Lehrstuhl für Connected Mobility
Fakultät für Informatik
Technische Universität München

Grundlagen von Betriebssystemen und Systemsoftware

WS 2018/19

Übung 1: Einführung

zum 22. Oktober 2018

- Die **Hausaufgaben** für dieses Übungsblatt müssen **spätestens** am **Sonntag, den 28.10.2018 um 23:59** in Moodle[1] hochgeladen worden sein.
- Zur Abgabe der Hausaufgaben ist eine Gruppe erforderlich. Bitte teilen Sie sich daher in fixe Gruppen von max. 4 Personen auf und tragen Sie diese im entsprechenden Moodle-Poll ein. Dies ist unabhängig von Ihrer Tutorium-Gruppe.
- Alle C-Programme müssen mit den folgenden Flags kompiliert werden:
  `gcc -Wall -o <progname> <prog.c>`.
- Hierzu stellen wir für jedes Übungsblatt jeweils ein Makefile bereit, das **nicht** verändert werden darf, um sicherzustellen, dass Ihre Abgabe auch korrekt von uns getestet und bewertet werden kann. Zum Kompilieren genügt der Aufruf von `make`. Wenn Sie zwischenzeitlich Änderungen vornehmen wollen, um etwa bestimmte Teile mittels `#ifdef` einzubinden und zu testen, dann kopieren Sie am besten das Makefile und modifizieren Ihre Kopie. Mit `make -f <Makefile>` können Sie dann eine andere Datei zum Übersetzen verwenden.
- Des Weiteren werden zu den Programmieraufgaben Testfälle in Form von ausführbaren Programmen bereit gestellt, sodass Sie Feedback für Ihre Lösungsansätze bekommen. Hierzu legen Sie Ihr eigenes sowie das Testprogramm im gleichen Verzeichnis ab und führen letzteres aus (`./xx_test`). Das Bestehen aller Testfälle bedeutet nicht, dass Ihre Lösung zwangsweise korrekt ist. Sie soll lediglich als Hilfestellung dienen und ein Indikator dafür sein, ob Ihr Ansatz im Groben in die richtige Richtung geht.
- Die Abgabe der Programme erfolgt als Archivdatei, die die verschiedenen Quelldateien (.{c|h}) umfasst und **nicht** in Binärform. Nicht kompilierfähiger Quellcode wird **nicht gewertet**.
- Um die Abgabe zu standardisieren enthält das Makefile ein Target "submit" (`make submit`), was dann eine Datei blatt01.tar zum Hochladen auf Moodle erzeugt.
- Damit die richtigen Dateien hochgeladen und ausgeführt werden, geben wir bei allen Übungen die jeweils zu verwendenden Dateinamen für den Quellcode und auch für das Executable an.
- Um den Notenbonus zu erhalten, muss Ihre Gruppe 80% der möglichen Gesamtpunkte am Ende des Semesters erreicht haben. Beachten Sie, dass das Plagiieren von Lösungen zum Ausschluss vom Bonus für die gesamte Gruppe führt.

[1] GBS Moodle: `https://www.moodle.tum.de/course/view.php?id=42019`

1 Vorbereitung auf das Tutorium

1.1 C-Programmierung – Einarbeitung

Lesen Sie das Tutorial *A C Primer*[2] von J. Pfoh.

Sie sollten danach ein erstes, grundlegendes Verständnis insbesondere folgender Aspekte besitzen:

- Pointer, `NULL` und die zugehörigen Operatoren `*` und `&`
- das enge Zusammenspiel von Pointern und Arrays
- `char`-Pointer respektive `char`-Arrays als Ersatz für `String`

Konkret sollten Sie ohne Recherche beantworten können, wie man die zweite Zuweisung alternativ ausführen könnte und wieso:

```
char mystring[] = "test";
mystring[1] = 'a';
```

Erfahrungsgemäß unterschätzen viele Studierende, vor allem in den Nebenfächern, den Anteil von C-Programmierung in GBS. Diese hat jedoch einen hohen Stellenwert in Übung und Klausur, insbesondere der Aspekt der Pointerarithmetik. Überfliegen Sie daher das Tutorial nicht, sondern nehmen Sie sich die Zeit für eine intensive Einarbeitung. Die eigenständige Aneignung entsprechender Programmierfertigkeiten wird auf Basis Ihres vorhandenen Java-Wissens von Ihnen erwartet. Der Schwierigkeitsgrad der Aufgaben wird graduell zunehmen.

[2] C Primer: `https://www.cm.in.tum.de/fileadmin/w00bvd/www/gbs-1819/c_primer.pdf`

1.2 Infrastruktur

Um Ihre Hausaufgaben zu programmieren und zu testen verwenden Sie bitte die unten beschriebene virtuelle Maschine. Ihre Programme müssen auf dieser Infrastruktur kompilierbar und ausführbar sein, verwenden Sie daher abweichende Infrastruktur auf eigenes Risiko.

1.2.1 Importieren der VM

Die VM (**GBS_VM.ova**[3])kann in VirtualBox[4] importiert und ausgeführt werden. Der Import ist im folgenden Bildschirmfoto abgebildet.

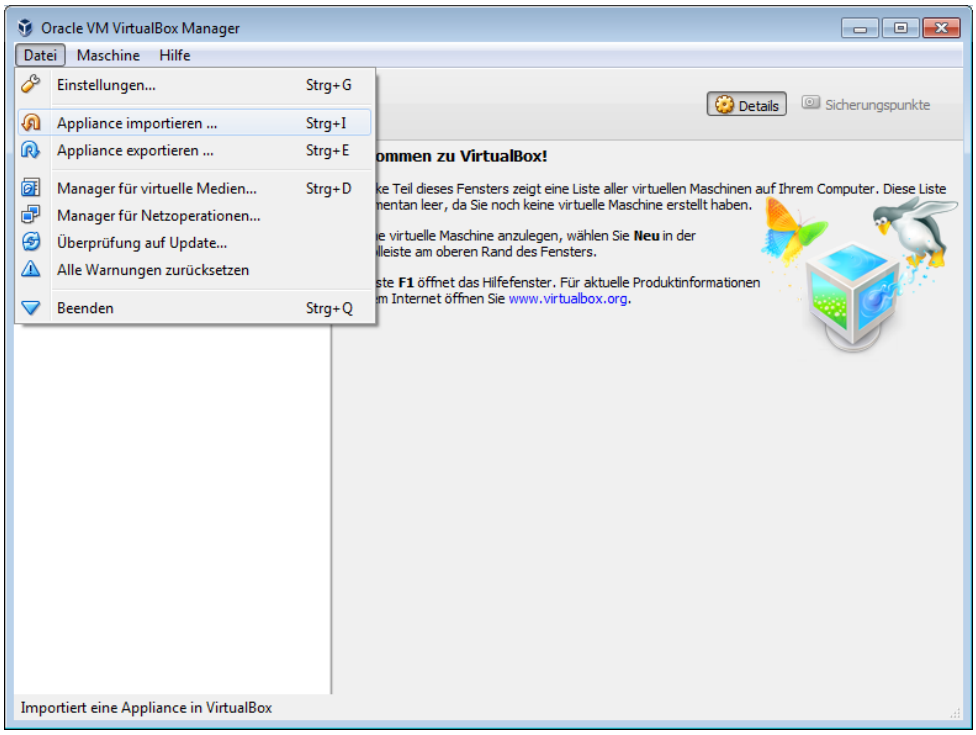

Figure 1: VM importieren

Danach kann die VM direkt gestartet werden und man kann sich nach dem Abschluss des Bootvorgangs mit dieser verbinden.

[3]VM Image: `http://gbs.cm.in.tum.de/GBS_VM.ova` (ca. 2 GB)
[4]Virtual Box: `https://www.virtualbox.org/`

1.2.2 Verbinden mit der VM

Sie können sich per SSH (Secure Shell) mit der VM verbinden. Unter Linux, macOS und dem Windows Linux Subsystem kann das direkt im Terminal, wie unten aufgelistet, durchgeführt werden:

```
ssh -p 22222 student@localhost
```

Alternativ *können Sie unter Windows mittels PuTTY* [5] *eine Verbindung zur VM aufbauen:*

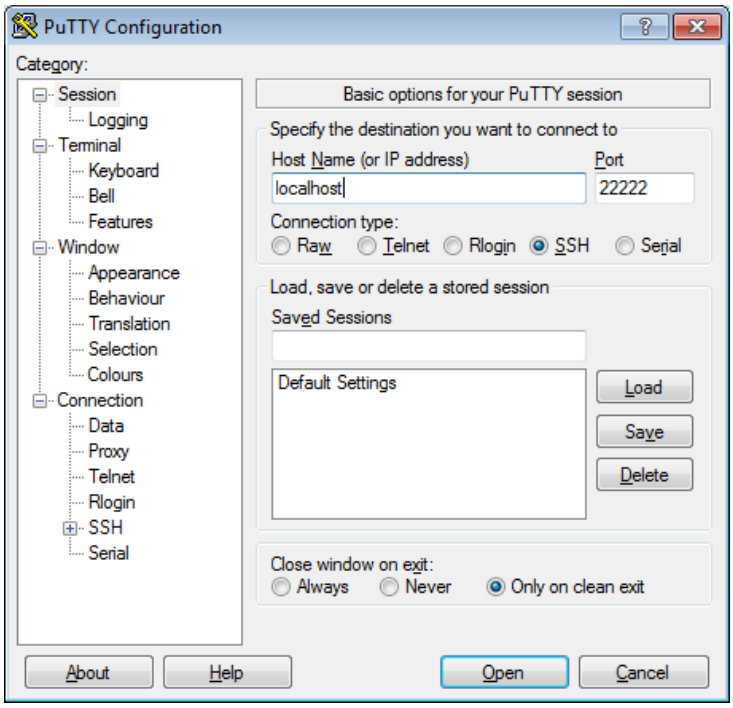

Figure 2: PuTTY

[5] PuTTY: `https://the.earth.li/~sgtatham/putty/latest/x86/putty.exe`

1.2.3 Dateiaustausch mit der VM

Unter Linux und macOS können Daten mittels des Befehls `scp` in die und aus der VM kopiert werden.

Kopieren einer Datei/Verzeichnis in das Homeverzeichnis der VM:

```
scp -P 22222 -r <Datei/Verzeichnis> student@localhost:
```

Kopieren einer Datei/Verzeichnis aus dem Homeverzeichnis der VM in das aktuelle Verzeichnis:

```
scp -P 22222 -r student@localhost:<Datei/Verzeichnis> .
```

Alternativ *kann auch ein beliebiger SFTP-Client (SSH File Transfer Protocol) z.B.: FileZilla* [6] *zum Datentransfer genutzt werden. In Abbildung 1.2.3 sind die Verbindungsdaten für die VM ersichtlich. Nach dem Verbinden können Daten mittels Drag & Drop zwischen dem Host-Betriebssystem und dem Gast-Betriebssystem ausgetauscht werden.*

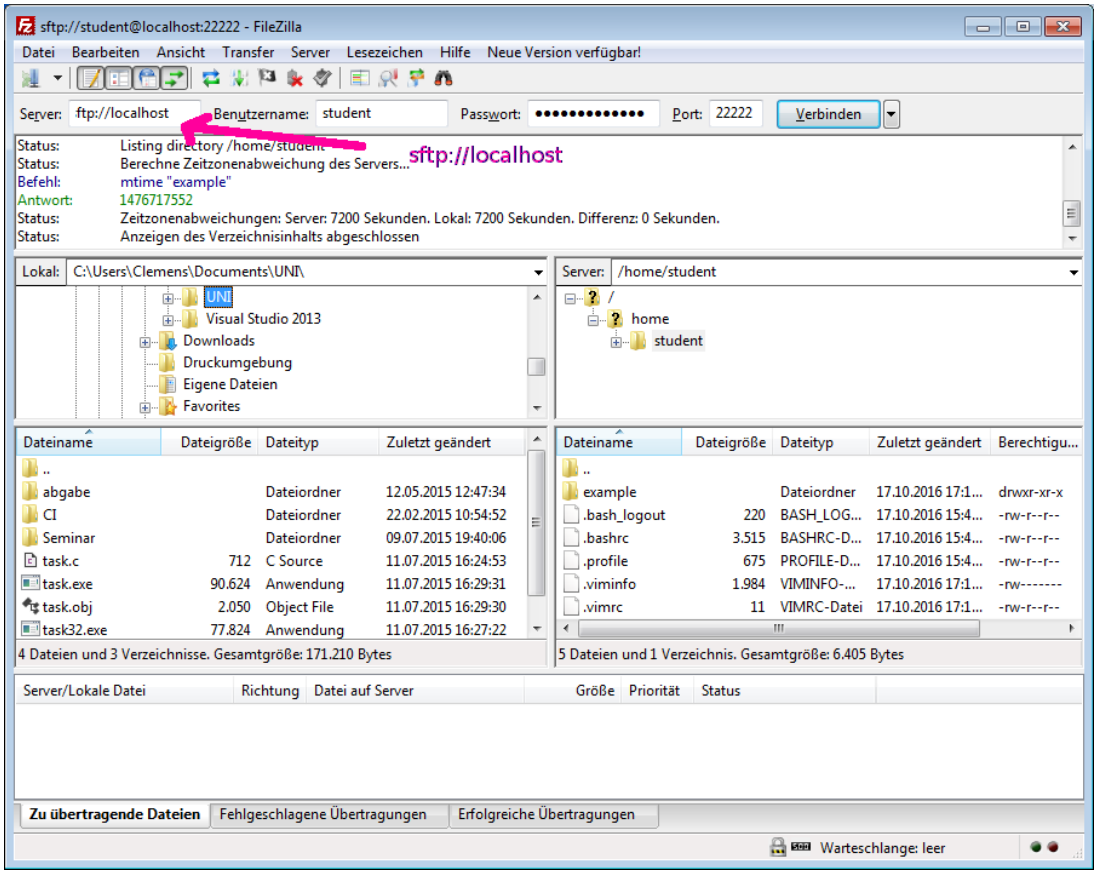

Figure 3: FileZilla

[6] FileZilla: `https://filezilla-project.org/`

1.2.4 Bekannte Probleme mit der VM

Im Folgenden finden Sie einige Beispiele, die bei potenziellen Problemen mit der VM aushelfen können.

- **VM startet nicht:**
  - Vergewissern Sie sich, dass auf Ihrem System **(HW) Virtualization Support** im BIOS aktiviert ist.
  - Falls Virtualization Support aktiv ist, kann es sein, dass Hyper-V unter Windows den Virtualization Support blockiert (vor Windows 10). Ab Windows 10, unterstüzt Hyper-V **nested Virtualization**, das wiederum den Hypervisor nicht behindern sollte.

1.3 POSIX-Funktionen

Lesen Sie die Man-Pages zu den unter 2.2 aufgelisteten Funktionen.

2 Tutoraufgaben

2.1 C-Programmierung – From Java to C

Im Rahmen des Programmierpraktikums haben Sie die Programmierung mit Java erlernt. Um Ihnen den Einstieg in die C-Programmierung zu erleichtern, wird Ihr Tutor im Rahmen der ersten Sitzung einige zentrale Unterschiede zwischen Java und C erläutern. Darauf aufbauend werden Sie sich in der Hausaufgabe tiefer in C einlesen. Ihr Tutor wird die folgenden Aspekte behandeln:

- Mittels *javac* übersetzt der Java-Compiler Ihren menschenlesbaren Quellcode in sog. architekturneutralen Bytecode. Dieser kann auf jedem Computer mit installiertem Java Runtime Environment *interpretiert* werden, also in Echtzeit in Maschinencode übersetzt werden. In C hingegen übersetzt der Compiler Ihren Quellcode direkt in Maschinencode. Da das *kompilierte* Programm somit direkt an die Rechnerarchitektur angepasst ist, kann es schneller laufen, ist aufgrund der Maschinencode-Spezialisierung jedoch i.A. nicht auf anderen Computerarchitekturen lauffähig.
- Vor der Kompilierung scannt der sog. Preprocessor Ihren Code und führt alle mit # beginnenden Anweisungen aus, d.h. er ersetzt Konstanten und importiert benötigte Dateien.
- In C wird prozedural programmiert. Klassen und Objekte existieren nicht, es existieren nur Funktionen. Diese müssen vor ihrem ersten Aufruf bekannt sein ("weiter oben im Code stehen"). Entweder Sie werden vorher *definiert* oder als Prototyp *deklariert*.
- Die Low-level Syntax von Java und C ist nahezu identisch, objektorientierte Konstrukte fehlen jedoch in C.
- Sämtliche in einer Funktion verwendeten Variablen sind am Anfang der Funktion, vor Zuweisungen und Aufrufen, zu deklarieren. Je nach Compiler sind Variablendeklarationen inmitten der Funktion unzulässig.
- Die Größe eines Arrays ist bei dessen Deklaration mit anzugeben.
- C kennt im Gegensatz zu Java sog. Pointer und kann Arbeitsspeicherbereiche direkt manipulieren. Im Rahmen der Hausaufgabe erfahren Sie hierzu mehr.
- C-Main-Funktion: int main (int argc, char *argv [], char *envp [])

2.2 POSIX-Funktionen

Was ist die Rolle der folgenden POSIX (Portable Operating System Interface) Funktionen? Erläutern Sie kurz die Parameter und den Rückgabewert jeder Funktion. [7]

- `int printf(const char *format, ...);`
- `ssize_t read(int fildes, void *buf, size_t nbyte);`
- `ssize_t write(int fildes, const void *buf, size_t nbyte);`
- `void *malloc(size_t size);`
- `void free(void *ptr);`

[7] Erläuterungen und Beispiele der einzelnen Funktionen finden Sie auch in den **man**-Pages.

3 Hausaufgaben

3.1 Elementare Ein-/Ausgabe: Hello, world (1 Punkt)

Erstellen Sie ein Programm, das "**Hello, world\n**" auf der Konsole (Standardausgabe, **stdout**) ausgibt. Nutzen Sie dafür die Ausgabefunktion `printf`.

Quellen: `hw.c`

Executable: `hw`

3.2 Elementare Ein-/Ausgabe: ASCII-Tabelle (2 Punkte)

Der American Standard Code for Information Interchange (ASCII) ist eine weitverbreite Zeichencodierung. Standardmäßig wird hierbei jeder Zahl von 0 bis 127 ein **character** zugeordnet. Die Großbuchstaben von A bis Z sind bspw. durch die Zahlen 65 bis 90 im Dezimalsystem codiert. Um einen Überblick über die enthaltenen Zeichen zu bekommen, erstellen Sie ein Programm, das folgende Tabelle für alle ASCII-Codes in verschiedenen Zahlensystemen ausgibt:

```
Oct     Dec     Hex     Char
000     0       00
...
060     48      30      0
061     49      31      1
...
101     65      41      A
102     66      42      B
...
```

Nutzen Sie für die Formatierung der Zeilen `printf` mit geeigneten Formatstring-Platzhaltern (`man 3 printf`). Achten Sie auf die verschieden formatierten Zahlen: **oktal** mit führender 0, **dezimal** linksbündig, **hexadezimal** mit 2 Stellen (a-f in Kleinbuchstaben) dargestellt.

Quellen: `asc.c`

Executable: `asc`

Beantworten Sie sich die Frage, warum die ersten 32 Zeichen des ASCII-Codes nicht darstellbar sind. Erklären Sie Teile Ihrer Antwort mithilfe des Terminalbefehls `man ascii` **(keine Abgabe)**.

3.3 Elementare Ein-/Ausgabe: Umwandlung von Zeichen (3 Punkte)

Erstellen Sie ein Programm, das einen Namen als Zeichenkette (String) von **stdin** einliest. Nutzen Sie dafür die Funktion `read`. Wandeln Sie alle Kleinbuchstaben der eingelesenen Zeichenkette in Großbuchstaben um. Achten Sie darauf, dass Sie wirklich nur Buchstaben umwandeln: bei der Eingabe eines Zeichens außerhalb von `[a..z]` bzw. `[A..Z]` soll das Programm bis zum letzten gültigen Buchstaben ausgeführt werden.

Die resultierende Zeichenkette soll zusätzlich mittels des ROT13-Algorithmus "verschlüsselt" werden. Nutzen Sie die Funktion `write` um das Ergebnis in der Form "Hallo: [Name] -- [ROT13-Name]\n" auszugeben. Informationen zu ROT13 finden Sie im Web. Der eingegebene String soll also zuerst in Großbuchstaben umgewandelt und anschließend um 13 Stellen innerhalb der Großbuchstaben weiterrotiert werden. Achten Sie hierbei darauf, dass die Eingabe üblicherweise mit einem \n endet, welches Sie für die Ausgabe entfernen müssen. Auch hier soll die Funktion bei nicht-alphabetischen Zeichen bis zum letzten gültigen Buchstaben ausgeführt werden, wie im folgenden Beispiel zu sehen ist:

```
$ ./rotup
Ingo123!ABC                              // Eingabe
Hallo: Ingo123!ABC -- VATB               // Ausgabe
```

Quellen: `rotup.c`

Executable: `rotup`

3.4 hexdump (5 Punkte)

Schreiben Sie eine Funktion:

```
void hexdump (FILE *output, char *buffer, int length);
```

zur Ausgabe eines Puffers als "Hexadezimal-Dump". Eine solche Funktion eignet sich bestens zum Debuggen. Sie gibt einen Puffer in Form der ASCII-Codes der Zeichen im Hexadezimalformat einerseits und der entsprechenden Zeichen (sofern sie darstellbar sind) andererseits aus.

Ein Beispiel:

```
000000 : 47 72 75 6e 64 6c 61 67 65 6e 20 42 65 74 72 69   Grundlagen Betri
000010 : 65 62 73 73 79 73 74 65 6d 20 75 6e 64 20 53 79   ebssystem und Sy
000020 : 73 74 65 6d 73 6f 66 74 77 61 72 65 00            stemsoftware.
```

Ganz links steht der relative Offset zum Beginn des Puffers. Rechts werden die entsprechenden Zeichen lesbar ausgegeben, sofern ihr ASCII-Code zwischen 0x20 (dez. 32) und 0x7e (dez. 126) liegt und damit darstellbar ist. Bei Werten außerhalb dieses Bereichs wird statt des Zeichens ein Punkt (.) ausgegeben.

Das Ausgabeformat soll exakt dem folgenden Aufbau entsprechen:

- 6 Zeichen Puffer-Offset mit führenden Nullen
- 1 Leerzeichen, 1 Doppelpunkt, 1 Leerzeichen
- 16 x die Ausgabe des nächsten Bytes im Puffer formatiert in 2 Zeichen mit führender 0 im Hexadezimalformat, wobei jeweils zwei aufeinanderfolgende Ausgaben durch genau ein Leerzeichen getrennt sind
- 3 Leerzeichen
- 16 x die Ausgabe der zuvor hexadezimal dargestellten Bytes als Zeichen, sofern dieses "druckbar" ist (siehe oben)
- Newline

Beachten Sie, dass die letzte Zeile weniger als 16 Zeichen im Puffer haben kann und dass dementsprechend Leerzeichen ergänzt werden müssen, um die korrekte Darstellung zu erhalten.

Testen Sie die Funktion, indem Sie vier Character-Arrays unterschiedlicher Länge im Quellcode definieren und dann diese als Hexdump ausgeben.

Erweitern Sie Ihre Funktion so, dass der auszugebende Puffer als Kommandozeilenparameter übergeben wird. Werden mehrere übergeben, so soll das Programm jeden Puffer einzeln ausgeben, wobei jeweils zwei aufeinanderfolgende durch eine Leerzeile getrennt sind. Achten Sie darauf, dass Ihr Programm sich auch dann sinnvoll verhält, wenn kein Parameter angegeben wird.

Implementieren Sie die Funktion `hexdump ()` in einer Datei `hexdump.c` und die Funktion `main ()` in einer separaten Datei `hd.c`; denken Sie daran, in letzterer den Funktionsprototypen korrekt zu deklarieren.

Quellen: `hexdump.c hd.c`

Executable: `hd`

Beispiel:

```
$ ./hd Hello "Operating Systems" now.
000000 : 48 65 6c 6c 6f 00                                      Hello.

000000 : 4f 70 65 72 61 74 69 6e 67 20 53 79 73 74 65 6d        Operating System
000010 : 73 00                                                  s.

000000 : 6e 6f 77 2e 00                                         now..
```