Gruppe: _____

Name: _____

Matr.Nr.: _____

Erfolg: _____

Übung 8

Systemtheorie 1

Wintersemester 2007/2008

Abgabe 11.01.2007 10:30

Institut für Formale Modelle und Verifikation, Dr. Toni Jussila, Dipl.-Ing. Robert Brummayer

Aufgabe 29

a) Berechnen Sie mit `bad_string_hash`, `very_bad_string_hash`, und `classic_string_hash` aus der Vorlesung den Hash-Wert von dem String `otto` unter der Annahme einer ASCII-Kodierung des Strings. Geben Sie jeweils alle Zwischenergebnisse an.

b) Geben Sie für die Hash-Funktion `bad_string_hash` zwei unterschiedliche Strings mit demselben Hash-Wert an.

c) Geben Sie für die Hash-Funktion `very_bad_string_hash` zwei unterschiedliche Strings mit demselben Hash-Wert an.

Aufgabe 30

Wir betrachten nun eine weitere Hash-Funktion. Sei `s` die als Argument übergebene Zeichenfolge, `h` der Hashwert und `asc(c)` der ASCII Code vom Zeichen `c`.

- Zu Beginn gilt `h = 0`.
- Die Funktion iteriert über alle Zeichen von `s`.
- Wird das Zeichen `c` besucht, dann berechnet sich der neue Wert von `h` folgendermaßen:

$$h_{new} = h_{old} \veebar ((h_{old} << 5) + (h_{old} >> 2) + \texttt{asc(c)})$$

- Nachdem das letzte Zeichen des Strings besucht wurde, gibt die Funktion den Wert von `h` als Ergebnis zurück.

$\veebar$ bezeichnet das bitweise exklusive Oder und $<<$ bzw. $>>$ steht für die Links- bzw. Rechts-Shift-Operation.

a) Implementieren Sie die oben angegebene Hash-Funktion in der Programmiersprache C. Gleich neben dem Link zur Übungsangabe finden Sie die Datei `testhashex.c`. Laden Sie diese Datei runter, suchen Sie das Skelett der Funktion `sax_string_hash` und vervollständigen Sie die Funktion.

   Sie können ihre Implementierung testen, indem Sie `testhashex.c` mit ihrem C-Compiler übersetzen und anschließend das Programm mit dem Argument `--sax` und einer beliebigen Anzahl von Quellcode-Dateien aufrufen.

   Drucken Sie ihre Implementierung aus und geben Sie den Ausdruck mit ab. Rufen Sie weiters `testhashex` mit ihrer Hash-Funktion `(--sax)` und der Datei `testhashex.c` auf und geben Sie einen Screenshot der Programmausgabe ab.

b) Neben dem Link zur Übungsangabe finden Sie ein Archiv `sources.zip`, das eine Auswahl von Quellcode-Dateien aus dem aktuellen Linux-Kernel enthält. Entpacken Sie dieses Archiv und testen Sie alle Hash-Funktionen auf den Source-Dateien. Ein Beispiel für einen Aufruf unter Linux, der die Hash-Funktion `bad_string_hash` testet, ist: `testhashex --bad sources/*`. Welche Hash-Funktion schneidet am Besten und welche am Schlechtesten ab? Wie gut verhält sich die von Ihnen implementierte Hash-Funktion?

Aufgabe 31

Gegeben seien zwei Hashtabellen `HT1` und `HT2` mit jeweils einer Initialgröße von 100. Um ein schnelles Auffinden der Elemente einer Hashtabelle gewährleisten zu können, muss die Größe der Hashtabelle dynamisch angepasst werden. Wird die Hashtabelle im Laufe der Zeit zu klein, so muss sie vergrößert und anschließend ein Rehashing durchgeführt werden. Bei diesem Rehashing wird die Hashtabelle umstrukturiert, wobei die Hash-Werte der bereits vorhandenen Elemente neu berechnet werden müssen.

`HT1` verwendet folgende Strategie: Ist nach der Einfüge-Operation die Anzahl der Elemente in der Hashtabelle dividiert durch die Größe der Hashtabelle gleich 0,5, so wird die Größe der Hashtabelle um 100 erhöht und anschließend ein Rehashing durchgeführt.

`HT2` verwendet eine andere Strategie: Ist nach der Einfüge-Operation die Anzahl der Elemente in der Hashtabelle gleich der Größe der Hashtabelle, so wird die Größe der Hashtabelle verdoppelt und anschließend ein Rehashing durchgeführt.

Es werden nun 10000 beliebige Strings eingefügt.

a) Wie oft wird bei `HT1` und bei `HT2` die Größe angepasst?

b) Wieviele Hash-Wert-Berechnungen werden bei `HT1` und bei `HT2` durchgeführt? Berücksichtigen Sie bei ihren Berechnungen das Rehashing.

Aufgabe 32

Wir betrachten nun den in der Vorlesung behandelten Super-Trace-Algorithmus.

a) Wie kann der Super-Trace-Algorithmus in Verbindung mit einer nicht-kollisionsfreien Hash-Funktion sinnvoll im Bereich des Model-Checkings eingesetzt werden?

b) In einem archaischen Betriebssystem mit einem Adressraum von 24 Bit können Sie die Hälfte des addressierbaren Speichers für das Bit-State-Hashing verwenden. Wieviel Mega-Byte verbraucht das Array für das Bit-Set und wie groß muss der Wertebereich der Hash-Funktion sein bzw. mit wievielen gültigen Bits sollte das Ergebnis der Hash-Funktion kodiert sein?