BERGISCHE UNIVERSITÄT
WUPPERTAL
GAUSS-STRASSE 20
42119 WUPPERTAL
TELEFAX (0202) 439-2901
TELEFON (0202) 439-0
WWW www.uni-wuppertal.de

Fachbereich C

MATHEMATIK UND NATURWISSENSCHAFTEN
Fachgruppe Mathematik und Informatik

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Hans-Juergen.Buhl@math.uni-wuppertal.de
WWW: http://www.math.uni-wuppertal.de/∼buhl

Betriebssysteme: Konzepte, Dienste, Schnittstellen (Betriebssysteme und betriebssystemnahe Programmierung)

SS 2005 – Übungsblatt 8

Ausgabe: 13. Juni 2005

Abgabe: bis spätestens 20. Juni 2005
im Fachschaftsraum Mathematik
oder per email an `c.markmann@uni-wuppertal.de`

Aufgabe 1. *dynamisch gelinkte Executables*

Informieren Sie sich in

http://docs.sun.com/app/docs/doc/816-1386

über die Unterschiede des (statischen) Ladens eines dynamisch gelinkten Executables bei Programmstart, des „lazy“ Ladens bzw. des dynamischen Ladens mittels der dl-Library.

Schildern Sie in eigenen Worten die Unterschiede. Welche Einsatzgebiete sehen Sie jeweils?

Aufgabe 2. *ddd auf Fehlersuche*

Das folgende Programm enthält einige gravierende Fehler. Mittels des Debuggers ddd können diese bequem genauer untersucht werden:

```
////////////////////////////////
// Datei:   vektor1.cc
// Version: 1.0
// Zweck:   Vektor als Klasse
// Autor:   Hans-Juergen Buhl
// Datum:   26.01.99
////////////////////////////////
```

```
#include        <iostream>

using namespace std;

class vektor{

  const int low;         // v(low..high)
  const int high;

  double* v;             // Startadresse fuer dyn. verwaltetes Exemplar

public:

  vektor(int h, int l = 1, double d = 0.0);
                                                  //  v(l..h) = d

  vektor(const double x[], int n);                // v(1..n) = x[0..n-1]

  ~vektor(){ delete []v; };

  double& operator()(int i);
  double  operator()(int i) const;

  int lo() const { return low;  };
  int hi() const { return high; };

  friend ostream& operator<<(ostream& os, const vektor& v);

};

vektor::vektor(int h, int l, double d) : low(l), high(h)
{
  int size(h-l+1);
  if (size < 1) throw "falsche Vektor-Länge in Konstruktor";
  v = new double[size];
  if (v == 0) throw "kein freier Speicherplatz mehr verfügbar";
  for (int j=0; j < size; j++)
    v[j] = d;
};

vektor::vektor(const double x[], int n) : low(1), high(n)
{
  if (n < 1) throw "falsche Vektor-Länge in Konstruktor";
  // ...    z u    e r g ä n z e n
};

double& vektor::operator()(int i)
{
```

```
  if ( (i < low) || (i > high))
     throw "Indexverletzung bei Komponentenzugriff";
  return v[i-low];
};

double  vektor::operator()(int i) const
{
  if ( (i < low) || (i > high))
     throw "Indexverletzung bei Komponentenzugriff";
  return v[i-low];
};

ostream& operator<<(ostream& os, const vektor& w)
{
  os << "( ";
  os << w(w.lo());
  for (int i=w.lo()+1; i <= w.hi(); i++)
    os << " , " << w(i);
  os << " )";
  return os;
};

int main()
{
  vektor y(5, 1, 3.0);
  cout << y.lo() << " " << y.hi() << endl;
  cout << y  << endl;
  for (int i=y.lo(); i <= y.hi(); i++)
    y(i) = i*2;
  cout << y  << endl;

  vektor x(8, 2);
  cout << x.lo() << " " << x.hi() << endl;
  cout << x  << endl;
  for (int k=x.lo(); k <= x.hi(); k++)
    x(k) = k*2;
  cout << x << endl;

  double zh[] = { 1.0, 3.0, 2.0, 4.0 };
  vektor z(zh, 4);
  cout << z.lo() << " " << z.hi() << endl;
  for (int j=z.lo(); j <= z.hi(); j++)
    z(j) *= z(j);
  cout << z << endl;

  return 0;
}
```

Nach abnormalem Programmabbruch

```
./vektor1
1 5
( 3 , 3 , 3 , 3 , 3 )
( 2 , 4 , 6 , 8 , 10 )
2 8
( 0 , 0 , 0 , 0 , 0 , 0 , 0 )
( 4 , 6 , 8 , 10 , 12 , 14 , 16 )
1 4
( 16 , 0 , 16.5493 , 0 )
Segmentation fault
```

und Überprüfung mittels

```
g++ -g vektor1.cc   -o vektor1
gdb vektor1
(gdb) run
Starting program: /home/buhl/vektor1
1 5
( 3 , 3 , 3 , 3 , 3 )
( 2 , 4 , 6 , 8 , 10 )
2 8
( 0 , 0 , 0 , 0 , 0 , 0 , 0 )
( 4 , 6 , 8 , 10 , 12 , 14 , 16 )
1 4
( 16 , 0 , 16.5493 , 0 )

Program received signal SIGSEGV, Segmentation fault.
0x40184d2e in _int_free () from /lib/libc.so.6
gdb) bt
#0  0x40184d2e in _int_free () from /lib/libc.so.6
#1  0x40183a5f in free () from /lib/libc.so.6
#2  0x400b76b1 in operator delete(void*) () from /usr/lib/libstdc++.so.5
#3  0x400b770d in operator delete[](void*) () from /usr/lib/libstdc++.so.5
#4  0x0804908b in ~vektor (this=0xbffff6c0) at vektor1.cc:27
#5  0x08048f54 in main () at vektor1.cc:100
#6  0x401228ae in __libc_start_main () from /lib/libc.so.6
(gdb) q
```

ist man nur wenig schlauer. Ändern Sie das Programm, um die Fehlerstelle genauer zu untersuchen. Benutzen Sie `ddd` statt `gdb` zur schrittweisen (Knopf `Next` bzw. `Step`) Programmausführung. Beheben Sie den Fehler.

Aufgabe 3. *cat1.c*

Bringen Sie das folgende Programm zum Ablauf

```
/* cat1.c -- simple version of cat */
#include <stdio.h>
#include <unistd.h>

/* While there is data on standard in (fd 0), copy it to standard
   out (fd 1). Exit once no more data is available. */

int main(void) {
    char buf[1024];
    int len;

    /* len will be >= 0 while data is available, and read() is
       successful */
    while ((len = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
        if (write(STDOUT_FILENO, buf, len) != len) {
            perror("write");
            return 1;
        }
    }

    /* len was <= 0; If len = 0, no more data is available.
       Otherwise, an error occurred. */
    if (len < 0) {
        perror("read");
        return 1;
    }

    return 0;
}
```

Lesen Sie die Manual-Seiten von `read` und `write`. Erklären Sie die Wirkungsweise des Programms. Ist das Programm korrekt?

Aufgabe 4. *cat2.c*

Bringen Sie das folgende Programm zum Ablauf

```
/* cat2.c -- simple 2nd version of cat */
#include <stdio.h>
#include <unistd.h>

/* While there is data on standard in (fd 0), copy it to standard
   out (fd 1). Exit once no more data is available. */

int main(void) {
    char buf[1024];
    int len;
    int wlen, n;

    /* len will be >= 0 while data is available, and read() is
```

```
   successful */
while ((len = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
    if (len == -1){
        perror("write");
        return 1;
    }
    wlen = 0;
    do {
        if ((n = write(STDOUT_FILENO, &buf[wlen], len-wlen)) == -1) {
            perror("write");
            return 1;
        }
        wlen += n;
    } while (wlen < len);
}

/* len was <= 0; If len = 0, no more data is available.
   Otherwise, an error occurred. */
if (len < 0) {
    perror("read");
    return 1;
}

return 0;
}
```

und erklären Sie seine Wirkungsweise Zeile für Zeile. Warum unterscheidet es sich von `cat1.c`?

Aufgabe 5. *cat.c aus coreutils*

Besorgen Sie sich von

ftp://alpha.gnu.org/gnu/fetish/

die neuste Version der `coreutils`. Dekomprimieren und entpacken Sie die Datei. Wechseln Sie ins Verzeichnis `coreutils-5.3.0/src` und studieren Sie die Datei `cat.c`:

Ist hier die Problematik der „interrupted system calls“ richtig berücksichtigt? Welche zusätzlichen Funktionalitäten zum `cat` der vorigen Aufgabe sind hier eingebaut?