DV1620: Projektspecifikation 1

Antivirus i C++

Martin Boldt & Anton Borg
Department of Computer Science and Engineering
Blekinge Institute of Technology

2022-01-20

Introduktion

Projektuppgiften i kursen DV1620 går ut på att skriva ett eget förenklat antivirusprogram i C++ (alltså *inte* Python). Det finns en befintlig virusdatabas som ert program skall använda sig av. Denna databas innehåller beskrivningar av filer som är klassade som virus. Er uppgift är att skriva ett program som traverserar (går igenom) en katalog i filsystemet samt alla dess underkataloger, och dess underkataloger osv. För varje fil programmet hittar bland dessa kataloger skall det kontrollera ifall den matchar någon av beskrivningarna i virusdatabasen. Varje gång programmet påträffar en fil som finns med i virusdatabasen så skall detta rapporteras till en loggfil.
För att göra uppgiften mer överskådlig så bryter vi ner uppgiften i mindre och mer lätthanterliga delar (eng. "divide and conquer"). Genom att lösa respektive del för sig så kommer arbetet förhoppningsvis att gå lättare.
Projektet kan brytas ner i följande tre delar: filtraversering, virusdatabasen och filidentifiering. Vidare ska ni även lämna in en skriftlig **kravspecifikation** tillsammans med ert program, där ni går igenom den kravställning som ni arbetat utefter.

Gruppindelning

Samtliga delar i denna specifikation ska uppfyllas. Vidare ska uppgiften lösas i grupper om två (2) studenter. Studenterna ansvarar själva för att samordna gruppindelningen. Observera att *båda* i gruppen måste kunna förklara *all* kod som gruppen skrivit. Om någon i gruppen inte kan utföra detta så kommer denna person att självständigt få genomföra en omredovisning.

Systemspecifikation och examinering

Examinationen kommer att utföras under Ubuntu 18.04.6[1] (tips: installera en Developer-miljö). Under Ubuntu skall en fungerande *Makefile* bifogas. Vid examineringstillfället så kommer ert antivirusprogram att testas genom att köra det på en testkatalog där ett förutbestämt antal filer skall identifieras som virus. Det går bra att exekvera Ubuntu i en virtuell miljö i ert vanliga operativsystem, se t.ex. VirtualBox[2].

[1] URL: https://www.ubuntu.com/desktop/developers
[2] URL: https://www.virtualbox.org

Del 1 - Filtraversering

Ert program skall ges ett startbibliotek och skall sedan gå igenom alla filer i detta samt alla dess underkataloger, dess underkataloger osv. Denna typ av fillistningar brukar lösas med *rekursiva* funktionsanrop, alltså en funktion som anropar sig själv för varje underkatalog. Namnet på startbiblioteket skickar ni in till programmet som ett argument i kommandoprompten, t.ex:

```
./dv1620program MyStartDir/
```

Notera att pathen till startbiblioteket ska tillåta relativa pather också, t.ex. att gå upp två steg i kataloghierarkin. På så vis ska följande kommando i Ubuntu fungera givet att testkatalogen faktiskt finns på den pathen:

```
./dv1620program ../../MyStartDir/
```

Observera även att ni inte får använda externa program för att traversera filsystemet eftersom ni skall skapa en egen lösning i C++. Om ni stöter på problem med att det inte går att öppna filer så kan det bero på att ni har för många bibliotek öppna samtidigt. Kom ihåg skillnaden mellan *DepthFirst*[3] och *BreadthFirst*[4] vid rekursiv traversering genom en katalogstruktur. Unixtips, se `opendir()` och `readdir()`.

Del 2 - Virusdatabasen

Det är i virusdatabasen som beskrivningarna av virus finns lagrade. I vårt fall är det egentligen ingen databas utan en helt vanlig textfil, med namn `signatures.db`. Varje rad i textfilen innehåller en virusdefinition. Ni kan alltså läsa tecken för tecken på varje rad tills ni stöter på ett radslut `'\n'` eller så använder ni er av en funktion för att läsa filer rad för rad. Varje virusdefinition innehåller två värde som är åtskilda av ett '='. Det första värdet är en textsträng som innehåller *namnet* på viruset. Denna textsträng får inte överstiga 32 tecken, inkl. NULL-terminering. Efter namnet följer ett '=' och därefter kommer en beskrivning av innehållet i virusfilen, se exempel nedan.

```
TestVirus.D=255044462d312e340a332030206f62
```

Det andra värdet, alltså virusbeskrivningen, har ingen maxlängdsbegränsning. Vidare är beskrivningen lagrad i hexadecimal notation. Detta betyder att varje byte i virusfilen (*en* byte) beskrivs med *två* tecken i virusdefinitionen. Se exempel för `TestVirus.D` nedan.

När programmet startar kan det förutsätta att signaturdatabasen finns i samma katalog som programmet för tillfället har sparats i. Notera att det *inte* är tillåtet att i programmets källkod hårdkoda in pathen till signaturdatabsen.

Byte nr:	0	1	2	3	4	5	5	7
Hexvärde:	0x25	0x50	0x44	0x46	0x2d	0x31	0x2e	0x34
Decimalt:	37	80	68	70	45	49	46	52
Ascii:	'%'	'P'	'D'	'F'	'-'	'1'	'.'	'4'

Virusbeskrivningen börjar alltid från byte 0 (noll) i de filer som skannas (alltså de potentiella virusfilerna). Enligt tabellen ovan ser vi att den första byten i virusbeskrivningen är en textsträng som innehåller `"%PDF-1.4"`. Vi kan även sluta oss till att antalet byte som finns beskrivet i virusbeskrivningen är längden på beskrivningen delat med två. Så om vi har en virusbeskrivning som är 50 tecken lång så beskriver den de 50/2=25 första byten i virusfilen.

Det format på AV-databas som vi använder här är detsamma som opensource programmet ClamAV tidigare har använt.

[3]Se följande länk: https://en.wikipedia.org/wiki/Depth-first_search
[4]Se följande länk: https://en.wikipedia.org/wiki/Breadth-first_search

Del 3 - Filidentifiering

För varje fil ni stöter på vid filtraverseringen skall ni jämföra den mot virusdefinitionerna i virusdatabasen. Observera att ni alltid börjar med att jämföra första byten i filen mot första byten i virusbeskrivningen. Om filen stämmer överens med en av virusdefinitionerna så skall ni logga detta i en logfil med namn `dv1620.log`. I logfilen skall det finnas information om vilken fil som är infekterad, sökvägen till filen samt namnet på det virus som antas ha infekterat filen.

1 Felhantering

Observera att ni inte kan vara säkra på att virusdatabasen verkligen finns i samma katalog som programmet när det startar. Vidare kan ni exempelvis inte hller vara säkra på att virusdatabasen *inte* innehåller syntaxfel vid examineringen. Ni skall därför inkludera utförlig felhantering i ert program.

2 Kravspecifikation

Ni ska dokumentera de krav ni använt då ni utvecklat ert AV-program. Dokumentera kraven i en rapport där ni kort förklarar hur ni jobbat med kraven, samt tar med en tabell över era krav. Tabellen ska inkludera följande kolumner:

1. ID-nummer
2. Namn
3. Kort beskrivning
4. Hur ni testat kravet
5. Beroende till andra krav (list isåfall de kravens ID-nummer)
6. Om kravet *Ska* eller *Bör* implementeras

Nedan visas ett exempel på en sådan tabell inklusive ett krav.

ID	Namn	Beskrivning	Beroende	Test	Ska
E.1	Utskrift	Programmet ska dokumentera resultatet.		Fungerar	JA

Inlämning

Deadline för uppgiften hittar ni på Canvas. Innan ni skickar in er inlämningsuppgift skall ni kontrollera att:

- ni uppfyller samtliga krav i denna specifikation,
- ert projekt både kompilerar och exekverar korrekt under en vanlig Ubuntu 18.04.6 Developer installation,
- ert program exekverar när man står i samma katalog som programmet och skriver `./mitt-program-namn katalog-path/`
- ni har bifogat er kravspecifikation som en **PDF-fil**,
- ni har skickat in er *Makefile*, och

- ni har angett namn samt mailadress **i samtliga filer** ni lämnar in, alltså även era kodfiler,
- ni har packat alla era inlämningsfiler som ett tar.gz eller zip-arkiv.

Lycka till!