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. Fehler 1:
   1. Wo entstanden die Fehler (Lokalisierung)?
   2. Wie entstanden die Fehler (Fehlerquellen & -ursachen)?
   3. Wieso fielen die Fehler nicht auf?
   4. Wie wurden sie behoben (Lösungsstrategie)?
3. Fehler 2 (analog zu Fehler 1)
4. Fehler 3 (analog zu Fehler 1)
5. Generelles Vorgehen beim Testen
6. Quellen

(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.

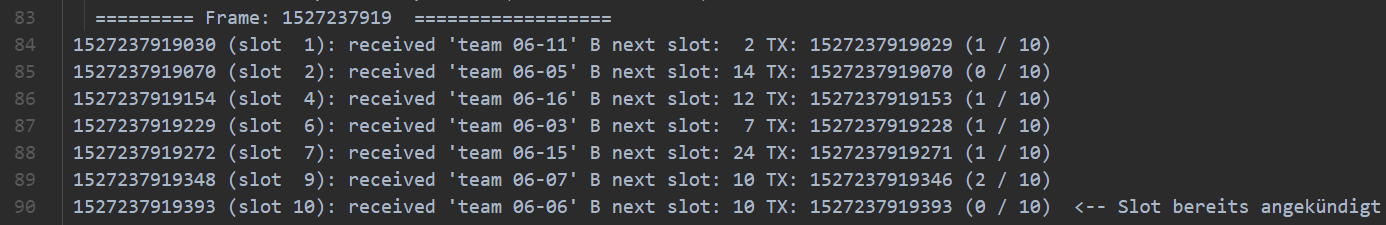
1. Was waren die Fehler?

Das Programm führte zu Kollisionen die nicht hätten sein dürfen.

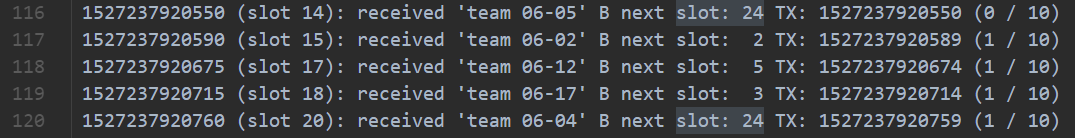
(Fehler 1 & 2) Auffällig war, dass schon reservierte Slots noch einmal reserviert wurden. Dies darf nicht passieren. Jeder Slot darf nur einmal reserviert werden!

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.



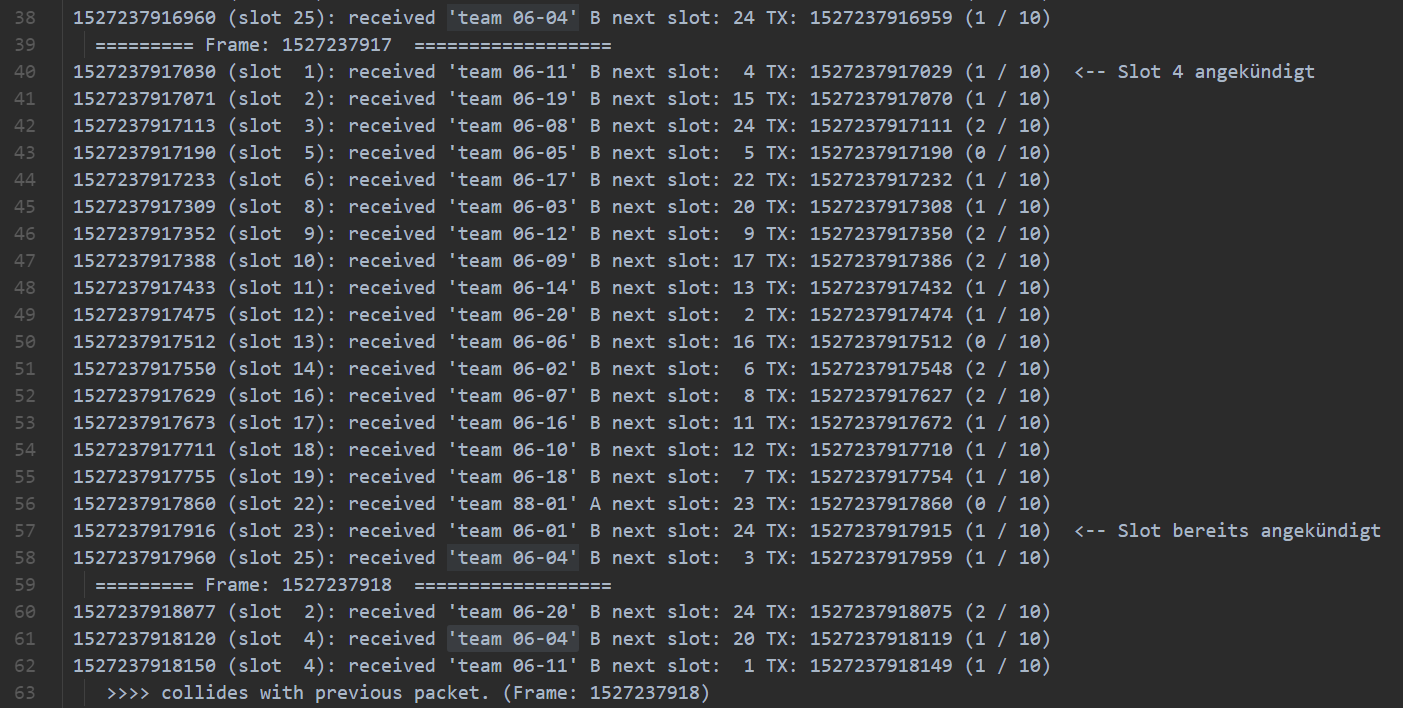
*Abb. 1, Visualisierung von Fehler 1*

**

*Abb. 2, Visualisierung von Fehler 2*

(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 oder die innere Uhr der Station falsch lief.

**

*Abb. 3, Visualisierung von Fehler 3*

1. Fehler 1
   1. Wo entstand der Fehler?

(Fehler 1 & 2)

Der Entwurf ist noch richtig.

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.

Die Implementation war nicht fehlerfrei.

Es fiel auf, dass es immer der Fall war, dass es bei Stationen war die direkt auf einander folgten. Zum Beispiel (Abb. 1) sendete Station 07 in Slot 9, dass diese im nächsten Frame Slot 10 reserviert. Station 06 in Slot 10, wollte auch die 10. Alle anderen Stationen wurden richtig berücksichtigt.

So lag es nahe, dass die Nachricht von Station 07 in Station 06 ggf. nicht berücksichtigt wurde, wenn es darum ging einen eigenen Slot im nächsten Frame zu reservieren.

* 1. Wie entstand der Fehler?

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).

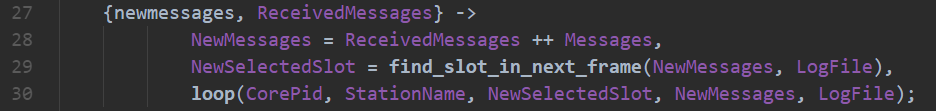
Es handelt sich um einen logischen Fehler.

* 1. Wieso fiel der Fehler nicht eher auf?

Zwischen Abgabe und Nachreichung wurde ein Problem gefixt.

Und zwar bekommt der SlotFinder die kollisionsfreie Nachricht (geschaltet in einer Liste) und merkt sich diese (fügt sie hinten an die bereits erhaltenen Nachrichten an), um sie in find\_slot\_in\_next\_frame auszuwerten.

Irrtümlich wurde allerdings im Aufruf von find\_slot\_in\_next\_frame (Zeile 29) jedoch nicht NewMessages sondern Messages (also die vor dem aktuellen Slot empfangenen Nachrichten) übergeben. So wurde die letzte empfangene Nachricht nicht direkt ausgewertet.



*Abb. 4, der nun richtige 1. Parameter der find\_slot\_in\_next\_frame Funktion*

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 wäre uns Fehler 1 noch aufgefallen. Jedoch hatten wir Samstagnachmittag bis abends kein Zugriff auf einem Linux Rechner. Weshalb wir es nicht testen konnten.

Zudem wurde dieser Fall so nicht getestet.

Der Grund hierfür ist, dass ein solcher Testcaste sehr viele Komponenten umfasst und dementsprechend zu viel Zeit gebraucht hätte, die wir Zeit jedoch nicht hatten, da wir an der Abgabe am Freitag bis halb 9 im Labor saßen und raus geworfen wurden, da keine Aufsichtsperson mehr vorhanden war.

* 1. Wie wurde er behoben?

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 bis dahin empfangenen Nachrichten des Frames berücksichtigt werden, was nun den Slot direkt vor dem eigenen miteinschließt.

(Wie im Entwurf spezifiziert)

1. Fehler 2
   1. Wo entstand der Fehler?

(Siehe Fehler 1, selber Abschnitt)

Die Implementation war nicht fehlerfrei.

Bei Testläufen hatten wir uns ausgeben lassen welcher Slot nun schon reserviert ist und welche noch nicht reserviert wurden. Dabei viel auf, dass obwohl die 24 schon reserviert war sie dennoch noch in der Liste der nicht reservierten enthalten war. Beim genauerem Hinsehen konnten wir feststellen wie sich zu Beginn eines neuen Frames stets zwei Mal die Zahl 24 in der Liste der nicht reservierten Slots befand.

Da bei Framebeginn die nicht reservierte Liste initial mit der Konstanten Liste in SlotFinder besetzt wurde, musste also in der Konstante schon etwas falsch sein.

Zusehen in Abb. 2, obwohl die 24 schon im weit entfernten Slot 14 reserviert wurde (von Station 06) wird sie dennoch noch einmal in Slot 20 (von Station 04) reserviert.

Die große zeitliche Entfernung wird hier hervorgehoben, um klar zu machen, dass Fehler 2 kein Aspekt von Fehler 1 ist, sondern ein selbstständiger Fehler mit eigener Fehlerursache und -wirkung.

* 1. Wie entstand der Fehler?

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.

* 1. Wieso fiel der Fehler nicht eher auf?

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.

Zudem kommt, dass wir Sets nicht verwenden wollten, da sie in den ersten beiden Aufgaben verboten waren. Und wir nun eine Liste wie ein Set behandelten aber nie logisch sicherstellten, dass es auch wirklich wie ein Set arbeitet (vor allem die Eigenschaft, dass jedes Element nur einmal vorkommt).

* 1. Wie wurde er behoben?

Die zweite 24 wurde aus der Liste gelöscht.



*Abb. 5, die nun gefixte Liste ohne die zweite 24 in SlotFinder.erl*

1. Fehler 3
   1. Wo entstand der Fehler?

Auch hier passt es im Entwurf noch.

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

Jedoch auch in der Implementation Fehler:

Wie man in Abb. 3 sieht, führt diese Zeitverschiebung zu erheblichen Problemen. Zunächst reserviert Station 04 Slot 24 für den nächsten Frame (Zeile 38). Im nächsten Frame sendet die Station jedoch im nächsten Frame in Slot 25 (Zeile 58). Es war Glück, dass es hier noch nicht zu einer Kollision kam. Im selben Zug reserviert sie Slot 3, was eine korrekte (und die letzte mögliche) Wahl ist. Sende dann jedoch wiederum in Slot 4. Also war hier etwas falsch mit der Überprüfung der Sendezeit oder der internen Uhr.

In den Stations spezifischen Logs viel uns nicht auf, dass die Station an sich falsch sendet. Die Sendezeit wurde richtig errechnet. Und es wurde auch richtig geprüft ob die Sendezeit eingehalten wird oder nicht. Deswegen haben wir untersucht auf welcher Basis diese Sendezeit errechnet wird und stellten fest, dass die Frame beginne immer später wurden. So als bräuchten die Frames eben nicht exakt 1000 MS sondern länger. Und diese Zeit jenseits der 1000 MS akkumulierte sich und verschob den nächsten Framebeginn immer wieder ein klein wenig nach hinten. Den Fehler vermuteten wir deshalb in der core.erl Datei im frame\_loop. Und wie die Frames aufeinander aufbauen.

* 1. Wie entstand der Fehler?

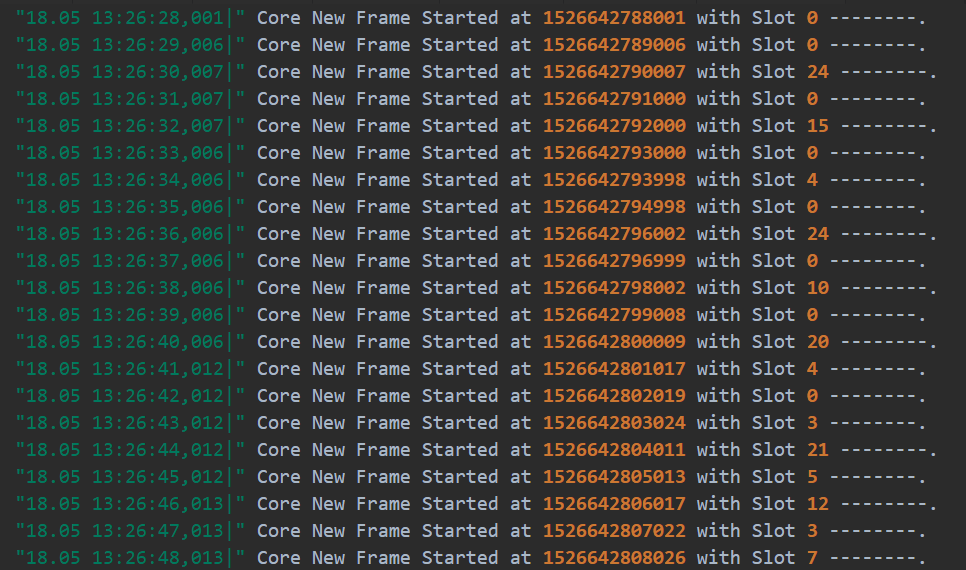
Auch hier war die Implementation der Fehler. Dieses Mal technischer Natur, da „timer“ in Erlang nie 100% exakt arbeitet und nur eine Mindestwartezeit statt die Gesamtwartezeit garantiert.

Unser Programm hat als Kernstück core.erl. Dieses Modul besitzt die frame\_loop Funktion. Diese repräsentiert einen Frame und sollte exakt 1000 MS lang sein. Ob ein Frame auf die Sekunde genau anfängt wurde nie kontrolliert, sondern es wurde fest damit gerechnet, dass die Framelänge genau eingehalten wird.

Immer wieder war sie jedoch länger als 1000 MS.

Weshalb der Frame für eine Station immer später begann (Weil die Loops sequentiell ausgeführt wurden, also aufeinander aufbauten). Weshalb die Station dachte sie sende in Slot 3, war global jedoch gesehen schon in Slot 4.

Diese Zeitverschiebung kann man ganz gut in Abb. 6 sehen. Man sieht wie die Framestartzeit nur selten auf die Sekunde genau los geht, sondern meist später. Man sieht wie gegen Ende die Startzeit immer später wird. Bis sie schon mehr als einen halben Slot versetzt anfängt. Am Offset kann dies nicht liegen. Da die Framestartzeit ja aus aktueller Systemuhrzeit (vsutil:getUTC()) und offset (addition) berechnet wird und auch dann immer auf die Sekunde genau sein soll.



*Abb. 6, aus Run 12, team 06-01.log*

* 1. Wieso fiel der Fehler nicht eher auf?

Der Fehler fiel bei genauem Blick auf die Logs auf, doch hatten wir erst spät eine Idee wie wir ihn lösen können. Und kümmerten uns zunächst um ebenso große Probleme für die wir bereits eine gute Spur hatten (Fehler 1 & 2).

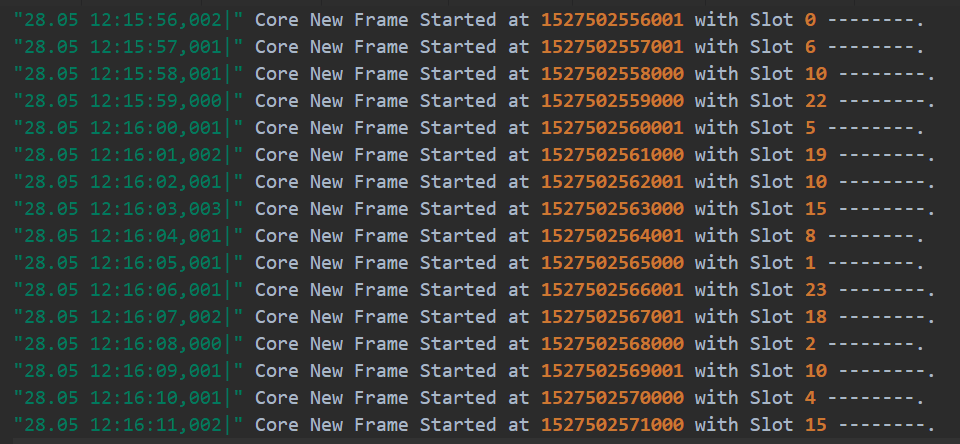
* 1. Wie wurde er behoben?

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‘). 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 wiederum eine bestimmte Nachricht an den Core schickt. Für den Core 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.

Dies kann man in Abbildung 7 gut beobachten. Dort kann man nun auch sehen, dass die Änderung direkt geholfen hat den Framestart wesentlich genauer zu machen.



*Abb. 7, die verbesserte Version*

1. Generelles Vorgehen beim Testen

Bisher haben wir Fehler durch das Testen im Labor behoben. Vieles war unteranderem durch unsere geschriebenen Tests abgedeckt. Dabei gingen wir folgendermaßen vor:

Zunächst haben wir unseren Code kompiliert und überprüft, ob der Compiler bereits Fehler warf. Waren hier keine Syntax-Fehler zu finden, wurde die Anwendung mit den Eunit Tests getestet.

Meist fingen wir an die Anwendung mit nur einer Station zu starten, um zu prüfen, ob es einen Laufzeit-Fehler gibt. Gab es diese nicht, wurden die Logs überprüft, ob die Daten in den Logs stimmen. War auch hier kein Fehler zu entdecken, haben wir die Anzahl der Stationen erhöht und die Anwendung erneut gestartet. Zusätzlich haben wir beim Testen mit mehreren Stationen, den Sniffer von Herrn Schulz parallel laufen lassen.

Hierbei wurden dann die Logs von den Stationen und von dem Sniffer von Herrn Schulz zur Überprüfung verwendet.

Nachdem Unstimmigkeiten entdeckt wurden, die durch Logik-Fehler auftraten, wurden diese dann versucht zu lokalisieren. Dabei haben wir gemeinsam Gebrainstormt wie und wann dieser bestimmte Fehler auftreten kann, und wo im Code dieser Fehler auftritt.

Gab es zu diesem Punkt noch keine Ideen, so wurde mit Hilfe der Sequenzdiagramme versucht den Fehler zu lokalisieren. Brachte auch dies keinen Lösungsansatz, so haben wir im Code Backtracking betrieben, bis wir den Fehler lokalisierten.

Hiernach wurde gemeinsam nach einer Lösung gesucht und gemeinsam die Lösung implementiert.

1. Quellen

Abbildungen 1, 2 und 3 entstanden aus Ihrer gesendeten Log Datei.

Abbildungen 4, 5 sind aus dem aktuellen Code entnommen.

Abbildungen 6, 7 entstanden aus den Stations Logs die in unserer Owncloud zu finden sind: <https://owncloud.informatik.haw-hamburg.de/index.php/f/26095377>