Tätigkeiten während des ersten praktischen Studiensemesters

24. Januar 1992

Hubert Feyrer

- Data Dictionary
  - Aufgabenstellung
  - Implementierung
- Operatortätigkeiten
  - zap

Meta-Daten

```
Name   : Karl_________________
Vorname: Huber________________
Strasse: Hinterm Berg________
PLZ    : 4321
Ort    : Grossberg__

Name   : ____________________       (20 alnum)
Vorname: ____________________       (20 alnum)
Strasse: ____________________       (20 alnum)
PLZ    : ____                       (4 num)
Ort    : __________                 (10 alnum)
```

Informationen über Satzarten-Köpfe

- Das letzte Änderungsdatum des Macros
- Der Name der Datei, in der die jeweiligen Sätze abgestellt werden (`NAME=`, `NAME2=`)
- Den Dateityp: SAM, ISAM
- Die Satzlänge
- Schlüssellänge und -position
- Der Name des Macros, mit dem die jeweilige Satzart in Anwendungsprogrammen eingebunden werden

---

Informationen über Satzarten-Felder

- Logische Nummer des Feldes im Satz
- Symbolbezeichnung
- Feldbezeichnung
- Stellenzahl

- Datentyp: C (alphanumerisch) oder P (numerisch)
- Position im Satz
- Anzahl Nachkommastellen
- Gruppierungskennzeichen

Informations-Blöcke in MS-Macros

```
            .
            .
            .
L077        DFFLAB      NAME=IMOS0000,NAME2=SMLDRUCK,                F
                        ORG=ISAM,RES=1000,RECL=40,                   F
                        KEYL=13,KEYP=0
*
            .
            .
            .
SA077       DS          0L44
*
SA077K1     DC          L1C' '          "SML-TYP",E=M,Z
SA077K2     DC          L8C' '          "SML-SCHLÜSSEL"
SA077K3     DC          L4P'0'          "LFD. ZEILENNR."
*
SA077MK     DC          L3C' '          "KURZKENNUNG"
SA077ZE     DC          L4P' '          "ZEILE"
SA077SP     DC          L2P' '          "SPALTE"
SA077LA     DC          L2P'0'          "LÄNGE"
            .
            .
            .
SA077XXX    DC          A'SA077K1,SA077K2,SA077K3'
            DC          A'SA077MK,SA077ZE,SA077SP,SA077LA'
            .
            .
            .
            MEND
            MODEND
```

KILL & ZAP (1)

```
% ps -u hubert
   PID TTY      TIME COMMAND
 29712  03       0:02 sh
 29181  a12      0:02 sh
 29229  04       0:02 sh
 29197  a12      0:01 csh
 29728  03       0:00 csh
 24676  ?        0:02
 29732  03       0:00 cf
 24920  ?        0:01
 28648  04       0:01 uimos.ru
 29746  04       0:00 csh
 29773  a12      0:00 ps
```

```
% kill 28648
```

```
% kill -9 28648
```

```
% ps -u hubert | fgrep sh
 29712  03       0:02 sh
 29181  a12      0:02 sh
 29229  04       0:02 sh
 29197  a12      0:01 csh
 29728  03       0:00 csh
 29746  04       0:00 csh
```

```
% ps -u hubert | fgrep sh | cut -c1-7
 29712
 29181
 29229
 29197
 29728
 29746
```

```
% kill -9 ‘ps -u hubert | fgrep sh | cut -c1-7‘
```

KILL & ZAP (2)

```
#
# Ein kleines Utility, um das Abbrechen von Programmen
# zu vereinfachen.
#                                                  H. Feyrer
#

if [ $# -eq 1 ]
then
    if [ `who | fgrep -i $1 | wc -l` -eq 1 ]
    then
        ps -u $1 | fgrep sh
        kill -9 `ps -u $1 | fgrep sh | cut -c1-7`
    else
        echo 'Sie sind an mehreren Terminals eingeloggt: '
        ps -u $1 | cut -c7-13,22-99
        echo
        echo -n 'Bitte Terminal angeben: '
        read t

        ps -u $1 -t $t | fgrep sh
        kill -9 `ps -u $1 -t $t | fgrep sh | cut -c1-7`
    fi
else if [ $# -eq 2 ]
then
        ps -u $1 -t $2 | fgrep sh
        kill -9 `ps -u $1 -t $2 | fgrep sh | cut -c1-7`
else
        # Usage
        echo "Aufruf:  $0 Name [ Terminal ]"
        echo Wird keine Terminal-Bezeichnung angegeben, werden
        echo alle aktiven sh-Prozesse des jeweiligen Benutzers
        echo gekillt, ansonsten nur die auf dem angegebenen
        echo Terminal.
    fi
fi
```

KILL & ZAP (3)

```
%
% zap
Aufruf:  zap Name [ Terminal ]
Wird keine Terminal-Bezeichnung angegeben, werden alle aktiven
sh-Prozesse des jeweiligen Benutzers gekillt, ansonsten nur die
auf dem angegebenen Terminal.
%
% zap hubert
Sie sind an mehreren Terminals eingeloggt:
 TTY   COMMAND
  a12   sh
  ?
  a12   csh
  04    csh
  ?
  ?
  a12   csh
  a12   tee
  a12   sh
  a12   sh
  a12   cut
  a12   ps

Bitte Terminal angeben: a12
 28025  a12      0:02 sh
 28055  a12      0:02 csh
 29141  a12      0:01 csh
 29161  a12      0:00 sh
 29168  a12      0:00 sh

scodev

Welcome to SCO System V/386

scodev!login:
```

Der Vortrag besteht aus zwei Teilen: Zum einen eine kurz gehaltene Erläuterung der eigentlichen Praktikumstätigkeit, der Erstellung des Data Dictionarys. Dies kann jedoch im Praktikumsbericht weitaus detaillierter nachgelesen werden.

Gemäß den Richtlinien zum Praktikum sollten auch eine oder mehrere Wochen des Praktikums aus Operatortätigkeiten bestehen. Der zweite Teil dieses Vortrags wird sich mit etwas größerer Genauigkeit diesem Themenbereich widmen.

1 Data Dictionary

Die Hauptbeschäftigung im Praktikum war die Erstellung eines Data Dictionarys oder Satzart-Verzeichnisses.

1.1 Aufgabenstellung

Das von der Firma UBG produzierte Produktions-Planungs- und -Steuerungs-System (PPS-System) "IMOS" enthält eine Vielzahl verschiedener Datenstrukturen, sog. "Satzarten", in denen die Strukturen der zu verwaltenden Daten gespeichert werden. Die Satzarten werden in Include-Dateien, den "MS-Macros", definiert. Im Prinzip kann die Form, in der eine Satzart definiert wird, mit einer `struct`-Anweisung der Sprache C oder einem `record` in PASCAL verglichen werden, jedes Satzart-Feld entspricht dabei einem Feld der `struct`/des `record`s.

Die Aufgabenstellung im Praktikum bestand darin, ein Programm zu schreiben, das aus den in Assembler geschriebenen MS-Macros ein *Data Dictionary* erstellt, d. h. es sollen Daten über die in den MS-Macros enthaltenen Datenstrukturen gesammelt und zentral abgespeichert werden.

FOLIE: **Beispiel DD**

Im vorliegenden Fall interessieren zum einen allgemeine Informationen über die Datenstrukturen, z. B.

FOLIE: **SAKOP**

- Das letzte Änderungsdatum des Macros
- Der Name der Datei, in der die jeweiligen Sätze abgestellt werden (`NAME=`, `NAME2=`)
- Den Dateityp: SAM, ISAM (`ORG=`)
- Die Satzlänge (`RECLEN=`)
- Schlüssellänge und -position (`KEYL=`, `KEYP=`)
- Der Name des Macros, mit dem die jeweilige Satzart in Anwendungsprogrammen eingebunden werden (3. Zeile)

FOLIE: **SAFEL**

Außer diesen allgemeinen Informationen interessieren die folgenden Angaben:

- Logische Nummer des Feldes im Satz (→Ausgabesteuerung)
- Symbolbezeichnung (Label)
- Feldbezeichnung (Text, `E=`)
- Stellenzahl (aus Typ, Breite)
- Datentyp: C oder P
- Position im Satz ($\sum$ Bytes)
- Anzahl Nachkommastellen (aus Kommentar, `NK=`)
- Gruppierungskennzeichen (Kommentar: `G=`)

Sie werden für jedes einzelne Feld einer Satzart-Definition abgestellt.

Desweiteren existiert eine dritte Satzart, die Erläuterungen zu den jeweiligen Feldern enthält.

1.2 Implementierung

Das Programm wurde auf dem CTM-Rechner der Fa. UBG geschrieben. Die dabei verwendete Sprache war Assembler. Da der Rechner keinen der bekannten Intel- oder Motorola-Prozessoren besitzt, mußte zuerst der Assembler selbst gelernt werden. Dies geschah mit Hilfe des Manuals und durch Auskünfte verschiedener Mitarbeiter.

Bei der Implementierung wurde auf einen modularen Aufbau geachtet, was sich auch bei späteren Änderungsanforderungen bezahlt machte. Das Programm setzt auf mehreren Basis-Funktionen auf, die v. a. für die Ein-/Ausgabe verantwortlich sind.

Die Module des eigentlichen Macro-Scanners wurden sofort, nachdem sie konzipiert und implementiert waren, ausgiebig getestet. Dieses Vorgehen erleichterte die Endmontage des Scanners, da die einzelnen Routinen praktisch nur noch zusammengesetzt werden mußten.

Für jeden der Folgenden Blöcke eines Macros existiert ein eigenes Unterprogramm, das die entsprechenden Informationen herausliest:

FOLIE: **Macro-Blöcke**

.
.
.

```
L077        DFFLAB      NAME=IMOS0000,NAME2=SMLDRUCK,                F
                        ORG=ISAM,RES=1000,RECL=40,                   F
                        KEYL=13,KEYP=0
*
            .
            .
            .
SA077       DS          0L44
*
SA077K1     DC          L1C' '          "SML-TYP",E=M,Z
SA077K2     DC          L8C' '          "SML-SCHLÜSSEL"
SA077K3     DC          L4P'0'          "LFD. ZEILENNR."
*
SA077MK     DC          L3C' '          "KURZKENNUNG"
SA077ZE     DC          L4P' '          "ZEILE"
SA077SP     DC          L2P' '          "SPALTE"
SA077LA     DC          L2P'0'          "LÄNGE"
            .
            .
            .
SA077XXX    DC          A'SA077K1,SA077K2,SA077K3'
            DC          A'SA077MK,SA077ZE,SA077SP,SA077LA'
            .
            .
            .
            MODEND
```

Zum Austesten stand der Debugger `DMON1` zur Verfügung. Dabei handelt es sich jedoch nicht um eine Source-Level-Debugger. Vielmehr muß man stets ein (aktuelles!) Listing zur Hand haben, um den Befehl identifizieren zu können, an dem der Befehlszähler gerade steht. Natürlich ist es mit dem `DMON1` auch möglich, Breakspoints zu setzen, Register und Speicherinhalte zu betrachten sowie diese zu verändern.

2 Operatortätigkeiten

Anstatt hier näher auf die Programmierung des Macro-Scanners einzugehen soll nun auf die Tätigkeiten im Bereich System-Betreuung näher eingegangen werden.

2.1 `zap`

Da mein Rechner in der Nähe des UNIX-Entwicklungsrechners stand, fiel mir, daß die Konsole dieses Rechners meist dazu verwendet wird, um "hängengebliebene" Programme zu killen, d. h. dem Betriebssystem

ein Signal zu senden, um das Programm abzubrechen. Unter UNIX braucht man dazu die Prozeßnummer (Process ID, PID) des abzubrechenden Programmes.

Für einen bestimmten Benutzer erhält man die Prozeßnummern der momentan laufenden Prozesse mit dem folgenden Kommando:

```
% ps -u hubert
   PID TTY      TIME COMMAND
 29712  03       0:02 sh
 29181  a12      0:02 sh
 29229  04       0:02 sh
 29197  a12      0:01 csh
 29728  03       0:00 csh
 24676  ?        0:02
 29732  03       0:00 cf
 24920  ?        0:01
 28648  04       0:01 uimos.ru
 29746  04       0:00 csh
 29773  a12      0:00 ps
```

Nachdem man die Prozeßnummer des zu killenden Programms aus der ersten Spalte bestimmt hat, sendet man ein Signal an den Prozeß, sich zu beenden. Dies geht mit Hilfe des `kill`-Kommandos, woher auch der Vorhang seinen Namen hat. Um z. B. das Programm `uimos.run` abzubrechen, ist folgende Eingabe nötig:

```
% kill 28648
```

Dieses Kommando sendet das Signal #15 (SIGTERM), es gibt dem Programm also noch die Möglichkeit, sich standardgemäß zu beenden und evtl. belegte Resourcen wieder freizugeben. Ignoriert ein Programm das Signal, so sendet man das Signal #9 (SIGKILL) senden. Dieses Signal kann nicht ignoriert oder abgefangen werden. Man gibt dazu die entsprechende Signalnummer als Argument des `kill`-Befehls an:

```
% kill -9 28648
```

Dieser Befehlsablauf `ps -u ...` — `kill ...` kann automatisiert werden. Außerdem reicht es aufgrund der Prozeßhierarchie unter UNIX, den jeweiligen Shell-Prozell des Anwenders zu terminieren, da damit alle von der Shell gestarteten Programme ebenfalls automatisch beendet werden.

Die Shell-Prozesse können aus der vom `ps`-Kommando angegebenen Liste mit Hilfe des `grep`-Befehls herausgefiltert werden. Für das vorliegende Problem reicht sogar `fgrep`, eine eingeschränkte, dafür aber schnellere Version von `grep`. Die Verkettung der Ausgabe von `ps` mit der Eingabe von `fgrep` erfolgt mit Hilfe von (asynchronen) Pipes:

```
% ps -u hubert | fgrep sh
 29712  03       0:02 sh
 29181  a12      0:02 sh
 29229  04       0:02 sh
```

```
 29197  a12      0:01 csh
 29728  03       0:00 csh
 29746  04       0:00 csh
```

Die in den Spalten 1 bis 7 stehende Prozessnummer kann mit dem Programm `cut` herausgeschnitten werden:

```
% ps -u hubert | fgrep sh | cut -c1-7
 29712
 29181
 29229
 29197
 29728
 29746
```

Die so eliminierte Prozessnummer kann nun mit dem ‘-Konstrukt direkt in den `kill`-Befehl eingesetzt werden. Dabei werden die Zeilentrenner in Worttrenner (Leerzeichen) umgewandelt:

```
% kill -9 ‘ps -u hubert | fgrep sh | cut -c1-7‘
```

Es ist einfacher, die Login-Shell zu killen und sich danach neu einzuloggen, als nur die Anwendung anzubrechen, da IMOS unter Curses läuft und dabei die Terminal-Einstellungen verändert werden. Diese werden vom `getty`-Kommando, welches für das Einloggen ins System verantwortlich ist, automatisch zurückgesetzt.

Zusätzlich zum Argument “`-u`” kann mit der Option “`-t`” die auf einem bestimmten Terminal laufenden Programme abgefragt werden. Dies ist wichtig, wenn ein Benutzer an mehreren Bildschirmen gleichzeitig eingeloggt ist, aber nur einer durch ein fehlerhaftes Programm blockiert ist.

Werden die oben genannten Befehle in ein Shell-Script geschrieben, das unter der (weit verbreiteten) Bourne-Shell läuft, so ergibt sich folgenden Shell-Script:

```
%
% cat zap
#
# Ein kleines Utility, um das Abbrechen von Programmen
# zu vereinfachen.
#                                                  H. Feyrer
#

if [ $# -eq 1 ]
then
    if [ ‘who | fgrep -i $1 | wc -l‘ -eq 1 ]
    then
        ps -u $1 | fgrep sh
        kill -9 ‘ps -u $1 | fgrep sh | cut -c1-7‘
    else
```

```
        echo 'Sie sind an mehreren Terminals eingeloggt: '
        ps -u $1 | cut -c7-13,22-99
        echo
        echo -n 'Bitte Terminal angeben: '
        read t

        ps -u $1 -t $t | fgrep sh
        kill -9 `ps -u $1 -t $t | fgrep sh | cut -c1-7`
    fi
else if [ $# -eq 2 ]
then
        ps -u $1 -t $2 | fgrep sh
        kill -9 `ps -u $1 -t $2 | fgrep sh | cut -c1-7`
else
        # Usage
        echo "Aufruf:  $0 Name [ Terminal ]"
        echo Wird keine Terminal-Bezeichnung angegeben, werden
        echo alle aktiven sh-Prozesse des jeweiligen Benutzers
        echo gekillt, ansonsten nur die auf dem angegebenen
        echo Terminal.
    fi
fi

%
```

Ein Beispiellauf ergibt folgendes Ergebnis:

```
%
% zap
Aufruf:  zap Name [ Terminal ]
Wird keine Terminal-Bezeichnung angegeben, werden alle aktiven
sh-Prozesse des jeweiligen Benutzers gekillt, ansonsten nur die
auf dem angegebenen Terminal.
%
% zap hubert
Sie sind an mehreren Terminals eingeloggt:
 TTY   COMMAND
  a12   sh
  ?
  a12   csh
  04    csh
  ?
  ?
  a12   csh
  a12   tee
  a12   sh
```

```
  a12   sh
  a12   cut
  a12   ps

Bitte Terminal angeben: a12
 28025  a12      0:02 sh
 28055  a12      0:02 csh
 29141  a12      0:01 csh
 29161  a12      0:00 sh
 29168  a12      0:00 sh

scodev

Welcome to SCO System V/386

scodev!login:
```

Das selbe Ergebnis hätte man auch mit dem Befehl

```
% zap hubert a12
```

erreichtn können. Ein weiterer Punkt, der bei der Benutzung des Programmes nicht außer Aucht gelassen werden sollte, ist die Kennung, unter der zum einen das Programm läutf, das gekillt werden soll, zum anderen die Benutzerkennung, unter der `zap` gestartet wird.

Startet man `zap` unter der selben Kennung wie das fehlerhafte Programm, so wird man wenig Schwierigkeiten haben, dieses zu entfernen. Sollte man jedoch unter einer anderen Kennung eingeloggt sein, so wird es nicht gelingen, das Programm abzubrechen, da unter UNIX nur der Besitzer eines Prozesses diesen abbrechen darf.

Keine Regel ohne Ausnahme: Unter UNIX gibt es einen privilegierten Benutzer mit dem Namen `root` (oft auch als Superuser bezeichnet), der jeden Prozeß abbrechen kann.

Läuft `zap` unter dessen Benutzerkennung, ist darauf zu achten, daß die richtigen Werte angegeben werden, da sonst die Programme anderer Benutzer angebrochen werden. Dieses Risiko ist jedoch bei `zap` wesentlich geringer als bei der bisherigen Verwendung der Kombination `ps`–`kill`.

2.2 check

```
% cat check
#
# Ueberpruefen, wieviel Speicher von einer Datei-Art auf Platte
# belegt wird. Aufruf:
#
# check Verzeichnis 'Datei-Muster'
#
```

```
case $# in
0|1)  echo Usage: $0 directory file-pattern
      exit 0
esac

a=0
for i in `find $1 -name "$2" -exec ls -l {} \;      \
          | fgrep -v total                          \
          | cut -c33-40`
do
    a=`expr $a + $i`
done
echo "$1/.../$2" : $a Bytes.

%
% find . -name '*.c' -print
./test1.c
./sprog_dir/sam.c
./sprog_dir/sprog.c
%
%
% find . -print
.
./test1.c
./check.log
./sprog_dir
./sprog_dir/sprog.o
./sprog_dir/sam.o
./sprog_dir/sprog.tex
./sprog_dir/sam.h
./sprog_dir/sam.c
./sprog_dir/sprog.c
./sprog_dir/makefile
% find . -name '*.c' -exec ls -l {} \;
-rw-rw-rw-   1 hubert   group         20 Jan 20 12:39 ./test1.c
-rw-rw-rw-   1 hubert   group      25970 Jan 20 12:40 ./sprog_dir/sam.c
-rw-rw-rw-   1 hubert   group       8284 Jan 20 12:40 ./sprog_dir/sprog.c
% check .
Usage: check directory file-pattern
% l
total 8
-rw-rw-rw-   1 hubert   group       1541 Jan 20 12:48 check.log
drwxrwxrwx   2 hubert   group        144 Jan 20 12:43 sprog_dir
-rw-rw-rw-   1 hubert   group         20 Jan 20 12:39 test1.c
% sh
$ check . *.c
```

```
./.../test1.c : 20 Bytes.
$ check . '*.c'
./.../*.c : 34274 Bytes.
$
```