Prof. Dr. M. Jäger
FB MNI

16. Januar 2019

Betriebssysteme
Lösungsvorschlag zu Übungsblatt 4

Aufgabe 1 (Peterson-Algorithmus)

Widerspruchsbeweis:

Annahme: Es gibt eine Verklemmung.

Aus der Annahme und dem Code folgt offensichtlich:

(1) Beide Prozesse sind in der Warteschleife (d.h. sie führen die while-Anweisung aus)

Die Betrachtung des Codes ergibt weiter:

(2) In der while-Anweisung wird *TURN* nicht modifiziert
(3) Vor Eintritt der Verklemmung wurden *TURN* nacheinander die Werte 1 und 2 zugeweisen, die Reihenfolge ist nicht bekannt

Daraus folgt: (4) TURN ist in der Verklemmungssituation konstant, $TURN \in \{1, 2\}$

Fallunterscheidung bzgl. der möglichen Werte von *TURN*:

- **Fall 1:** $TURN = 1$

  Aus der Wartebedingung von Prozess 1, sieht man, dass Prozess 1 nicht wartet und damit keine Verklemmung vorliegen kann.

- **Fall 2:** $TURN = 2$

  Aus der Wartebedingung von Prozess 2, sieht man, dass Prozess 2 nicht wartet und damit keine Verklemmung vorliegen kann.

Aufgabe 2 (Speicherverwaltung)

Betrachten Sie das nachfolgende C-Programm

```
#define SPALTEN 8192
#define ZEILEN 100000
int feld[ZEILEN][SPALTEN];
int main(){
  int i,j;
  for ( i=0; i<ZEILEN; i++ )
    for ( j=0; j<SPALTEN; j++ )
      feld[i][j] = 1;
}
```

a) Wieviel Speicherplatz wird das Programm etwa benötigen?

Der Code ist vernachlässigbar klein, für den Stack wird eine Standardgröße genommen, die man ebenso vernachlässigen kann. Der größte Speicherbedarf entsteht für das Array „feld“. Ist `sizeof(int) = 4`, so werden für „feld“ `8192*100000*4` Byte also ca. 3200 MB benötigt.

b) Im logischen Adressraum des Prozesses ist Speicherplatz für „feld“ verfügbar. Zu welchem Zeitpunkt wird dieser Speicherplatz bereitgestellt?

Da „feld“ eine globale Variable ist, wird eine statische Speicherverwaltung verwendet. Daher wird der logische Speicherplatz zu Beginn der Programmausführung reserviert.

c) Überprüfen Sie in der Prozesstabelle jeweils den realen Speicherbedarf zu Beginn, in der Mitte und am Ende der Ausführung. Stimmen die Werte mit ihrer Erwartung überein? Erklären Sie die Werte.

Den aktuellen Speicherverbrauch eines Programms kann man mit dem „ps“-Kommando anzeigen lassen, z.B.

```
ps -o "sz vsz rsz"
```

Der „SZ“-Wert ist die virtuelle Größe in Seiten, der „VSZ“-Wert die virtuelle Größe in kB und der „RSZ“-Wert der reale Speicherverbrauch in kB.

Im Programm lässt sich ein Shell-Befehl durch „system(...)“ ausführen. Beispiel für die Programmdatei „memtest.c“:

```
#include <unistd.h>
#include <stdlib.h>

#define SPALTEN  8192
#define ZEILEN 100000

int feld[ZEILEN][SPALTEN];

void show(){
  // Speicherbedarf anzeigen
  system("ps -o \"sz vsz rsz\" -p $(pidof memtest)");
}

int main(){
      int i,j;
      for ( i=0; i<ZEILEN; i++ )
        for ( j=0; j<SPALTEN; j++ ){
          feld[i][j] = 1;
          if (i==0 && j==0 ||
              i==(int)ZEILEN/2 && j==0 ||
              i==ZEILEN-1 && j==SPALTEN-1)
            show();
        }
    }
```

Die Ausgabe könnte wie folgt aussehen:

```
   SZ    VSZ   RSZ
801010 3204040 660
   SZ    VSZ   RSZ
801010 3204040 1600840
   SZ    VSZ   RSZ
801010 3204040 3200932
```

Der reale Speicherbedarf wächst während der Schleife stetig. Daran sieht man, dass realer Speicher erst beim Zugriff durch das Programm reserviert wird.

d) Messen Sie die Zeit für die Feld-Inititialisierung mit dem Programm *time*. Von der angezeigten „user"-Zeit wird ein Teil auf die CPU-internen Berechnungen und ein Teil auf die Hauptspeicherzugiffe entfallen. Wie könnte man den für die Berechnungen benötigte Zeit ermitteln? Wie lange dauert ein Hauptspeicherzugriff im Durchschnitt? Wieviele Systemtakte sind das?

Zeitmessung

Beispiel für die Laufzeitermittlung für das Programm *timetest*:

```
$ time ./timetest
real    0m1,071s
user    0m0,894s
sys     0m0,076s
```

Entscheidend ist hier die „user"-Zeit, als ca. 0,9 Sekunden.

Anzahl der Hauptspeicherzugriffe

Die Anzahl der logischen Hauptspeicherzugriffe ergibt sich aus der Anzahl der Schleifenwiederholungen. Da die Indexvariablen *i* und *j* in Registern der CPU stehen, bleibt pro Schleifendurchlauf der inneren Schleife nur jeweils ein einziger schreibender Hauptspeicherzugriff auf die Array-Komponente `feld[i][j]`. Die Gesamtanzahl der Speicherzugriffe ist `ZEILEN*SPALTEN`, also `819.200.000`.

Zeitanteil Hauptspeicherzugriffe

Die Zeit für die Berechnungen könnte man durch die Laufzeit eines vergleichbaren Programms ohne Hauptspeicher-Zugriffe abschätzen, z.B.

```
int main(){
    register int i,j;
    for ( i=0; i<ZEILEN; i++ )
      for ( j=0; j<SPALTEN; j++ )
        (j*SPALTEN+i)*ZEILEN+1;
}
```

Am Ende erhält man vielleicht 0,5s für die logischen Hauptspeicher-Zugriffe.

TLB-Einfluss

Das Programm schreibt die Feldelemente in der Reihenfolge, die ihrer physikalischen Abspeicherung entspricht. Dies verringert die Anzahl der für Seitentabellen-Lookups benötigten Hauptspeicherzugriffe drastisch.

Pro Zugriff würde ohne TLB noch ein Seitentabellen-Lookup kommen. Durch den TLB kann man die Anzahl der Seitentabellenzugriffe vernachlässigen, da bei einer Seitengröße von 4096 Byte pro Seite 1000 Array-Elemente gespeichert sind und daher nur bei jedem 1000. Array-Element die Seitentabelle benötigt wird.

Dauer pro logischem Speicherzugriff

Die Dauer pro Zugriff ist also `0,5s/819.200.000`, das sind 0,6ns (Nanosekunden). Bei einer Systemtaktung von 2,5GHz dauert ein Takt 0,4ns, also 1,5 Takte pro logischem Speicherzugriff.

Dauer pro physikalischem Speicherzugriff

Es ist klar, dass Hauptspeicherzugriffe nicht so schnell sein können. Die tatsächlichen Hauptspeicherzugriffe sind durch die Caching-Mechanismen bestimmt. Bei einer Cache-Line-Größe von 64 Byte werden bei jedem physikalischen HS-Zugriff 16 Feldelemente in den Cache kopiert, so dass insgesamt nur `819.200.000/16`, also `51.200.000` Zugriffe erfolgen, was wiederum 24 Takten pro Zugriff entspricht.

e) Erwarten Sie, dass sich die Laufzeit ändert, wenn man das Feld spaltenweise bearbeitet?

```
for ( j=0; j<SPALTEN; j++ )
  for ( i=0; i<ZEILEN; i++ )
    feld[i][j] = 1;
```

Begründen Sie ihre Antwort und überprüfen Sie es durch eine Zeitmessung.

Hier greift das Programm bei jedem Schleifendurchlauf der inneren Schleife auf eine andere Speicherseite zu. Sowohl das Adress-Caching durch den TLB als auch das Daten-Caching werden durch diese ungünstige Programmierung nicht ausgenutzt. Die Laufzeit vervielfacht sich. Auf meinem PC ist die Laufzeit 6-7 Mal länger. Im Experiment ergab das 11 Takte pro Zugriff inklusive Seitentabellen-Lookup.

f) Bei einem modernen System wird sich die Laufzeit bei spaltenweiser Bearbeitung des Felds nicht nur verdoppeln sondern vielleicht verzehnfachen. Dies ist nicht durch den TLB-Verzicht erklärbar. Welche Effekte spielen hier noch mit?

siehe oben

g) In welcher Situation konnte sich die Laufzeit durch die spaltenweise Bearbeitung verhundertfachen? Wenn bei Hauptspeicherknappheit länger nicht genutzte Seiten des Prozess in den Swapbereich ausgelagert werden.

Aufgabe 3 (Virtueller Speicher)

Ein System verwendet virtuellen Speicher mit Paging. Für die Hauptspeicherverwaltung wird eine dreistufige Seitentabelle benutzt. Physikalische Adressen sind 64-Bit groß. Eine virtuelle Adresse ist 32 Bit groß und von der Form

p_1	p_2	p_3	D

mit folgender Aufteilung:

p_1: 5 Bit für die Seitentabelle der 1. Stufe
p_2: 8 Bit für die Seitentabelle der 2. Stufe
p_2: 8 Bit für die Seitentabelle der 3. Stufe
D: 11 Bit für die Distanz

Annahme: Der virtuellen Adresse v=0x00000901 entspricht die reale Adresse r=0x0010100000000101

a) Wie groß sind die Seitenrahmen für das obige Beispiel?

2^{11} Byte

Da 11 Bit für die Distanz verfügbar sind, lassen sich damit 2^{11} Byte adressieren.

b) Geben Sie an, wie der TLB-Eintrag zu (v,r) aussieht

(0x1,0x20200000000)

Der TLB-Eintrag (S,R) enthält die Seitennummer S als Schlüssel und die zugehörige Rahmennummer R als Wert. Der TLB-Eintrag enthält keine Distanzen, da diese in der virtuellen und der realen Adresse identisch sind.

Zur Bestimmung von S müssen also die letzten 11 Bit von v weggelassen werden, zur Bestimmung von R die letzten 11 Bit von r. Dazu verwendet man die binäre Darstellung:

v = 0000 0000 0000 0000 0000 1001 0000 0001
S = 0000 0000 0000 0000 0000 1

r = 0000 0000 0001 0000 0001 0000 0000 0000 0000 0000 0000 0000 0001 0000 0001
R = 0000 0000 0001 0000 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0

Daraus kann man wieder die hexadezimale Darstellung berechnen:

S=0x1 und R=0x20200000000

c) Geben Sie an, wie die Rahmennummer im Beispiel bestimmt wird. Zeichnen Sie dazu eine Skizze mit den benötigten Seitentabelleneinträgen.

Die Werte von p1, p2 und p3 lassen sich mit Hilfe der o.g. Bitbreiten aus der binären Darstellung von v entnehmen.

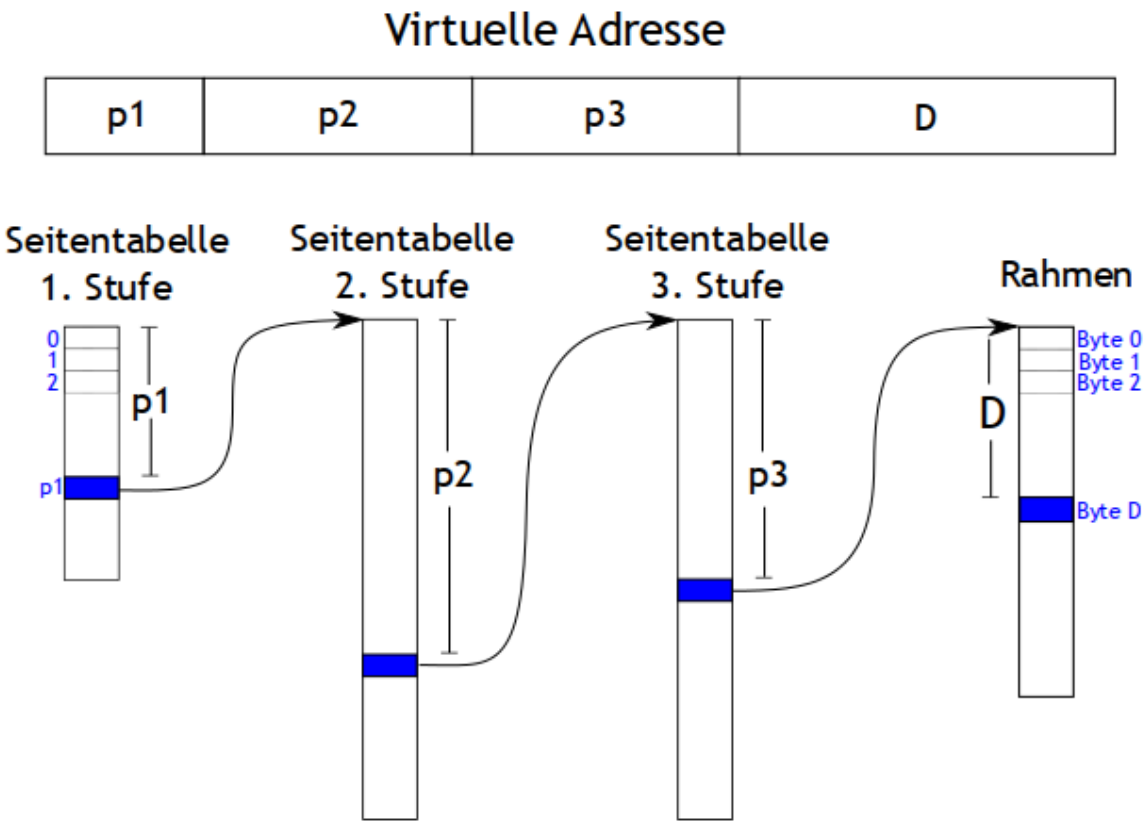
v = 0000 0000 0000 0000 0000 1001 0000 0001

Adressteil	Bitbreite	Bitpositionen in v	Wert (binär)
p1	5	0-4	00000
p2	8	5-12	00000000
p3	8	13-20	00000001
D	11	21-31	00100000001

Im Beispiel erfolgt die Bestimmung also wie folgt

1) Im 1. Eintrag (p1=0) der Seitentabelle Stufe 1 steht die Anfangsadresse der benötigten Seitentabelle Stufe 2.
2) Im 1. Eintrag (p2=0) der Seitentabelle Stufe 2 steht die Anfangsadresse der benötigten Seitentabelle Stufe 3.
3) Im 2. Eintrag (p3=1) der Seitentabelle Stufe 3 steht die Rahmenadresse.

Die nachfolgende Skizze verdeutlicht dies allgemein für beliebige virtuelle Adressen.



d) Die Umrechnung von virtuellen in reale Speicheradressen in einem System mit Paging kostet Zeit.

Dauert die Berechnung einer realen Adresse in einem Rechner mit 3-stufigen Seitentabellen **im Durchschnitt wesentlich** länger als in einem Rechner mit 2-stufigen Seitentabellen (Begründung)?

Bei einer 3-stufigen Tabelle dauert die Bestimmung der realen Adresse mit Hilfe der Seitentabellen natürlich länger als bei einer 2-stufigen, da ein zusätzlicher Hauptspeicherzugriff benötigt wird. Die Bestimmung erfolgt aber in den meisten Fällen nicht mittels Seitentabelle. In mehr als 90% aller Fälle wird die reale Adresse durch TLB-Lookup ermittelt. Deshalb ist die durchschnittliche Umrechnungszeit trotz der zusätzlichen Tabellenstufe nur unmerklich höher.