LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

CRC32 (A401)

Projektaufgabe – Aufgabenbereich Theorie der Informationsverarbeitung

1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung festgesetzt, die Sie über die Praktikumshomepage [1] aufrufen können.

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. Die Teile der Aufgabe, in denen Assembler-Code anzufertigen ist, sind für die 64-Bit ARMv8-Architektur (AArch64) zu schreiben.

Der **Abgabetermin** ist der **3. Februar 2019, 23:59 Uhr (MEZ)**. Die Abgabe erfolgt per Git in für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der Praktikumsordnung angebene Liste von abzugebenden Dateien.

Der **erste Teil Ihrer Projektpräsentation** ist eine kurze Vorstellung Ihrer Aufgabe im Umfang von ca. 2 Minuten in einer Tutorübung in der Woche **10.12.2018 – 14.12.2018**. Erscheinen Sie bitte **mit allen Team-Mitgliedern** und wählen Sie bitte eine Übung, in der mindestens ein Team-Mitglied angemeldet ist.

Die **Abschlusspräsentationen** finden in der Woche vom **04.03.2019 – 08.03.2019** statt. Weitere Informationen zum Ablauf und die Zuteilung der einzelnen Präsentationstermine werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihren Tutor.

Wir wünschen Ihnen viel Erfolg und Freude bei der Bearbeitung Ihrer Aufgabe!

Mit freundlichen Grüßen
Die Praktikumsleitung

PS: Vergessen Sie nicht, sich rechtzeitig in TUMonline zur Prüfung anzumelden. Dies ist Voraussetzung für eine erfolgreiche Teilnahme am Praktikum im laufenden Semester.

[1] `https://www.caps.in.tum.de/lehre/ws18/praktika/asp/`

2 CRC32

2.1 Überblick

Die Informationstheorie ist ein Feld der Mathematik, in welchem man sich allgemein mit dem Kodieren von Nachrichten aufgrund von statistischen Gegebenheiten beschäftigt. Sie werden in Ihrer Projektaufgabe einen bestimmten Teilbereich näher beleuchten und einen bestimmten Algorithmus in Assembler implementieren.

2.2 Funktionsweise

Die Zyklische Redundanzprüfung ist ein Fehler erkennender Algorithmus, welcher gerne zum Erstellen einfacher Prüfsummen von Dateien verwendet wird.
Die Berechnung der CRC32 beruht auf Polynomdivision. Dabei wird die Bitfolge einer zu prüfenden Nachticht N der Länge l als Polynom mit binären Koeffizienten K_i aufgefasst:

$$N = x_1 x_2 x_3 \dots x_l \qquad = \sum_{i=0}^{l} K_i \cdot x^i = N(x)$$

Als Dividend für die Polynomdivision kommt ein sogenanntes Generatorpolynom $G(x)$ zum Einsatz. Typischer Weise wählt man $G(x)$ als

$$G(X) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0 \quad (1)$$

Die zu berechnende Prüfsumme P ist dann der Rest der Polynomdivision von $N(x)/G(x)$. Formal gilt:

$$\mathrm{CRC32}(N, G) = P(x) \equiv N(x) \mod G(x)$$

Die Prüfsumme P wird nach der Berechnung einfach an die ursprüngliche Nachricht N angehängt: $N' = (N||P)$
Der Empfänger führt nach Erhalten der Nachricht N' die gleiche Polynomdivision mit dem gleichen $G(x)$ aus. Ergibt sich als Rest 0, so kann davon ausgegangen werden, dass die Nachricht mit großer Wahrscheinlichkeit korrekt übermittelt wurde.

2.3 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Alle Antworten auf konzeptionelle Fragen sollten in Ihrer Ausarbeitung erscheinen. Besprechen Sie nach eigenem Ermessen außerdem im Zuge Ihres Vortrags einige der konzeptionellen Fragen. Die Antworten auf die Implementierungsaufgaben werden durch Ihrem Code reflektiert.

2.3.1 Theoretischer Teil

- Erarbeiten Sie anhand geeigneter Sekundärliteratur die genaue Funktionsweise des CRC32-Algorithmus'. Rechnen Sie zunächst auf Papier nach, dass das oben skizzierte Verfahren funktionieren kann. Überlegen Sie sich auch, wie Addition und Subtraktion innerhalb eines binären Feldes funktionieren.
- Welche Arten von Fehlern erkennt CRC32? Zeigen Sie, dass es mindestens eine Klasse von Fehlern gibt, für die CRC32 nicht funktioniert. Geben Sie dazu geeignete Beispiele für zwei Nachrichten N und M mit $N \neq M$ an, für die aber CRC32(N) =CRC32(M) gilt.
- Suchen Sie nach alternativen Implementierungen von CRC32 und vergleichen Sie sie mit der Ausgabe Ihres fertigen Programms. Vergleichen Sie auch die Geschwindigkeit Ihrer Implementierung und einer alternativen (beispielsweise C) Implementierung.
- Für ein konstantes G können die Werte der Polynomdivision vorausberechnet werden. Tun Sie dies für alle 256 verschiedenen Werte, die ein Eingabebyte annehmen kann. Verwenden Sie das G aus Angabe 1.

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie Ihrem Assemblerprogramm die Inhalte einer eingelesenen Datei als Pointer übergeben können.
- Implementieren Sie eine C-Funktion

  ```
  int CRC32_C(char *message)
  ```

  welche einen Pointer auf eine Nachricht `message` übergeben bekommt und die Prüfsumme P als Integer zurückgibt. Die Funktion soll dabei das oben definierte Generatorpolynom G als Konstante verwenden. Verwenden Sie für die Berechnung der CRC32-Prüfsumme (mindestens) eine Schleife und implementieren Sie die binäre Polynomdivision.
- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

  ```
  int CRC32_ASM(char *message)
  ```

  welche einen Pointer auf eine Nachricht `message` übergeben bekommt und die Prüfsumme P als Integer zurückgibt. Die Funktion soll intern mit einem Tabellen-Lookup für die 256 verschiedenen Ergebnisse der Polynomdivision pro Byte arbeiten. Eine einfache Übersetzung Ihres C-Programms in Assembler wird hier **nicht** gewertet!

2.3.3 Bonusaufgabe

Erweitern Sie Ihre Implementierung derartig, dass das Programm IEEE802.3 Frames (Ethernet) lesen kann und überprüfen Sie die in der Frame Check Sequence enthaltene CRC32 Prüfsumme.

2.4 Checkliste

Die folgende Liste soll Ihnen als Gedächtnisstütze beim Bearbeiten der Aufgaben dienen. Beachten Sie ebenfalls die in der Praktikumsordnung angegebenen Hinweise. Sollten Sie eine Reverse-Engineering-Aufgabe erhalten haben, sind diese Punkte für Sie weitestgehend hinfällig.

- Verwenden Sie keinen Inline-Assembler. Assembler muss separate geschrieben werden
- I/O-Operationen dürfen grundsätzlich in C implementiert werden.
- Sie dürfen die Signatur der in Assembler zu implementierenden Funktion nur dann ändern, wenn Sie dies (in Ihrer Ausarbeitung) rechtfertigen können.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz) und behandeln Sie alle möglichen Eingaben, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Verwenden Sie SIMD-Befehle, wenn möglich.
- Fügen Sie Ihrem Projekt ein funktionierendes `Makefile` hinzu, welches durch den Aufruf von `make` Ihr Projekt kompiliert.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden.
- Verwenden für die Ausarbeitung Sie die bereitgestellte TEX-Vorlage und legen Sie sowohl die PDF-Datei als auch sämtliche TEX-Quellen in das Repository.
- Stellen Sie Performanz-Ergebnisse grafisch oder tabellarisch dar.
- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Bonusaufgaben (sofern vorhanden) müssen nicht implementiert werden.
- Geben Sie die Folien für Ihre Abschlusspräsentation im PDF-Format ab. Achten Sie auf hinreichenden Kontrast (schwarzer Text auf weißem Grund!) und eine angemessene Schriftgröße. Verwenden Sie 4:3 als Folien-Format.