Vorlesung Systemnahe Programmierung – WS 2019/20

Dr. Matthias Frank, Stephan Plöger

1. Übungszettel

Ausgabe: Mi 09.10.2019; Abgabe bis zum (Sonntag) 20.10.2019, 23:59 Uhr
(Die folgenden Abgaben ab Blatt 2 liegen wie angekündigt jeweils Freitag bis 23.59 Uhr)
Die Vorführung erfolgt in der Woche vom 21.10.2019 bis zum 27.10.2019 in Ihrer Übungsgruppe.

Alle Programme müssen im Poolraum unter Linux kompilierbar bzw. lauffähig und ausreichend kommentiert sein. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen.

Aufgabe 1: String- und Byteblock-Funktionen (6 Punkte)

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-**.

(a) 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**.

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

(i) **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.

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

(iii) **char *mystrcat(char *dest, const char *src)** — Diese Funktion hängt **src** an das Ende von **dest** an (engl. concatenate), sodass anstatt des (ehemaligen) Null-bytes 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.

(iv) **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,
- 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.

(c) 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:

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

(ii) **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 2: 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 ausführlich 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.

WICHTIG: 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 1 demonstrativ verwendet. Schreiben Sie anschließend ein Makefile für ihr Demo-Programm, das

a) 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

b) 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.

WICHTIG: Künftig sollen alle C Aufgaben mit Makefile abgegeben werden. Wenn korrekt erstellt, ist das hier erstellte Makefile ein gutes Template für die weiteren Abgaben.

Aufgabe 3: 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-sysprog.informatik.uni-bonn.de

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

Hinweis: *Falls Sie sich noch nicht zu dritt in Ihrer Übungsgruppe angemeldet haben holen Sie das bitte noch nach. Sollten Sie noch keinen Abgabepartner haben suchen Sie einen Partner z.B. über unsere Mailingliste. Nach Schließung von TVS werden dann die verbleibenden Einzelpersonen und Zweiergruppen eines Abgabetermins in Dreier-Abgabegruppen zusammengefasst.*

WICHTIG*: Nach der Zuweisung zu einer Gruppe in TVS und dem erstmaligen anmelden im Gitlab, bitten wir Sie Ihrem Betreuer eine E-Mail mit Ihrem Benutzernamen zu schicken damit er/sie Sie für Ihre Gruppe und Repository freischalten kann.*

Aufgabe: Importieren Sie die in Aufgabe 1 und 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` im

Repository eingepflegt. Mittels `push` werden dann die Änderungen von Ihrer lokalen Kopie ins zentrale Repository überführt.

WICHTIG: 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.

Aufgabe 4: Fehlersuche bei Speicherverwaltung (3 Punkte)

Bemerkung: Ausnahmsweise 3 Punkte, pro Codefragment 1 Punkt, Bepunktung mit 0 oder 1 Punkten bei keiner/falscher bzw. richtiger Antwort. 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)

```
1 int* i;
2 for (int k=0; k<10; k++) {
3   i = malloc(sizeof(int));
4   if (i == NULL) { /* hier ausgelassen: Fehlermeldung, Programm beenden */ }
5   *i = k;
6 }
7 free(i);
```

(b)

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

(c)

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