Übungsblatt 11

Dr. Matthias Frank, Dr. Matthias Wübbeling

Ausgabe Mittwoch, 20. Dezember 2023
Abgabe bis
Freitag, 12. Januar 2024, 23:59 Uhr
Vorführung vom 15. bis zum 19. Januar 2024

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

Hinweis:	Manche Browser sind nicht dazu in der Lage die in die PDFs eingebetteten Dateien anzuzeigen. Mittels eines geeigneten Readers, wie dem Adobe Reader, lassen sich diese jedoch anzeigen und verwenden.

Hinweis:	Die Inhalte von Aufgabe 3 werden in der Vorlesung am 09. Januar 2024 besprochen.

Aufgabe 1 Stackexperimente in Assembler (4 Punkte)

a)

Ihr Vorgänger hat ein Programm geschrieben, das für Ihre Firma extrem wichtig ist. Sie finden die disassemblierte Version names **stack.S** eingebettet in diese PDF-Datei. Leider ist der C-Quelltext verloren gegangen, und es wurde ein Fehler gefunden. Sie möchten jetzt diesen Fehler beseitigen, aber können das Programm selbst leider nicht verändern. Sie haben allerdings die Kontrolle über eine Funktion, die das Programm aufruft.

Sie möchten, dass die Funktion `f`, die das Programm aufruft, stattdessen an die Funktion `f2` delegiert. Normalerweise könnten Sie dazu einfach `call` verwenden, um innerhalb Ihrer Funktion die andere aufzurufen. Leider hat Ihr Chef verboten, dass Sie `call`-Instruktionen in Ihrem Code verwenden. Außerdem dürfen Sie keine Sprungbefehle (bedingt oder unbedingt) verwenden.

Ihnen kommt aber eine geniale Idee: Auf dem Stack liegt von dem Programm eine Rücksprungadresse. Wenn Sie diese verändern, springt der Prozessor nicht zurück in das Programm, sondern in `f2`. Gleichzeitig müssen Sie natürlich trotzdem irgendwie zurück in das Programm kommen. Überlegen Sie sich also, wie Sie die originale Rücksprungadresse so hinlegen, dass der Prozessor nach `f2` dahin zurückspringt.

Versuchen Sie zunächst, das Problem auf Papier zu lösen. Überlegen Sie sich, wo Stackpointer und Basepointer hinzeigen und welche Werte auf dem Stack liegen, und zeichnen Sie die Veränderungen auf dem Stack ein. Geben Sie eine Zeichnung ab, was Ihr Programm tut und warum das so funktioniert.

b)

Wie müssten Sie Ihre Lösung verändern, wenn die Funktion keinen Stackframe aufgebaut hätte?

c)

Nach `f2` wollen Sie noch nicht direkt in das Programm zurückspringen, sondern zunächst wieder nach `f` zurückkehren. Ändern Sie Ihr Programm so ab, dass der Ablauf so ist, als hätten Sie `call` verwendet!

Aufgabe 2 Experimente mit netcat (4 Punkte)

Diese Aufgabe soll in die Netzwerkkommunikation einstimmen, bevor in folgenden Aufgaben eigene Programme mit Socket-Programmierung erstellt werden.

> „Netcat, auch nc genannt, ist ein einfaches Werkzeug, um Daten von der Standardein- oder -ausgabe über Netzwerkverbindungen zu transportieren. Es arbeitet als Server oder Client mit den Protokollen TCP und UDP. Die Manpage bezeichnet es als TCP/IP swiss army knife (Schweizer Taschenmesser für TCP/IP).“
> Quelle: `https://de.wikipedia.org/wiki/Netcat`

a) (1 Punkt)

Recherchieren Sie zu netcat/nc und machen sich mit der Nutzung grob vertraut.

b) (1 Punkt)

Nutzen Sie nun netcat mit UDP (auswählbar über entsprechende Option). Starten Sie eine netcat-Instanz als Server (Empfänger), eine weitere als Client (Sender). Sie können zwischen verschiedenen Rechnern kommunizieren, bzw. auch auf dem selben Rechner mit der localhost-IP-Adresse 127.0.0.1.

Tauschen Sie nun mehrere Zeilen (an der Konsole eingegebenen) Text zwischen den beiden Instanzen aus. Protokollieren Sie in einer Textdatei jeweils für Server und Client das eingegebene netcat Kommando mit Parametern sowie die eingetippten bzw. empfangenen Textzeilen.

c) (1 Punkt)

Starten Sie nun auf demselben Rechner zeitlich versetzt mehrere Server-Instanzen von netcat mit UDP und *derselben* Portnummer. Senden Sie dabei gleichmäßig die ganze Zeit zeilenweise Texte vom selben netcat Client-Prozess an den Rechner und die gewählte Portnummer. Was beobachten Sie bei den Server- Instanzen, jeweils nach Start der 1. Instanz, nach der 2. Instanz, usw.?

d) (1 Punkt)

Starten Sie auf einem Rechner einen netcat Server mit TCP auf dem Port 8080. Starten Sie nun einen Webbrowser und geben als URL die IP-Adresse des Rechners, gefolgt von „:8080/“ ein (beispielhaft für localhost `http://127.0.0.1:8080/`). Protokollieren Sie wieder den Aufruf von netcat mit Parametern sowie die Ausgabe der empfangenen Daten. Was kann man an den Daten erkennen? Wenn möglich, experimentieren und protokollieren Sie mit verschiedenen Webbrowsern von unterschiedlichen Endgeräten (PC, Laptop, Smartphone, . . .). Dokumentieren sie ihre Experimente, beispielsweise anhand von Vorgehensweise, Skripten mit ausgeführten Befehlen, Screenshots, etc.

Aufgabe 3 Byte-Order/Domain Name System (4 Punkte)

Schreiben Sie eine C-Funktion `big_endian()`, die zurückliefert, ob das aktuelle System in Big-Endian-Byteorder arbeitet oder nicht. Benutzen Sie in dieser Funktion keine weiteren Funktionsaufrufe oder Makros. Entwickeln Sie weiterhin eine Funktion
`char* my_inet_ntoa(struct in_addr in)`,
welche genau wie `inet_ntoa` IP-Adressen in Network- Byteorder in Dotted-Decimal-Notation umwandelt (siehe Manpages zu `inet_ntoa()`). Interpretieren Sie dazu die IP-Adresse als 32-Bit-Integerzahl und berechnen Sie die einzelnen Bytes mit arithmetischen Operationen. Achten Sie dabei auf die Byteorder des jeweiligen Systems. Ihre Funktion könnte z.B. folgendermaßen beginnen:

```
char* my_inet_ntoa(struct in_addr in) {
        /* interpret IPv4-Address as 32bit uint */
        unsigned int addr = *(unsigned int *) &in;
```

Schreiben Sie nun ein Programm, welches als erstes die Byteorder des aktuellen Systems ausgibt (Little/Big-Endian). Weiterhin soll das Programm über die Kommandozeile einen Hostnamen entgegennehmen und eine zugehörige IP-Adresse ermitteln. Benutzen Sie dabei statt der häufig verwendeten Funktion `gethostbyname` die Kommunikationsprotokoll-unabhängige Funktion `getaddrinfo` (siehe Manpages). Diese Adresse soll nun einmal mittels der Funktion `inet_ntoa` und ein weiteres Mal unter Benutzung Ihrer Funktion `my_inet_ntoa` ausgegeben werden. Vergessen Sie dabei nicht, den von `getaddrinfo` allokierten Speicher mittels `freeaddrinfo` wieder freizugeben.

Integrieren Sie anschließend in Ihr Programm die Funktionalität, dass zu einer gegebenen IP-Adresse der zugehörigen Hostnamen ermittelt wird. Benutzen Sie dazu die Funktion `getnameinfo`.

Die (herkömmlichen) Funktionen `gethostbyname` und `gethostbyaddr` sollen nicht mehr benutzt werden. Hinweise (teilw. samt Beispielen) zu den (besseren) kommunikationsprotokollunabhängigen Funktionen finden sich z.B. unter `http://linux.die.net/man/3/gethostbyaddr`.