Linux som utvecklingsmiljö

Tema 6
Bibliotek

HT-13

Petter Lerenius, 790703-0295

1. En beskrivning av det egna biblioteket

Mitt bibliotek räknar ut den resulterande resistansen av ett fritt antal parallella eller seriellt kopplade resistorer med valfritt värde. Den enda exponerade funktionen är:

```
float calc_resistance(int count, char conn, float *array);
```

En användare skickar in antalet resistorer, hur de är kopplade (seriellt eller parallellt) och slutligen en pekare till det första resistansvärdet i en array med lika många värden som första parametern angivit. Funktionen returnerar sedan den resulterande resistansen som en float.

Jag har använt mig av följande två formler.

Seriellt:

```
R=R1+R2+R3+...
```

Parallellt:

```
1/R=1/R1+1/R2+1/R3+...
```

För att testa mitt bibliotek skapade jag ett testprogram som tar in h-filen och sedan länkar in mitt bibliotek. Det skickar in olika värden och verifierar resultatet mot det förväntade. Jag valde att prova de två algoritmerna med ett par olika värden var, samt specialfallen med ett resistorvärde på noll samt en NULL pekare till arrayen. Bibiliteket skapades mha komandot:

```
gcc -g -c -fPIC src/resistance.c
gcc -g -shared -fPIC -o libresistance.so resistance.o
```

Och testprogrammet med:

```
gcc -g -Isrc -o testRes test/test.c -L. -lresistance -Wl,-rpath,.
```

För att göra det enklare skapades en makefil som kunde bygga biblioteket samt bygga testkoden. Kommandot `make test` bygger det som behövs samt kör testet, vilket var väldigt smidigt då man vill kunna testa programmet ofta och på ett enkelt sätt.

Växlar som använts i makefilen till att bygga:

-g	Bygg med debuginformation så att det är möjligt att felsöka med gdb om det inte fungerar
-c	Kompilera koden men länka inte
-fPIC	genererar “position independant code” för att användas i dynamiska bibliotek
-o	Vad utfilen ska heta
-Isrc	Lägg till katalogen src för att leta efter include-filer
-L	Titta efter bibliteksfiler i den här katalogen
-l	Länka in det här biblioteket, förutsätter att biblioteket börjar med lib och slutar med .so
-Wl	används för att skicka parametrar till länkaren, i det här fallet -rpath . , vilket säger till länkaren var den ska börja leta efter .so-filer
-shared	genererar shared objects-fil för dynamiska bibliotek

Del 2 Kompilering av electrotest med makefile

Vi som jobbat tillsammans med denna uppgift är Björn Rikte, Petter Lerenius och Klaus Virtanen.

Huvudprogrammet använder sig av de olika bibliotekens h-fil för att få in funktionsdefinitionerna, sedan länkas biblioteken in vid bygget av huvudprogrmmet. Växlarna som använts är de samma som redovisats i föregående avsnitt. Makefilen återfinns i det bifogade katalogträdets rot. Vi har följande byggmöjligheter:

make lib :	De tre bibliotekskomponenterna kompileras och länkas, var och en med samma kommandon och växlar som beskrivs i del 1. Enskilda bibliotekskomponenterna kan också skapas med kommandona make libcomponent, make libresistance eller make libpower om så skulle önskas.
make all:	Huvudprogrammet electrotest kompileras och länkas på samma sätt som beskrivs i del 1 men med alla tre bibliotekskomponenterna (-l) och sökvägar till alla includefiler (-I) på kommandoraden.
make install:	Anropas under sudo vid behov. Före installation till /usr/bin och /usr/lib kompilerar vi om huvudprogrammet utan länkaroptionen -Wl,rpath,./lib för att utesluta bibliotekskräff i lokal katalog vid körning av installerat program (och /usr/lib finns ju redan på default sökväg).
make uninstall:	Tar bort filer installerade med make install. Anropas under sudo vid behov
make clean:	Tar bort lokala objektsfiler, libfiler och programfiler från arbetskatalogen.
make test:	Testar programmet med data från filen testdata.txt

Givetvis är alla sökvägar till källkodsfiler anpassade efter katalogstrukturen.

Del 3 Hur det gick och vad bör man tänka på i liknande fall

Vi fick ganska snabbt kontakt med varandra via mail och bestämde oss då för att använda oss av svn för koddelning och mail för kommunikation. Eftersom uppgifterna var väl avgränsade och interfacen var tydligt definierade krävdes det inte så mycket koordinering, mer än att bestämma vem som gjorde vad.

Det tog lite tid innan alla var igång, men sen flöt sammarbetet på riktigt bra och vi fick relativt fort ihop ett program som fungerade. Efter lite testande av varandras kod, genom huvudprogrammet, kunde vi ge feedback på vilka buggar och tveksamheter som hittats.

Generellt sätt tycker jag att det gick oväntat lätt att samarbeta kring uppgiften, men å andra sidan behövde vi inte komma fram till så många gemensamma beslut som krävde diskussioner, för då hade vi nog varit tvugna att ha ett telefonmöte eller liknande för att inte det skulle bli alltför många mail fram och tillbaka. Jag tyckte att den var en rolig uppgift!

För att ett större projekt ska fungera bör man nog ha någon som är ytterst ansvarig och som har rätten att ta beslut i ärenden som är svåra att enas kring. Om det är många som bidrar med kod kan det vara bra med något sätt att få sin ändring granskad och godkänd innan den accepteras in i huvudprogrammet. Detta kan lösas med t.ex. leverans-brancher, eller så kan man använda ett kodgranskingsverktyg, typ Gerrit.

Det är också viktigt att man är överrens om interfacen, alltså funktionerna i biblioteken som ska användas, så att dessa är spikade tidigt och sedan inte ändras utan att alla är medvetna om att ändringen kommer ske, eftersom det är svårt för en utvecklare att veta var det kommer att påverka i biblioteksanvändarnas kod.

Det är nödvändigt att ha ett versionshanteringsverktyg när man är flera som sammarbetar som kan hålla koll på vad som ändras och av vem. Å andra sidan klarade sig Linux med patchar via mail väldigt länge...

Summa sumarum var det en kul uppgift!