Systemprogrammierung

Einleitung

3. November 2008

Überblick

Betriebssysteme. . .

. . . bilden das Rückgrat jedes Rechensystems, ob groß oder klein

- was Betriebssysteme sind, kann „Glaubenskriege“ hervorrufen
  - das Spektrum reicht von „Winzlingen“ bis hin zu „Riesen“
  - simple Prozeduren $\Leftrightarrow$ komplexe Programmsysteme
- entscheidend ist, dass Betriebssysteme nie dem Selbstzweck dienen

. . . sind unerlässliches Handwerkszeug der Informatik

- das zu beherrschen und gelegentlich auch noch anzufertigen ist

. . . verstehen, hilft, Phänomene zu begreifen und besser einzuschätzen

- d.h., Eigenschaften (*features*) von Fehlern (*bugs*) zu unterscheiden
  - um Fehler kann ggf. „herum programmiert“ werden
  - um zum Anwendungsfall unpassende Eigenschaften oft jedoch nicht
- Systemverhalten kann sich positiv/negativ „nach oben“ auswirken

Fallstudie: Phänomen Speicherverwaltung

Vorbelegung einer Matrix

Fall A: j wächst schneller als i

```
void by_row (int *mx, unsigned int n, int v) {
    unsigned int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            mx[i * n + j] = v;
}
```

Fall B: i wächst schneller als j

```
void by_column (int *mx, unsigned int n, int v) {
    unsigned int i, j;
    for (j = 0; j < n; j++)
        for (i = 0; i < n; i++)
            mx[i * n + j] = v;
}
```

`mx` Zeiger auf ein Feld `m[n][n]` von `n`×`n` Elementen (`n` Reihen, `n` Spalten)

`mx[i * n + j]` Selektion des `j`-ten Spaltenelements der `i`-ten Reihe; entspricht `m[i][j]` — sollte C eine solche Operation erlauben

Wirkt sich der Unterschied auf das Laufzeitverhalten aus?

Fallstudie (Forts.)

Instanzenbildung und Initialisierung der Matrix

```
#include <stdlib.h>

extern void by_row (int*, unsigned int, int);
extern void by_column (int*, unsigned int, int);

main (int argc, char *argv[]) {
    if (argc == 3) {
        unsigned int n;
        if ((n = atol(argv[2])) {
            int *mx;
            if ((mx = (int*)calloc(n*n, sizeof(int)))) {
                if (*argv[1] == 'R') by_row(mx, n, 42);
                else by_column(mx, n, 42);
                free(mx);
            }
        }
    }
}
```

In Abhängigkeit von der Initialisierungsvariante (`argv[1]`) wird das dynamisch angelegte zweidimensionale Feld (`mx`) in verschiedener Weise mit demselben Wert (`42`) vorbelegt. Die Ausdehnung des Felds ist ein weiterer Programmparameter (`argv[2]`).

Fallstudie (Forts.)

Laufzeitverhalten für $N \times N$ Matrix, mit $N = 11\,174 \approx 500\,\text{MB}$ Speicherplatzbedarf

Betriebssystem	Zentraleinheit	Speicher	by_row()	by_column()	
Solaris	2 × 1 GHz UltraSPARC IIIi	8 GB	3.64r 2.09u 1.11s	27.07r 24.68u 1.10s	(a)
Windows XP (Cygwin)	2 × 3 GHz Pentium 4 XEON	4 GB	0.87r 0.65u 0.21s	11.94r 11.62u 0.21s	(b)
Linux 2.6.20	2 × 3 GHz Pentium 4 XEON	4 GB	0.89r 0.48u 0.40s	14.73r 14.34u 0.39s	(c)
	2 × 2.8 GHz Pentium 4	512 MB	51.23r 0.47u 2.17s	39.84r 14.34u 2.09s	(d)
Mac OS X 10.4	1.25 GHz PowerPC G4	512 MB	10.24r 0.69u 2.12s	106.72r 23.15u 17.08s	(e)
	1.5 GHz PowerPC G4	512 MB	10.11r 0.46u 2.08s	93.68r 23.71u 6.85s	(f)
		1.25 GB	2.17r 0.43u 1.50s	27.95r 22.35u 4.22s	(g)

Fallstudie (Forts.)

Analyse des Laufzeitverhaltens

(a)–(g) Linearisierung
- zweidimensionales Feld $\mapsto$ eindimensionaler Arbeitsspeicher

(a)–(g) Zwischenspeicher (engl. *cache*)
- Zugriffsfehler (engl. *cache miss*), Referenzfolgen

(d) Kompilierer
- semantische Analyse, Erkennung gleicher Zugriffsmuster

(d)–(f) virtueller Speicher
- Seitenfehler (engl. *page fault*), Referenzfolgen

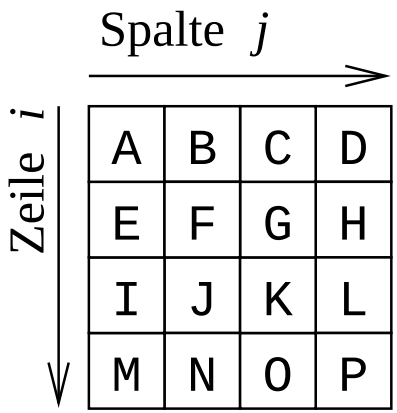
(e)–(g) Betriebssystemarchitektur
- Verortung der Funktion zur Seitenfehlerbehandlung

Sir Isaac Newton

Was wir wissen, ist ein Tropfen, was wir nicht wissen, ist ein Ozean.

Linearisierung

Abspeicherung und Aufzählung



zeilenweise Abspeicherung/Aufzählung

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P

spaltenweise Abspeicherung/Aufzählung

A	E	I	M	B	F	J	N	C	G	K	O	D	H	L	P

abstrakte Repräsentation zweidimensional

wirkliche Repräsentation eindimensional

- zeilen- oder spaltenweise Abspeicherung des Feldes
- zeilen- oder spaltenweise Aufzählung der Feldelemente

Fallstudie, (a)–(g)

- Abspeicherung zeilenweise (C bzw. `main()`)
- Aufzählung zeilen- (`by_row()`) und spaltenweise (`by_column()`)

Linearisierung (Forts.)

Referenzfolgen

Entwicklung der Adresswerte A beim Zugriff auf die Elemente einer zeilenweise abgespeicherten Matrix mit Anfangsadresse γ:

$i \setminus j$	0	1	2	$\cdots$	$N-1$
0	0	1	2	$\cdots$	$N-1$
1	$N+0$	$N+1$	$N+2$	$\cdots$	$N+N-1$
2	$2N+0$	$2N+1$	$2N+2$	$\cdots$	$2N+N-1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$N-1$	$(N-1)N+0$	$(N-1)N+1$	$(N-1)N+2$	$\cdots$	$(N-1)N+N-1$

- $A = \gamma + (i * N + j) * \texttt{sizeof(int)}$, für $i, j = 0, 1, 2, \ldots, N-1$

Fallstudie, (a)–(g)

- linear im homogenen Fall (`by_row()`) $\rightsquigarrow$ starke Lokalität
  - Abspeicherung und Aufzählung sind gleichförmig
- sprunghaft, sonst (`by_column()`) $\rightsquigarrow$ schwache Lokalität

Zwischenspeicher

Abpufferung langsamerer Zugriffe auf den Hauptspeicher

Normalfall Datum befindet sich im Zwischenspeicher

- Zugriffszeit $\approx$ Zykluszeit der CPU, Wartezeit $= 0$

Ausnahmefall Datum befindet sich *nicht* im Zwischenspeicher

- Zugriffsfehler $\rightsquigarrow$ Einlagerung (Zwischenspeicherzeile, *cache line*)
- Zugriffszeit $\geq$ Zykluszeit des RAM, Wartezeit > 0

GAU Zwischenspeicher voll im Ausnahmefall

- Zugriffsfehler $\rightsquigarrow$ Ein- und ggf. Auslagerung (Zwischenspeicherzeile)
- Zugriffszeit $\geq 2\times$Zykluszeit des RAM, Wartezeit $\gg 0$

Problem: Lokalität bzw. Linearität der Zugriffe

- starke Lokalität erhöht die Trefferwahrscheinlichkeit erheblich!!!

Fallstudie, (a)–(g)

- beide Varianten verursachen bei Ausführung den GAU
- `by_column()` $\rightsquigarrow$ schwache Lokalität: schlechte Trefferquote

Kompilierer
Semantische Analyse

funktionale Eigenschaft $\mapsto$ „was“ etwas tut

- beide Varianten tun das gleiche — nur in verschiedener Weise

nicht-funktionale Eigenschaft $\mapsto$ „wie“ sich etwas ausprägt

- `by_row()` zählt Feldelemente entsprechend Feldabspeicherung auf
- `by_row()` zeigt für gegebene Hardware günstigere Zugriffsmuster

⋮

- `by_row()` wird schneller als `by_column()` ablaufen können

Fallstudie, (d): Beispiel eines wahren Mysteriums. . .

- `gcc -O6 -S` zeigt identischen Assemblersprachenkode
  - `by_column()` ist Kopie von `by_row()`
  - statische Analyse sagt gleiches Verhalten beider Varianten voraus
- Experiment brachte Messreihen mit extremen Ausschlägen hervor
  - „dynamische Umgebung“ verhält sich zugunsten von `by_column()`

Virtueller Speicher

Überbelegung des Hauptspeichers

wenn z.B. gilt: `sizeof(Arbeitsspeicher)` $>$ `sizeof(Hauptspeicher)`

- Hauptspeicher ist Zwischenspeicher $\mapsto$ Vordergrundspeicher
  - von Teilen eines oder mehrerer voranschreitender Programme
- ungenutzte Bestände im Massenspeicher $\mapsto$ Hintergrundspeicher
  - z.B. Plattenspeicher, SSD oder gar Hauptspeicher anderer Rechner

Problem: vgl. Zwischenspeicher, S. 4-9

- Ein-/Auslagerung löst zeitaufwändige Ein-/Ausgabevorgänge aus
- Zugriffszeit verlangsamt sich um einige Größenordnungen: ns $\rightsquigarrow$ ms

Fallstudie, (a)–(g): Zwickmühle wegen Hauptspeicherkapazität...

(d)–(f) beide Varianten verursachen bei Ausführung den GAU (S. 4-9)

- kontraproduktiver Seitenvorabruf (engl. *pre-paging*)

sonst fallen „nur“ Einlagerungsvorgänge an: Münchhausen gleich...

- Programm zieht sich selbst komplett in den Hauptspeicher

Betriebssystemarchitektur

Repräsentation und Ausprägung von Systemfunktionen

Schönheit, Stabilität, Nützlichkeit — *Venustas*, *Firmitas*, *Utilitas*: Die drei Prinzipien von Architektur [1]:

- je nach Bedarf sind Systemfunktionen verschieden ausgelegt [2]
  - sie teilen sich dieselben Domänen oder eben auch nicht
  - bzgl. Adressraum, Ausführungsstrang, Prozessor oder Rechnersystem
- Funktionsauslegung und funktionale Eigenschaft sind zu trennen
  - beide rufen jedoch gewisse nicht-funktionale Eigenschaften hervor
  - z.B. verursachen domänenübergreifende Aktionen ggf. Mehraufwand

Fallstudie, (e)–(g)

- Mac OS X = NeXTStep ∪ Mach 2.5 $\rightsquigarrow$ mikrokernbasiertes System
  - Systemfunktionen laufen als *Tasks* in eigenen Adressräumen ab
  - Tasks bieten Betriebsmittel für ggf. mehrere Ausführungsstränge
  - Ausführungsstränge sind die Zuteilungseinheiten für Prozessoren
- Ein-/Auslagerungstask als „externes Rufgerät“: *external pager*
  - außerhalb des klassischen Kerns $\rightsquigarrow$ domänenübergreifende Aktionen

Zwischenzusammenfassung

Es wird noch nicht erwartet, das alles zu verstehen. . . [3]

(a)–(g) Linearisierung
- Algorithmen und Datenstrukturen ✓

(a)–(g) Zwischenspeicher
- Grundlagen der Technischen Informatik ✓
- Grundlagen der Rechnerarchitektur und -organisation ✓

(d) Kompilierer
- Grundlagen des Übersetzerbaus ✂

(d)–(f) virtueller Speicher
- Systemprogrammierung ✗

(e)–(g) Betriebssystemarchitektur
- Systemprogrammierung ✗
- Betriebssysteme ✂

Phänomen hin oder her...

Was macht ein Betriebssystem [aus] ?

Glossar

Be'triebs·sys·tem *<n.; -s, -e; EDV> Programmbündel, das die Bedienung eines Computers ermöglicht. [4]*

Summe derjenigen Programme, die als residenter Teil einer EDV-Anlage für den Betrieb der Anlage und für die Ausführung der Anwenderprogramme erforderlich ist. [5]

Eine Softwareschicht, die alle Teile des Systems verwaltet und dem Benutzer eine Schnittstelle oder eine virtuelle Maschine anbietet, die einfacher zu verstehen und zu programmieren ist [als die nackte Hardware]. [6]

Glossar (Forts.)

Ein Programm, das als Vermittler zwischen Rechnernutzer und Rechnerhardware fungiert. Der Sinn des Betriebssystems ist eine Umgebung bereitzustellen, in der Benutzer bequem und effizient Programme ausführen können. [7]

Ein Betriebssystem kennt auf jeden Fall keinen Prozessor mehr, sondern ist neutral gegen ihn, und das war es vorher noch nie. Und auf diese Weise kann man eben jeden beliebigen Prozessor auf jedem beliebigen anderen emulieren, wie das schöne Wort lautet. [8]

Glossar (Forts.)

The operating system is itself a programm which has the functions of shielding the bare machine from access by users (thus protecting the system), and also of insulating the programmer from the many extremely intricate and messy problems of reading the program, calling a translator, running the translated program, directing the output to the proper channels at the proper time, and passing control to the next user. [9]

Es ist das Betriebssystem, das die Kontrolle über das Plastik und Metall (Hardware) übernimmt und anderen Softwareprogrammen (Excel, Word, ...) eine standardisierte Arbeitsplattform (Windows, Unix, OS/2) schafft. [10]

Glossar (Forts.)

Der Zweck eines Betriebssystems [besteht] in der Verteilung von Betriebsmitteln auf sich bewerbende Benutzer. [11]

Eine Menge von Programmen, die die Ausführung von Benutzerprogrammen und die Benutzung von Betriebsmitteln steuern. [12]

Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen. [13] ☺

Abriss des Vorlesungsstoffes

Wir werden. . .

1. einen „Hauch“ Rechnerorganisation und Softwaretechnik „einatmen“
2. Rechnerbetriebsarten kennen- und unterscheiden lernen
3. Betriebssysteme in ihrer Grobfunktion „von aussen“ betrachten
4. Funktionen von Betriebssystemen im Detail untersuchen
5. den Stoff rekapitulieren

Grundzüge der Lehrveranstaltung

Zusammenhänge stehen im Vordergrund, Spezialitäten im Hintergrund

- Leitfaden ist die ganzheitliche Betrachtung von Systemfunktionen
- skizziert wird eine logische Struktur ggf. vieler Ausprägungsformen
- klassischer Lehrbuchstoff wird ergänzt, weniger repetiert oder vertieft

Ausprägungen von Betriebssystemen dürfen nicht dogmatisiert werden

- etwa: ☆❇■◆❙ „ist besser als“ ✷❇■❄❑◑▲ — umgekehrt dito
- oder: ☆❁❁☆✻ „schlägt beide um Längen“ . . .

Betriebssysteme sind immer im Anwendungskontext zu sehen/beurteilen

Ein Betriebssystem (engl. *operating system*)...

...ist eine Menge von Programmen, die

- Programme, Anwendungen oder BenutzerInnen assistieren sollen
- die Ausführung von Programmen überwachen und steuern
- den Rechner für eine Anwendungsklasse betreiben
- eine abstrakte Maschine implementieren

...hat die Aufgabe, die Betriebsmittel des Rechners zu verwalten

- d.h., Ressourcen zu kontrollieren und ggf. gerecht zu verteilen

...definiert sich nicht über die Architektur, sondern über Funktionen

Literaturverzeichnis

[1] Vitruv Marcus Vitruvius Pollio.
De Architectura Libris Decem.
27 v.Chr. (Primus Verlag, 1996).

[2] Hugh C. Lauer and Roger M. Needham.
On the duality of operating system structures.
ACM Operating Systems Review, 13(2):3–19, April 1979.

[3] Dennis MacAlistair Ritchie.
/* You are not expected to understand this */.
`http://cm.bell-labs.com/cm/cs/who/dmr/odd.html`, 1975.

[4] Renate Wahrig-Burfeind.
Universalwörterbuch Rechtschreibung.
Deutscher Taschenbuch Verlag, September 2002.

Literaturverzeichnis (Forts.)

[5] Hans-Jochen Schneider, editor.
Lexikon der Informatik und Datenverarbeitung.
Oldenbourg-Verlag, München, Wien, fourth edition, 1997.

[6] Andrew Stuart Tanenbaum.
Operating Systems: Design and Implementation.
Prentice-Hall, Inc., second edition, 1997.

[7] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne.
Operating System Concepts.
John Wiley & Sons, Inc., sixth edition, 2001.

[8] Friedrich Kittler.
Interview.
`http://www.hydra.umn.edu/kittler/interview.html`, 1993.

Literaturverzeichnis (Forts.)

[9] Douglas Richard Hostadter.
Gödel, Escher, Bach: An Eternal Golden Braid — A Metaphorical Fugue on Minds and Machines in the Spirit of Lewis Carrol.
Penguin Books, 1979.

[10] Birgit Ewert, Kerstin Christoffer, Uwe Christoffer, and Saban Ünlü.
FreeHand 10.
Galileo Design, 2001.

[11] Peer Brinch Hansen.
Betriebssysteme.
Carl Hanser Verlag, erste edition, 1977.

[12] Arie Nicolaas Habermann.
Introduction to Operating System Design.
Science Research Associates, 1976.

Literaturverzeichnis (Forts.)

[13] Deutsches Institut für Normung.
Informationsverarbeitung — Begriffe.
DIN 44300. Beuth-Verlag, Berlin, Köln, 1985.