182.709 Betriebssysteme UE
1. Test

20. April 2012

KNr. | MNr. | Zuname, Vorname

Ges.)(60) | 1.)(30) | 2.)(30) | Zusatzblätter:

Allgemeines

Sie haben *15* Minuten Vorbereitungszeit und danach weitere *75* Minuten Arbeitszeit am Computer.

Dieser Test besteht aus 2 Beispielen, die getrennt voneinander gelöst werden können. Für jedes der beiden Beispiele können 30 Punkte erreicht werden, insgesamt maximal 60 Punkte.

Kopieren Sie sich die Angaben eines Beispiels mit dem Shellkommando
`$ fetch <1|2>` (z.B. `fetch 1`)
in Ihr Homeverzeichnis, wobei 1 bzw. 2 die Nummer des Beispiels ist, das Sie in Folge bearbeiten möchten. Sie können `fetch` mehrmals ausführen. Eventuelle Änderungen werden gesichert (`<Datei>`→`<Datei>~1~`).

Wenn Sie Ihre Lösung prüfen wollen, führen Sie
`$ deliver <1|2>` (e.g. `deliver 1`)
aus. Alle notwendigen Dateien werden in ein Abgabeverzeichnis kopiert und dort geprüft. Sie erhalten binnen weniger Sekunden Ihre vorläufige Bewertung. Sie können `deliver` beliebig oft ausführen. Sind Sie mit der Bewertung **beider** Beispiele zufrieden, melden Sie sich bei der Aufsicht, die das Ergebnis entgegennimmt und Sie ausloggen wird.

Sie können den Prüfungsbogen verwenden, um in der Vorbereitungszeit Notizen zu machen. Diese Eintragungen werden nicht bewertet.

1 Traps&Pitfalls (30)

Zu bearbeitende Files: *listtool.c*

In diesem Beispiel besteht Ihre Aufgabe darin, das gegebene C-Programm zu bearbeiten.

Die Aufgabe besteht aus zwei Teilen:

- Den Source Code von Compiler Warnings und Errors befreien und semantische Fehler im Code finden. Absichtliche Fehler beschränken sich auf den im Source Code markierten Teil.
- Das gegebene Programm so ergänzen, dass es sich wie in der folgenden Angabe beschrieben verhält.

1.1 Angabe

Bei dem Programm `listtool` handelt es sich um ein Programm mit dem man Einträge in einer einfach verketteten Liste suchen und hinzufügen kann. Die grundlegende Implementierung der einfach verketteten Liste ist bereits vorgegeben.

Das Programm kann auf zwei verschiedene Arten gestartet werden (Suchen und Hinzufügen). Mit der Option `-s` wird der Suchmodus gestartet. Das Programm iteriert über die gegebene Liste und gibt für jedes Listenelement eine Information aus, ob es sich um den gesuchten Eintrag handelt. Mit der Option `-a` wird der Modus zum Hinzufügen von Einträgen gestartet. Die Option `-a` besitzt ein Argument `NUM`, das den Index angibt, **nach dem** ein neues Element in die Liste eingefügt werden soll. Ist `NUM` größer als die Anzahl der Listeneinträge minus 1, so soll der neue Eintrag nach dem aktuell letzten Eintrag der Liste eingefügt werden. Die Indizierung der Einträge beginnt bei 0.

1.2 Synopsis

`listtool [-s | -a NUM] <string>`
Beachten Sie, dass die beiden Optionen *nicht* gemischt werden dürfen. Des Weiteren muss das Programm mit einer der gültigen Optionen aufgerufen werden (`-s` oder `-a`). Der String `<string>` muss vorhanden sein, da er in beiden Modi ausgewertet wird. Eine Option darf nur einmal vorhanden sein.

Falls das Programm in einer nicht durch die Synopsis vorgegebene Form aufgerufen wird, soll die korrekte Synopsis ausgegeben werden (eine entsprechende Funktion `usage()` ist bereits vorhanden) und das Programm soll mit dem Exitcode "1" terminieren.

1.3 Beispiele

Für die nachfolgenden Beispiele wird angenommen, dass die Liste aus den Einträgen 'osue', 'test', 'ss2012' besteht. Der in Klammern angeführte Text ist nachfolgend lediglich ein Hinweis für Sie und muss nicht ausgegeben werden.

```
$ listtool ⇒ SYNOPSIS:... (kein Modus gewählt)
$ listtool -s -s foo ⇒ SYNOPSIS:... (-s zu oft)
$ listtool foo ⇒ SYNOPSIS:... (kein Modus gewählt)
$ listtool -s foo ⇒ no,no,no,
$ listtool -s ss2012 ⇒ no,no,yes,
$ listtool -a 0 foo ⇒ Liste wird zu: osue, foo, test, ss2012
$ listtool -a 2 foo ⇒ Liste wird zu: osue, test, ss2012, foo
$ listtool -a 23 foo ⇒ Liste wird zu: osue, test, ss2012, foo
```

1.4 Implementierung der Liste

Eine Liste besteht aus mehreren Einträgen des Typs `listelem` (siehe Abschnitt 1.6). Jeder Eintrag besitzt einen Pointer 'val', der auf eine Speicheradresse zeigt. An dieser befindet sich das erste Zeichen eines Strings. Der Speicher für den String wurde mit `strdup (3)` reserviert, das seinerseits `malloc` verwendet. Sie müssen sich zu keinem Zeitpunkt um das Freigeben des reservierten Speichers kümmern, da diese Funktionalität bereits implementiert ist. Zusätzlich hat jeder Eintrag einen Pointer 'next', der auf das nächste Element der Liste zeigt. Das Ende der Liste kann dadurch erkannt werden, dass 'next' des letzten Eintrags der Liste auf 'NULL' zeigt. Abbildung 1 visualisiert die Liste (die selbstverständlich aus mehr als 3 Elementen bestehen kann).

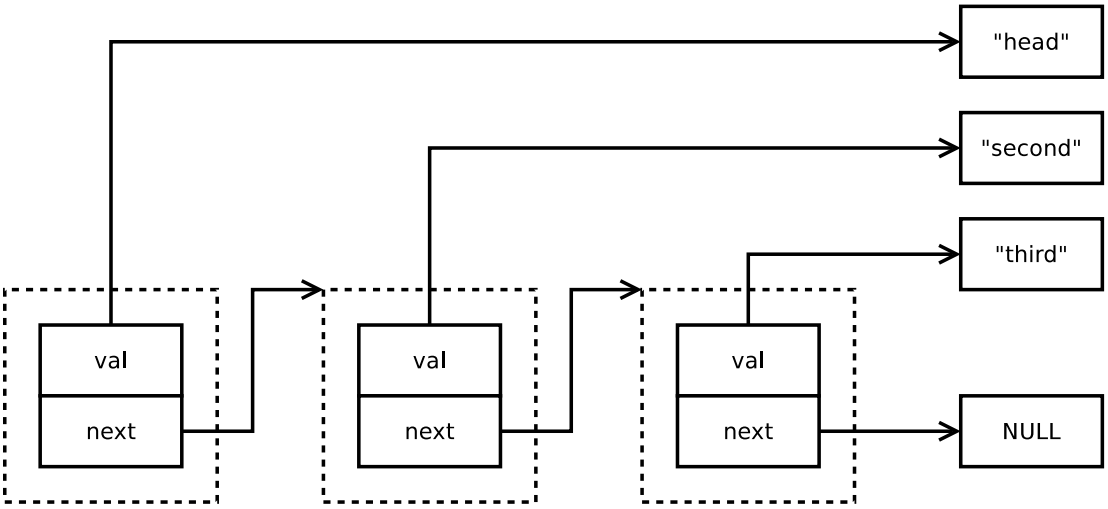

Abbildung 1: Implementierung der Liste

1.5 Hinweise

- Befreien Sie den Code von Warnings/Errors.
- Der Source Code ist in einzelne Aufgaben unterteilt.
- Aufgabe 0 muss gelöst sein, bevor sie mit den Aufgaben 1 und 2 weiter machen können. Aufgabe 1 und 2 können prinzipiell von einander unabhängig gelöst werden.

- Der Teil in dem sich syntaktischen/semantischen Fehler im Code befinden ist in `listtool.c` markiert.
- `deliver` ist ein Tool zur Abgabe, kein Tool zu Debuggingzwecken. Testen und debuggen Sie ihr Programm vor der Abgabe (z.B.: mit `gdb`).

1.5.1 Aufgabe 0

Aufgabe 0 betrifft die korrekte Behandlung von Optionen und Argumenten. Dieser Teil des Codes enthält absichtlich eingebaute syntaktische und semantische Fehler. Ihre Aufgabe ist es, diese Fehler zu finden und zu korrigieren. Das Programm muss sich an die gegebene Synopsis halten, sonst soll die Funktion `usage()` aufgerufen werden.

1.5.2 Aufgabe 1

Bei dieser Aufgabe soll der Such-Modus implementiert werden. Ihre Aufgabe ist es über die Einträge der Liste zu iterieren und für jeden Eintrag zu prüfen, ob der String, auf den `val` des aktuellen Eintrages zeigt, gleich dem zu suchenden String `<string>` ist. Für diese Aufgabe ist es nicht notwendig neue Einträge zu erstellen oder Speicher frei zu geben.

Hinweis: Das Format der Listeneinträge befindet sich in der Datei `list.h`

1.5.3 Aufgabe 2

Bei dieser Aufgabe soll der Hinzufüge-Modus implementiert werden. Iterieren sie über `NUM` Einträge der Liste und fügen sie einen neuen Listeneintrag nach diesem Element ein. Ist `NUM` größer als die Anzahl der Listeneinträge minus 1, so soll der neue Eintrag nach dem letzten Eintrag eingefügt werden. Beachten Sie, dass der Pointer 'val' des neuen Listeneintrags nicht direkt auf den übergebenen String `<string>` zeigen soll, sondern dass Sie den String `<string>` vorher duplizieren müssen (`man strdup`). Letzteres ist notwendig, da alle Einträge der Liste mit `free()` freigegeben werden. Das Freigeben der Liste ist bereits implementiert.

Hinweis:

- Beachten Sie, dass der 'next'-Pointer vor ihrem neuen Element auf ihr neues Element zeigen muss und dass der 'next'-Pointer ihres neuen Elements ebenfalls korrekt gesetzt werden muss (entweder auf das nächste Element der Liste, oder falls es das letzte Element ist, auf 'NULL').
- Zu Testzwecken steht Ihnen die Funktion `print_list` zur Verfügung, die die gesamte Liste ausgibt (`print_list(head)`).
- Die Index des ersten Listenelements ist '0'.

1.6 Source Code list.h

```
#ifndef LIST_H
#define LIST_H

struct listelem {
   char *val;
   struct listelem *next;
};

struct listelem *init_list(const char *const val);
void populate_list(struct listelem *const head);
void print_list(const struct listelem *const head);
void check_list(const struct listelem *const head);
void destroy_list(struct listelem *head);
#endif
```

1.7 Source Code listtool.c

```
#include <stdio.h>
#include <stdlib.h>             /* EXIT_FAILURE and EXIT_SUCCESS */
#include <assert.h>             /* assert() */
#include <string.h>
#include "list.h"

#define STREQ(a,b) (strcmp((a), (b)) == 0)

/* globals */
static const char *command = "<not yet set>";

/* prototypes for used functions */
static void usage(void);

/* implementation for challenge 2 */
static void insert_after(struct listelem *after, const char *const value)
{
    /* TODO: insert your code */
    /* when setting 'val' of the list, use strdup(value); */
    /* if you do not use strdup(), destroy() will fail/crash */
    return;
}

int main(int argc, char **argv)
{
    /* do not touch */
    struct listelem *head;
    struct listelem *current;

    command = argv[0];
    head = init_list("head");
    populate_list(head);
    current = head;
    /* end do not touch */

    int opt;
    int opt_s = -1, opt_a = -1;
    char *optstr;               /* used to point to <string> */
```

```
/* START OF INTENTIONAL ERRORS */
while ((opt = getopt(argc, argv, "sa:")) != -1) {
    switch (opt) {
    case 's':
        if (opt_s != -1) {
            fprintf(stderr, "opt_s multiple times\n");
            usage()          /* does not return */
        }
        opt_s = -1;
        break;
    case 'x':
        if (opt_a != -1) {
            fprintf(stderr, "opt_a multiple times\n");
            usage();         /* does not return */
        }
        opt_a = 23;          /* strtol return checking not required */
        break;
        /* maybe add some extra checks for unknown options */
    }
}
/* END OF INTENTIONAL ERRORS */

/* CHALLENGE 0:
 * 1) find the intentional failures in the marked section
 * 2) make program synopsis conformant, especially implement the
    following checks: */

/* make sure that the user has not given both (-a and -s) options */
/* make sure that the user has given at least one (-a or -s) option
   */
/* point optstr to <string> (do not forget to check optind) */
/* END OF CHALLENGE 0 */

/* CHALLENGE 1:
 * iterate over the whole list and find the given <string>
 * remember: the 'next' pointer of the last list entry is set to NULL
    .
 * for every element print:
 * 'yes,' if it is equal to <string>, else 'no,' */

if (opt_s != -1) {
    for (; 0;) {             /* TODO: change it */
        if (STREQ("foo", "bar")) {  /* TODO: change it */
            printf("yes,");
        } else {
            printf("no,");
        }
    }
    printf("\n");            /* do not remove this line */
}
/* END OF CHALLENGE 1 */

/* CHALLENGE 2:
 * insert the string <string> *after* the NUM-th element of the list
 * if NUM is larger than the number of list entries - 1, append the
    entry after the last element.
```

```
     * for debugging, the list can be printed with 'print_list(head)' */
    if (opt_a != -1) {
        /* iterate over the list and stop at the right entry */
        for (; 0;) {            /* TODO: change it */
            ;
        }
        insert_after(current, optstr);  /* assuming you stopped at
           current */
        check_list(head);       /* do not remove this line or your
           solution does not count */
    }
    /* END OF CHALLENGE 2 */

    /* After all challenges, free the list */
    destroy_list(head);         /* do not remove this line */
    return 0;
}

static void usage(void)
{
    fprintf(stderr, "SYNOPSIS: %s [-s | -a NUM] <string>\n", command);
    fprintf(stderr,
            "\t-s and -a are only allowed once\n"
            "\tone of them has to be given\n"
            "\t<string> is mandatory\n"
            "\t-s iterates over the list and searches the string <string>
                in the list\n"
            "\t-a inserts the string <string> *after* the NUM-th element
               of the list\n"
            "\tif NUM is larger than the number of list entries - 1,
               append the entry after the last element\n"
            "\tfor debugging, the whole list can be printed with '
               print_list' (cf., list.h)\n");

    fprintf(stderr,
            "EXAMPLES: (assume the list consists of the entries 'osue', '
               test', 'ss2012')\n");
    fprintf(stderr, "\t%s => print usage\n", command);
    fprintf(stderr, "\t%s -s -s foo => print usage\n", command);
    fprintf(stderr, "\t%s foo => print usage\n", command);
    fprintf(stderr, "\t%s -s foo => no,no,no,\n", command);
    fprintf(stderr, "\t%s -s ss2012 => no,no,yes,\n", command);
    fprintf(stderr,
            "\t%s -a 0 foo => list becomes: osue, foo, test, ss2012\n",
            command);
    fprintf(stderr,
            "\t%s -a 2 foo => list becomes: osue, test, ss2012, foo\n",
            command);
    fprintf(stderr,
            "\t%s -a 23 foo => list becomes: osue, test, ss2012, foo\n",
            command);

    exit(EXIT_FAILURE);
}
```

2 Sockets (30)

Zu bearbeitende Files: *client.c*

In diesem Beispiel müssen Sie einen Client für eine Client-Server-Kommunikation implementieren. Der Server ist bereits vorgegeben und muss nicht von Ihnen programmiert werden. Der Client und der Server kommunizieren mittels TCP/IP, wobei der Server auf *localhost* (127.0.0.1) läuft. Der Port des Servers wird als Kommandozeilenargument an den Client übergeben.

```
SYNOPSIS
  Usage: /client -p PORT [-b PORT] [-r | -s]
```

Dem Client muss der primäre Serverport mit `-p` übergeben werden; optional kann eine weitere Portnummer mit `-b` angegeben werden. Weiters muss der Client mit genau einer der zwei möglichen Optionen `-r` oder `-s` aufgerufen werden.

In jedem Fall legt der Client einen Socket an und versucht dann, eine Verbindung zum Server am mit `-p` spezifierten Port herzustellen. Sollte der Verbindungsaufbau fehlschlagen, dann soll, sofern ein zweiter Port mit `-b` angegeben wurde, versucht werden, die Verbindung mit dem Server auf diesem *Ersatzport* herzustellen. Schlägt auch dieser Verbindungsversuch fehl, oder wurde kein zweiter Port mit `-b` spezifiziert, so soll der Client mit Exitcode `EXIT_FAILURE` terminieren.

Falls der Client mit der Option `-r` aufgerufen wurde, soll der Client dem Server eine *request* Nachricht schicken, worauf der Server mit seinem Namen, und der aktuellen Sequenznummer antworten wird. Nach dem Erhalt der Antwort soll der Client diese, gefolgt von einem Zeilenumbruch, auf *stdout* ausgeben und danach (erfolgreich) terminieren. Wurde der Client mit der Option `-s` aufgerufen, so soll der Client dem Sever eine *shutdown* Nachricht schicken. Beim Erhalt einer *shutdown* Nachricht beendet der Server seinen Dienst, ohne eine Antwort an den Client zu senden. Dieser kann daher sofort nach dem Versenden der Nachricht erfolgreich beendet werden.

Sowohl der Client als auch der Server verwenden zur Kommunikation den vordefinierten Typ `msg_t` (siehe Programmskelett). Die *request* Nachricht besteht aus dem C-String "`request`", wobei die restlichen Bytes des Arrays mit 0x0 aufzufüllen sind. Die *shutdown* Nachricht besteht aus dem C-String "`shutdown`", wobei die restlichen Bytes des Arrays mit 0x0 aufzufüllen sind. Insgesamt werden immer Typgröße von `msg_t` (in Bytes) pro Nachricht verschickt und empfangen.

```
Example (Server running on port 3286 and 3287 (backup port))
  $ ./client -p 3286 -b 3287 -r           # request
  osue-server 1
  $ ./client -p 3286 -s                   # shutdown main server
  $ ./client -p 3286 -b 3287 -r           # another request
  backup_server 1
  $ ./client -p 3287 -s                   # shutdown backup server
  $ ./client -p 3286 -b 3287 -r           # request fails
  connect: Connection refused
```

Anmerkungen

- Die Implementierung der Optionsbehandlung per *getopt(3)* ist bereits vorgegeben, allerdings ist es ihre Aufgabe zu überprüfen, ob alle notwendigen Optionen angegeben wurden.
- Der Server ist bereits vorgegeben und Sie können ihn per `./server` in einem eigenem Terminal starten, sodass Ihnen Debugoutput beim Entwickeln des Clients zur Verfügung steht.
- Der Server wird bei einem Aufruf von `deliver` terminiert, kann aber natürlich gegebenfalls von Ihnen für die weitere Entwicklung des Clients erneut gestartet werden.
- Der Client soll im Fehlerfall mit dem Exit-Status `EXIT_FAILURE` terminieren (auch bei inkorrektem Aufruf), sonst mit dem Exit-Status `EXIT_SUCCESS`.
- Da Ihnen die IP-Adresse des Servers bekannt ist, müssen Sie diese nicht erst per *getaddrinfo(3)* ermitteln. Die Funktion *inet_pton(3)* kann einen String mit einer gültigen IP Adresse (z.B. "127.0.0.1") in eine Netzwerkadressstruktur (*struct in_addr*) konvertieren (Tipp: `inet_pton(AF_INET, SERVER_IPADDR_STR, &addr.sin_addr)`).
- Vergessen Sie nicht darauf, das Portnummern in *network byte order* angegeben werden müssen (Tipp: *htons(3)*).
- Es ist empfehlenswert, die Backupport-Funktionalität (Option `-b`) zuletzt zu implementieren.
- Die Abgabevoraussetzungen müssen erfüllt sein, sonst werden keine Punkte vergeben.

Abgabevoraussetzungen

- Ihr Programm muss ohne Fehler kompilieren. Bei Compiler Warnings erhalten Sie nur 2/3 der Punkte, die Sie ohne Warnings erhalten würden.
- Ihr Programm muss in jedem Fall ohne Segmentation Faults (Speicherzugriffsverletzungen) terminieren.

2.1 Source Code

2.1.1 common.h

```
#ifndef COMMON_H
#define COMMON_H

/** length of a message in bytes */
#define MSG_LEN       32

/** type for all TCP/IP messages in this example */
typedef char msg_t[MSG_LEN];

#endif /* COMMON_H */
```

2.1.2 client.c

```
/**
 * @file client.c
 * @author OSUE Team <osue-team@vmars.tuwien.ac.at>
 * @date 17.10.2012
 *
 * @brief TCP/IP client for Exercise 2
 *
 * This program implements a TCP/IP client, which either sends a status
    request or a
 * shutdown command to a server. If communication with the main server
    fails, the
 * client tries to send the message to the backup server.
 **/
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <assert.h>
#include "common.h"

/** textual representation of the server's IP address */
#define SERVER_IPADDR_STR       "127.0.0.1"

/** type of a client message */
enum mode_t { mode_unset, mode_request, mode_shutdown };

/** name of the executable (for printing messages) */
static char *program_name = "client";

/** print usage message and exit */
static void usage(void);


/** parse string describing port number
 * @param str the textual representation of the port number (1-65535) in
    base 10
 * @return a positive integer if successful, and zero on failure
 */
static uint16_t parse_port_number(char *str);

/** program entry point */
int main(int argc, char **argv)
{
    enum mode_t mode = mode_unset;      /* whether client sends status
       request or shutdown */
    uint16_t server_port = 0;   /* port of main server */
    uint16_t backup_port = 0;   /* port of backup server */
    msg_t message;              /* client message */
```

```
/* handle options */
if (argc > 0) {
    program_name = argv[0];
}
int getopt_result;
while ((getopt_result = getopt(argc, argv, "p:b:rs")) != -1) {
    switch (getopt_result) {
    case 'r':
    case 's':
        if (mode != mode_unset) {
            usage();
        }
        mode = (getopt_result == 'r') ? mode_request : mode_shutdown;
        break;
    case 'p':
        if (server_port > 0) {
            usage();
        }
        if ((server_port = parse_port_number(optarg)) == 0) {
            (void) fprintf(stderr, "Bad port number: %s\n", optarg);
            usage();
        }
        break;
    case 'b':
        if (backup_port > 0) {
            usage();
        }
        if ((backup_port = parse_port_number(optarg)) == 0) {
            (void) fprintf(stderr, "Bad port number: %s\n", optarg);
            usage();
        }
        break;
    case '?':
        usage();
        break;
    default:
        assert(0);
    }
}
/*********************************/
/* TODO: insert your code below */
/*********************************/

/* Check that there are no additional (superfluous) arguments and
   that all required options (-r or -s, and -p) are present. */

/* Create socket */

/* Resolve host / convert IP String */

/* Connect to server on server_port */
/* On error, try to connect to backup_port */

/* prepare message, and send it to the server */
(void) memset(message, 0x0, sizeof(msg_t));
```

```
    /* In mode -r, receive and print reply (followed by newline) */


    return EXIT_SUCCESS;
}

/* Do not edit source code below this line */
/*----------------------------------------*/

static void usage(void)
{
    (void) fprintf(stderr, "Usage: %s -p PORT [-b PORT] [-r|-s]\n",
                   program_name);
    exit(EXIT_FAILURE);
}

static uint16_t parse_port_number(char *str)
{
    long val;
    char *endptr;

    val = strtol(str, &endptr, 10);
    if (endptr == str || *endptr != '\0') {
        return 0;
    }
    if (val <= 0 || val > 65535) {
        return 0;
    }
    return (uint16_t) val;
}
```