Klauck:

Damit ist Aufgabe 3 nicht erfolgreich beendet. Um weiter am Praktikum teilnehmen zu können bzw. die PVL noch erhalten zu können ist eine Fehleranalyse bis Dienstag 20:00 Uhr einzureichen. Sofern diese akzeptiert wird, kann die PVL noch erlangt werden.

Die Fehleranalyse ist ausführlich durchzuführen,

* Zudem bitte ich um eine Beschreibung, wie die aufgetretenen Fehler bisher behoben bzw. bearbeitet wurden. Ggf. kann eine aktuelle Architektur (z.B. welche Prozesse laufen nebenläufig) und Sequenzdiagrame/Abläufe (z.B. wie werden die zeitkritischen Abläufe behandelt bzw. wo finden sie statt) Bitte geben Sie das Dokument als pdf ab.

Done in erster Version:

* vom Entwurf (Lokalisierung der möglichen Fehlerquellen)
* über die technische Umsetzung (Code, konkrete Lokalisierung der logischen
* wie auch möglichen technischen Fehlerquellen (Fehlerursachen)
* bis hin zur Fehlerwirkung.
* Die Ihnen bekannten Fehler (Sie selbst haben im Test noch Fehler beobachtet)
* und die hier (siehe auch angehängtes log) aufgeführten Fehler.

Fehleranalyse

Aufgabe 3 – Team6

Verteilte Systeme – Prof. Dr. Klauck & Dipl.-Ing Schulz

Sommersemester 2018

Gliederung:

1. Was waren die Fehler (Fehlerwirkung)?
2. Wo entstanden die Fehler (Lokalisierung)?
3. Wie entstanden die Fehler (Fehlerquellen & -ursachen)?
4. Wieso fiehlen die Fehler nicht auf?
5. Wie wurden sie behoben (Lösungsstrategie)?

(Prologe) Was war die Aufgabe?

Die Anwendung die wir Realisieren soll das Zeitmultiplex verfahren umsetzen.

Dabei sollen die Stationen über UDP Multicast (auch Kanal genannt) miteinander kommunizieren. Wobei auf Kollisionen geachtet wird.

Die Vergabe der Sende-Slots wird bei jeder Station lokal individuell bestimmt, indem man Nachrichten aus dem Kanal auswertet. Zudem werden die Nachrichten von Typ A Stationen verwendet, um die Uhren der eigenen Station zu synchronisieren.

1. Was waren die Fehler?

Das Programm führte zu Kollisionen die nicht hätten sein dürfen.

(Fehler 1 & 2) Auffällig war, dass schon reservierte Slots noch einmal reserviert wurden. Dies darf nicht passieren. Jeder Slot darf nur einmal reserviert werden!

So kam es dazu, dass diese Regel nicht beachtet wurde und (meist 2) Stationen im selben Slot gesendet haben, was dann zu Kollisionen führte die nicht hätten stattfinden dürfen.

Hier hat also die Slotreservierung nicht funktioniert wie sie sollte.

(Fehler 3) Hinzu kommt in seltenen Fällen, dass richtig reserviert wurde, jedoch der Slot so verpasst wurde, dass man im Slot davor / danach sendete. Was wiederum zu falschen Kollisionen führte.

Hier hat also die Prüfung der Sendezeit beim Senden nicht richtig funktioniert.

1. Wo und Wie entstanden die Fehler?

Der Entwurf ist noch richtig.

(Fehler 1 & 2) In der Sendephase wird erst kurz vor Senden der Nachricht der Slot für den nächsten Frame vom SlotFinder angefordert. So ist garantiert, dass alle bisher empfangenen Nachrichten (und deren reservierten Slots) berücksichtigt wurden.

(Fehler 3) Auch hier passt es im Entwurf noch.

Die Frames sind exakt 1000 MS, die Slots genau 40 MS lang. Es sollt nicht gesendet werden wenn die Slot Mitte des Sendeslots verpasst wird.

Die Implementation war nicht fehlerfrei.

(Fehler 1) Bei der Implementation allerdings wurde ein logischer Fehler begangen.

Hier wurde schon lange vor dem Senden (ca. 20-30 MS) ein Slot für den nächsten Frame angefordert. Also eine Abweichung vom Entwurf die so hätte nicht passieren dürfen und die zu Problemen führte.

(Fehler 2) Jedoch war dies nicht der einzige logische Fehler.

Im SlotFinder gibt es eine Liste (als Konstante) mit allen möglichen Slots.

Diese Liste sollte die Zahlen 1 bis einschließlich 25 je einmal halten.

Jedoch kam die 24 zwei Mal vor.

(Fehler 3) Auch hier war die Implementation der Fehler. Dieses Mal technischer Natur, da „timer“ in Erlang nie 100% exakt arbeitet und nur eine Mindestwartezeit statt die Gesamtwartezeit garantiert.

Unser Programm hatte als Kernstück core.erl. Dieses Modul besitzt die frame\_loop Funktion. Diese repräsentiert einen Frame und sollte exakt 1000 MS lang sein.

Ab und zu war sie jedoch 999 MS bzw. 1001 MS lang.

Dies führte dazu das nach einiger Zeit es sich nicht ausgeglichen hat, sondern die Länge von 1001 MS wohl öfter vor kam als 999 MS. Weshalb der Frame für eine Station immer später begann (Weil die Loop sequentiell ausgeführt wurden, also aufeinander aufbauten). Weshalb die Station dachte sie sende in Slot 2, war global jedoch gesehen schon in Slot 3.

1. Wieso fielen die Fehler nicht auf?

(Fehler 1 & 2)

Zwischen Abgabe und Nachreichung wurde ein Problem gefixt.

Und zwar bekommt der SlotFinder ja die kollisionsfreien empfangenen Nachrichten und wertet diese aus. Hierbei wird aus der Liste der möglichen Slots (nicht reservierten) die Slots aus den Nachrichten (reservierte) gelöscht. Mit der übrig bleibenden Liste (nicht reservierter) wird dann bei Slot Anforderung per Zufall ein Slot ausgewählt.

Das Problem war, dass wir hier zunächst nicht die neue Liste (nicht reservierter) übergeben haben, sondern ausversehen noch die alte (mit allen generell möglichen).

Deswegen wurden hier die einkommenden Nachrichten nicht vollends verwertet.

Diesen Fehler haben wir bemerkt. Und dementsprechend die richtige Liste übergeben. Damit dachten wir wäre das Problem der fehleranfälligen Slot Findung gelöst. Und sendeten die Nachreichung ab.

Wir hätten vor dem Absenden noch einen Lauf machen sollen, dann wären uns Fehler 1 & 2 noch aufgefallen. Jedoch hatten wir Samstagnachmittag bis abends kein Zugriff auf einem Linux Rechner. Weshalb wir es nicht testen konnten.

(Fehler 1)

Dieser Fall wurde so nicht getestet.

Der Grund hierfür ist, dass ein solcher Testcaste sehr viele Komponenten umfasst und dementsprechend zu viel Zeit gebraucht hätte, wir die Zeit jedoch für \*\*\*\*\*\*\*\*\*.

(Fehler 2)

Die Liste an sich wurde nie selbst getestet.

In den Funktionen in denen die Liste gebraucht wird wurde sie nicht per „?Konstantenname“ genutzt sondern per Parameter mitgegeben. So wurde beim Testen immer eine eigene Liste mit möglichen Slotnummern mitgegeben statt die interne Konstante zu verwenden.

(Fehler 3)

Der Fehler fiel bei genauem Blick auf die Logs auf, doch hatten wir erst spät eine Idee wie wir ihn lösen können.

1. Wie wurden sie behoben?

(Fehler 1)

Es fiel auf, dass es immer der Fall war, dass es bei Stationen war die direkt auf einander folgten. Zum Beispiel sendete Station 10 in Slot 10, dass diese im nächsten Frame Slot 20 reserviert. Station 11 in Slot 11, wollte auch die 20. Alle anderen Stationen wurden richtig berücksichtigt.

So lag es nahe, dass die Nachricht von Station 10 ggf. nicht berücksichtigt wurde wenn es darum ging einen eigenen Slot im nächsten Frame zu reservieren. Der Fehler lag dann im Detail daran, dass der Slot zu früh angefordert wurde. Zu dem Zeitpunkt der Slot Anforderung sendete eine andere Station (im Slot davor) zeitgleich die Nachricht, die dann eben nicht berücksichtigt wurde. Bis diese nämlich voll ausgewertet wurde hat die eigene Station längst einen Slot ausgewählt (der mit ein wenig Pech der ist den die Station davor gerade reserviert hatte).

Die Slot Anforderung wurde nun zeitlich nach hinten verschoben, direkt vor dem Senden der Nachricht wird nun erst ein Slot angefordert. Somit ist garantiert, dass alle empfangenen Nachrichten des Frames berücksichtigt werden.

(Wie im Entwurf spezifiziert)

(Fehler 2)

Es fiel auf, wenn die Stationen nicht direkt nacheinander kamen (Fehler 1), dass die Kollisionen nur entstanden wenn es um Slot 24 ging. Also haben wir die Konstante Liste in SlotFinder näher angeschaut und realisiert, dass dort eben die 24 zwei Mal vor kommt.

Die zweite 24 wurde aus der Liste gelöscht.

(Fehler 3)

Man konnte sehen, dass nach langer Laufzeit, die Stationen in Slot X senden wollten, aber in X + 1 (laut Sniffer) erst ankamen. Also haben wir angefangen zu loggen wann ein Frame beginnt und endet. Dabei kam heraus, dass manche Frames 1001 MS brauchen und sich dies dann akkummuliert. Also darf diese Akkumulierung nicht geschehen.

Nun haben wir einen Timer in der UTCClock der nach 1000 MS sich selber ein Signal gibt.

Dieser Timer wird angepasst wenn der Offset angepasst wurde (also dementsprechende ‚Restzeit im aktuellen Frame‘ – 1). Der Timer wird beim starten der Uhr erstellt und jedes Mal resettet wenn eine neuer Frame beginnt.

Läuft der Timer ab, schickt der Timer eine bestimmte Nachricht an die UTCClock, die dann die ein bestimmtes atom, die aktuelle Uhrzeit und den aktuellen Offset an den Core schickt. Für ihn das Zeichen, dass ein neuer Frame eben begonnen hat.

Der Timer ist initial auf 1000 MS gestellt, da etwas Zeit vergeht bis die Nachricht (dass ein neuer Frame angefangen hat) beim Core angekommen ist und der Timer selbst manchmal zusätzliche Zeit benötigt, könnte es jedoch unter Umständen dazu kommen, das die Nachricht im Core 1 MS zu spät ankommt.