartend gesetzt.

`cond.signal()`: ein wartender Prozess wird aktiviert.

Ziel : Einführung wichtiger Realisierungskonzepte zur Synchronisierung paralleler Abläufe. Dazu Konkretisierung des Prozessbegriffs.

Anforderungen für wechselseitigen Ausschluss

Konzepte für wechselseitigen Ausschluss

Unterbrechungssperre

Test-and-Set Operationen

Dienste mit passivem Warten

Semaphor-Konzept

Das Semaphor-Konzept ermöglicht die Realisierung des w.A. auf einem höheren Abstraktionslevel als die bereits angesprochenen Hardwareoperationen.

Monitor-Konzept

Die Operationen P und V sind ***atomar*** ; sie sind unteilbar und sie werden wechselseitig ausgeschlossen ausgeführt.

Sei s die Koordinierungsvariable, dann sind die P und V Operationen wie nachfolgend definiert.

Mögliche Realisierung auf Hardware-Ebene mittels des Test-and-Set Maschinenbefehls.

Informelle Charakterisierung

```
public void P (int s) {
    s = s - 1;
    if ( s < 0 ) { Prozess in die Menge der bzgl. s wartenden Prozesse
    einreihen }
}

public void V (int s) {
    s = s + 1;
    if ( s ≤ 0 ) { führe genau einen der bzgl. s wartenden Prozesse in den
    Zustand rechenwillig über }
}
```

Binäres Semaphor : die Kontrollvariable s nimmt nur boolesche Werte an.

man spricht auch von Mutex.

Mutex in Posix

einfache Sperre zur Absicherung gemeinsamer Ressourcen

pthread_mutex_init(mutex) ⇒ initialisiert und konfiguriert ein Mutex.

pthread_mutex_lock(mutex) ⇒ entspricht der P-Operation; sperrt ein Mutex.

pthread_mutex_unlock(mutex) ⇒ entspricht der V-Operation; gibt ein Mutex frei.

Mutex Objekte und Semaphore mit Integer Werten stehen auch in Windows zur Verfügung.

Operationen

Die Operationen P und V sind ***atomar*** ; sie sind unteilbar und sie werden wechselseitig ausgeschlossen ausgeführt.

Sei s die Koordinierungsvariable, dann sind die P und V Operationen wie nachfolgend definiert.

Mögliche Realisierung auf Hardware-Ebene mittels des Test-and-Set Maschinenbefehls.

Informelle Charakterisierung

```
public void P (int s) {
    s = s – 1;
    if ( s < 0 ) { Prozess in die Menge der bzgl. s wartenden Prozesse
    einreihen }
}

public void V (int s) {
    s = s + 1;
    if ( s ≤ 0 ) { führe genau einen der bzgl. s wartenden Prozesse in den
    Zustand rechenwillig über }
}
```

Binäres Semaphor : die Kontrollvariable s nimmt nur boolesche Werte an.

man spricht auch von Mutex.

Mutex in Posix

Semaphore

Semaphore wurden 1968 von Dijkstra eingeführt. Ein ***Semaphor*** (Signalmast) ist eine ganzzahlige Koordinierungsvariable s, auf der nur die drei vordefinierten Operationen (Methoden) zulässig sind:

Initialisierung,

Prolog P (kommt von protekt),

Epilog V (kommt von vrej).