Opdrachten week 3 les 1 – Memory tests en test coverage

In les 2 van week 2 heb je testcode geschreven voor het UDT buffer en voor de temperatuurlogger waarin dit buffer toegepast wordt. Het bleek lastig te zijn om goed te testen of de buffer op correcte wijze gebruik maakt van dynamische geheugenallocatie. Hoe kun je zeker weten dat de code van de buffer geen geheugenlek bevat of vrijgegeven geheugen toch nog gebruikt[1]? Dit soort fouten kan met behulp van speciale tools eenvoudig opgespoord worden.

Ook heb je jezelf in les 2 van week 2 waarschijnlijk regelmatig afgevraagd of je al genoeg testcode had geschreven. Je kan er namelijk altijd nog wel een test bij bedenken.

Je leert deze les hoe je:

- fouten bij het gebruik van dynamische geheugenallocatie zoals geheugenlekken kunt opsporen met speciale tools zoals valgrind;
- weet dat je genoeg testcode hebt geschreven door gebruik te maken van test coverage metingen.

De tool die in de praktijk vaak gebruikt wordt om fouten in het geheugengebruik van een programma te vinden, is valgrind[2]. Dit is een zeer uitgebreid test framework waarmee allerlei dynamische tests[3] uitgevoerd kunnen worden. Het enige nadeel van deze tool is, dat het alleen op Linux of onder WSL te gebruiken is.

3.1.1 Test de implementatie van het UDT buffer, die je in les 2 van week 1 hebt ontwikkeld en in les 2 van week 2 hebt uitgebreid, zo goed mogelijk, op memory errors.

Een eenvoudige manier om te kijken of je genoeg testcode hebt geschreven is door te meten welke code wel en welke code niet uitgevoerd wordt door de testcode. Dit wordt test coverage of ook wel code coverage genoemd. Een stukje code dat tijdens het testen niet uitgevoerd wordt, is uiteraard nog niet goed genoeg getest en er moet een test bedacht worden die

[1] Als geheugen dat net met behulp van `free` is vrijgegeven, toch nog gebruikt wordt in een programma, dan leidt dit niet altijd tot een fout. Het levert alleen een fout op als de betreffende geheugenplaats inmiddels voor iets anders in gebruik is genomen. Dit soort fouten is zeer moeilijk op te sporen omdat ze soms wel maar soms ook niet optreden bij het uitvoeren van het programma.

[2] Zie: https://valgrind.org/.

[3] Dynamische tests zijn tests die het programma testen door het uit te voeren, dit in tegenstelling tot statische tests die het programma testen door de code van het programma te analyseren (zonder deze code uit te voeren).

dit stukje code uitvoert en test. Let erop dat het natuurlijk niet zo is, dat alle code die wel uitgevoerd wordt tijdens het uitvoeren van de tests, wel goed getest wordt.

Er zijn verschillende tools beschikbaar waarmee de test coverage bepaald kan worden. Wij maken gebruik van gcov[4]. Deze tool is onderdeel van gcc (de GNU Compiler Collection). De C-compiler die wij gebruiken is ook een onderdeel van gcc dus is het voor de hand liggend om gcov te gebruiken.

3.1.2 Installeer de volgende plug-ins in Visual Studio Code:

- Live Preview[5];
- Gcov Viewer[6].

Bepaal de test coverage voor de testcode die je in les 2 van week 2 hebt geschreven om het UDT buffer te testen.

3.1.3 Voeg zoveel mogelijk tests toe om delen van de code, die nog niet getest werden, alsnog te testen. Is het zinvol om een code coverage van 100 % na te streven? Waarom wel/niet.

[4] Zie: https://gcc.gnu.org/onlinedocs/gcc/Gcov.html.

[5] Zie: https://marketplace.visualstudio.com/items?itemName=ms-vscode.live-server.

[6] Zie: https://marketplace.visualstudio.com/items?itemName=JacquesLucke.gcov-viewer.