ESSAY IOT-SOFTWARE

Review van beroepsproduct op de ‘power of 10’

Abstract

In dit essay review ik de door mij geschreven code op basis van de regels uit het boek “the_power_of_10_rules_for_developing_safety_critic” en beoordeel ik in hoeverre mijn code overeenkomt met de regels uit dit boek. De opdracht is uitgevoerd in opdracht van Chris Meijs voor het vak IOT-Software van de HAN.

Sonny Selten (1654954)

Shg.selten@student.han.nl

Beoordeling en Verbetering van Code op Basis van de "Power of 10" Regels

De "Power of 10: Rules for Developing Safety-Critical Code" opgesteld door Gerard J. Holzmann vormt een kader voor het ontwikkelen van software die cruciaal is voor de veiligheid. Deze regels zijn bedoeld om de betrouwbaarheid en verifieerbaarheid van software aanzienlijk te verbeteren. Door deze regels toe te passen op een gegeven codebase, kunnen we niet alleen de veiligheid en prestaties van de software verbeteren, maar ook de onderhoudbaarheid en leesbaarhei. Hier volgt een beoordeling van het IOT-software beroepsproduct '**REST API**' op basis van de "Power of 10" regels.

Regel 1: Beperk alle code tot zeer eenvoudige controleflowconstructies

De eerste regel benadrukt het belang van eenvoudige controleflowconstructies. Het gebruik van constructies zoals `goto`, `setjmp`, `longjmp` en recursie wordt afgeraden vanwege de complexiteit die zij toevoegen aan de controleflow, wat de analyse bemoeilijkt. De gegeven code maakt geen gebruik van deze constructies, wat positief is. Maar daarbij moet ik ook zeggen dat ik deze keywords nooit eerder heb gebruikt. De controleflow is eenvoudig en goed te volgen. Er zijn geen recursieve functies en de controleflow wordt beheerd met reguliere conditionele statements en lussen.

Regel 2: Geef alle lussen een vaste bovengrens

Het doel van deze regel is om de mogelijkheid van oneindige of onvoorspelbare iteraties te elimineren door ervoor te zorgen dat alle lussen een duidelijke en vaste bovengrens hebben. In functies zoals `getBufferActual` en `setBufferSize` worden lussen gebruikt die itereren op basis van bufferomvang of -aantal, welke variabelen zijn. Dit betekent dat deze lussen potentieel onbegrensd kunnen zijn, wat een risico vormt voor de voorspelbaarheid en stabiliteit van de code. Echter aangezien dit wel volgens de requiremtens van de opdracht is en in mijn implementatie van de code hebben zowel de *while* als *for* loops allemaal een bovengrens die het aantal iteraties limiteert aan de variabele kan dit alleen mis gaan indien de gebruiker te hogen waardes aan de buffer geeft.

Regel 3: Gebruik geen dynamische geheugentoewijzing na initialisatie

Dynamische geheugentoewijzing kan onvoorspelbaar gedrag veroorzaken, vooral in omgevingen waar betrouwbaarheid cruciaal is. Deze regel beveelt aan om dynamische geheugentoewijzing te beperken tot de initialisatiefase van het programma. De code gebruikt `malloc` in de functies `initializeBuffer` en `setBufferSize`. Terwijl `initializeBuffer` typisch alleen tijdens de initialisatie wordt aangeroepen, kan `setBufferSize` ook dynamisch worden aangeroepen binnen de functie `handleRequest`, wat in principepe in strijd is met deze regel. Maar ook hier geld wel dat dit opgesteld is volgens de requirement uit de opdracht.

Regel 4: Geen enkele functie mag langer zijn dan één enkele bedrukte pagina

Korte en overzichtelijke functies verbeteren de leesbaarheid en onderhoudbaarheid van de code. Lange functies zijn moeilijker te begrijpen en te onderhouden. De functie `handleRequest` is vrij lang en overschrijdt mogelijk een enkele bedrukte pagina, wat de leesbaarheid en onderhoudbaarheid vermindert. Handle request zou eventueel opgesplitst kunnen worden naar kleinere functies. In het specifiek de tokenizer zou appart kunnen. Eigenlijk voldoet deze methode nu twee functies wat niet helemaal in lijn is met single responsibility.

Regel 5: De assertiedichtheid van de code moet gemiddeld minimaal twee asserties per functie zijn

Asserties helpen bij het vroegtijdig detecteren van fouten door aannames expliciet te maken en te controleren. Een hoge assertiedichtheid verbetert de betrouwbaarheid van de code. De code bevat geen asserties, wat betekent dat er geen mechanisme is om automatisch fouten en anomalieën te detecteren. De opgestelde code voldoet dus eigenlijk niet aan deze eis. Wel worden sommige functies zoals *getBufferActual*, *getRunningStandardDeviation* vroegtijdig gestopt als er niet genoeg data beschikbaar is.

Regel 6: Declareer alle gegevensobjecten op het kleinste mogelijke niveau van scope

Het beperken van de scope van variabelen voorkomt onbedoelde bijwerkingen en maakt de code gemakkelijker te begrijpen en te debuggen. Er worden verschillende globale variabelen gebruikt, zoals `sensorBuffer1`, `sensorBuffer2` en lopende statistieken. Echter is dit nodig om bepaalde data te kunnen berekenen. Ook lijden deze regel tot een filosofische discussie tussen mij en Chirs aangezien er aangeraden werd om in bijvoorbeeld *handleRequest* de *respone* return waarde boven in de functie aan te maken zodat dit niet elke keer in de return geconstrueerd hoeft te worden. Ondanks dat ik de intentie en meerwaarde van deze regel begrijp zal hier waarschijnlijk toch soms een persoonlijke afweging in spelen welk resultaat minder data gebruikt.

Regel 7: Elke aanroepende functie moet de terugkeerwaarde van niet-void functies controleren en parameters valideren

Het is belangrijk om de terugkeerwaarden van functies te controleren en de parameters te valideren om onverwachte gedrag en fouten te voorkomen. Functies zoals `initializeBuffer` en `malloc` controleren hun parameters of terugkeerwaarden niet consistent, wat kan leiden tot onvoorspelbaar gedrag. Maar in mijn persoonlijke mening zou dit helemaal niet hoeven als je een goede functie zou schrijven. Het kan leiden tot veel overhead en onnodige checks, wat de load op microcontrollers verder verhoogd. Ik denk persoonlijk dat een uitgebreide unit test meer garantie kan bieden in dit doel dan de beoogde regel uit het boek.

Regel 8: Beperk het gebruik van de preprocessor tot het opnemen van headerbestanden en eenvoudige macro's

Overmatig gebruik van de preprocessor kan de code ondoorzichtig maken en de leesbaarheid verminderen. Deze regel adviseert om de preprocessor alleen te gebruiken voor het opnemen van headerbestanden en eenvoudige macro's. De code gebruikt de preprocessor uitsluitend voor het opnemen van headerbestanden, wat in overeenstemming is met deze regel.

Regel 9: Het gebruik van pointers moet beperkt zijn

Pointers kunnen moeilijk te beheren en foutgevoelig zijn. Deze regel adviseert om het gebruik van pointers te beperken tot enkelvoudige dereferencing en om het gebruik van function pointers te vermijden. De code gebruikt pointers, maar houdt zich aan enkelvoudige dereferencing. Er worden geen function pointers gebruikt. Dit betekend dat de code in lijn is met deze regel.

Aanbeveling: Vermijd complexe pointermanipulaties en zorg ervoor dat alle pointergebruik veilig en noodzakelijk is. Dit helpt bij het behouden van de eenvoud en veiligheid van de code.

Regel 10: Compileer met alle warnings ingeschakeld en behandel warnings als fouten

warnings van de compiler kunnen wijzen op potentiële problemen in de code. Het inschakelen van alle warnings en het behandelen ervan als fouten zorgt ervoor dat deze problemen vroegtijdig worden aangepakt. Hoewel deze regel betrekking heeft op het bouwproces in plaats van de code zelf, is het belangrijk om te zorgen dat de code wordt gecompileerd met alle waarschuwingen ingeschakeld en dat de codebasis vrij is van waarschuwingen. In het beroepsproduct zitten verschillende warnings echter zijn sommige niet helemaal correct. Zo wordt er een warning gegeven over een exit in de functie *handleRequest* waarin geen return waarde zit echter kan ik dit niet terug vinden in de functie. Ook worden er warnings gegeven voor verschillende unused variabele en functies echter zijn deze ook niet altijd helemaal correct aangezien ze in Arduino worden aangeroepen en daarom niet geregistreerd worden of nodig zijn voor eventuele uitbreiding. Ik heb geprobeerd in de code de warning te supressed echter kreeg ik hierna nog steeds de meldingen. Ondanks dat er warning in het beroepsproduct zitten denk ik toch dat deze over het algemeen acceptabel en te overzien zijn.

Conclusie

Het toepassen van de "Power of 10" regels op de het beroepsproduct levert in een reeks specifieke aanbevelingen die de veiligheid, betrouwbaarheid en onderhoudbaarheid van de code kunnen verbeteren. Door asserties toe te voegen, functies op te splitsen, dynamische toewijzing na initialisatie te vermijden, parameters en terugkeerwaarden te valideren, en de scope van variabelen te beperken, kan de code voldoen aan de strikte eisen die nodig zijn voor het ontwikkelen van veiligheid-kritische software. Het naleven van deze regels, hoewel strikt, zal de robuustheid en voorspelbaarheid van de softwareverbeteren, wat essentieel is in omgevingen waar betrouwbaarheid cruciaal is. Echter blijft het persoonlijke oordeel ook waardevol en zullen we niet regels moeten toepassen omdat het een regel is, maar het achterliggende doel van de regel achterhalen en vervolgens hier op sturen. Ik ben van mening dat deze regels meerwaarde kunnen bieden in ontwerpen van code op microcontroller en zal deze zeker meenemen in toekomstige vergelijkbare projecten