Übungsblatt 1

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 11. Oktober 2023
Abgabe bis **Sonntag, 22. Oktober 2023, 23:59 Uhr**
(ab Blatt 2 wie angekündigt jeweils freitags bis 23:59 Uhr)
Vorführung vom 23. bis zum 27. Oktober 2023

Alle Programme müssen unter **Ubuntu 22.04** kompilierbar bzw. lauffähig und ausreichend kommentiert und mit **Makefile** (C, Assembler) versehen sein, um Punkte zu erhalten. Als Compiler sollen **clang** (C) und **nasm** (Assembler) verwendet werden. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Alle Gruppenmitglieder sollten die Abgabe erklären können. Die Abgabe erfolgt mittels Ihres Git-Repositories und der vorgegebenen Ordnerstruktur.

Die Punkte der Aufgaben sind relevant für die Zulassung. Die Punkte der Bonusaufgaben werden auf Ihren Punktestand addiert, werden aber nicht auf die für die Zulassung benötigten Punkte addiert.

Abgabestruktur: Jede Abgabe, die die folgende Struktur nicht umsetzt, wird **NICHT** bepunktet bzw. mit 0 Punkten bewertet

- die zu korrigierenden Lösungen müssen bis zur Deadline auf dem `master`-Branch liegen. Lösungen auf anderen Branches werden nicht gewertet
- alle Lösungen müssen in der vorgegebenen Ordnerstruktur (**blattXX/aufgabeYY**) abgelegt werden, wobei `XX` und `YY` durch die jeweiligen Nummern des Zettels und der Aufgaben ersetzt werden sollen. Der Name soll exakt nur aus diesen Zeichen bestehen und achtet auf Kleinschreibung
- sämtliche Aufgaben, mit Ausnahme der theoretischen Aufgaben, die eine PDF erfordern, benötigen zwingend ein **Makefile**. Abgaben ohne dieses werden nicht gewertet

Aufgabe 1 Betriebssystem einrichten (0 Punkte)

Die Programme sollen unter *Ubuntu 22.04* lauffähig sein. Um das zu gewährleisten, können Sie Ubuntu 22.04 entweder auf ihrem Computer installieren, oder in einer virtuellen Maschine; Anleitungen dafür finden Sie im Internet. Kostenlos herunterladen können Sie es unter [1]. Ein Tutorial für die Installation finden Sie unter [2]. Für die Bearbeitung müssen noch weitere Werkzeuge neben den standardmäßig vorhandenen installiert werden. Diese können Sie mithilfe des Kommandozeilenbefehls

```
apt install build-essential clang valgrind git nasm
```

installieren. Sie werden dabei aufgefordert, Ihr Administratorkennwort, das Sie bei der Installation gesetzt haben, einzugeben.

[1] `https://ubuntu.com/download/desktop/thank-you?version=22.04.3&architecture=amd64`
[2] `https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview`

Aufgabe 2 String- und Byteblock-Funktionen (6 Punkte)

> Hinweis: Dies ist Teil eines Aufgabenzyklus zur Entwicklung einer simplen Shell ohne Verwendung der C-Standardbibliothek.

In C stellen Zeichenketten (Strings) bekannterweise keinen eigenen Datentyp dar; stattdessen werden `char*`-s verwandt und die Konvention angenommen, dass Strings nullterminiert sind (d.h. ein String läuft von dem gegebenen Anfang bis zum ersten Nullbyte nach dem Anfang). Die Standardbibliothek realisiert verschiedene String-Operationen; Ziel dieser Aufgabe ist es, einige von ihnen manuell (also ohne die Standardbibliothek) zu implementieren. Um Namenskollisionen zu vermeiden, versehen wir die selbst implementierten Funktionen etc. meist mit dem Präfix `my-`.

1. Um auf die Funktionsprototypen und die Dokumentation aus der Standardbibliothek verweisen zu können, müssen wir zunächst einen Typ definieren. Erstellen Sie dazu eine Headerdatei **mystddef.h** und definieren Sie dort den Typ `size_t` als `unsigned long`.

2. Deklarieren Sie in einer Headerdatei **mystring.h** und implementieren Sie in einer Quelltext-Datei **mystring.c** die folgenden Funktionen:

   a) `size_t mystrlen(char *str)` — Diese Funktion gibt die Länge des übergebenen Strings zurück, d.i. die Anzahl an Bytes zwischen `str` (einschließlich) und dem ersten Nullbyte nach `str` (ausschließlich). Die Länge des leeren Strings (der nur ein Nullbyte enthält) ist insbesondere 0.

   b) `char *mystrcpy(char *dest, const char *src)` — Dies kopiert `mystrlen(src) + 1` Bytes von `src` nach `dest` und gibt `dest` (unverändert) zurück.

   c) `char *mystrcat(char *dest, const char *src)` — Diese Funktion hängt `src` an das Ende von `dest` an (engl. concatenate), sodass anstatt des (ehemaligen) Nullbytes am Ende von `dest` das erste Byte von `src` steht, etc., und `strlen(dest)` um `strlen(src)` größer wird. Die Funktion gibt `dest` unverändert zurück.

   d) `int mystrcmp(const char *s1, const char *s2)` — Diese Funktion vergleicht `s1` und `s2` anhand der lexikographischen Ordnung der Zeichen der Strings (wobei die Zeichen als vorzeichenlos interpretiert werden) und gibt

   - einen negativen Wert zurück, falls s1 vor s2 kommt,
   - Null zurück, falls s1 und s2 gleich sind, und
   - einen positiven Wert zurück, falls s1 nach s2 kommt.

   Beachten Sie, dass Sie **mystddef.h** aus **mystring.h** einbinden müssen, um `size_t` verwenden zu können.

3. Nebst String-Funktionen deklariert **string.h** aus der Standardbibliothek auch einige extrem nützliche Funktionen zum Umgang mit Byteblöcken. Deklarieren und implementieren Sie wie oben die folgenden Funktionen:

   a) `void *mymemcpy(void *dest, const void *src, size_t n)` — Kopiert `n` Bytes von `src` nach `dest` und gibt `dest` zurück.

   b) `void *mymemset(void *buf, int ch, size_t n)` — Überschreibt `n` Bytes beginnend bei `buf` mit `ch` (als `char` interpretiert) und gibt `buf` zurück.

Aufgabe 3 Makefiles und AddressSanitizer (4 Punkte)

Makefiles werden häufig verwendet, um Projekte schnell und einfach zu kompilieren. Auf der Vorlesungs-Website finden Sie ein PDF-Dokument **Makefile.pdf**, welches Ihnen eine Einführung in Makefiles gibt. Weitere Informationen finden Sie außerdem unter `http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/` bzw. eine ausführliche Anleitung unter `http://www.gnu.org/software/make/manual/make.html`. Außerdem sollen alle Programme mit *AddressSanitizer* kompiliert werden (Flag `-fsanitize=address`, s. u.). Dies soll Ihnen während des Programmierens dabei helfen, Fehler bei der Speicherverwaltung zu finden und zu beheben. Ausführliche Informationen über AddressSanitizer finden Sie in dem PDF-Dokument **ASAN.pdf** auf der Vorlesungs-Website.

> Hinweis: Für die Aufgaben soll standardmäßig der Compiler `clang` verwendet werden (nicht `gcc`!). Das Aufrufen von `clang` sowie die dabei verwendeten Optionen orientieren sich an `gcc`, die Verwendung von `clang` ist daher nahezu identisch zu `gcc`.

Aufgabe

Erstellen sie ein kurzes Programm **demo.c**, welches alle Funktionen aus Aufgabe 2 demonstrativ verwendet. Schreiben Sie anschließend ein Makefile für ihr Demo-Programm, das

1. folgende Variablen definiert und benutzt:
   OBJECTS (alle **.o**-Dateien),
   HEADERS (alle **.h**-Dateien),
   CFLAGS, das jedem kompilierenden clang-Aufruf übergeben werden soll und auf `-g -Wall -fsanitize=address` gesetzt werden soll,
   LDFLAGS, das jedem linkenden clang-Aufruf übergeben werden soll und auf `-g -Wall -fsanitize=address` gesetzt werden soll

2. alle **.c**-Dateien in **.o**-Dateien unter Verwendung von „%“-Pattern kompiliert (`clang -c`) und alle **.o**-Dateien zu einer ausführbaren Datei linkt.

Achten Sie darauf, dass alle **.c**-Dateien, die eine **.h**-Datei einbinden, neu kompiliert werden, falls die **.h**-Datei geändert wird.

> Warnung: Künftig sollen alle C Aufgaben mit Makefile abgegeben werden. Wenn korrekt erstellt, ist das hier erstellte Makefile ein gutes Template für die weiteren Abgaben.
>
> **Jede Aufgabe, die ohne Makefile abgegeben wird, wird mit 0 Punkten bewertet!**

Aufgabe 4 Anmeldung über PECAS (0 Punkte)

Die Übungszettel müssen in Gruppen von drei Studis gelöst werden. Die Registrierung der Kleingruppen zur Abgabe hierzu läuft über PECAS: `https://pecas.net.cs.uni-bonn.de` Bei den Kleingruppen zur Übungsabgabe ist es ratsam, dass alle Teilnehmer den gleichen Übungstermin haben. Für die Registrierung in PECAS wird es evtl. eine Deadline geben die dann auf der Vorlesungwebsite verkündet wird. Zur Registrierung in PECAS ist ein funktionierender Informatik-Account notwendig, denn der Login-Prozess in PECAS läuft über das GitLab der Informatik. Bei Problemen mit Ihrem Informatik-Account wenden Sie sich bitte an die Systemgruppe: `https://gsg.cs.uni-bonn.de`. Bei anderweitigen Problemen wenden Sie sich bitte an [3]. Bei zukünftigen, aufgabenbezogenen Problemen können Sie aber selbstverständlich Ihren Tutor/Ihre Tutorin direkt anschreiben. Nach der Registrierung wird der Zugriff auf die entsprechende Abgabegruppe auf `https://gitlab.cs.uni-bonn.de` freigeschaltet. Hierzu sollten Sie auch eine E-Mail erhalten.

[3] tut-sys-prog@lists.iai.uni-bonn.de

Aufgabe 5 Git (0 Punkte)

Git (`https://git-scm.com/`) ist ein dezentrales Versions-Verwaltungssystem, das hauptsächlich zum kollaborativen Arbeiten genutzt wird. Es soll im Folgenden sowohl für Ihr gemeinsames Arbeiten an den Aufgaben, als auch als Medium zur Abgabe der Übungsaufgaben an Ihren Tutor genutzt werden. Git verwaltet die Projektarchive (Repositories) als Dateisysteme und protokolliert Veränderungen an den Dateien, sodass auch auf ältere Versionen von Dateien zugegriffen werden kann.

Jeder Abgabegruppe wird ein Repository zugeordnet, auf das per Internet mit dem Informatik-Account (meistens `name@informatik.uni-bonn.de`) zugegriffen werden kann:

`https://gitlab.cs.uni-bonn.de/`

Benutzername und Passwort sind die gleichen wie für den Informatik-Account.

> Hinweis: Falls Sie sich noch nicht zu dritt angemeldet haben, holen Sie dies bitte noch nach und vergessen Sie nicht, dass die Abgabegruppen ausschließlich durch die zwingende Anmeldung in PECAS erstellt werden. Nutzen Sie also das angegebene Gruppen-Token zur Registrierung Ihrer Abgabegruppe.

Aufgabe Importieren Sie die in Aufgabe 2 erstellten Quellcode Dateien und das Makefile in Ihr Git-Repository. Vorher können Sie mit dem Befehl `git clone` das Gruppen-Repository auschecken. Die HTTPS-URL zum Auschecken sowie weitere Informationen finden Sie in der Gitlab Oberfläche unter Ihrem Gruppen-Repository.

Modifizieren Sie den Programmcode mit dem Tool GNU `indent`. Dieses sorgt für eine gut lesbare Formatierung. Der Aufruf von indent inklusive der verwendeten Parameter zur Formatierung soll in einer README Datei protokolliert und ebenfalls mittels `git add` ins Repository aufgenommen werden. Diese Änderungen werden mittels `git commit` in das Repository eingepflegt. Mittels `push` werden dann die Änderungen von Ihrer lokalen Kopie ins zentrale Repository überführt. Bis zur Deadline können Sie Ihre Lösungen unbegrenzt oft überarbeiten und aktualisieren.

> Warnung: Zukünftig geben Sie bitte analog zu dieser Aufgabe unaufgefordert alle Aufgaben der folgenden Übungsblätter mittels git ab. Dabei ist folgende feste Ordnerstruktur einzuhalten: Für jedes Blatt wird ein eigener Ordner `blattXX` erstellt, wobei `XX` durch die Nummer des Übungszettels ersetzt wird (also z. B. `blatt01`, `blatt02`, ...). Innerhalb dieser Ordner werden weitere Ordner `aufgabeXX` für die einzelnen Aufgaben des jeweiligen Blattes erstellt; `XX` wird auch hier durch die Nummer der Aufgabe ersetzt (also z. B. `aufgabe01`, `aufgabe02`, ...). Für jede Aufgabe werden die Dateien zur Abgabe in den entsprechenden Unterordner gelegt.
>
> Achtet darauf keine weiteren Zeichen zu verwenden und die Kleinschreibung einzuhalten.
>
> **Jede Aufgabe, die nicht dieser Ordnerstruktur folgt, wird mit 0 Punkten bewertet!**

Aufgabe 6 Fehlersuche bei Speicherverwaltung (3 Punkte)

> Hinweis: Bitte geben Sie die Antworten kurz stichwortartig in einer Textdatei ab, darüberhinausgehende Erläuterungen sind in der Besprechung mit den Tutoren möglich.

Identifizieren Sie in den folgenden C-Codefragmenten die Fehler, die in der Speicherverwaltung gemacht wurden. Beschreiben Sie kurz das Problem und wie es zu lösen wäre.

(a)

```
int* i;
for (int k=0; k<10; k++) {
        i = malloc(sizeof(int));
        if (i == NULL) { /* hier ausgelassen: Fehlermeldung, Programm beenden */ }
        *i = k;
}
free(i);
```

(b)

```
double* d;
d = malloc(2 * sizeof(double));
if(i == NULL) { /* hier ausgelassen: Fehlermeldung, Programm beenden */ }
d[1] = 4.772;
d[2] = -17 * 3.557;
free(d);
```

(c)

```
int** i;
i = malloc(sizeof(int*));
if (i == NULL) { /* hier ausgelassen: Fehlermeldung, Programm beenden */ }
**i = 5;
free(i);
```