Hochschule Augsburg
Fakultät für Informatik

Sommersemester 2015

Prüfung

Prüfungsfach:	Systemnahe Programmierung
Datum/Uhrzeit:	8. Juli 2015 / 13:30 Uhr
Raum:	wie angeküundigt
Prüfer:	Dr. Hubert Högl
Dauer:	**60** Minuten
Hilfsmittel:	keine

Hinweise:

1. Dieses Angabenblatt hat auch eine **Rückseite**! Bitte sofort überprüfen.
2. **Schreiben** Sie bitte **nicht** auf das **Angabenblatt**. Verwenden Sie für Ihre Antworten die separat ausgeteilten Bögen. **Die Angaben dürfen Sie behalten.**
3. Schreiben Sie **nicht mit Bleistift**.

Viel Glück!

Aufgabe 1 (4 Punkte)

Welche elementaren Datentypen kennt man in der Assemblerprogrammierung? Auf welche Weise könnte man die folgenden Daten interpretieren?

```
2B332C30
```

Aufgabe 2 (4 Punkte)

Schreiben Sie den Assembler-Code hin, um mit der Funktion `write()` aus der C Standardbibliothek einen String Ihrer Wahl auf die Standardausgabe auszugeben. Hier sind die Aufrufparameter von `write`:

```
write(int fd, char *buf, int count);
```

Aufgabe 3 (5 Punkte)

Punkte: a) 2, b) 3

Das Manual zum `execve()` Systemaufruf (`man 2 execve`) beschreibt den Aufruf wie folgt:

```
int execve(char *filename, char *argv[], char *envp[]);
```

Der Aufruf hat die Nummer 11.

a) Der Systemaufruf wird in C als Funktion aufgerufen, in Wirklichkeit ist es aber ein Software Interrupt. Wie passt das zusammen?

b) Beschreiben Sie den tatsächlichen Systemaufruf in Assembler. Die Schreibweise `char *ar[]` bezeichnet ein Array `ar` im Speicher, das als Elemente Zeiger auf Strings beinhaltet. Der erste Parameter `filename` ist ein Zeiger auf einen String.

Aufgabe 4 (12 Punkte)

Punkte: a) 2, b) 1, c) 4, d) 2 , e) 2, f) 1

Der folgende Auszug aus dem Hauptspeicher beginnt bei der konstanten Adresse `mem`. Nach oben hin steigen die Adressen an. In jeder Zeile sind 8 Byte enthalten. Rechts neben der Tabelle steht die ASCII Darstellung der Speichertabelle.

```
mem+32:   00 00 00 00 00 00 00 00
mem+24:   00 00 00 00 00 00 00 00
mem+16:   00 00 00 00 00 00 00 00
mem+8:    00 00 00 00 00 00 00 00
mem:      41 42 43 00 97 a0 bf 19  ABC.....
          -----------------------
          0  1  2  3  4  5  6  7
```

a) Führen Sie den folgenden Maschinenbefehl aus. Die Register sind wie folgt gesetzt: `eax = 32`, `ebx = 1`.

```
movb    $0x55,  mem(%eax, %ebx, 4)
```

b) Ab Adresse `mem` findet man den String `ABC`. Welche Bytes gehören zu diesem String?

c) Der String `ABC` ab Adresse `mem` soll an die Stelle `mem+8` kopiert werden. Schreiben Sie in Assembler eine Funktion `scopy(a1, a2)` mit zwei Parametern, die byteweise einen String von Adresse `a1` nach Adresse `a2` kopiert.

d) In welcher Weise könnte man den String noch im Speicher ablegen? Tragen Sie Ihren Vorschlag ab Adresse `mem+16` in den Speicher ein.

e) Ihr Programm schreibt mit einer `movl` Operation an die Adresse `mem+24` die Zahl `0x12345678` in den Speicher. Schreiben Sie die Zahl in die Speichertabelle.

f) Wie können Sie mit dem `gdb` Debugger die Speichertabelle ausgeben? Schreiben Sie das dazu nötige Kommando hin.

Aufgabe 5 (13 Punkte)

Sehen Sie sich folgenden Assembler-Quelltext an. Die drei Punkte `...` stehen für beliebigen Code, der uns nicht interessiert. Beantworten Sie bitte folgende Fragen:

1. Beschreiben Sie, was an den Stellen (1) bis (7) gemacht wird. Bitte keine trivialen Bemerkungen, sondern die übergeordnete Absicht des Codes deutlich machen.

2. Zeichnen Sie den Stack direkt nach der Ausführung von Zeile 5. Zeichnen Sie auch Framepointer und Stackpointer ein.

3. Wie greift man innerhalb der Funktion `tuwas()` auf die lokalen Daten zu? Nehmen Sie an, dass die 8 Byte aus zwei Integer (long) Werten bestehen. Schreiben Sie die Framepointer-relative Adressierung für den Integer mit der kleineren Adresse hin.

```
    pushl $2              # (1)
    pushl $4              # (1)
    call tuwas            # (2)
    addl $8, %esp         # (3)
    ...

tuwas:
    pushl %ebp            # (4)
    movl %esp, %ebp       # (4)
    subl %esp, 8          # (5)
    ...
    movl %ebp, %esp       # (6)
    popl %ebp             # (6)
    ret                   # (7)
```

Aufgabe 6 (4 Punkte)

Diese Frage dreht sich um das dynamische Linken von Objektdateien. Ihr Modul `main.o` soll aus der C Standardbibliothek die Funktion `printf()` zur Laufzeit verwenden.

1. Welche Komponente muss die ausführbare Datei enthalten, damit das dynamische Linken möglich wird?

2. Beschreiben Sie die einzelnen Schritte die beim dynamischen Linken gemacht werden. Verwenden Sie eine Skizze, so wie wir das in der Vorlesung gemacht haben.

3. Schreiben Sie die notwendigen Kommandozeilen für den GNU C Compiler hin.

Aufgabe 7 (4 Punkte)

Schreiben Sie in C **oder** Assembler ein Programm, das eine beliebig grosse Binärdatei in eine andere Datei kopiert, ähnlich wie der *copy* Befehl:

```
cp <infile> <outfile>
```

Aufgabe 8 (4 Punkte)

Was versteht man unter **robusten Programmen**, so wie es im Kapitel 7 im Buch von Bartlett steht? Beantworten Sie die folgenden Punkte:

a) Warum ist das **Testen** eines Programmes wichtig?

b) Was ist **error handling**?

c) Was wird im Programm `add_year()` (Kap. 7) gemacht, um es robuster zu machen?

Aufgabe 9 (4 Punkte)

Übertragen Sie die folgende C Funktion nach Assembler.

```
int vergleich(int a, int b)
{
    if (a == b) {
        return 1;
    }
    else {
        return 0;
    }
}
```

Aufgabe 10 (4 Punkte)

Sehen Sie sich die folgende GDB Sitzung an:

```
GNU gdb 6.8-debian

(gdb) file main
Reading symbols from /home/hhoegl/prog-3-1/main...done.
(gdb) list
1   .section .text
2   .globl _start
3
4   _start:
5       movl $1, %eax
6       movl $0, %ebx
7   b1:
8       int $0x80

(gdb) br b1
Breakpoint 1 at 0x804805e: file main.s, line 8.

(gdb) run ene mene miste
Starting program: /home/hhoegl/prog-3-1/main ene mene miste

Breakpoint 1, b1 () at main.s:8
8       int $0x80
```

```
Current language:  auto; currently asm

(gdb) x/5xw $esp
0xbfacd0c0: 0x00000004  0xbfacd748  0xbfacd763  0xbfacd767 0xbfacd76c

(gdb) x/s 0xbfacd748
0xbfacd748:  "/home/hhoegl/prog-3-1/main"
...
```

1. Kommentieren Sie die einzelnen gdb Ein- und Ausgaben.
2. Was sehen Sie, wenn Sie die weiteren Werte 0xbfacd763, 0xbfacd767 und 0xbfacd76c als Argumente an `x/s` übergeben?