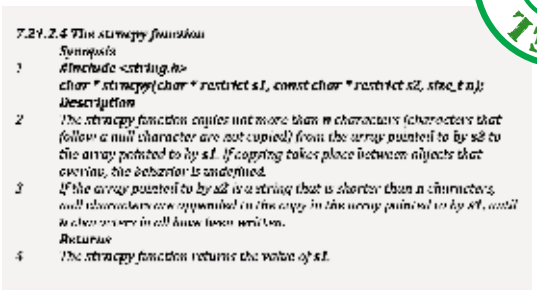
Så testar vi C-bibliotek för kritiska system

EXPERTARTIKEL

Av Marcel Beemster, Solid Sands

Marcel Beemster är kompilatorexpert med doktorshatt i datavetenskap från Universitetet i Amsterdam. Han har verkat som senior mjukvaruingenjör i ett antal kompilatorprojekt. I början av 2013 ändrade han fokus till stöd, underhåll och utveckling av SuperTest-verktyget som nämns i artikeln. Han var med och grundade Solid Sands 2014.



7.24.2.4 The strncpy function

Synopsis

1 *#include <string.h>*
*char *strncpy(char * restrict s1, const char * restrict s2, size_t n);*

Description

2 *The strncpy function copies not more than n characters (characters that follow a null character are not copied) from the array pointed to by s2 to the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined.*

3 *If the array pointed to by s2 is a string that is shorter than n characters, null characters are appended to the copy in the array pointed to by s1, until n characters in all have been written.*

Returns

4 *The strncpy function returns the value of s1.*

Bild 1. SuperGuard härleder strikta krav ur C-specifikationen, som är ganska informell.

Vid verifiering för funktionssäkerhet kan du inte utgå från att komponenter, kompilatorer och bibliotek är felfria. Allt måste kvalificeras, och i din egen utvecklingsmiljö, och för samma användningsfall som applikationen.

Vid kvalificering för en säkerhetsstandard är det utvecklaren och ingen annan som ska påvisa att mjukvara (och metoder, processer och verktyg) uppfyller standardens krav. Problemet är att delar av verktygskedjan ligger utanför hens kontroll. En betydande del av koden är troligen kompilerad för andra tillämpningsområden, med andra kompilatordirektiv och i en annan utvecklingsmiljö än den utvecklaren använder.

Detta gäller exempelvis funktioner i C-standardbiblioteket Libc, som ofta levereras som binärfiler i SDK:n (software development kit, utvecklingsmiljön).

Kvalificering av kodbibliotek är kritisk. Om en komponent är defekt äventyras hela tillämpningens funktionssäkerhet.

Det finns en utbredd tro att binärbibliotek är oberoende av tillämpning. Det stämmer inte i praktiken. Biblioteket kan vara förkvalificerat av SDK-leverantören med samma kompilator och man kan ändå nästan garantera att det finns skillnader av nämnda slag. Det skapar problem när du ska argumentera för att säkerhetsstandarden uppfylls.

Därför har Solid Sands – sedan tidigare en av de världsledande inom kompilatorvalidering – introducerat ett nytt kvalifikationsverktyg för just bibliotek: SuperGuard C Library Safety Qualification Suite. Det är ett kravbaserat testpaket för C:s standardbibliotek och det spårar enskilda testresultat tillbaka till krav som vi härlett ur ISO-specifikationen.

SuperGuard kan användas för kvalificering av säkerhetskritisk C för såväl omodifierade tredjepartsbibliotek, som för egenutvecklade implementeringar.

Den som önskar kvalificera sig enligt ISO 26262 (funktionssäkerhet i vägfordon) kan verifiera på två sätt som detaljeras i standardens del 8 respektive del 6. Här ska vi berätta om hur SuperGuard arbetar med del 8, men SuperGuard täcker även del 6.

Klausul 12 i del 8 handlar om”kvalificering av mjukvarukomponenter”. Kvalificeringen ska argumentera för deras ”lämplighet för återanvändning”.

Specifikt nämns bibliotek från tredje part, vilket därmed uppenbarligen omfattar standardbiblioteken i kommersiella SDK:er. Klausulen gäller även återanvänd intern mjukvara och öppen källkod.

Standarder för funktionssäkerhet har det gemensamma kravet att det ska verifieras att bibliotekets implementering överensstämmer med dess specifikation.

Därmed är en förutsättning för kvalificering att kodkomponentens krav har angetts. Bevis för att kraven uppfylls ska därefter främst baseras på kravbaserade tester vilket kan uppnås genom ”tillämpning av en dedikerad kvalificeringstestsvit”.

Bevisen ska omfatta både normala driftsförhållanden och beteende vid felsituationer. Inga kända fel som kan leda till överträdelse av säkerhetskrav ska uppträda.

Lyckligtvis finns biblioteksspecifikationer offentligt tillgängliga för både C och C++. De kan användas som utgångspunkt för kravbaserad testning.

Faktum är att möjligheten att testa mot språkens och bibliotekens specifikationer är en av anledningarna till att programmeringsspråken C och C++ har så bred användning. Detta trots att språkspecifikationen inte är skriven som en kravlista.

SuperGuard-testsviten baserar strikt sina tester av bibliotek på krav hämtade från ISO C-språkdefinitionen. Funktioner testas både innanför och utanför deras gränser för att verifiera att även felhanteringen följer specifikationen.

Ett robust bevis enligt ASIL-D (Automotive Safety Integrity Level D – den högsta integritetsnivån för fordon) kräver strukturell kodtäckningsanalys. SuperGuard tillhandahåller även detta, plus en hög MC/DC-täckning (Modified Condition/Decision Coverage).

SuperGuard inkluderar också analys och testning av ekvivalensklasser och gränsvärden, samt felgissning baserad på bästa tillgängliga kunskap och erfarenhet av bibliotekets beteende.

Hur SuperGuard-tester utvecklas

Problemet med att utveckla kravbaserad testning för C och C++ är att specifikationerna visserligen detaljerat beskriver funktionernas beteende – men utan att definiera explicita krav. Kraven måste härledas.

För att kunna stödja de olika standarderna måste SuperGuard gå mycket längre än SuperTest när det gäller rapportering, dokumentationskrav och enskilda tester och testresultat.

TESTERNA I SUPERGUARDS testsvit är utformade enligt följande principer:

- **De är beteendebaserade.** Testerna jämför förväntade modellresultat med faktiska resultat.
- **De körs i en exekveringsmiljö.** Hela verktygskedjan, inklusive målprocessorn, är involverad i varje test. Därmed kan SuperGuard användas för hardware-in-the-loop-verifiering.
- **Testerna för den fristående delen av biblioteket** (som vanligtvis används i bare-metal-system) kräver minimala resurser. De flesta SuperGuard-tester kan köras på system med mindre än 4 kbyte, vilket betyder att SuperGuard kan användas på mycket små inbyggda system.

SUPERGUARD BRYTER DETALJERAT ner hur varje C-specifikation omvandlas till krav och kopplar för spårning varje individuellt testresultat till specifikation, testkrav och biblioteksfunktion.

BILD: MIDJOURNEY

Även förkvalificerade C-bibliotek kan skapa buggar och måste testas.

Varje bibliotekstest i SuperGuard-sviten utvecklas metodiskt. Vi tar biblioteksfunktionen strncpy() som exempel – se bild 1!

Det lustiga med specifikationen i bild 1 är att stycke 2 inte specificerar någon lägre gräns för hur många tecken strncpy() ska kopiera. Den säger ”kopierar högst n tecken” men kräver aldrig att några tecken alls ska kopieras. I stycke 3 nämns att s1 fylls ut med nulltecken. Tolkat bokstavligt skulle det vara en korrekt implementering av strncopy() att inte göra något mer än att fylla s1 med nulltecken.

Men det är förstås inte vad funktionen förväntas göra. Den ska kopiera så många tecken som möjligt från s2 till s1 tills antingen strängen s2 eller n är uttömd. Det finns ingen tvekan om detta, och ingen tycks ha klagat sedan ANSI C89 eftersom formuleringen finns kvar i C18.

FÖR ATT DEFINIERA KRAV måste vi vara lite mer exakta.

Det första steget är att extrahera krav (REQs, requirements) från beskrivningen som tar hänsyn till vad funktionen faktiskt är tänkt att göra. Dessa är:

REQ-copystring: Om strängen s2 har en längd l2 (definierad av strlen()) som är kortare än n, ska l2 tecken kopieras, i ordning, från s2 till s1.

REQ-copyn: Om s2 inte är kortare än n, ska de första n tecknen kopieras, i ordning, från s2 till s1.

REQ-shorter: Om s2 är kortare än n, ska nulltecken läggas till efter de kopierade tecknen i s1 tills sammanlagt n tecken har skrivits.

REQ-nomore: strncpy() ska inte skriva något i s1 bortom de första n tecknen.

REQ-nochange: strncpy() ska inte modifiera strängen s2.

REQ-return: strncpy() ska returnera värdet av s1.

REQ-NOCHANGE FÖLJER visserligen i princip av att s2 deklarerats konstant. Men deklarationen i sig garanterar inte att en implementation av strncpy() inte skriver till s2.

För varje krav utvecklas en testspecifikation som definierar hur ett test verifierar att kravet är sant. En enskild testspecifikation ger vanligen ett flertal testfall som täcker funktionens in- och utdatadomäner. Testfallen implementeras av testet. Testspecifikationen kopplar kravet till testerna.

TILL EXEMPEL är testspecifikationerna för REQ-copystring och REQ-nomore som följer:

- **REQ-copystring:** Anropa strncpy()-funktionen med olika värden på n (inklusive n==0) som är lika med och större än längden på ursprungssträngen. Verifiera att ursprungssträngen kopieras upp till det avslutande nulltecknet.
- **REQ-nomore:** För alla testfall, verifiera att tecknet med index n i målarrayen s1 inte ändras. Om det passar med testet, verifiera också att inga tecken bortom n ändras.

I DETTA FALL läggs REQ-nomore i samma testfil som övriga testfall för strncpy(). Eftersom kravet måste uppfyllas ovillkorligen för varje anrop till strncpy() i vilket fall som helst, implementeras testet genom att helt enkelt lägga till en extra kontroll på varje testfall för övriga krav, istället för ge den egna test.

Headerfiler och funktionsliknande makron

Det finns ytterligare ett sätt på vilken språket C ställer till det för testare. Alla standardfunktioner i C levereras inte i form av förkompilerade binärer. Många av dem beror av headerfiler.

Där definieras typer, globala variabler, makron, och annat. De är lika mycket en del av biblioteket som de förkompilerade biblioteksfunktionerna. Vissa funktioner finns både som riktiga funktioner och som makron. Vanligen används makrovarianten eftersom det är snabbast och effektivast. SuperGuard testar båda.

Funktionsmakrona kompileras tillsammans med källkoden. Det är därför viktigt att de, och resten av headerfilen, verifieras för det aktuella användningsfallet. I C++ har header-makron en ännu viktigare roll i definitionen av generiskt typade mallar. ■