Laboration 4: Filsystem

DV1460 - Realtid- och Operativsystem

Per Sandgren och Caj Lokkin

Contents

0.1 Introduktion

Rapportens syfte är att visa grundläggande teknisk förståelse för hur ett filsystem är organiserat och hur de nödvändigaste funktionerna kan implementeras. I den här rapporten visas ett implementerat filsystem till Linux.

0.2 Metod

Datorer från BTHs labbsalar användes, med Debian Linux installerat, samt två laptops med Debian och Arch Linux installerat för att skriva och testa koden som skrevs. När information saknades om hur olika funktioner implementeras så användes egen programmeringserfarenhet, när erfarenheten var för liten så användes programmeringssajter så som stackoverflow. Struktur att gå efter - I filsystemet så är det en trädstruktur med mappar som delar i trädet, varje mapp har en pekare till förälder och alla dess barn samt pekare till filer, root har nullptr som förälder. När filer skapas så skapas en pekare från vilken katalog den gjordes under, när en katalog skapas så skapas en pil från föräldern till barnet och från det nya barnet till föräldern. Det är den här strukturen som arbetet följde.

En katalog är en mapp i filsystemet, en katalog kan ha katalog i sig samt filer. När filer skapas så skrivs deras innehåll till det virtuella minnet, inte metadatan.

För implementering av funktionerna så kollade man först på vad som krävdes för funktionerna som skulle implementeras, därifrån försökte vi ta fram vart funktionen skulle grundas, beroende då på vad funktionen behöver, om funktionen behövde komma åt virtuell hårddisk så såg vi till att funktionen fanns där minneshanterarobjektet fanns som vari- abel. För att funktionerna i shell sen skulle komma åt dessa gjorde vi underfunktioner som anropade funktionerna i klassfilerna som låg under. EX: Shells funktion "createImage" anropar en funktion från FileSystem-klassen "createImage", inkluderar det som behövs, alltså ett namn för filen, funktionen finns i FileSystem-klassen för att vi behöver komma åt MemblockDevice-klassen för att kunna hämta minnesblocken för att skriva ut de i filen vi skapar med "createImage". Funktioner i shell som skulle fungera var:

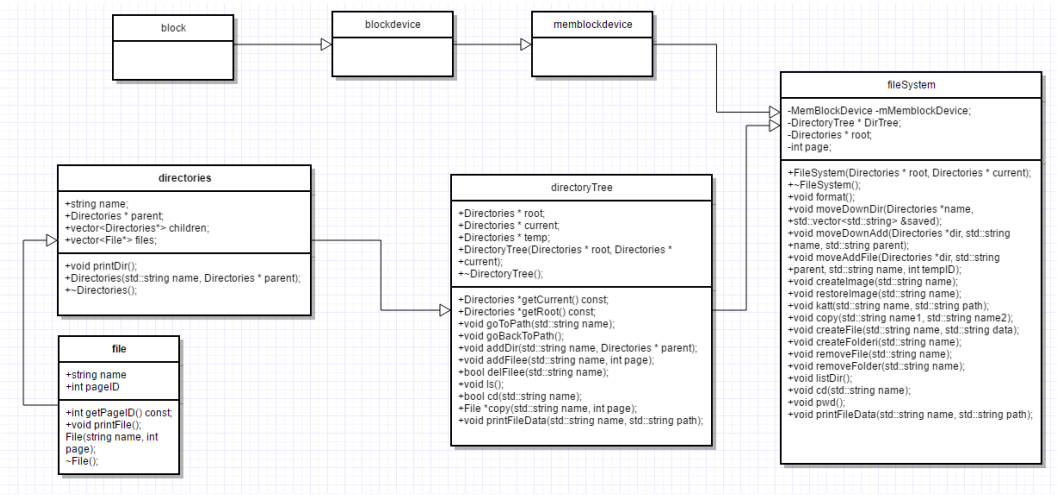

Figure 1: klassdiagram över filsystemet

Klasser:

Block, en klass med dynamisk char-array som allokerar 512 byte med konstruktorn.

Blockdevice, en rent virtuell klass, har en dynaimisk allokerad fält med Block-objekt.

MemBlockDevice, ärver från BlockDevice, genererar 250 block med blockstorlek 512 byte i konstruktorn.

File, fil, innehållandes namn och pageID (till för att hålla koll på vart i det virtuella minnet dens innehåll ligger).

Directories, kataloger, innehållande pekare till namn och förälder, har även en vector med Fil-objekt.

DirectoryTree, hanterar Directories och vad som finns i de, funktioner för att använda, lägga till o redigera i Directories och dess filer.

FileSystem, innehåller ett MemBlockDevice-objekt, DirectoryTree-objekt och funktioner för att understödja APIt.

0.3 Resultat

I det API som gjorts så har ett antal funktioner skapats för shell-commandona som skulle implementeras, till exempel create eller ls, att fungera.
Shellkommandon och API bakom:

format(), formatera disken, tar ingen input och används för att bygga upp ett tomt filsystem.

quit(), lämna körning, tar ingen input.

createImage(std::string name) och restoreimage(std::string name), spara ner systemet och bygga upp det, tar in ett namn att döpa filen till eller namn på fil som ska läsas in.

create, (kallar på createFile), skapa fil, tar in input för sökväg/namn och filinnehåll, anges ingen sökväg så skapas den i den katalog man är i.

cat, kallar på katt, läsa innehåll i fil, tar in input för sökväg/namn på fil som ska läsas, anges ingen sökväg så skapas den i den katalog man är i.

ls, kallar på listDir,, lista innehåll i katalog, tar input för namn på katalog som ska lista innehåll ifrån och sökväg, anges ingen sökväg så listas den katalog man är i.

copy(std::string path1, std::string path2), skapa en kopia av en fil, tar in input för källnamn, nya filnamnet, källans sökväg och den nya sökvägen, anges ingen sökväg så kopieras och skapas den i det katalog man är i.

mkdir(kallar på createFolderi), skapa katalog, tar in input för sökväg/katalogsnamn, anges ingen sökväg så skapas den i den katalog man är i.

cd(std::string name), göra en katalog till nuvarande katalog, tar in sökväg för att gå till en katalog.

pwd(), skriva ut fullständig sökväg, tar inte in någon input.

rm(kallar på removeFile), ta bort fil, tar in input för sökväg/filnamn, anges ingen sökväg så försöker funktionen ta bort filen i den katalog man är i.

help(), tar ingen input, skriver ut hjälp för de olika kommandon som finns. Samt några som inte blev implementerade: mv och append.

void createFile(std::string path, std::string data), kallas på av create, lägger till en fil med filinnehåll i systemet om namn inte kolliderar, tar in input för sökväg/namn och filinnehåll, anges ingen sökväg så skapas den i den katalog man är i.

void createFolderi(std::string path), lägger till en katalog i systemet om namn inte kolliderar, tar in input för sökväg/namn, anges ingen sökväg så skapas den i den katalog man är i.

void moveDownDir(Directories * name, std::vector<std::string> saved), till för createImage som rekursiv funktion, används för att söka igenom hela kataloger o spara ner vad den hittar.

void moveDownAdd(Directories * dir, std::string name, std::string parent), till för funktionen restoreImage, rekursiv funktion till för restoreImage-funktionen, följer sparade filen och lägger till vad moveDownDir hittade i rätt ordning, katalogen återskapas i rätt ordning.

void moveAddFile(Directories *dir, std::string parent, std::string name, int tempID), rekursiv funktion till för att lägga till filer i rätt katalog i restoreImage-funktionen. katt, kollar innehåll av fil o skriver ut det.

void katt, kallas på av cat, kallar på printFileData.

void removeFile(std::string path), tar bort fil med det filnamn som angetts i den katalog som angetts, om ingen sökväg angets används nuvarande katalog.

void listDir(std::string path), kallas på av ls, tar in katalogs namn, skriver ut filer och kataloger som finns i angiven katalog, om ingen sökväg angets används nuvarande katalog.

void printFileData(std::string path), kallas på av katt, tar in sökväg/namn och skriver ut data för angiven fil, om ingen sökväg angets används nuvarande katalog.

0.4 Analys och Diskussion

Eftersom att det finns så många olika sätt att gå till väga när man gör alla funktioner så finns det troligen många bättre sätt att implementera allt, om mer tid hade lagts på laborationen när projektet startade hade man kunnat skrota all skräpkod som ligger lite över allt och göra en mer effektiv och snygg kod, hade man haft flera skärmar när man hade arbetat så hade man även kunnat se mer av koden och strukturerat bättre, skärmen till datorn i laborationssalen på bth användes i början följt av små laptop-skärmar. Filsystemet funkar, kan finnas buggar som gör att säkerheten är låg då inget fokus gjorts på hur säker APIn är, dock så om man hade mer tid eller om det vore krav att täppa till potentiella säkerhetshål så skulle man nog gjort ett säkrare system.

0.5 Slutsats

Grundläggande teknisk förståelse om hur filsystem är uppbyggda, organiserade och hur de fungerar är erhållen. I ett filsystem måste mycket implementeras, att börja med struktur är viktigt, att veta vart filer allokeras och hur de allokeras är en bra grund i förståelse hur ett operativsystem talar med minne/hårddisken.