Seminar
Nebenläufige und verteilte Programmierung
Ein Bakery Algorithmus mit beschränkten Werten

Jan Waller

Institut für Informatik und Praktische Mathematik
Technische Fakultät
Christian-Albrechts-Universität zu Kiel

28. Juni 2005

Inhalt

Dijktras Concurrent Programming Problem

Eine Definition

- N asynchrone Prozesse, shared-memory
- zyklisches Programm, *kritische* und *nicht kritische* Sektionen

Definition

1. Nur ein Prozess darf in seiner kritischen Sektion sein.
2. Jeder Prozess muss irgendwann seine kritische Sektion erreichen (oder abstürzen).
3. Jeder Prozess darf in seiner nicht kritischen Sektion abstürzen.

- Es kann keine Aussage über die Geschwindigkeit der Prozesse gemacht werden.

Dijktras Concurrent Programming Problem

Einige Annahmen - Speicher

Speicher

- Jeder Prozess hat seinen eigenen Speicher.
- Jeder Prozess kann nur in seinem eigenen Speicher schreiben.
- Jeder Prozess kann aus jedem Speicher lesen.

Atomare Operationen

- Jede Lese- oder Schreib-Operation ist atomar.

Dijktras Concurrent Programming Problem

Einige Annahmen - Ausfall von Prozessen

Ein Prozess kann jeder Zeit ausfallen.

Fällt ein Prozess aus

- stoppt der Prozess sofort in seiner nicht kritischen Sektion.
- liefern Lese-Operationen auf seinen Speicher immer Null.

Bakery Definability Property

Ein typischer Algorithmus besteht aus 4 Teilen:

- Eingang (the doorway)
- Warteraum (the wait)
- kritischer Sektion (the critical section)
- Ausgang (the exit)

Definition - Bakery Definability Property

Die Reihenfolge, in der die kritische Sektion betreten wird, wird nur von der Reihenfolge der Abarbeitung des Einganges bestimmt!

Zu Beachten: Sowohl Eingang, als auch Ausgang, bestehen nur aus einer beschränkte Anzahl von Schritten.

ein erster Algorithmus

Algorithmus

ein erster Bakery-Algorithmus

Eingang

```
n[i] = 1 + max(n[1], n[2], ..., n[N]);
```

Warteraum

```
for (j=1 to N)
  do () while (n[j] ≠ 0 ∧ (n[j], j) < (n[i], i));
```

kritische Sektion

```
CRITICAL_SECTION;
```

Ausgang

```
n[i] = 0;
```

ein erster Algorithmus
Problem

Problem

Der gegenseitige Ausschluss ist hier nicht gewährleistet.
2 Prozesse...

Idee

Eine Variable c[] zur Kontrolle, ob ein Prozess grade seine Zahl wählt, wird eingeführt. Falls grade gewählt wird, muss gewartet werden.

Lamports Algorithmus

Algorithmus

Lamports Bakery-Algorithmus

Eingang

```
    c[i] = TRUE;
    n[i] = 1 + max(n[1], n[2], ..., n[N]);
    c[i] = FALSE;
```

Warteraum

```
    for (j=1 to N)
      do () while (c[j] = TRUE);
      do () while (n[j] ≠ 0 ∧ (n[j], j) < (n[i], i));
```

Lamports Algorithmus

Problem

Problem

Die Zahlen in `n[]` können unbeschränkt große Werte annehmen.

Idee

Sobald die Grenze `NMAX` erreicht ist, müssen alle Prozesse mit `n[]` $>$ `NMAX` warten, bis alle mit `n[]` $\leq$ `NMAX` die kritische Sektion verlassen haben, und verringern dann ihren Wert um `NMAX`.

Vijayaraghavans Algorithmus

ein erster Algorithmus

ein beschränkter Bakery-Algorithmus

```
Eingang
    c[i] = TRUE;
    n[i] = 1 + max(n[1], n[2], ..., n[N]);
    c[i] = ENTER1;
    if( n[i] > NMAX )
      for(j=1 to N)
        do () while (n[j] ≠ 0 ∧ n[j] ≤ NMAX);
      c[i] = WAIT;
      for(j=1 to N)
        do () while (c[j] = TRUE ∨ c[j] = ENTER1);
      n[i] = n[i] - NMAX;
      c[i] = READY;
      for(j=1 to N)
        do () while (c[j] = WAIT);

Warteraum
    for (j=1 to N)
      do () while (c[j] = TRUE);
      do () while (n[j] ≠ 0 ∧ (n[j], j) < (n[i], i));

kritische Sektion
    CRITICAL_SECTION;

Ausgang
    n[i] = 0;
    c[i] = FALSE;
```

Vijayaraghavans Algorithmus

ein erster Algorithmus - Problem

Problem

Da wir die Nummern um `NMAX` verringern wollen, darf kein neuer Prozess mehr `n[]` wählen, sonst kann die Reihenfolge nicht sicher gestellt werden.

Idee

Alle neu ankommenden Prozesse werden ab dann in einen neuen Wartebereich umgeleitet.

Vijayaraghavans Algorithmus

ein zweiter Algorithmus

ein beschränkter Bakery-Algorithmus

```
Eingang
   c[i] = TRUE;
   if( x = TRUE )
      warten
      x = FALSE;
   else
      n[i] = 1 + max(n[1], n[2], ..., n[N]);
      c[i] = ENTER1;
      if( n[i] > NMAX )
         x = TRUE;
         for(j=1 to N)
            do () while (n[j] ≠ 0 ∧ n[j] ≤ NMAX);
         c[i] = WAIT;
         for(j=1 to N)
            do () while (c[j] = TRUE ∨ c[j] = ENTER1);
         n[i] = n[i] - NMAX;
         c[i] = READY;
         for(j=1 to N)
            do () while (c[j] = WAIT);

Warteraum
   for (j=1 to N)
      do () while (c[j] = TRUE);
      do () while (n[j] ≠ 0 ∧ (n[j], j) < (n[i], i));

kritische Sektion
   CRITICAL_SECTION;

Ausgang
   n[i] = 0;
   c[i] = FALSE;
```

Vijayaraghavans Algorithmus

ein zweiter Algorithmus - Problem

Problem

Wie können wir nun das ***warten*** so implementieren, das die Reihenfolge der Prozesse erhalten bleibt?

Idee

Ein weiterer Zähler `b[]` zählt die Reihenfolge der im Wartebereich eintreffenden Prozesse.

Vijayaraghavans Algorithmus

ein dritter Algorithmus

ein beschränkter Bakery-Algorithmus

Eingang

```
c[i] = TRUE;
if( x = TRUE )
  b[i] = 1 + max(b[1], b[2], ..., b[N]);
  c[i] = ENTER2;
  for(j=1 to N)
    do () while (c[j] = TRUE);
    do () while (n[j] ≠ 0);
  c[i] = READY;
  for(j=1 to N)
    do () while(c[j] = TRUE ∨ c[j] = ENTER2);
  x = FALSE;
else
  n[i] = 1 + max(n[1], n[2], ..., n[N]);
  c[i] = ENTER1;
  if( n[i] > NMAX )
    x = TRUE;
    for(j=1 to N)
      do () while (n[j] ≠ 0 ∧ n[j] ≤ NMAX);
    c[i] = WAIT;
    for(j=1 to N)
      do () while (c[j] = TRUE ∨ c[j] = ENTER1);
    n[i] = n[i] - NMAX;
    c[i] = READY;
    for(j=1 to N)
      do () while (c[j] = WAIT);
```

Warteraum

```
for (j=1 to N)
  do () while (c[j] = TRUE);
  do () while (n[j] ≠ 0 ∧ (n[j], j) < (n[i], i));
```

kritische Sektion

```
CRITICAL_SECTION;
```

Ausgang

```
n[i] = 0;
b[i] = 0;
c[i] = FALSE;
```

Vijayaraghavans Algorithmus

ein zweiter Algorithmus - Problem

Problem

Nachdem alle Prozesse mit $\texttt{n[]} \neq 0$ abgearbeitet wurden, müssen die Prozesse mit $\texttt{b[]} \neq 0$ bevorzugt in die kritische Sektion.

Idee

Anpassen des Warteraumes mit zwei weiteren Abfragen.

Vijayaraghavans Algorithmus

der fertige Algorithmus

Vijayaraghavans Bakery-Algorithmus

```
Eingang
    c[i] = TRUE;
    if( x = TRUE )
      b[i] = 1 + max(b[1], b[2], ..., b[N]);
      c[i] = ENTER2;
      for(j=1 to N)
        do () while (c[j] = TRUE);
        do () while (n[j] ≠ 0);
      c[i] = READY;
      for(j=1 to N)
        do () while(c[j] = TRUE ∨ c[j] = ENTER2);
      x = FALSE;
    else
      n[i] = 1 + max(n[1], n[2], ..., n[N]);
      c[i] = ENTER1;
      if( n[i] > NMAX )
        x = TRUE;
        for(j=1 to N)
          do () while (n[j] ≠ 0 ∧ n[j] ≤ NMAX);
        c[i] = WAIT;
        for(j=1 to N)
          do () while (c[j] = TRUE ∨ c[j] = ENTER1);
        n[i] = n[i] - NMAX;
        c[i] = READY;
        for (j = 1 to N)
          do () while (c[j] = WAIT);
Warteraum
    for (j=1 to N)
      do () while (c[j] = TRUE);
      do () while ((b[j] ≠ 0 ∧ (b[j], j) < (b[i], i)) ∨
                   (b[j] ≠ 0 ∧ b[i] = 0 ∧ x = FALSE) ∨
                   (n[j] ≠ 0 ∧ (n[j], j) < (n[i], i)));
kritische Sektion
    CRITICAL_SECTION;
Ausgang
    n[i] = 0;
    b[i] = 0;
    c[i] = FALSE;
```

Vielen Dank für die Aufmerksamkeit!

Bibliographie

S. Vijayaraghavan.
A Variant of the Bakery Algorithm with Bounded Values as A Solution to Abraham's Concurrent Programming Problem.
Proc. of Design, Analysis and Simulation of Distributed Systems, 2003.

L. Lamport.
A New Solution of Dijkstra's Concurrent Programming Problem.
Communication of the ACM, Vol.17, 8 pp.453-455, 1974.