Betriebs- und Kommunikationssysteme

WS 15/16

Prof. Dr-Ing. Volker Roth

Freie Universität Berlin

4. Übung

Ausgabe	Abgabe
6.10.15	20.11.15

Bitte bei der Abgabe Name der Mitglieder einer Gruppe, Nummer der Übung/Teilaufgabe und Datum auf den Lösungsblättern nicht vergessen! Darauf achten, dass die Lösungen beim richtigen Tutor abgegeben werden. Achten Sie bei Programmieraufgaben außerdem darauf, dass diese im Linuxpool kompilierbar sind. Nutzen Sie dazu die Flags `-std=c99`, `-Wall` und `-pedantic`. Es sollten keine Warnungen auftauchen.
Zu spät abgegebene Lösungen werden nicht berücksichtigt!

Aufgabe 1: Prozesse (2 Punkte)

In dieser Aufgabe sollen Sie sich genauer mit dem *Process Control Block* (PCB) befassen und für die Sicherheit in Betriebessystemen mit Hilfe eines Fallbeispiels sensibilisiert werden. Ziel ist es `nicht`, dass Sie den im Artikel beschrieben Angriff reproduzieren können oder die Sicherheitslücke im kleinsten Detail verstehen. Sie sollten vor der Bearbeitung die Hinweise lesen.

(a) Grenzen Sie *Process* und *Thread* voneinander ab.

(b) Beschreiben Sie die Funktionen der Felder *Process identifiers*, *CPU state*, *Control information* und erklären Sie die Notwendigkeit dieser Felder.

(c) Erklären Sie den *Program Counter* und *Stack Pointer*.

(d) Recherchieren Sie den Begriff *Call Stack* (Aufrufstapel) und beschreiben Sie diesen. Was ist eine Rücksprungadresse und warum ist die notwendig?

(e) Lesen Sie den verlinkten Artikel [One98]und beantworten Sie folgende Fragen:

(1) Welche Datenstruktur in C soll ausgenutzt werden?
(2) Was passiert im Codebeispiel `example2.c`? Gegeben Sie ggf. eine Zeichnung an.
(3) Welches Sicherheitsproblem wird durch das Codebeispiel `example2.c` deutlich?
(4) Was kann der Programmierer dagegen beim Programmieren tun?
(5) Welchen Wert auf dem *Stack* möchte der Angreifer manipulieren und warum?
(6) Was möchte der Angreifer erreichen und warum ist es lohnenswert?
(7) Warum muss der *Shellcode* in Assemblersprache übersetzt werden?
(8) Warum kann der Anfang des *Stacks* leicht ermittelt werden?
(9) Es ist schwer die genaue Rücksprungadresse zu finden. Wie kann dieses Problem umgangen werden?

(f) Fassen Sie den Angriff *Buffer overflow* zusammen.

(g) Wie kann ein Betriebssystem einen *Buffer overflow* verhindern?

Hinweise:

- Ein guter Startpunkt zur Recherche ist die Einführung des verlinkten Artikels.
- Die Codebeispiele sind für die Beantwortung der Fragen hilfreich aber nicht notwendig.
- Für die Fragen 1 bis 6 der Teilaufgabe e reicht es aus die ersten sechs Seiten zu verstehen.

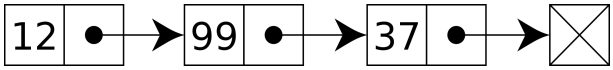

Abbildung 1: Aufbau einer verketteten Liste

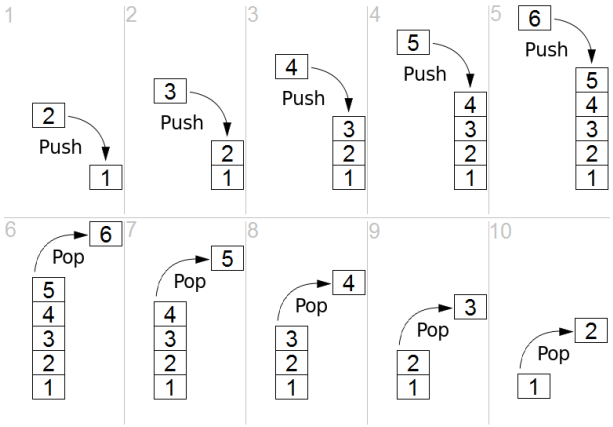

Abbildung 2: Funktionsweise eines Stapels.

Aufgabe 2: Einführung in C (3 Punkte)

In dieser Aufgaben sollen Sie sich mit *structs* (komplexeren Datenstrukturen) und *pointers* (Zeiger) in C vertraut machen. Recherchieren Sie zu nächst folgende Begriffe:

struct, *pointer*, *malloc* , *free* und *sizeof()*

Machen Sie sich anschließend mit der Verwendung vom $\star$-Operator vertraut.

Eine einfache *linked list* (verketteten Liste) ist eine dynamische Datenstruktur und besteht aus einer Menge von Knoten, die untereinander verkettet sind. Jeder Knoten besteht aus dem zu speichernden Objekt und einem Zeiger auf das nächste Element. Das letzte Element der Liste zeigt auf NULL und mittels eines Zeigers *head* wird auf den ersten Knoten in der Liste gezeigt. Abbildung 1 illustriert eine einfach verkettete Liste.

Eine besondere Form einer einfach verketteten Liste ist der *Stack* (Stapel). Neue Listenelemente werden immer „oben“ auf dem Stack abgelegt. Gelesen werden kann ebenfalls nur das oberste Element. Ein Stack arbeitet also nach dem LIFO-Prinzip (*last in, first out*). Abbildung 2 illustriert die Funktionsweise eines Stacks.

a) Verwenden Sie das bereitgestellte Framework und implementieren Sie einen Stack mittels einer verketteten Liste mit den folgenden Funktionen:

   `bool is_empty()`: testet, ob der Stack leer ist

   `void push(item *elem)`: legt ein Element auf dem Stack ab

   `void *pop()`: entfernt das oberste Element vom Stack und gibt es zurück

   Halten Sie sich an die vorgegebenen Funktionsdefinitionen. Der Typ `bool` wird C-intern wie `int` gehandhabt, `void *` wird verwendet, um einen allgemeinen Zeigertyp zu deklarieren.

b) Stacks werden z.B. oft in mathematischen Koprozessoren eingesetzt (z.B. auch im Intel-Pentium). Dabei wird zunächst die nötige Anzahl Operanden auf dem Stack abgelegt, dann wird die arithmetische Funktion aufgerufen und schließlich kann das Ergebnis vom Stack

abgeholt werden. Die arithmetische Funktion selbst liest die nötige Anzahl von Operanden vom Stack, führt die Operation aus und legt das Ergebnis auf dem Stack ab. Modifizieren Sie den Stack von Aufgabeteil a) derart, dass er `int`-Zahlen aufnimmt. Implementieren Sie die Funktion `multiplication()`, die wie beschrieben auf dem Stack operiert. Sie holt sich zunächst die Zahl der Faktoren und dann die einzelnen Operanden vom Stack.

Kommentieren und testen Sie Ihr Programm.

Literatur

[One98] Aleph One. Smashing the stack for fun and profit. *Phrack Magazine*, 49(14), 1998.