Vorlesung Systemnahe Programmierung – WS 2019/20

Dr. Matthias Frank, Stephan Plöger

4. Übungszettel

Ausgabe: Mi 30.10.2019; Abgabe bis zum (Freitag) 08.11.2019, 23:59 Uhr.
Die Vorführung erfolgt in der Woche vom 11.11.2019 bis zum 15.11.2019 in Ihrer Übungsgruppe.

Alle Programme müssen im Poolraum unter Linux kompilierbar bzw. lauffähig und ausreichend kommentiert und mit einem Makefile oder Cargo.toml versehen sein. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Die Abgabe erfolgt mittels Ihres Git-Repositories und der vorgegebenen Ordnerstruktur.

Aufgabe 11: Hello World in Assembler (6 Punkte (1+3+1+1))

Für diese und die folgenden Assembler-Aufgaben verwenden Sie bitte den Netwide Assembler (NASM). Die ausführbare Datei heißt üblicherweise `nasm`. NASM benutzt als Standard-Einstellung die Intel-Syntax.

Denken Sie bitte auch daran, ihre Programme stets gründlich und verständlich zu kommentieren.

a) (1 Punkt)

Gegeben ist das folgende, unkommentierte "Hello world"-Programm.

```
1     SECTION .data
2
3     str:    db `Hello world!\n`
4     strlen: equ $ - str
5
6     num:    dq 1337
7
8     SECTION .text
9
10    global _start
11
12    _start:
13        mov     rax, 1
14        mov     rdi, 1
15        mov     rsi, str
16        mov     rdx, strlen
17        syscall
18
19        mov     rax, 60
20        mov     rdi, 0
21        syscall
```

Speichern Sie den Quellcode unter `helloworld-a.asm` (auch zu finden auf der Sys-Prog Homepage). **Ergänzen Sie den Quellcode um aussagekräftige Kommentare.** Assemblieren, linken und testen Sie dann das Programm.

b) (3 Punkte)

Kopieren Sie die Quellcodedatei aus Teilaufgabe a) nach `helloworld-b.asm`. Ergänzen Sie das Programm so, dass der Wert der Variable `num` in einen String umgewandelt und in der Konsole ausgegeben wird. Tipp: Sie erreichen dies durch eine wiederholte Division durch 10 unter Beachtung des Rests. Sie können zu diesem Zweck analog zu `str` einen String deklarieren, der als Puffer für die auszugebende Zahl dient. Sie können davon ausgehen, dass es sich bei `num` um eine ganze, nicht negative Zahl handelt, also $n \in \mathbb{N}_0$ gilt. Testen Sie ihr Programm mit verschiedenen Werten für `num`.

c) (1 Punkt)

Kopieren Sie ihre Lösung aus Teilaufgabe b) nach `helloworld-c.asm`. Lagern Sie nun den Code zur Umwandlung einer Zahl des Typs `.long` in einen String in eine Funktion `printnumber` der Form

```
printnumber:
      ; ...
      ret
```

aus. Diese soll aus dem Hauptprogramm folgendermaßen aufgerufen werden:

```
mov   eax, num          ; Auszugebende Zahl wird in eax übergeben
call  printnumber       ; Funktion aufrufen
```

Hinweis: Es gibt in Assembler keine echte Unterscheidung zwischen den Begriffen Funktion, Prozedur und Subprogramm. Ebenso gibt es keine vorgeschriebene Art, Parameter zu übergeben und Rückgabewerte zurückzugeben (deshalb gibt es Konventionen wie die cdecl).

d) (1 Punkt)

Beschreiben Sie, was die Befehle `call` und `ret` genau tun. Warum ist der Stack dafür wichtig?

Aufgabe 12: Speicherverwaltung (6 Punkte)

Dies ist Teil eines Aufgabenzyklus zur Entwicklung einer simplen Shell ohne die C-Standardbibliothek.

Eine wichtige Funktion der C-Standardbibliothek ist die Bereitstellung einer *dynamischen Speicherverwaltung*; ebendiese soll in dieser Aufgabe implementiert werden. Dafür sollen Sie einen 1 MiB (2^{20} Bytes) großen fixen und statisch allokierten Speicherblock verwenden, den Sie z. B. durch folgende (globale) Dekleration anlegen können:

```
char memory[1048576];
```

Implementieren Sie die in der Vorlesung vorgestellte Free-Liste, um diesen Speicher zu verwalten:

(a) Um auf Funktionsprototypen und Dokumentation aus der Standardbibliothek verweisen zu können, müssen wir den Null-Pointer definieren. Ergänzen Sie dazu die Datei `mystddef.h` aus STRING- UND BYTEBLOCK-FUNKTIONEN (oder aus

SYSTEMAUFRUFS-WRAPPER) um eine Definition des Präprozessormakros `NULL` als `((void*)0)`.

(b) Deklarieren Sie in einer Headerdatei `mystdlib.h` und implementieren Sie in einer Quelltextdatei `mystdlib.c` die folgenden Funktionen:

- `void *mymalloc(size_t size)` — Reserviert `size` Bytes Speicher und gibt einen Zeiger an den Anfang des reservierten Blocks zurück. Falls size = 0, soll `NULL` zurückgegeben werden. Falls nicht genügend Speicher verfügbar ist, gibt dies `NULL` zurück. Falls der Aufruf erfolgreich war (d.i. nicht `NULL` zurückgab), müssen sämtliche Bytes des Speicherblocks erfolgreich lesbar und schreibbar sein, der initiale Inhalt des Speicherblocks ist nicht spezifiziert, und der neue Speicherblock darf keinen bereits reservierten aber noch nicht freigegebenen Speicherblock überlappen.
- `void myfree(void *ptr)` — Gibt den Speicherblock beginnend bei `ptr` frei, sodass er wiederverwendet werden kann. Falls `ptr` der Nullzeiger `NULL` ist, passiert nichts. Falls `ptr` nicht `NULL` ist, muss `ptr` vorher von `mymalloc` zurückgegeben und seitdem *nicht* freigegeben worden sein, ansonsten ist das Verhalten undefiniert (d. h. Sie müssen diesen Fall nicht gesondert behandeln).

Hinweis: `mystdlib.c` *ist auch der richtige Ort, um den zu verwaltenden statischen Speicherblock anzulegen (s. o.).*

(c) Die Standardbibliothek definiert zwei weitere sehr nahe verwandte Speicherverwaltungsfunktionen. Deklarieren und implementieren Sie wie oben:

- `void *mycalloc(size_t nmemb, size_t size)` — Reserviert `nmemb` · `size` Bytes an Speicher, initialisiert sie mit Nullen, und gibt einen Zeiger auf den Anfang des Speicherblocks zurück. Siehe `mymalloc` für weitergehende Anforderungen an den zurückgegebenen Speicherblock.
- `void *myrealloc(void *ptr, size_t size)` — Verändert die Größe des bei `ptr` startenden Speicherblocks zu `size`, u.U. den Block an eine neue Position verschiebend. Gibt die neue Addresse des Speicherblocks zurück (falls sie unterschiedlich von `ptr` ist, zählt `ptr` als freigegeben und darf nicht an `myfree` *etc.* übergeben werden). Nach dem Aufruf ist das Minimum von der alten Größe des Speicherblocks und `size` Bytes am neuen Zeiger startend gültig; der Inhalt von evtl. neu entstandenen Bytes ist nicht spezifiziert. Falls das Reservieren von neuem Speicher nicht gelingt, gibt dies `NULL` zurück und lässt den Speicher an `ptr` unberührt. Siehe `mymalloc` für weitergehende Anforderungen an den zurückgegebenen Speicherblock. Falls `ptr` der Nullzeiger `NULL` ist, verhält sich dies wie `mymalloc(size)`. Falls `size` = 0 und `ptr` != NULL, verhält sich dies wie `myfree(ptr)` und gibt `NULL` zurück.

Auf der Vorlesungs-Website finden Sie ein Programm „test.c“, welches Sie benutzen können, um ihre Implementierung zu testen.