Hochschule RheinMain
Studiengang Medieninformatik
Sommersemester 2012
Prof. Dr. Robert Kaiser
M. Sc. Kai Beckmann

Betriebssysteme und Rechnerarchitekturen

LV 4112

Übungsblatt 5

05.05.12

Aufgabe 5.1 (<assert>):

(a) Die Standard-Headerdatei `<assert.h>` definiert mit dem Macro `assert()` ein einfaches Hilfsmittel zur Laufzeit-Überprüfung von Annahmen. Schauen Sie sich diese Headerdatei an, beachten Sie die Verwendung der im Präprozessor „eingebauten" Macros `__LINE__` und `__FILE__`. Erklären Sie die Arbeitsweise.

(b) Warum sollte so etwas:

```
...
assert(i++ != 0);
...
```

in einem Programm nicht vorkommen?

(c) Schreiben Sie ein kleines C-Programm `assert-demo.c`. mit dem Sie die Funktion des Macros `assert()` erproben können.

(d) Erzeugen Sie unter Verwendung des Präprozessors das expandierte Hauptprogramm und geben Sie diese Ausgabe an.

(e) Kompilieren Sie Ihr Programm einmal mit und einmal ohne die Compileroption `-DNDEBUG` und erproben Sie beide Varianten. Erklären Sie Ihre Beobachtungen.

Aufgabe 5.2 (Setjmp()/Longjmp()):

Mit Hilfe der Standard-Bibliotheksfunktion `setjmp()` kann eine Kopie des momentanen Prozessorzustandes erstellt werden. Mit `longjmp()` kann ein solcher Prozessorzustand wiederhergestellt werden. Machen Sie sich anhand der Manualseite mit diesem Funktionspaar vertraut.

(a) Schreiben Sie ein kleines C-Programm `setjmp-demo.c`, bei dem eine vom Hauptprogramm aufgerufene Funktion sich selbst (rekursiv) aufruft, und nach 5 solcher Rekursionen mit Hilfe von `longjmp()` direkt ins Hauptprogramm zurückkehrt.

(b) Erproben Sie dieses Programm mit Hilfe des Debuggers. Verfolgen Sie dabei die Adressen der lokalen Variablen der Funktion. Was können Sie dabei beobachten? Wie ist es möglich, das ein und dieselbe lokale Variable unterschiedliche Adressen besitzt?

Aufgabe 5.3 (Adressräume):

Auf der Webseite der Veranstaltung finden Sie Quellcode und Makefile für die C-Bibliothek `libmemprobe.a`. Diese bietet eine Funktion `memprobe()`, mit der versuchsweise auf beliebige Speicheradressen zugegriffen werden kann. Der bei ungültigen Adressen normalerweise auftretende Programmabbruch wird mit Hilfe eines Signal Handlers vermieden.

(a) Verstehen Sie den Code. Beachten Sie besonders die Verwendung des Funktionspaares `setjmp()` und `longjmp()` im Zusammenspiel mit dem Signal Handler. Erklären Sie die Arbeitsweise.

(b) Schreiben Sie ein Hauptprogramm `memprobe.c`, das unter Verwendung der Funktion `memprobe()` seinen gesamten Adressraum abtastet und anzeigt, auf welche Speicherbereiche Lese-, bzw. Schreib- und Lesezugriff besteht.

Hinweise:

(1) Die unterlagerte Hardware (MMU) kann Zugriffsrechte nur jeweils für ganze Speicherseiten verwalten. Es genügt daher bei der Abtastung, nur jeweils einen Zugriffsversuch pro Speicherseite vorzunehmen. Die Größe der Speicherseiten Ihres Rechners (und damit die Schrittweite Ihrer Abtast-Schleife) können Sie mit der Funktion `getpagesize()` ermitteln.
(2) Falls Ihr Rechner mehr als 32 Bit Adressbreite verwendet, beschränken Sie die Abtastung auf die ersten 4 Gigabyte des Adressraums, da eine vollständige Abtastung des gesamten Adressraumes zu lange dauern würde.

Aufgabe 5.4 (Benutzeradressraum):

Die Speicherobjekte eines Programmes sind verschiedenen Bereichen zugeordnet, die mit unterschiedlichen Zugriffsrechten und Eigenschaften im Adressraum eines Benutzerprozesses erscheinen:

- **Code:** Der Programmcode, der ausgeführt wird. Nur Leserechte, Größe zur Laufzeit konstant.
- **Daten:** Globale (statische) Daten des Programmes. Schreib- und Leserechte, Größe zur Laufzeit konstant. Ggf. weitere Unterteilung in initialisierte und nicht initialisierte Variablen.
- **Heap:** Dynamisch (z.B. mit `malloc()`) allozierter Speicher. Schreib- und Leserechte. Größe kann sich zur Laufzeit ändern (wachsen).
- **Stack:** Speicher für lokale Variablen, Rücksprungadressen, etc. Schreib- und Leserechte, Größe kann sich zur Laufzeit ändern (wachsen).

(a) Erweitern Sie das Programm `memprobe.c` aus Aufgabe 2.3 zu einem Programm `memprobe1.c`, das für die gefundenen, zugreifbaren Speicherbereiche zusätzlich jeweils nach Möglichkeit angibt, zu welchem der vier genannten Speicherbereiche sie gehören. Deklarieren, bzw. allozieren Sie dazu jeweils Speicherobjekte der genannten vier Klassen und vergleichen Sie deren Adressen mit den Grenzen der gefundenen Speicherbereiche.
(b) Deklarieren Sie in Ihrem Programm eine Zeichenkette auf zwei verschiedene Arten:

(1) `static char *string = "Hallo Welt\n";`
(2) `static char string[]= "Hallo Welt\n";`

In beiden Varianten bezeichnet das Symbol `string` einen Zeiger auf eine (null-terminierte) Zeichenkette. Überprüfen Sie mit den Ihnen nun zu Gebote stehenden Mitteln, in welchem der genannten vier Bereiche die Zeichenkette jeweils liegt. Bei welcher Variante ist ein Verändern der Zeichenkette zur Laufzeit möglich?