Blekinge Tekniska Högskola

DV1492, DV1556 – (Realtids- och) Operativsystem

Projekt - Simulering av filsystem

Författare: Carina Nilsson, Cuong Phung, Erik Bergenholtz
Datum: september, 2017

Innehåll

1 Syfte

Syfte med laborationen är att du ska utveckla en grundläggande teknisk förståelse för hur ett filsystem är organiserat och hur de nödvändigaste funktionerna kan implementeras.

2 Förutsättningar

Det finns ett antal hjälpfiler att ladda ner från kurshemsidan på It's Learning.

Det är ett krav att filsystemet ska byggas på en simulerad disk i form av en 2-dimensionell byte-array 250×512 byte stor (250 block à 512 byte vardera). Den simulerade "disken" får bara läsas och skrivas blockvis, dvs. man läser eller skriver alltid ett 512 byte-block i taget.

I övrigt är du fri att definiera dina datastrukturer hur du vill.

3 Redovisning

Uppgiften ska utföras i grupper om två. Annan gruppstorlek ska beviljas av handledaren. Grupper med fler än tre deltagare godtas inte.

Redovisningen sker gruppvis. Uppgiften ska redovisas genom att ladda upp källkod och dokumentation i .pdf-format på It's Learning i form av en arkivfil (exempelvis .zip eller .tar).

Projektuppgiften kommer att betygsättas A-F (se bedömningskriterierna i avsnitt 6 i dokumentet). Vid inlämningen ska det tydligt anges vilken betygsgrad projektet ska testas för i kommentarsfältet inlämningen på lärplattformen.

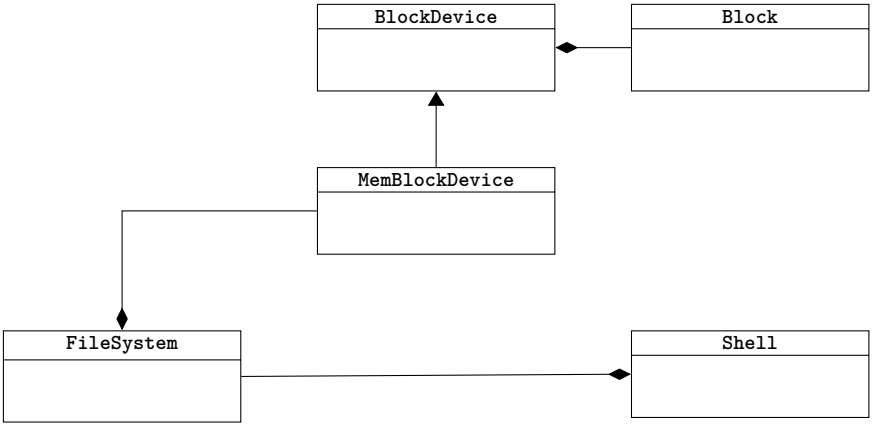
Observera att det är inte tillåtet att kopiera kod som någon annan än gruppmedlemmarna har konstruerat.

4 Hjälpfiler

Till din hjälp finns det redan ett halvklart filsystem i form av några klasser (Se figur 1) skrivna i C++. Det finns en färdiggjord `Makefile` för att underlätta för dig som väljer att kompilera i kommandotolken under Linux. Instruktioner på hur du kan kompilera med make-filen finns nedan.

Det är också tillåtet att ignorera hjälpfilerna och implementera din helt egna lösning.

4.1 Klassdiagram



Figur 1: Klassdiagram för hjälpfilerna.

Block är en klass som har en dynamisk `char`-array, vars ”default-konstruktor” allokerar minne för 512 tecken (byte).

BlockDevice är en rent virtuell klass, där flera funktioner inte är implementerade, dessa är implementerade i klassen **MemBlockDevice**. **BlockDevice** har ett dynamisk allokerat fält innehållande **Block**-objekt.

MemBlockDevice ärver från **BlockDevice** och där finns flera funktioner. Standardkonstruktorn kommer att generera 250 block med en blockstorlek på 512 byte. Nedan följer en kort beskrivning på de nödvändigaste funktionerna:

- `int writeBlock(int blockNr, ...)` har som inparameter `std::string`, `std::vector<char>` eller `char[]`. Funktionen returnerar -2 när inparametern och blockstorleken inte är av samma längd, -1 vid anrop av ett felaktigt blocknummer och 1 vid lyckad skrivning.
  Notera! om man använder `char[]` så görs ingen kontroll på fältets längd.
- `Block readBlock(int blockNr)` läser ett block. Funktionen kastar ett `std::out_of_range` exception om man anger ett blocknummer som inte existerar.

FileSystem har ett **MemBlockDevice**-objekt. Meningen är att du ska implementera API-funktionerna som filsystemet tillhandahåller: `createFile(...)`, `createFolder(...)`, `read(...)`, `write(...)`osv. Observera att API-funktionena ska vara elementära och sammanfaller inte automatiskt med funktionerna i `Shell`. Exempelvis bör `Shell`-funktionen `cp` vara sammansatt av en läsning av fil följd av en skrivning till fil.

Shell är ingen klass utan en samling funktioner, däribland `main()`. Här finns redan en kommandotolk implementerad. Här ska funktionerna för `Shell`-kommandona implementeras med hjälp av filsystemets API.

I övrigt är du fri att ta bort, lägga till och modifiera kod enligt ditt tycke. Vissa funktioner kanske behöver modifieras för att uppnå vissa betygskriterier.

4.2 Kompilering och körning

Det finns redan en s.k. `Makefile`, denna fil innehåller kommandon för hur man kompilerar koden. Enklast är att gå till din projektmapp och skriva:

```
make all
```

vilket kommer kompilera all kod och generera en körbar fil (`Shell`) som ligger i katalogen `bin/`. Tänk på att om du väljer att lägga till fler filer i projektet så måste du uppdatera `Makefile`.

Det går också bra att importera alla källkodsfiler (.cpp och .h) till ett IDE (t.ex Visual Studio, Eclipse etc.) och kompilera och köra programmet därifrån.

5 Uppgift

Ett filsystem ska konstrueras. Sekundärminnet som organiseras av filsystemet är simulerat som en 2-dimensionell byte-array (se avsnittet Hjälpfiler). Om du väljer att ignorera hjälpfilerna, se då till att din implementation har en matris av 250 st block med 512 byte i varje. Filsystemet ska vara en hierarkisk trädstruktur som kan ha godtyckligt många undernivåer. Systemet ska klara sökvägar av typen `/aaa/bbb/ccc...`. Roten ska betecknas `/`. Se också till att din lösning inte genererar minnesläckor.

Vad som måste implementeras för att ge respektive betyg anges i avsnitt Bedömning.

- **Tips 1** Tänk noga igenom hur strukturen för filsystemet ska se ut innan du börjar koda.
- **Tips 2** Om ni är osäkra på hur ert filsystem är tänkt att bete sig för olika kommandon går det bra att testa hur respektive kommando fungerar i valfritt Linux/UNIX-system, exempelvis i laborationssalen. De flesta kommandona har en motsvarighet som heter likadant där.

6 Bedömning

6.1 Krav för godkänt, betyg E

För att godkännas på uppgiften med lägsta betyg ska följande genomföras:

Generella krav:

- Inga utskrifter i filsystemets API (klassen `FileSystem` i klassdiagrammet). Utskrifterna ska hanteras i `Shell`.

Följande Shell-kommandon ska implementeras med hjälp av filsystemets API:

`format` bygger upp ett tomt filsystem, dvs. formatterar disken.

`quit` lämnar körningen (finns rean implementead)

`createImage <filepath>` sparar det simulerade systemet på en fil på datorns fysiska hårddisk så att den går att återskapa vid ett senare tillfälle.

`restoreImage <filepath»` återställer filsystemet från en fil på datorns fysiska hårddisk.

`create <filepath>` skapar en fil på den simulerade disken (datainnehållet skrivs in på en extra tom rad)

`cat <filepath>` skriver ut innehållet i filen filnamn på skärmen.

`ls` listar innehållet i aktuell katalog (filer och undermappar).

`cp <oldfilepath> <newfile>` skapar en ny fil som är en kopia av den existerande filen.

`mkdir <dirpath>` skapar en ny tom katalog på den simulerade disken.

`cd <dirpath>` ändrar aktuell katalog till den angivna katalogen på den simulerade disken.

`pwd` skriver ut den fullständiga sökvägen ända från roten till ”current directory”.

`rm <filepath>` tar bort angiven fil från den simulerade disken.

I dokumentationen ska följande finnas:

- En tydlig beskrivning av den logiska strukturen i *ert* filsystem.
- Tydlig beskrivning över metoderna i ert API (hur metoderna i klassen `FileSystem` anropas och vad de gör).
- Ett simpelt klassdiagram samt en kort beskrivning av varje klass.

6.2 Krav för betyg C

För att nå betygsnivån C ska alla krav för godkänt (betyget E) vara uppfyllda. Dessutom tillkommer följande:

- Filer ska kunna vara större än 1 block.
- Kommandot `copy` ska tillåta att kopiera en fil till en annan mapp än den mapp filen redan finns i.
- Kommandot `ls` ska även ange storlek i (i byte) på varje element i listan.

Dessutom ska följande funktioner implementeras:

- `append <filepath1> <filepath2>` lägger till innehållet från första filen i slutet av den andra.
- `mv <sourcepath> <destpath>` ändrar namn och/eller flyttar på fil från `source` till `dest`.

6.3 Krav för betyget A

Alla krav för betyg C ska vara uppfyllda. Dessutom tillkommer följande:

- Systemet ska hantera både absoluta och relativa sökvägar för alla Shell-funktioner.
- De relativa katalognamnen `.` och `..` ska kunna användas. Där `.` betyder aktuell katalog, och `..` betyder föräldrakatalogen i katalogträdet.
- System ska kunna hantera accessrättigheter till filer. Implementera följande kommando (kommandotolken i `Shell` måste då utvidgas).

  ```
  chmod <accessrights> <filepath>
  ```

  Exempelvis: `chmod 4 FILUR`, får effekten att `FILUR` får läsas men inte skrivas.

  Dokumentera tydligt vilka koder som slår av/på rättigheter att läsa/skriva en till en fil. Glöm inte att uppdatera `help()`.

- Se till att filsystemets `read()` och `write()` tar hänsyn till rättighetena så att man inte läser en fil man saknar läsrättighet till eller skriver till en fil man saknar skrivrättighet till.

6.4 Krav för övriga betyg

För betyg D ska samtliga krav för betyg E vara uppfyllda samt de flesta av kraven för betyg C.

För betyg B ska samtliga krav för betyg C vara uppfyllde samt de flesta av kraven för betyg A.