Природно-математички факултет Крагујевац
Институт за математику и информатику

Семинарски рад из предмета Оперативни системи 2

тема: **Fast file system**

Студент:
Александар Вујиновић 55/11

Професор:
др. Милош Ивановић

У Крагујевцу, јануар 2016.

Садржај

1. “Стари” фајл систем

Када је први **Unix** оперативни систем представљен, један од његових креатора, Кен Томсон, створио је и први фајл систем, који је био прилично једноставан.

Састојао се из три главна дела:
- Супер блок **C** који је садржао информације о целокупном фајл систему: његовој величини, броју и-чворова, показивач на листу слободних блокова, итд.
- Индекс чворови (**и-чворови**): структуре које садрже метаподатке о сваком фајлу на диску, попут величине, локације фајла на диску, правима приступа, итд.
- Део са подацима

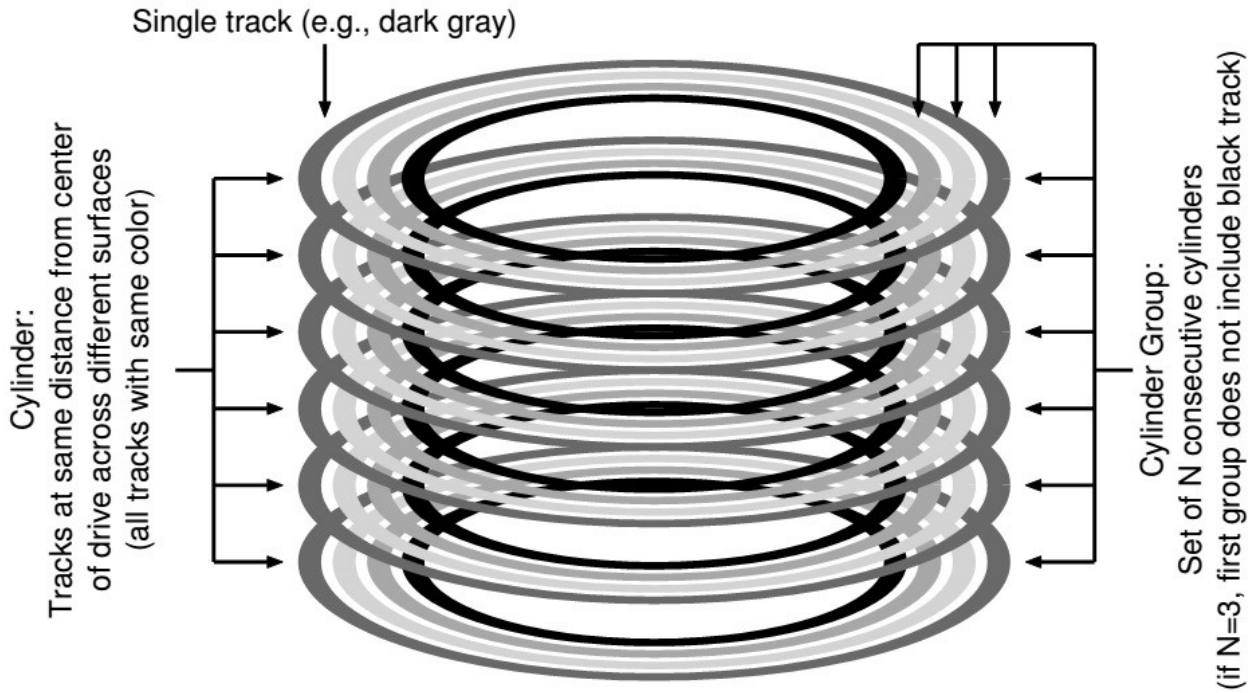
Добра ствар код овог фајл система је та што је био једноставан, а испуњавао је све основне захтеве. Чак је успео и да отклони мане претходних фајл система, уводи функционалну хијерархију директоријума, и једноставан је за употребу. Величина блока је 512 бајта.
Проблем овог фајл система су биле јако лоше перформансе. Нека испитивања показују да су перформансе у старту биле лоше, а да су се временом само погоршавале. Након одређеног времена, користило би се само 2% укупног протока диска.
Једна од главних мана је што је фајл систем третирао хард диск као RAM; Подаци су смештани по њему без неког посебног реда, што је после кратког времена доводило до тога да се један фајл у деловима налази разбацан по целом диску. Његово учитавање је захтевало много више У/И операција читања блокова него што је његова стварна величина.
Ово је након одређеног броја уписивања и брисања фајлова доводило до фрагментације меморије диска, тј. свуда около су се налазили мали делови слободне меморије, који су одвојено били бескорисни.
Други проблем је била мала величина блока, од само 512 бајта. Са једне стране је то било добро, јер није било пуно неискоришћене меморије у делимично попуњеним блоковима. Али уколико се фајл простирао на више блокова, његово учитавање је захтевало пуно операција учитавања, којима је претходило позиционирање глава за читање на различитим локацијама.
Све ово је приморало програмере да развију нови, ефикаснији фајл систем, и у томе су успели. Тако је настао **FFS – Fast file system**.

2. Fast file system

Идеја групе научника са Берклија је била да конструишу фајл систем који ће имати „паметан“ диск, чиме ће отклонити проблеме алокације меморије и побољшати перформансе претходника. Њихов **FFS** је покренуо еру истраживања фајл система; задржали су исти интерфејс према фајл систему (API, укључујући функције *open(), read(), write(), close()*), али су променили њихову унутрашњу имплементацију. Та нова конструкција фајл система је у употреби и данас.

2.1. Групе цилиндара

Први корак у конструкцији је била логичка подела диска. Диск је подељен у групе **цилиндара**. Цилиндар је скуп трака које су на истом растојању од центра, али се налазе на различитим плочама диска. **FFS** групише сваких **H** суседних цилиндара у групу, па се цео диск може посматрати као колекција група цилиндара.



Слика 1. Групе цилиндара на диску

И-чворови се налазе на спољним тракама сваке групе цилиндара. У почетку, сви и-чворови су се налазили на првој плочи диска. Отказом те плоче, неповратно су се губиле све битне информације. Овај проблем је решен тако што свака група цилиндара чува своје и-чворове на произвољној плочи.

Савремени фајл системи током рада не знају коју групу цилиндара користе. То и није потребно, јер диск скрива своју унутрашњу организацију од клијента, кога се то и не тиче. Уместо тога, фајл системи посматрају дискове као **групе блокова**, који имају узастопне адресе.

Ове групе блокова су централни механизам које **FFS** користи да побољша перформансе. Смештањем два повезана фајла у исту групу, фајл систем се осигурава да следеће читање оба фајла једног за другим, неће резултовати дугом претрагом њихових локација на диску(**seek time**).

Да би могао да складишти фајлове и директоријуме у исту групу блокова, фајл систем мора имати детаљне податке о свакој групи. Пример једне групе цилиндара је приказан на слици 2.

Слика 2. Изглед једне групе блокова

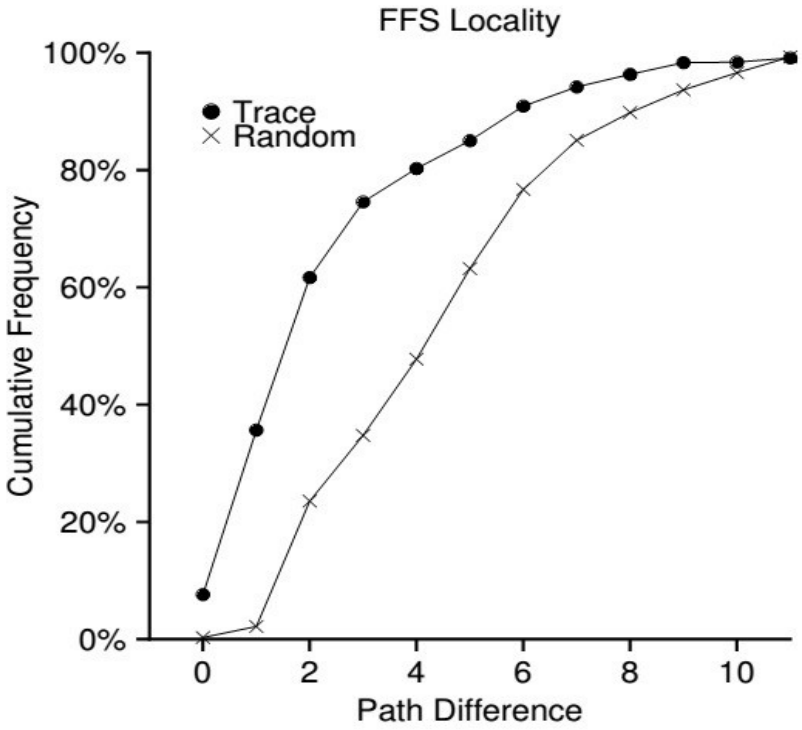
FFS чува копију супер блока у свакој групи цилиндара. Тиме осигурава да, у случају отказа једне групе, не остане без важних информација. У оквиру сваке групе, **FFS** такође мора знати који су и-чворови и блокови података заузети. За то се користе **битмапе и-чворова** и **битмапе података**. Битмапе су одличан начин за вођење евиденције о слободном простору на диску. Помоћу њих фајл систем лако може наћи довољно велико парче слободне меморије и алоцирати је за нови фајл. И-чворови и део са подацима је исти као на старим фајл системима.
Оно по чему се посебно разликује **FFS** од претходних фајл система је величина једног блока. Сада она износи минимално 4096 бајтова, тј. 4KB, за разлику од пређашних, који је био величине 512 бајтова.

2.2. Политика смештања фајлова и директоријума

Користећи податке о свакој групи цилиндара, **FFS** мора донети одлуку како да смешта нове фајлове и директоријуме на диск, тако да побољша његове перформансе. Основна идеја је проста: чувај повезане фајлове заједно на једном месту, а независне фајлове одвојено. Наравно, на фајл систему је да закључи шта су повезани фајлови и да их смешта близу једне другима.
Прво треба одредити где сместити нов директоријум. Користи се следећи приступ: налази се група блокова са малим бројем директоријума (због одржавања баланса између група), и великим бројем слободних и-чворова (да би се лакше алоцирао велики број нових фајлова). У групу која испуњава наведене услове смешта се директоријум.
Приликом смештања фајлова, **FFS** ради две ствари. Прво, труди се да алоцира блокове података за фајл у истој групи блокова где се налази и и-чвор тог фајла. Овим скраћује време приступа и-чвору и подацима. Друго, труди се да смести све фајлове једног директоријума у исту групу блокова где је смештен и сам директоријум.
Пример. Уколико се креирају четири фајла, /дир1/1.txt, /дир1/2.txt, /дир1/3.txt, /дир99/4.txt, фајл систем ће покушати да смести прва три фајла у исту групу блокова, док ће четврти фајл бити смештен у сасвим другу групу.

2.3. Мера локалности фајлова

Да је управо описана политика складиштења ефикасна, показује следећи пример. Нека је отворен фајл **f** из директоријума **dir** (**dir/f**), и фајл **g**, из истог директоријума (**dir/g**). Дистанца између ова два фајла је 1, јер се налазе у истом директоријуму, али нису исти фајлови. Мерна јединица дистанце између фајлова је заправо мера колико далеко се морамо кретати кроз стабло директоријума да би се стигло од једног до другог фајла. На слици 3. је приказан график локалности у зависности од броја фајлова.



Слика 3. График локалности у зависности од броја отворених фајлова

На графику се види да око 7% приступа припада претходно отвореним фајловима, док се око 40% приступа односи на исте, или фајлове који су у истом директоријуму.
Следећих 25% приступа се односи на фајлове између којих је дистанца 2. **FFS** не покрива овај тип локалности, па стога овакви приступи фајловима захтевају више времена за тражење и позиционирање између два читања.
Зарад поређења, на графику је приказана локалност и за „Random“ смештања. Као што се види, нема пуно локалности међу фајловима. Ипак, сви фајлови припадају **root** фолдеру, тако да ипак постоји неки тип локалности, што је довољно за почетак.

2.4. Изузетак смештања великих фајлова

У **FFS** постоји један важан изузетак који се тиче политике смештања великих фајлова на диск. Уколико се користи досадашње правило локалности, цео фајл ће бити смештен на једном месту. Ово доводи до ризика да ће, уколико је фајл довољно велики, заузети једну комплетну групу блокова, или чак и више њих. Овакав приступ ремети правило локалности, јер у тој групи блокова неће бити места за друге, мање фајлове који су можда у истом директоријуму као и велики фајл.
Зато фајл систем ради следеће. Фајл се дели на више једнаких делова (сем последњег), и смешта у више различитих група блокова. Први део, величине 12 блокова нпр., се смешта у исту групу блокова где се налази и и-чвор тог фајла. Остатак фајла се дели на блокове подједнаке величине и смешта у друге групе. Ево како то изгледа на примеру.

Слика 4. Смештање великог фајла у једну групу блокова

На слици 4. је приказано како би изгледало смештање великог фајла без изузетка. Са изузетком, би изгледало као на слици 5.

Слика 5. Смештање великог фајла на више различитих група блокова

Логична претпоставка је да оваква подела фајла умањује перформансе читавог система, посебно код секвенцијалног приступа, где је потребан цео фајл. Онда се пуно времена губи на позиционирање свих делова. Али овај проблем се може решити пажљивим одабиром величине делова који ће се смештати по групама.
Уколико је величина једног дела довољно велика, фајл систем ће потрошити већину времена на трансфер фајла са диска у главну меморију, док ће се врло мали део потрошити на позиционирање главе диска између делова. Овај процес смањења overhead-а повећањем величине делова се назива **амортизација** и то је најчешћа техника у рачунарским системима.

Ево како то изгледа на примеру. Претпоставимо да је просечно време позиционирања главе диска 10 ms, и нека је брзина преноса података 40 MB/s. Ако желимо да нам времена позиционирања и читања делова фајла буду иста, 10 ms, величина једног дела фајла по групи ће бити следећа:

$$\frac{40\ \cancel{MB}}{\cancel{sec}} \cdot \frac{1024\ KB}{1\ \cancel{MB}} \cdot \frac{1\ \cancel{sec}}{1000\ \cancel{ms}} \cdot 10\ \cancel{ms} = 409.6\ KB$$

Уколико желимо да однос времена буде већи у корист величине делова, тј. искоришћења протока, треба повећати управо величину сваког дела.

Ипак, **FFS** не користи ову врсту прорачуна да би поделио велике фајлове на делове. Уместо тога, користи једноставнији приступ. Првих 12 блокова се смешта у исту групу са и-чвором фајла. Са величином блока од 4KB, и 32-битном адресом на диску, ова стратегија имплицира да ће сваких 1024 блока фајла (4MB), бити смештено у различитим групама, са јединим изузетком првог дела фајла, који ће имати додатних 48KB због и-чвора који описује фајл.

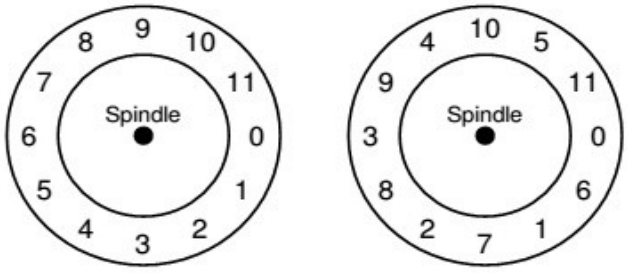
2.5. Оптимизација коришћења диска

Повећање величине једног блока података са 512 бајтова на 4KB довело је до појаве новог проблема. Иако је ова величина блока погодовала смештању већих фајлова, приликом смештања мањих фајлова, величине око 2KB, долазило је до беспотребног трошења меморије. Уколико би фајл био мало већи од 2KB, у блок од 4KB поред њега не би било места за још један фајл, па је тако више од 1KB на сваких 4KB било неискоришћено, што је

на глобалном плану износило више од 25% меморије целог диска.
Проблем је решен дељењем сваком блока на фрагменте величине 512 бајтова. Уколико би требало да се смести фајл величине 2KB, 4 фрагмента би била алоцирана, док би остатак био слободан за употребу, посебно ако би се величина фајла повећавала и захтевала још простора.

2.6. Распоред блокова

Још једна новина коју је **FFS** донео и која је допринела побољшању перформанси, био је распоред блокова. На дотадашњим фајл системима, диск није био толико самосталан, и захтевао је много процесорског времена за позиционирање главе и сличне радње.
Проблем се појавио када је учитаван фајл који је био смештен у суседним секторима диска.



Слика 6. Различите нумерације сектора

Као што је приказано на левој страни слике 6, сектори су били нумерисани по распореду. Нека фајл систем учита блок 0. Док се подаци са њега обраде, и процес закључи да су му потребни подаци и са блока 1, за учитавање тог блока фајл систем мора чекати читаву ротацију диска, јер се блок 1 више не налази испод главе читача.
Проблем је решен на следећи начин. Блокови су нумерисани по принципу „сваки други“, као што се може видети на десној страни слике 6.
Данас је то решено принципом **предвиђања**, тј. ако је учитан блок 0, фајл систем предвиђа да ће и блок 1 бити потребан убрзо, тако да и њега смешта у кеш диска. Ипак, у то време, то је била заиста корисна новина.

2.7. Именовање фајлова

FFS је донео још нешто ново. Омогућио је корисницима да фајловима дају дужа имена, за разлику од дотадашњег имена фиксне дужине од 8 карактера. Овај нови концепт је омогућила употреба **симболичних линкова**. Дотадашње именовање фајлова је имало бројна ограничења. Симболични линкови су омогућили кориснику да креира псеудониме за било који фајл или директоријум на систему. **FFS** је такође први увео функцију **rename()** за преименовање фајлова.

2.8. Закључавање фајлова

Стари фајл системи нису имали добро решење за закључавање фајлова. Процеси који су морали да измене фајл би покушавали да закључају фајл. Ако не би успели, морали би да чекају одређено време, па да покушају поново. Овакав приступ има више недостатака. Прво, процеси су трошили пуно процесорског времена покушавајући да закључају фајлове. Друго, у случају пада система, претходно закључан фајл би остао у том стању, иако се процес који га

је закључао не би више извршавао. Коначно, администраторски процеси који имају највећа права за управљање фајловима, морали би да имају другачији механизам приступа, да би се разликовали од обичних корисника.
FFS омогућава ексклузивно и дељено закључавање фајла, попут читалац/писац односа. Више процеса може истовремено читати фајл, али само га један процес у једном тренутку може мењати.
Закључавање или откључавање фајла је могуће само над отвореним фајловима. То значи да се начини закључавања могу мењати без затварања и поновног отварања фајла. Ово је корисно у следећем случају. Процес изврши дељено закључавање фајла, за читање. Уколико утврди да је фајлу потребно ажурирање, примењује ексклузивно закључавање над фајлом и врши ажурирање.

3. Закључак

Увођење **FFS** је био преломни тренутак у историји фајл система. Постало је јасно да је проблем управљања датотекама један од најзанимљивијих питања везаних за оперативне системе, и показало се да је најважније успешно управљање уређајима, посебно диском. Од тог времена, развијено је на стотиине нових фајл система, али и данас многи системи имају сличности са **FFS**. Сви модерни системи су научили најбитнију лекцију **FFS –** третирају диск као диск, а не RAM.

4. Литература

- “A Fast File System for UNIX ” Marshall K. McKusick, William N. Joy, Sam J. Leffler, Robert S. Fabry
- http://pages.cs.wisc.edu/~remzi/OSTEP/file-ffs.pdf
- http://www.cs.pomona.edu/~markk/filesystems/bsd.html
- http://www.cse.unsw.edu.au/~cs9242/02/lectures/09-fs/node4.html
- https://courses.cs.washington.edu/courses/cse451/05wi/lectures/14-ffs.pdf
- http://www.cs.columbia.edu/~junfeng/11sp-w4118/lectures/l24-fs-example.pdf
- http://minnie.tuhs.org/CompArch/Lectures/week11.html