Vorlesung Systemnahe Programmierung – WS 2014/15

Prof. Dr. Matthew Smith, Dr. Matthias Frank, Sergej Dechand

4. Übungszettel

Ausgabe: Mi 29.10.2014; Abgabe bis zum (Freitag) 07.11.2014, 23:59 Uhr
Die Vorführung erfolgt in der Woche vom 10.11.2014 bis zum 14.11.2014 in Ihrer Übungsgruppe.

Alle Programme müssen im Poolraum unter Linux kompilierbar, lauffähig und ausreichend kommentiert sowie mit einem Makefile versehen sein. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Die Abgabe erfolgt mittels Ihres SVN-Repositorys jeweils in dem Verzeichnis /tag/AufgabeXY.

Aufgabe 11: Fehlersuche in Programmen (4 Punkte)
In das folgende Programm haben sich einige syntaktische und semantische Fehler eingeschlichen, die Sie erkennen und korrigieren sollen.
Finden und korrigieren Sie die vorhandenen Fehler. Beseitigen Sie alle Warnungen (-Wall), die beim compilieren erzeugt werden. Protokollieren Sie die Fehler in einer Datei namens *README.txt*. Das Programm können Sie auch von unserer Homepage herunterladen:

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

int palindrom_check1(const char *st1)
{
  int st1_len = strlen(st1), i;

  for(i = 0; i < st1_len / 2; i++)
    if (toupper(st1[i]) != toupper(st1[st1_len -i])) return 0;

  return 1;
}

int main(void) {
  char *st1, *st2;
  int st1_len, i;
  char *exit = 'exit';

  st1 = malloc(100);
  st2 = malloc(100);

  printf("Zum Beenden exit als Eingabe angeben. \n");
  while (1) {
    printf("Eingabe: ");
    scanf("%s", st1);
    if (strcmp(st1, exit) == 0) return 0;

    st1_len = strlen(st1);
```

```
    for (i=0; i<st1_len; i++) {
      st2[i] = toupper(st1[st1_len -1 -i]);
    }

    if (palindrom_check1(st1)) printf("1. Test: Palindrom \n");
    else printf("1. Test: Kein Palindrom \n");

    if (st1 == st2) printf("2. Test: Palindrom\n");
    else printf("2. Test: Kein Palindrom\n");
  }
  return 0;
}
```

Aufgabe 12: Byte-Order / Domain Name System (4 Punkte)

Schreiben Sie eine C-Funktion *big_endian()*, die zurückliefert, ob das aktuelle System in Big-Endian-Byteorder arbeitet oder nicht. Benutzen Sie in dieser Funktion keine weiteren Funktionsaufrufe oder Makros. Entwickeln Sie weiterhin eine Funktion *char *my_inet_ntoa(struct in_addr in)*, welche genau wie *inet_ntoa(...)* IP-Adressen in Network-Byteorder in Dotted-Decimal-Notation umwandelt (siehe Manpages zu *inet_ntoa()*). Interpretieren Sie dazu die IP-Adresse als 32 Bit Integerzahl und berechnen Sie die einzelnen Bytes mit arithmetischen Operationen. Achten Sie dabei auf die Byteorder des jeweiligen Systems. Ihre Funktion könnte z.B. folgendermaßen beginnen:

```
char *my_inet_ntoa(struct in_addr in)
{
  /* interpret IPv4-Address as 32bit int */
  unsigned int addr = *(unsigned int *)&in;
```

Schreiben Sie nun ein Programm, welches als erstes die Byteorder des aktuellen Systems ausgibt (Little- bzw. Big-Endian). Weiterhin soll das Programm über die Kommandozeile einen Hostnamen entgegennehmen und eine zugehörige IP-Adresse ermitteln. Benutzen sie dabei statt der häufig verwendeten Funktion *gethostbyname(...)* die Kommunikationsprotokoll-unabhängige Funktion *getaddrinfo(...)* (siehe Manpages zu *getaddrinfo*). Diese Adresse soll nun einmal mittels der Funktion *inet_ntoa(...)* und ein weiteres Mal unter Benutzung Ihrer Funktion *my_inet_ntoa(...)* ausgegeben werden. Vergessen Sie dabei nicht, den von *getaddrinfo(…)* allokierten Speicher mittels *freeaddrinfo(…)* wieder freizugeben.

Schreiben Sie ein weiteres Programm, das zu einer IP-Adresse den zugehörigen Hostnamen ermittelt. Benutzen Sie dazu die Funktion *getnameinfo(...)*.

Die (herkömmlichen) Funktionen *gethostbyname(...)* und *gethostbyaddr(...)* sollen nicht mehr benutzt werden. Hinweise (teilw. samt Beispielen) zu den (besseren) kommunikationsprotokollunabhängigen Funktionen finden sich zB unter http://linux.die.net/man/3/gethostbyaddr.

Aufgabe 13: Nachrichtenübertragung mit UDP (4 Punkte)

Hinweis: Es wird auf einem der folgenden Übungszettel eine Aufgabe geben, die auf dieser Aufgabe aufbaut.

In dieser Aufgabe soll ein System zur Übertragung von Nachrichten implementiert werden. Das System soll aus zwei Anwendungen bestehen, eine Anwendung zum Empfangen von

Nachrichten (*Empfänger*) und eine weitere Anwendung zum Senden von Nachrichten (*Sender*). Die Kommunikation zwischen den beiden Anwendungen soll über UDP erfolgen.

Empfänger: *Dem Empfänger wird beim Start der Anwendung ein Port vom Betriebssystem zugewiesen (siehe Folie 55 aus Kapitel 1 der Vorlesung). Dieser Port soll direkt nach dem Start auf der Standardausgabe (stdout) ausgegeben werden. Benutzen Sie dazu nach dem Aufruf von* bind(...) *die Funktion* getsockname(...) *(siehe Manpages zu* getsockname*). Erhält der Empfänger eine Nachricht, gibt er die IP-Adresse, den Hostnamen und den Port des Senders sowie die übertragene Nachricht aus.*

Sender: *Dem Sender wird beim Start der Anwendung vom Benutzer mitgeteilt, an welchen Host und Port gesendet werden soll. Den eigenen Port bekommt er vom Betriebssystem zugewiesen. Das Programm soll die Möglichkeit bieten, wiederholt Textnachrichten von der Standardeingabe (stdin) entgegen zu nehmen und diese an den zu Beginn angegebenen Empfänger zu senden.*

Hinweis: Sie können Ihre Programme auch auf einem einzelnen Rechner testen, indem Sie als Hostnamen *localhost* (IP: 127.0.0.1) verwenden.

Frage: Ist es möglich, den korrekten Empfang der Nachrichten zu überprüfen, ohne einen weiteren Mechanismus zu implementieren? Begründen Sie Ihre Antwort in der Datei *Protokoll.txt*, die ebenfalls Teil Ihrer Abgabe ist.

Aufgabe 14: Zulässigkeit von Zuweisungen (2 Punkte)
Gegeben seien `feld` (ein `int`-Array) und `p` (ein `int`-Zeiger). Welche der folgenden Zuweisungen sind zulässig, welche nicht? Begründen Sie Ihre Antwort.

(a) `p = feld;`
(b) `feld = p;`
(c) `p = &feld[3];`
(d) `feld[2] = p[5];`