Perbandingan Unjuk Kerja Beberapa *Filesystem*
yang Dapat Berjalan di GNU/Linux
Oleh: Ema Utami, S.Si, M. Kom

1.1Pendahuluan

Filesystem merupakan salah satu komponen yang penting dan merupakan bagian yang paling telihat oleh pengguna dalam sistem operasi. Pengguna sistem operasi akan sering berinteraksi dengan filesystem. Filesystem sering digunakan sebagai tempat penyimpanan data. Program yang dijalankan dalam sistem operasi paling tidak akan menggunakan sebuah file untuk dibaca atau ditulis. Terdapat berbagai macam jenis filesystem dan pada umumnya 'dibawa' oleh sistem operasi seperti MSDOS filesystem atau FAT16 merupakan filesystem yang digunakan oleh sistem operasi MSDOS, FAT32 merupakan filesystem yang digunakan oleh MS Windows9X, NTFS oleh MS Windows NT dan 2000 dan lain sebagainya. Sistem operasi Microsoft DOS dan Windows9X hanya dapat mengenali sebuah filesystem yaitu FAT(16/32) sedangkan Microsoft Windows NT dan 2000 dapat mengenali dua macam file system yakni NTFS dan FAT. Kecederungan penggunaan sistem operasi dari Microsoft membuat sebagian besar dari kita hanya mengenal dan menggunakan filesystem bawaan dari sistem operasi yakni FAT atau NTFS. Dengan hanya adanya sebuah filesystem dalam sistem operasi tidak memungkinkan pengguna untuk mendapatkan alternatif filesystem lain yang mungkin lebih baik dari pada filesystem bawaan. Perbandingan unjuk kerja filesystem akan menjadikan pengguna sistem operasi mendapatkan informasi yang berguna untuk memilih filesystem yang digunakan. Sistem operasi GNU/Linux memiliki kelebihan salah satunya mampu menangani lebih dari satu macam filesystem. Tulisan ini merupakan hasil penelitian yang dilakukan penulis dengan memperbandingkan ujuk kerja beberapa *filesystem* yang dapat berjalan di GNU/Linux. Tujuan penelitian adalah untuk mendapatkan data yang didasarkan atas ujicoba mengenai waktu yang diperlukan untuk melakukan penulisan, pembacaan dan penghapusan suatu file dalam suatu filesytem. Selain itu penelitian ini juga bertujuan untuk mendapatkan karakteristik suatu filesystem terhadap pembacaan, penulisan dan penghapusan. Dengan hipotesa sementara: *Filesystem* dengan tipe *journal* secara umum akan memiliki unjuk kerja yang lebih baik dari *filesystem*dengan tipe *non-journalling*.Penelitian direncanakan dilakukan dengan metode ujicoba langsung dan studi literatur. Perangkat keras yang digunakan dalam penelitian ini mempunyai spesifikasi sebagai berikut :

- Processor AMD Athlon(tm) XP 2500+ cache size : 512 KB
- RAM 256 Mbyte
- Hardisk Seagate ST340014A, 78165360 sectors (40021 MB) w/2048KiB Cache, CHS=4865/255/63, UDMA(100)
- Controler : nVidia Corporation nForce2 IDE

Sedangkan perangkat lunak yang digunakan dalam penelitian ini adalah :

- Sistem Operasi GNU/Linux Distribusi Slackware 9.1 Kernel 2.6.7.
- Program *Benchmark* Bonnie++
- Program bantu seperti, *touch, mkdir* dan *rm.*
- Program pembuat *filesystem* mkreiserfs, mkfs.msdos, mkfs.ext3, mkfs.xfs dan mkfs.jfs

1.2 Manajemen File

Manajemen File merupakan salah satu servis yang paling terihat dari sekian banyak servis yang diberikan oleh sistem operasi. Komputer dpat menyimpan file dalam berbagai media dari bentuk *floppy, flash, tape* maupun *hardisk.* Secara umum *file* berisi suatu data yang diperlukan baik itu sebagai program maupun obyek lainnya. Sebuah file akan memiliki nama, tipe, waktu pembuatan dan berbagai properti lainya. Keberadaan *file* akan sangat bergantung dari jenis *filesystem* yang digunakan oleh sistem operasi. *Filesystem* secara sederhana dapat

dikatakan sebagai sebuah metoda untuk menyimpan *file* dalam struktur tertentu.

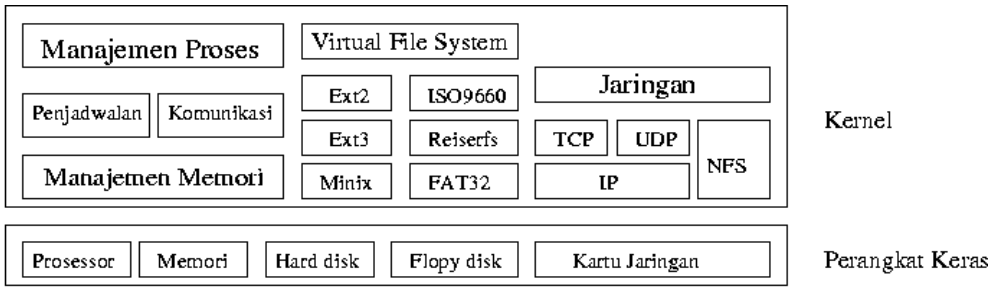
1.3 Kernel Linux

Keberadaan filesystem tidak terlepas dari bagian sistem operasi. Salah satu Sistem Operasi yang pesat perkembangannya adalah GNU/Linux atau lebih sering disebut dengan Linux. Linux sendiri sebenarnya hanya merupakan kernel yang merupakan bagian penting dari sebuah sistem operasi yang berinteraksi langsung dengan perangkat keras komputer. Kernel Linux pertama kali dibangun oleh Linus B Trovalds pada tahun 1991. Secara garis besar kernel Linux terdiri atas fungsi :

1. Menajemen Proses
   Manajemen proses seperti penjadwalan proses, pengaturan *time slice* juga pengaturan hubungan antara proses anak dan proses induk.
2. Manajemen Memori
   Manajemen memori baik memori fisik maupun virtual memori dilakukan oleh kernel.
3. Manajemen Sistem File
   Manajemen sistem file di dalam Linux dilakukan oleh kernel dengan menggunakan *Virtual File System* yang memungkinkan penggunaan lebih dari satu sistem file dalam Linux.
4. Komunikasi dan Jaringan
   Dukungan terhadap protokol TCP/IP dalam level kernel.

Struktur kernel Linux yang terdiri atas beberapa fungsi di atas dapat digambarkan seperti gambar 1

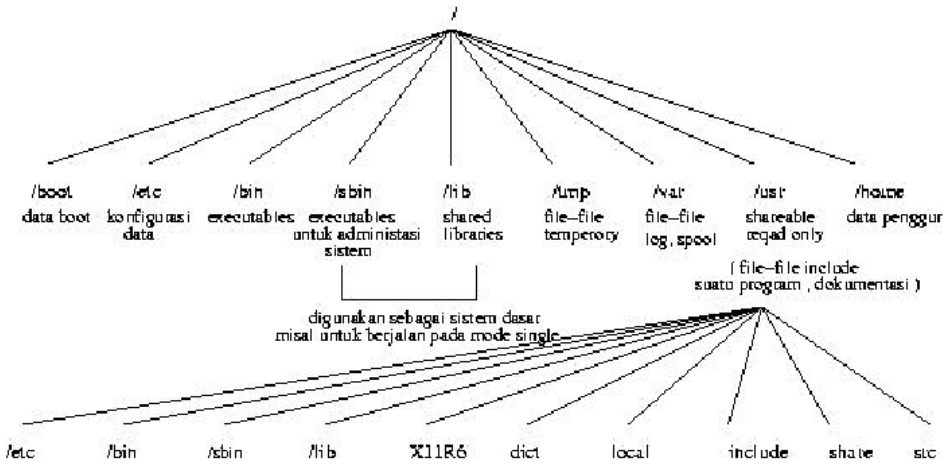
Gambar 1 Struktur Kernel Linux



1.4 Linux Filesystem

Pengorganisasian *Linux File System* (LFS) mempunyai kemiripan dengan sistem DOS, Windows atau sistem operasi lain yaitu memiliki sistem file hirarki. Hirarki dari LFS dimulai dengan direktori *root* yang ditandai dengan tanda *slash* (/). Di bawah direktori *root* dapat berisi file dan direktori dan setiap direktori dapat berisi file dan direktori dan seterusnya sehingga dapat terlihat pada gambar 2.

Gambar 2. Linux File System



LFS dan sistem file sistem operasi varian UNIX lainnya didefinisikan dalam *Filesystem Hierarchy Standard* (FHS). FHS menyatakan bahwa dalam sistem file varian UNIX dapat dibagi menjadi dua buah konsep dasar yaitu :

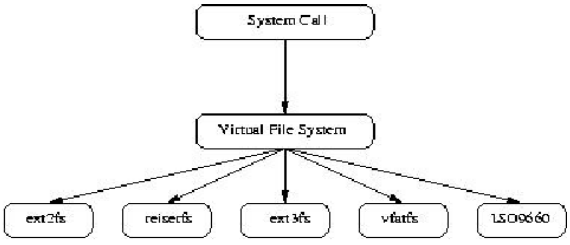
1. Data statis dan data variabel
   Data statis merupakan file/direktori yang isinya tidak dapat diubah tanpa campur tangan *system administrator* (sysadmin) sedangkan data variabel merupakan file/direktori yang dapat diubah tanpa interfensi sysadmin.
2. Data *shareable* dan data *unshareable*
   Data *shareable* merupakan file/direktori yang isinya dapat dibagi pakai oleh beberapa *hosts* sedangkan *unshareable* merupakan file/direktori yang spesifik terhadap suatu *host*.

Tabel 1 berikut menunjukkan beberapa direktori yang dapat bersifat statis, variabel, *shareable* dan *unshareable*.

Tabel 1 Direktori dalam Dua Konsep Dasar Filesystem Hierarchy Standard

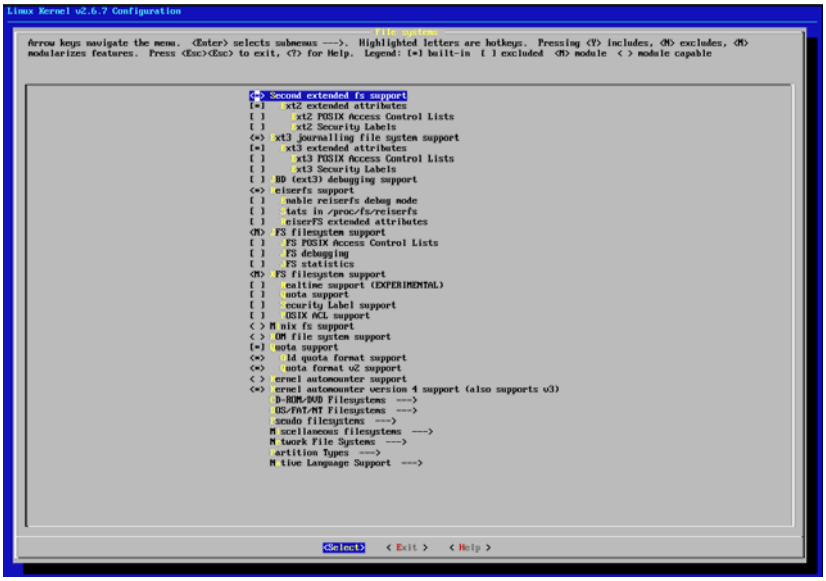
	shareable	*unshareable*
static	`/usr & /etc`	`/opt & /boot`
variable	`/var/mail` `var/spool/news`	`/var/run` `/var/lock`

Linux menggunakan *Virtual File System* (VFS) untuk menangani beberapa sistem file pada Linux. VFS merupakan layer software pada kernel yang menyediakan antarmuka sistem file kepada pengguna. VFS akan menterjemahkan *system call* untuk mengakses sistem file sesungguhnya. Gambar 3 menunjukkan hubungan antara VFS dan beberapa sistem file dalam kernel.



Gambar 3 Hubungan Antara Virtual File System dan File System

VFS memiliki kemampuan untuk menjadikan beberapa sistem file dapat digunakan dalam LFS. VFS menjadikan seluruh sistem file menjadi satu kesatuan yang global. Pada LFS hanya dikenal sebuah *root* direktori yang dapat menaungi berbagai macam sistem file yang dapat terdiri dari beberapa device. Pengaksesan sistem file pada suatu *device* dilakukan dengan cara *mounting* sistem file tersebut ke suatu direktori tertentu yang dinamakan *mount point* sehingga dalam sebuah direktori *root* memungkinkan terdiri lebih dari satu sistem file. Sistem file yang didukung Linux antara lain : *Acorn Advanced Disc Filing System, Amiga Fast File System, SCO UnixWare Boot File System (BFS), CODA filesystem, Ext2 filesystem, OS/2 HPFS, ISO 9660 (CDROM) filesystem, NTFS filesystem (Windows NT), Linux's /proc filesystem, ROMFS filesystem, SystemV/V7/Xenix/Coherent filesystem, UDF filesystem, UFS filesystem, VFAT filesystem*, dan lain-lain. Sehingga dari gambar 2 memungkinkan satu direktori berbeda *filesystem* dengan direktori lainnya. Dukungan *filesystem* yang bermacam-macam tersebut harus disiapkan dalam kernel linux. Konfigurasi ulang terhadap kernel kemudian dilanjutkan dengan kompilasi kernel serta menggunakan kernel baru tersebut adalah merupakan keharusan yang dilakukan. Perintah *make menunconfig, make bzImage, make modules* dan *make modules_install* di irektori sumber kernel merupakan langkah-langkah konfigurasi dan kompilasi kernel. Gambar 4 merupakan konfigurasi kernel Linux pada bagian dukungan *filesystem.*



Gambar 4 Dukungan Kernel Linux 2.6.7 pada Beberapa *Filesystem*

1.5 Tipe Filesystem

Filesystem memiliki beberapa tipe yakni *filesystem* dengan mode *journalling*dan tanpa dengan *journalling*. Secara umum *filesystem* dengan tipe *journalling* adalah *filesystem* dengan kemampuan tambahan yakni lebih cepat melakukan *recovery* jika terjadi inkosistensi yang terjadi karena (pada umumnya) tidak dimatikannya komputer dengan prosedur yang benar. Di GNU/Linux terdapat berberapa *filesystem* dengan tipe *journalling* misalnya XFS, JFS, EXT3 dan ReiserFS dan tipe non journal seperti EXT2 dan VFAT. File system melakukan *update* informasi struktural (*metada*) dengan metode *synchronous write*. Setiap *metadata* dalam melakukan *update*dapat membutuhkan penulisan yang terpisah. Jika terjadi *crash* pada waktu penulisan tersebut maka *metadata* dapat dalam keadaan tidak konsisten. Setelah poses

booting setelah terjadi *crash* maka utilitas *filesystem*akan melakukan pengecekan terhadap *metadata* dan memperbaikinya. Proses perbaikan ini dapat memakan waktu yang lama jika terjadi pada *filesystem* dengan kapasitas besar. *Filesystem* dengan tipe *journaling* menggunakan pendekatan yang berbeda dalam melakukan *update metada*. Sebelum informasi diperbaharui maka informasi tersebut disimpan dulu dalam *log* atau *journal* sebelum ditulis sebenarnya dalam *metada*. Jika terjadi *crash* maka *log* yang tersimpan akan digunakan untuk melakukan perbaikan *filesystem*. Perbaikan *filesystem* yang mengalami *crash* menggunakan *journaling* akan lebih cepat daripada *non-journaling*. Penulisan *filesystem*tipe *journalling* juga akan lebih cepat karena *update* dilakukan pada *journal.*

1.6 Metode Akses Filesystem

Informasi yang disimpan dalam suatu *file* jika diperlukan maka *file* tersebut harus diakses dan dibaca ke dalam memori komputer. Beberapa *filesystem*menerapakan metode yang berbeda dalam menulis maupun membaca suatu *file.* Metode akses tersebut antara lain adalah *sequential access* dan *direct access*

1.7 Konfigurasi Kernel Linux

Penelitian ini akan membandingkan unjuk kerja beberapa *filesystem*di GNU/Linux yakni VFAT (FAT32), EXT2, EXT3, XFS, JFS dan ReiserFS. Konfigurasi ulang kernel terhadap *filesystem* supaya mendukung ketiga *filesystem*tersebut dilakukan dengan cara :

1. Mengambil kode sumber kernel Linux, dapat diambil melalui situs web `www.kernel.org`, penulis menggunakan kernel versi 2.6.7
2. Mengekstrak kode sumber dengan perintah `# tar -zxvf linux-2.6.7.tar.gz` pada direktori `/usr/src` yang akan menghasilkan direktori `linux-2.6.7`.
3. Membuat *soft link* dengan nama linux yang menunjuk pada direktori linux-2.6.7 dengan perintah `ln -s linux-2.6.7 linux`.
4. Berpindah pada direktori tempat kernel Linux yang telah diekstrak `# cd /usr/src/linux`
5. Mengkonfigurasi kernel dengan perintah : `gmake config` atau `gmake menuconfig` atau `gmake xconfig`. Pada konfigurasi server kernel memerlukan dukungan pada *filesystem* VFAT, EXT2, EXT3, XFS, JFS dan ReiserFS. Dukungan *filesystem* pada server dapat ditanam langsung di kernel ataupun berupa modul.
6. Mengkompile kernel yang telah dikonfigurasi dengan perintah :
   - `# gmake all`
   - `# gmake modules_install`

Setelah kernel dikompile ulang dan digunakan dengan melakukan booting ulang maka dukungan *filesystem* tersebut dapat dilihat pada direktori `/proc/filesystem`. Perintah `cat /proc/filesystem` dapat dijalankan untuk melihatnya.

```
root@server:~# cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   binfmt_misc
nodev   usbfs
nodev   usbdevfs
nodev   futexfs
nodev   tmpfs
nodev   pipefs
nodev   eventpollfs
nodev   devpts
        reiserfs
        ext3
        ext2
nodev   ramfs
        msdos
        vfat
```

```
        iso9660
nodev   nfs
nodev   nfsd
nodev   smbfs
        ntfs
nodev   autofs
        udf
nodev   mqueue
nodev   rpc_pipefs
```

1.8 Partisi di Linux

Sebuah partisi dari Hardisk Seagate sebesar 10G dipersiapkan untuk digunakan dalam penelitian ini. Setelah partisi dibuat pada awal instalasi sistem operasi maka pembagian partisi tersebut dapat dilihat di file `/proc/partitions`, dengan program `cat` dapat dilihat sebagai berikut,

```
root@server:~# cat /proc/partitions
major minor  #blocks  name
   3     0   39082680 hda
   3     1    9767488 hda1
   3     2    9775552 hda2
   3     3          1 hda3
   3     5    9775552 hda5
   3     6    9285569 hda6
   3     7     473917 hda7
```

Dari hasil di atas maka partisi pada hda5 akan digunakan sebagai media penelitian yang besarnya 9775552 block atau sekitar 10GB.

1.9 Keadaan Sistem

Penelitian dijalankan pada mode tunggal (*single mode*), pada mode ini hanya sedikit proses yang dijalankan oleh sistem. Proses yang ada dapat dilihat dengan perintah top seperti terlihat pada gambar berikut,

```
 20:48:39  up  3:52,  4 users,  load average: 0.08, 0.43, 0.28
18 processes: 17 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:   4.2% user   0.5% system   0.0% nice   1.7% iowait  93.4% idle
Mem:   255500k av,   18252k used,  237248k free,       0k shrd,    4208k buff
         5920k active,              2680k inactive
Swap:  473908k av,      24k used,  473884k free                    3064k cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME CPU COMMAND
    1 root      16   0   496  148   464 S     0.0  0.0   0:00   0 init
    2 root      34  19     0    0     0 SWN   0.0  0.0   0:00   0 ksoftirqd/0
    3 root       5 -10     0    0     0 SW<   0.0  0.0   0:00   0 events/0
    4 root      15 -10     0    0     0 SW<   0.0  0.0   0:00   0 khelper
    5 root       5 -10     0    0     0 SW<   0.0  0.0   0:00   0 kblockd/0
   42 root      13 -10     0    0     0 SW<   0.0  0.0   0:00   0 aio/0
   41 root      15   0     0    0     0 SW    0.0  0.0   0:01   0 kswapd0
  163 root       5 -10     0    0     0 SW<   0.0  0.0   0:00   0 reiserfs/0
 1054 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 kapmd
 2737 root      15   0  2932 1804  2084 S     0.0  0.7   0:01   0 bash
 2739 root      16   0  1496  440  1336 S     0.0  0.1   0:00   0 agetty
 2740 root      16   0  1496  440  1336 S     0.0  0.1   0:00   0 agetty
 2741 root      16   0  1496  440  1336 S     0.0  0.1   0:00   0 agetty
 2742 root      16   0  1496  440  1336 S     0.0  0.1   0:00   0 agetty
 2743 root      16   0  1496  440  1336 S     0.0  0.1   0:00   0 agetty
 3108 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 pdflush
 3109 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 pdflush
 3127 root      17   0  2012  992  1784 R     0.0  0.3   0:00   0 top
```

Gambar 5 Proses yang berjalan pada single mode

1.10 Membuat filesystem

Pembuatan *filesystem di* GNU/Linux hanya dapat dijalankan oleh *root* berikut perintah-perintah yang digunakan :

1. Membuat *filesystem* Ext2

```
root@server:~# mkfs.ext2 /dev/hda5
mke2fs 1.34 (25-Jul-2003)
Filesystem label=
```

```
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
1224000 inodes, 2443888 blocks
122194 blocks (5.00%) reserved for the super user
First data block=0
75 block groups
32768 blocks per group, 32768 fragments per group
16320 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

2. Membuat *filesystem* FAT32

```
root@server:~# mkdosfs -F32  /dev/hda5
mkdosfs 2.8 (28 Feb 2001)
```

3. Membuat *filesystem* Ext3

```
root@server:~# mkfs.ext3 /dev/hda5
mke2fs 1.34 (25-Jul-2003)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
1224000 inodes, 2443888 blocks
122194 blocks (5.00%) reserved for the super user
First data block=0
75 block groups
32768 blocks per group, 32768 fragments per group
16320 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 20 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

4. Membuat *filesystem* Reisefs

```
root@server:~# mkreiserfs /dev/hda5
mkreiserfs 3.6.11 (2003 www.namesys.com)

A pair of credits:
Oleg Drokin was the debugger for  V3 during most of the time that  V4 was under
development,  and was quite  skilled and fast at it.  He wrote  the large write
optimization of V3.

Hans Reiser was the project initiator,  source of all funding for the first 5.5
years. He is the architect and official maintainer.

Guessing about desired format.. Kernel 2.6.7 is running.
Format 3.6 with standard journal
Count of blocks on the device: 2443888
Number of blocks consumed by mkreiserfs formatting process: 8286
Blocksize: 4096
Hash function used to sort names: "r5"
Journal Size 8193 blocks (first block 18)
Journal Max transaction length 1024
inode generation number: 0
UUID: 0bc8d7f9-18f2-4935-9ea4-44e20a5864cf
ATTENTION: YOU SHOULD REBOOT AFTER FDISK!
        ALL DATA WILL BE LOST ON '/dev/hda5'!
Continue (y/n):y
Initializing journal - 0%....20%....40%....60%....80%....100%
Syncing..ok

Tell your friends to use a kernel based on 2.4.18 or later, and especially not a
kernel based on 2.4.9, when you use reiserFS. Have fun.

ReiserFS is successfully created on /dev/hda5.
```

5. Membuat *filesystem* XFS

```
root@server:~# mkfs.xfs -f /dev/hda5
meta-data=/dev/hda5       isize=256    agcount=10, agsize=262144 blks
         =                sectsz=512
data     =                bsize=4096   blocks=2443888, imaxpct=25
         =                sunit=0      swidth=0 blks, unwritten=1
naming   =version 2       bsize=4096
log      =internal log    bsize=4096   blocks=2560, version=1
         =                sectsz=512   sunit=0 blks
realtime =none            extsz=65536  blocks=0, rtextents=0
```

6. Membuat filesystem JFS

```
mkfs.jfs version 1.1.3, 05-Sep-2003
Warning!  All data on device /dev/hda5 will be lost!

Continue? (Y/N) y
   \

Format completed successfully.

9775552 kilobytes total disk space.
```

7. Benchmarking filesystem

Pengujian unjuk kerja *filesystem*dilakukan dengan dua cara yakni :

1. Menggunakan program "umum" seperti touch untuk membuat file, mkdir untuk membuat direktori, cp untuk mengkopi, rm untuk menghapus.
2. Menggunakan program benchmark yakni Bonie++

1.11 Hasil Penelitian dan Pembahasan

Penelitian dilakukan dengan dua cara yaitu dengan menggunakan program umum dan program *benchmarking*.

1.11.1 Penelitian Pertama

Penelitian pertama adalah dengan menggunakan perintah umum yang sering digunakan dalam operasi sistem operasi. Adapun perintah yang digunakan dalam penelitian pertama ini adalah:

1. Membuat 10.000 file dengan perintah `touch` dan menghapusnya
2. Membuat sebuah file dengan ukuran 1GB dengan perintah `dd` dan menghapusnya
3. Membuat 10.000 direktori dengan perintah `mkdir` dan menghapusnya
4. Mengkopi source kode kernel Linux dari direktori lain.
5. Mengekstrak kernel source tersebut
6. Menghapus kernel source tersebut

Jumlah seluruh perintah yang digunakan pada penelitian ini ada 10 perintah yang dibagi dalam beberapa script yang digunakan untuk mempermudah penelitian yakni :

1. `buatdir.sh`

   Script ini berisi perintah untuk membuat 10.000 direktori dengan perintah mkdir, adapun isi dari script ini adalah :

```
#!/bin/bash
# This File Part of benchmark.sh script
# by Ema Utami (ema@nrar.net, ema@infohalal.com)

for ((x=0;x<10000;x++))
{
  mkdir "test_$x";
}
```

2. `buafile.sh`

   Script ini berisi perintah untuk membuat 10.000 file dengan perintah `touch`, adapun isi dari script ini adalah :

```
#!/bin/bash
# This File Part of benchmark.sh script
# by Ema Utami (ema@nrar.net, ema@infohalal.com)

for ((x=0;x<10000;x++))
{
```

```
    touch "test_$x";
        }
```

3. ss.sh

Script ini berisi perintah untuk melakukan *flushing* terhadap *buffer* suatu filesystem, adapun isi dari script ini adalah :

```
#!/bin/bash
#This File Part of benchmark.sh script
#by Ema Utami (ema@nrar.net, ema@infohalal.com)

echo "Syncronizing File System Waiting and sleeping for a while"
sync; # Flushing FileSystem
sleep 10; # Give CPU take e breath
```

4. benchmark.sh

Script ini berisi script utama yang merupakan kumpulan dari script-script di atas dan perintah bantu lainya, misalnya dd, rm dan cp. Adapun isi dari script ini adalah :

```
#!/bin/bash
## V 1.0
## benchmark.sh is script file for Bencmarking File System using "ordinary"
command
## buatfile.sh is script file for creating file using touch command
## buatdir.sh is script file for creating directory using mkdir command
## ss.sh is script to syncronizing file system and sleeping
## by Ema Utami (ema@nrar.net, ema@infohalal.com)

export TIMEFORMAT="%P%% CPU %E Second"
k_VERSION=2.6.7


# 1. Creating tiny files using touch command
echo " Creating 10.000 files ...."
time ./buatfile.sh

./ss.sh

# 2. Removing those tiny files using rm command
echo " Removing 10.000 file ...."
time rm test_*;

./ss.sh

# 3. Creating one big file using dd command
echo "Creating one big file using dd ............."
/bin/dd if=/dev/zero of=/mnt/penelitian/hugefile bs=100M count=10

./ss.sh

# 4. Deleting one big file using rm command
echo " Deleting big file ...."
time rm hugefile;

./ss.sh

# 5. Making 10.000 Directory using mkdir command
echo " Creating 10.000 directory ...."
time ./buatdir.sh

./ss.sh

# 6. Deleting 10.000 Directory using rm -rf command
echo " Removing 10.000 directory ...."
time rm -rf test_*;

./ss.sh

# 7. Copy Kernel Source tarbal from other directory
echo " Copying Linux kernel source from /usr/src/ ..." # Make sure you have it
!!
time cp /usr/src/linux-$K_VERSION.tar.gz . # Please change the version and dir
acording u'r kernel

./ss.sh

# 8. Untar Kernel Source using tar command
echo " Untar Kernel source ..."
time tar -zxvf linux-$K_VERSION.tar.gz

./ss.sh
```

```
# 9. Removing Kernel Source Tar bal
echo " Removing Linux kernel tree ...."
time rm -f linux-$K_VERSION.tar.gz;

./ss.sh


# 10. Removing Kernel Source tree
echo " Removing Linux kernel tree ...."
time rm -f linux-$K_VERSION;

./ss.sh
```

Pada script di atas setiap perintah diawali dengan perintah `time`. Perintah ini digunakan untuk mencatat waktu penyelesaian suatu perintah dan mencatat besarnya prosentase penggunaan prosesor.

5. Menjalankan script

Sebelum menjalankan script maka partisi di mount dulu kemudian script dikopikan pada partisi tersebut dan langkah terakhir adalah menjalankan script `benchmark.sh`, sebagai contoh langkah ini adalah seperti berikut,

```
root@server:~# mount /dev/hda5 /mnt/penelitian/
root@server:~# cd /mnt/penelitian/
root@server:/mnt/penelitian# cp /home/ema/penelitian/script/*.sh .
root@server:/mnt/penelitian# ls -l
total 16
-rwxr--r--    1 root     root         1771 Aug 10 19:30 benchmark.sh*
-rwxr--r--    1 root     root          145 Aug 10 19:30 buatdir.sh*
-rwxr--r--    1 root     root          145 Aug 10 19:30 buatfile.sh*
-rwxr-xr-x    1 root     root          232 Aug 10 19:30 ss.sh*
root@server:/mnt/penelitian#./benchmark.sh
1. Creating 10.000 files ....
99.46% CPU 5.580 Second
Syncronizing File System and sleeping for a whilebenchmark.sh
 2. Removing 10.000 file ....
99.87% CPU 0.390 Second
Syncronizing File System and sleeping for a while
3. Creating one big file using dd ............
10+0 records in
10+0 records out
22.01% CPU 22.346 Second
Syncronizing File System and sleeping for a while
 4. Deleting big file ....
81.03% CPU 0.247 Second
Syncronizing File System and sleeping for a while
 5. Creating 10.00 directory ....
94.57% CPU 6.048 Second
Syncronizing File System and sleeping for a while
 6. Removing 10.000 directory ....
93.83% CPU 0.746 Second
Syncronizing File System and sleeping for a while
 7. Copying Linux kernel source from /usr/src/ ...
4.47% CPU 5.589 Second
Syncronizing File System and sleeping for a while
 8. Untar Kernel source ...
26.87% CPU 18.120 Second
Syncronizing File System and sleeping for a while
 9. Removing Linux kernel tree ....
22.20% CPU 0.045 Second
Syncronizing File System and sleeping for a while
 10. Removing Linux kernel tree ....
79.68% CPU 1.042 Second
Syncronizing File System and sleeping for a while
```

6. Hasil Program

Script `benchmark.sh` dijalankan pada partisi `/mnt/penelitian` yang telah diberikan *filesystem.* Pembuatan *filesystem* secara bergatian kemudian dilanjutkan dengan menjalankan script tersebut. Seluruh percobaan dilakukan sebanyak tiga kali dan kemudian diambil nilai rata-ratanya.

1. Filesystem FAT32/ VFAT

Hasil dari ujicoba pada FAT32/VFAT tertampil pada tabel 2 berikut,

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*
1	Membuat 10.000 file	99.9	144.24
2	Menghapus 10.000 file	99.93	18.76
3	Membuat 1GB file	12.81	26.11
4	Menghapus 1GB file	63.05	0.38
5	Membuat 10.000 direktori	99.79	164.95
6	Menghapus 10.000 direktori	99.95	20.86
7	Mengkopi kernel source (tar.gz)	4.08	6.51
8	Mengekstrak kernel source	19.06	25.2
9	Menghapus hasil ekstrak	93.32	9.87
10	Menghapus kernel source (tar.gz)	100	0.03

Tabel 2 Tabel Hasil Ujicoba pada FAT32

Pada percobaan 8 (melakukan ekstrak terhadap kode sumber Linux) program `tar` akan melakukan ekstrak terhadap file kompresi dan membutuhkan pengubahan hak permisi. FAT32 tidak mempunyai fasilitas untuk melakukan perubahan hak permisi maka opsi program tar ditambahkan opsi `-no-same-owner`

2. Filesystem EXT2

Hasil dari ujicoba pada ETX2 tertampil pada tabel 3 berikut,

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*
1	Membuat 10.000 file	99.42	9.91
2	Menghapus 10.000 file	99.88	0.12
3	Membuat 1GB file	10.93	20.89
4	Menghapus 1GB file	9.35	0.55
5	Membuat 10.000 direktori	90.49	7.76
6	Menghapus 10.000 direktori	91.26	0.33
7	Mengkopi kernel source (tar.gz)	3.79	5.15
8	Mengekstrak kernel source	38.72	8.79
9	Menghapus hasil ekstrak	15.53	0.06
10	Menghapus kernel source (tar.gz)	15.65	1.24

Tabel 3 Tabel Hasil Ujicoba pada EXT2

Dari tabel 3 terlihat waktu paling lama yang digunakan pada EXT2 adalah pada saat melakukan eksekusi pembuatan 1GB file dengan menggunakan perintah dd (20.89 detik) sedangkan waktu tercepat yang diselesaikan adalah pada saat menghapus ekstrak kode sumber Linux (0.06 detik). Penggunan sumberdaya prosesor yang paling besar adalah pada saat menghapus 10.000 file (99.88 %) sedangkan penggunaan paling kecil adalah pada saat melakukan operasi *copy* kernel yakni 3.79 %.

3. Filesystem ReiserFS

Hasil dari ujicoba pada ReiserFS tertampil pada tabel 4 berikut,

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*
1	Membuat 10.000 file	99.51	6.17
2	Menghapus 10.000 file	99.96	0.39
3	Membuat 1GB file	11.5	24.24
4	Menghapus 1GB file	89.4	0.26
5	Membuat 10.000 direktori	99.39	6.07
6	Menghapus 10.000 direktori	95.18	0.86
7	Mengkopi kernel source (tar.gz)	3.55	5.55
8	Mengekstrak kernel source	33.75	16.24
9	Menghapus hasil ekstrak	47.2	0.04
10	Menghapus kernel source (tar.gz)	95.33	0.98

Tabel 4 Tabel Hasil Ujicoba pada ReiserFS

Dari tabel 4 terlihat waktu paling lama yang digunakan pada ReiserFS adalah pada saat melakukan eksekusi pembuatan 1GB file dengan menggunakan perintah `dd` (24.24 detik) sedangkan waktu tercepat yang diselesaikan adalah pada saat menghapus ekstrak kode sumber Linux (0.06 detik). Penggunan sumberdaya prosesor yang paling besar adalah pada saat menghapus 10.000 file (99.96 %) sedangkan penggunaan paling kecil adalah pada saat melakukan operasi *copy* kernel yakni 3.55 %.

4. Filesystem EXT3

Hasil dari ujicoba pada EXT3 tertampil pada tabel 5 berikut,

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*
1	Membuat 10.000 file	99.47	14.23
2	Menghapus 10.000 file	99.95	0.25
3	Membuat 1GB file	19.81	22.22
4	Menghapus 1GB file	67.28	0.32
5	Membuat 10.000 direktori	30.54	50.66
6	Menghapus 10.000 direktori	92.36	0.53
7	Mengkopi kernel source (tar.gz)	5.14	5.68
8	Mengekstrak kernel source	28.97	13.71
9	Menghapus hasil ekstrak	20.78	0.07
10	Menghapus kernel source (tar.gz)	81.33	0.45

Tabel 5 Tabel Hasil Ujicoba pada EXT3

Dari tabel 5 terlihat waktu paling lama yang digunakan pada EXT3 adalah pada saat melakukan eksekusi pembuatan 10.000 direktori dengan menggunakan perintah mkdir (50.66 detik) sedangkan waktu tercepat yang diselesaikan adalah pada saat menghapus ekstrak kode sumber Linux (0.07 detik). Penggunan sumberdaya prosesor yang paling besar adalah pada saat menghapus 10.000 file (99.95 %) sedangkan penggunaan paling kecil adalah pada saat melakukan operasi *copy* kernel yakni 5.14 %.

5. Filesystem XFS

Hasil dari ujicoba pada XFS tertampil pada tabel 6 berikut,

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*
1	Membuat 10.000 file	95.33	7.86
2	Menghapus 10.000 file	24.44	2.36
3	Membuat 1GB file	12.4	19.32
4	Menghapus 1GB file	63.1	0.09
5	Membuat 10.000 direktori	70.68	10.5
6	Menghapus 10.000 direktori	25.94	2.77
7	Mengkopi kernel source (tar.gz)	3.81	5.5
8	Mengekstrak kernel source	20.77	21.72
9	Menghapus hasil ekstrak	100	0.01
10	Menghapus kernel source (tar.gz)	22.04	5.08

Tabel 6 Tabel Hasil Ujicoba pada XFS

Dari tabel 6 terlihat waktu paling lama yang digunakan pada XFS adalah pada saat melakukan ekstrak source kode linux dengan menggunakan perintah tar (21.72 detik) sedangkan waktu tercepat yang diselesaikan adalah pada saat menghapus ekstrak kode sumber Linux (0.01 detik). Penggunan sumberdaya prosesor yang paling besar adalah pada saat menghapus hasil ekstrak source kode (100 %) sedangkan penggunaan paling kecil adalah pada saat melakukan operasi *copy* kernel yakni 3.81 %.

6. Filesystem JFS

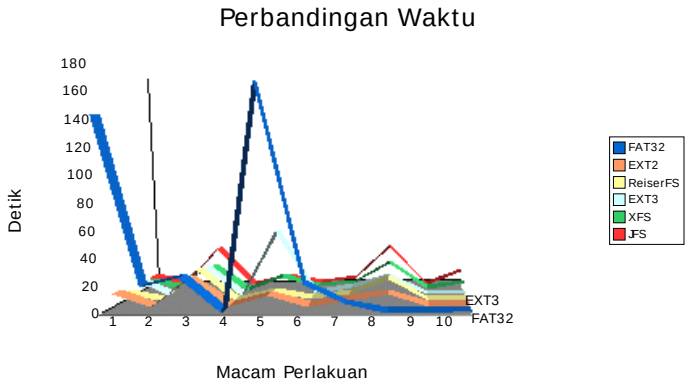
Hasil dari ujicoba pada JFS tertampil pada tabel 7 berikut,

No	*Perlakuan*	*Penggunaan CPU (%)*	*Waktu(detik)*
1	Membuat 10.000 file	91.69	7.08
2	Menghapus 10.000 file	27.08	1.16
3	Membuat 1GB file	10.71	31.66
4	Menghapus 1GB file	32.21	0.1
5	Membuat 10.000 direktori	89.43	7.3
6	Menghapus 10.000 direktori	26.66	1.64
7	Mengkopi kernel source (tar.gz)	4.03	5.38
8	Mengekstrak kernel source	12.05	31.62
9	Menghapus hasil ekstrak	11.37	0.02
10	Menghapus kernel source (tar.gz)	2.73	10.32

Tabel 7 Tabel Hasil Ujicoba pada JFS

Dari tabel 7 terlihat waktu paling lama yang digunakan pada JFS adalah pada saat melakukan pembnuatan 1GB file dengan menggunakan perintah `dd` (31.66 detik) sedangkan waktu tercepat yang diselesaikan adalah pada saat menghapus ekstrak kode sumber Linux (0.02 detik). Penggunan sumberdaya prosesor yang paling besar adalah pada saat membuat 10.000 direktori (89.43 %) sedangkan penggunaan paling kecil adalah pada saat melakukan penghapusan kernel source yakni 2.73 %.

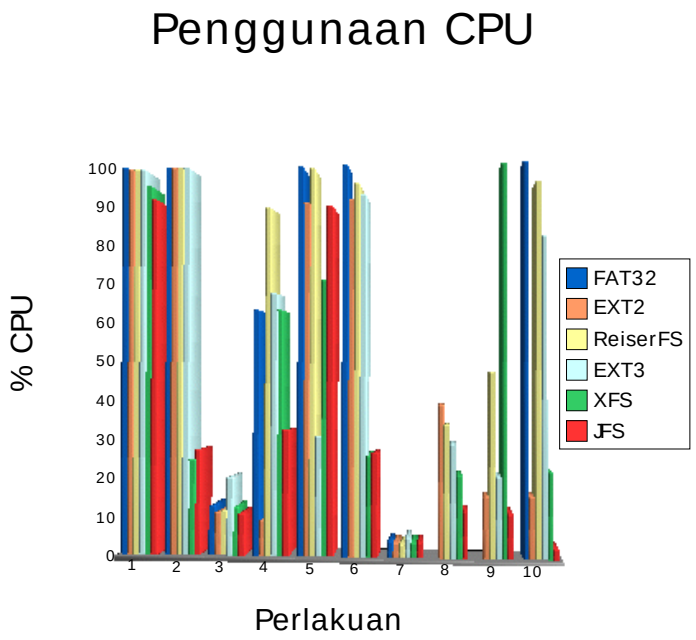
Grafik perbandingan waktu antara semua filesystem tertampil pada gambar 6



Gambar 6 Grafik Perbandingan Waktu Akses Filesystem

Grafik perbandingan penggunaan Prosesor antara semua filesystem tertampil pada gambar 7

Gambar 7 Grafik Perbandingan Penggunaan CPU



1.11.2 Penelitian Metode Kedua

Pada penelitian kedua digunakan program *benchmarking* yakni Bonnie++ versi 1.03a. Setelah kode sumber program ini diekstrak dan dikompile maka program ini dijalankan dengan perintah :

```
bonnie++ -d /mnt/penelitian -m Server -r 256 -s 512 -u root
```

Direktori `/mnt/penelitian` merupakan direktori target yang akan diformat dengan

filesystem yang berbeda-beda. Server adalah nama komputer 256 adalah besarnya RAM yang digunakan dan 512 adalah besarnya file yang digunakan untuk melakukan benchmarking. Program ini akan melakukan IO test sebagai berikut:

1. Sequential Output
   a. Per-Character.
   File ditulis dengan menggunakan perintah putc() yang merupakan salah satu fungsi dalam stdio.
   b. Block.
   File ditulis dengan menggunakan perintah write(2)
   c. Rewrite.
   File dibaca dengan perintah read(2) kemudian ditulis kembali dengan menggunakan perintah write()
2. Sequential Input
   a. Per-Character.
   File dibaca dengan menggunakan perintah getc() yang merupakan salah satu fungsi dalam stdio.
   b. Block
   File dibaca dengan perintah read(2)
3. Random Seeks
   Test ini akan menjalankan proses SeekProcCount secara pararel (default 3), menjalankan total 8000 proses lseek() pada suatu lokasi file yang ditentukan dengan perintah random() pada sistem bsd systems atau drand48() pada sysV
4. Sequential Create/Read/Delete
   Pada test ini akan dilakukan penulisan, pembacaan dan penghapusan file secara sequential dengan menggunakan 16 file.
5. Random Create/Read/Delete
   Pada test ini akan dilakukan penulisan, pembacaan dan penghapusan file secara random dengan menggunakan 16 file.

Pada test ini terbagi menjadi 6 test pertama dan 6 test kedua. Test pertama adalah menggunakan 512 Mbyte kapasitas hardisk pada partisi yang di uji, sedangkan test berikutnya adalah menggunakan 16 file yang merupakan simulasi pada penulisan banyak file.

Hasil Penelitian Kedua

1. Filesystem VFAT

Hasil pada filesystem VFAT adalah seperti tertampil pada tabel berikut,

No	*Perlakuan*	*K/Sec*	*% CPU*
1	Sequential Output Per Char	13495	76
2	Sequential Output Per Block	41310	12
3	Sequential Output rewrite	22175	11
4	Sequential Input Per Char	15008	74
5	Sequential Input Per Block	40417	14
6	Random Seeks	201.1	13
		/Sec	*% CPU*
7	Sequential Create	59	99
8	Sequential Read	112	99
9	Sequential Delete	709	99
10	Random Create	84	99

No	*Perlakuan*	*K/Sec*	*% CPU*
11	Random Read	112	99
12	Random Delete	193	99

Tabel 8 Test Bonnie++ pada VFAT

Dari tabel 8 terlihat bahwa pada 6 test pertama VFAT mampu melakukan pembacaan paling cepat per block dengan besar 40417 Kbyte per detik, sedangkan paling lama adalah saat pencarian secara random yang hanya mampu membaca sebanyak 201.1 Kbyte per detik. PenggunaanCPU terbesar pada test pertama adalah pada saat penulisan per char yakni 76%. Sedangkan paling kecil adalah pada saat melakukan penulisan ulang yakni 11%. Pada 6 test kedua penghapusan secara sequential merupakan test yang paling unggul pada VFAT yakni mampu melakukan penghapusan sebanyak 709 file per detik sedangkan paling kecil adalah saat melakukan pembuatan file secara sequential yakni 59 file per detik. Penggunaan CPU pada test kedua ini sama yakni 99%.

2. Filesystem EXT2

Hasil pada filesystem EXT2 adalah seperti tertampil pada tabel berikut,

No	*Perlakuan*	*K/Sec*	*% CPU*
1	Sequential Output Per Char	19785	99
2	Sequential Output Per Block	51725	11
3	Sequential Output rewrite	21202	6
4	Sequential Input Per Char	18981	88
5	Sequential Input Per Block	42921	7
6	Random Seeks	235.2	0
		/Sec	*% CPU*
7	Sequential Create	3994	99
8	Sequential Read	+++++	+++
9	Sequential delete	+++++	+++
10	Random Create	3995	99
11	Random Read	+++++	+++
12	Random Delete	12356	100

Tabel 9 Test Bonnie++ pada EXT2

Dari tabel 9 terlihat bahwa pada 6 test pertama EXT2 mampu melakukan penulisan paling cepat per block dengan besar 51725 Kbyte per detik, sedangkan paling lama adalah saat pencarian secara random yang hanya mampu membaca sebanyak 235.2 Kbyte per detik. PenggunaanCPU terbesar pada test pertama adalah pada saat penulisan per char yakni 99%. Sedangkan paling kecil adalah pada saat melakukan pencarian secara random yakni 0%. Pada 6 test kedua beberapa test tidak tercatat karena terlalu besar file yang dpat dibuat, pada test ini hasil terburuk EXT2 adalah pada saat melakukan pembuatan file secara sequential yakni 3994 file per detik.

3. Filesystem EXT3

Hasil pada filesystem EXT3 adalah seperti tertampil pada tabel berikut,

No	*Perlakuan*	*K/Sec*	*% CPU*
1	Sequential Output Per Char	17271	98
2	Sequential Output Per Block	53577	21
3	Sequential Output rewrite	19540	6

No	*Perlakuan*	*K/Sec*	*% CPU*
4	Sequential Input Per Char	16813	77
5	Sequential Input Per Block	43388	7
6	Random Seeks	224.5	0
		/Sec	*% CPU*
7	Sequential Create	2175	99
8	Sequential Read	+++++	+++
9	Sequential delete	+++++	+++
10	Random Create	2192	98
11	Random Read	+++++	+++
12	Random Delete	8387	100

Tabel 10 Test Bonnie++ pada EXT3

Dari tabel 10 terlihat bahwa pada 6 test pertama EXT3 mampu melakukan penulisan paling cepat per block dengan besar 53577 Kbyte per detik, sedangkan paling lama adalah saat pencarian secara random yang hanya mampu membaca sebanyak 224.5 Kbyte per detik. PenggunaanCPU terbesar pada test pertama adalah pada saat penulisan per char yakni 98%. Sedangkan paling kecil adalah pada saat melakukan pencarian secara random yakni 0%. Pada 6 test kedua beberapa test tidak tercatat karena terlalu besar file yang dpat dibuat, pada test ini hasil terburuk EXT3 adalah pada saat melakukan pembuatan file secara sequential yakni 2175 file per detik.

4. Filesystem ReiserFS

Hasil pada filesystem ReiserFS adalah seperti tertampil pada tabel berikut,

No	*Perlakuan*	*K/Sec*	*% CPU*
1	Sequential Output Per Char	18132	99
2	Sequential Output Per Block	58232	19
3	Sequential Output rewrite	19627	5
4	Sequential Input Per Char	19031	87
5	Sequential Input Per Block	40468	7
6	Random Seeks	230.4	0
		/Sec	*% CPU*
7	Sequential Create	28660	98
8	Sequential Read	+++++	+++
9	Sequential delete	24511	99
10	Random Create	28537	99
11	Random Read	+++++	+++
12	Random Delete	22179	100

Tabel 11 Test Bonnie++ pada ReiserFS

Dari tabel 11 terlihat bahwa pada 6 test pertama ReiserFS mampu melakukan penulisan paling cepat per block dengan besar 58232 Kbyte per detik, sedangkan paling lama adalah saat pencarian secara random yang hanya mampu membaca sebanyak 230.4 Kbyte per detik. PenggunaanCPU terbesar pada test pertama adalah pada saat penulisan per char yakni 99%. Sedangkan paling kecil adalah pada saat melakukan pencarian secara random yakni 0%. Pada 6 test kedua beberapa test tidak tercatat karena terlalu besar file yang dpat dibuat, pada test ini hasil terburuk ReiserFS adalah pada saat melakukan penghapusan file secara

random yakni 22179 file per detik.

5. Filesystem XFS

Hasil pada filesystem XFS adalah seperti tertampil pada tabel berikut,

No	*Perlakuan*	*K/Sec*	*% CPU*
1	Sequential Output Per Char	19916	98
2	Sequential Output Per Block	52079	12
3	Sequential Output rewrite	21024	6
4	Sequential Input Per Char	20542	95
5	Sequential Input Per Block	44257	7
6	Random Seeks	220.4	0
		/Sec	*% CPU*
7	Sequential Create	3915	24
8	Sequential Read	+++++	+++
9	Sequential delete	3366	22
10	Random Create	3639	27
11	Random Read	+++++	+++
12	Random Delete	892	5

Tabel 12 Test Bonnie++ pada XFS

Dari tabel 12 terlihat bahwa pada 6 test pertama XFS mampu melakukan penulisan paling cepat per block dengan besar 52079 Kbyte per detik, sedangkan paling lama adalah saat pencarian secara random yang hanya mampu membaca sebanyak 220.4 Kbyte per detik. PenggunaanCPU terbesar pada test pertama adalah pada saat penulisan per char yakni 98%. Sedangkan paling kecil adalah pada saat melakukan pencarian secara random yakni 0%. Pada 6 test kedua beberapa test tidak tercatat karena terlalu besar file yang dpat dibuat, pada test ini hasil terburuk XFS adalah pada saat melakukan penghapusan file secara random yakni 892 file per detik.

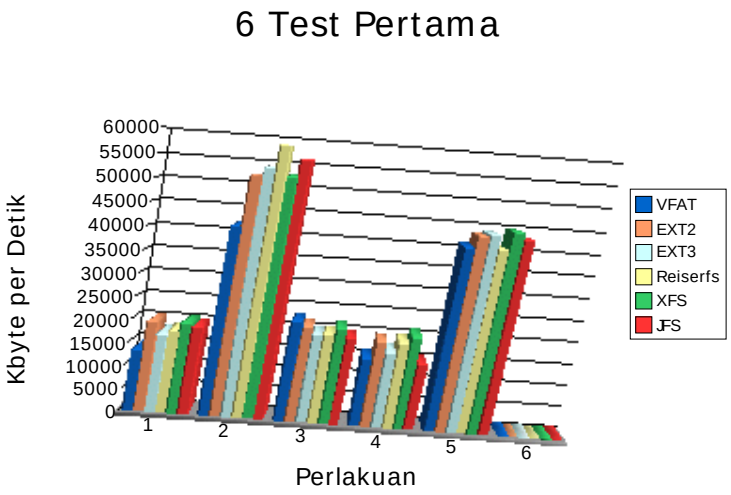
6. Filesystem JFS

Hasil pada filesystem JFS adalah seperti tertampil pada tabel berikut,

No	*Perlakuan*	*K/Sec*	*% CPU*
1	Sequential Output Per Char	19309	99
2	Sequential Output Per Block	55684	13
3	Sequential Output rewrite	19213	4
4	Sequential Input Per Char	14683	67
5	Sequential Input Per Block	42760	7
6	Random Seeks	239.1	0
		/Sec	*% CPU*
7	Sequential Create	7813	13
8	Sequential Read	+++++	+++
9	Sequential delete	5615	10
10	Random Create	4455	22
11	Random Read	+++++	+++
12	Random Delete	4512	14

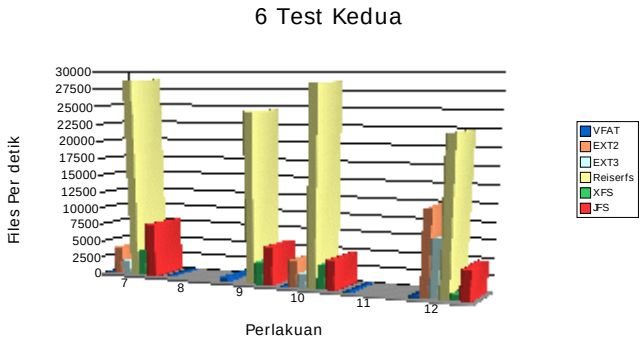
Tabel 13 Test Bonnie++ pada JFS

Dari tabel 13 terlihat bahwa pada 6 test pertama JFS mampu melakukan penulisan paling cepat per block dengan besar 55684 Kbyte per detik, sedangkan paling lama adalah saat pencarian secara random yang hanya mampu membaca sebanyak 239.1 Kbyte per detik. PenggunaanCPU terbesar pada test pertama adalah pada saat penulisan per char yakni 99%. Sedangkan paling kecil adalah pada saat melakukan pencarian secara random yakni 0%. Pada 6 test kedua beberapa test tidak tercatat karena terlalu besar file yang dpat dibuat, pada test ini hasil terburuk XFS adalah pada saat melakukan pembuatan file secara random yakni 4455 file per detik. Grafik perbandingan perlakuan 6 test pertama tertampil pada gambar 8



Gmbar 8 GrafikPerbandingan 6 Test Pertama

Sedangkan grafik perbandingan 6 test kedua tertampil pada gambar 9 berikut,



Gambar 9 GrafikPerbandingan 6 Test Kedua

Penutup

Dari hasil penelitian yang dilakukan dapat disimpulkan bahwa filesystem dengan tipe *journalling* memiliki kecenderungan yang sama dalam berberapa perlakuan. Hasil penelitian menunjukkan juga bahwa filesystem EXT3 dalam beberapa test masih lebih buruk dari EXT2. Jika masalah *recovery* diabaikan maka EXT2 masih dapat diandalkan sebagai *filesystem.* ReiserFS menunjukkan hasil terbaik saat melakukan penulisan banyak file, sehingga filesystem ini menjadi pilihan sebagai untuk digunakan pada program yang membutuhkan banyak penulisan file seperti proxy server dan mail server. Hasilpenelitian juga memperlihatkan bahwa VFAT merupakan Filesystem yang paling buruk. JFS merupakan filesystem yang dapat mengelola penggunaan CPU dengan baik. Semoga tulisan ini bermanfaat bagi pembaca yang membutuhkan informasi mengenai suatu *filesystem* khususnya mengenai karakteristiknya sehingga dapat diambil keputusan untuk menggunakan suatu *filesystem* berdasarkan penggunaan.

Referensi

Muller, G .(1999), *A Visual Introduction to Linux*, Technical Report, Philips Research, Prof Holstlaan 4 (WLO1) 5656 AA Eindhoven The Netherlands

Quinlan, D. & Russel, R. (2001), *Filesystem Hierarchy Standard --version 2.2 final*

Tanembaum, A.S (2001), *Modern Operating Systems*, Prentice-Hall, Inc.

Florido, J.I.S (2001), *Journal File System*, Linux Gazette, Issue 55