VIT®

UNIVERSITY

(Estd. u/s 3 of UGC Act 1956)

CSE-2005 OPERATING SYSTEMS

J COMPONENT REPORT

COMPARITIVE STUDY OF FILE SYSTEMS

D SABARESHWAR[1] - 19BCI0105

PANJAM GOUTHAM REDDY[2] - 19BCI0027

R SENGOSHARAN[3] - 19BCI0059

DEVANSH JAIN[4] - 19BCI0173

ANKITA MANDAL[5] - 19BCE0438

FACULTY - RAJKUMAR S

SLOT - F1

Sabareshwar.d2019@vitstudent.ac.in[1]

Panjamgouthamredd.y2019@vitstudent.ac.in[2]

Sengosharanr.2019@vitstudent.ac.in[3]

Devansh.jain2019@vitstudent.ac.in[4]

Ankita.mandal2019@vitstudent.ac.in[5]

ABSTRACT

Proposed Framework with detailed description of each module

1) Comparative study of file systems in windows.

   In this module, we are going to explore about the different file systems present in windows and
   their detailed explanation.

   We have selected some widely used file systems like NTFS, ReFS, FAT, FAT32 in Windows.

   We made a tabular comparison of all the widely used Windows file systems.

2) Comparative study of file systems in linux.

3) Comparative study of three File allocation methods.

4) Demonstration of Implementation of selected file systems.

KEYWORDS

1) **File systems**: In computing, a file system or filesystem (often abbreviated to fs) controls how data is stored and retrieved. Without a file system, data placed in a storage medium would be one large body of data with no way to tell where one piece of data stops and the next begins.Taking its name from the way paper-based data management system is named, each group of data is called a "file." The structure and logic rules used to manage the groups of data and their names is called a "file system."

2) **FAT file systems**: [1]File Allocation Table (FAT) is a file system developed for personal computers. Originally developed in 1977 for use on floppy disks, it was adapted for use on hard disks and other devices. It is often supported for compatibility reasons by current operating systems for personal computers and many mobile devices and embedded systems, allowing interchange of data between disparate systems[1].

3) **File Allocation Methods:** The allocation method defines how the files are stored in the disk blocks. The direct access nature of the disks gives us the flexibility to implement the files. In many cases, different files or many files are stored on the same disk. The main problem that occurs in the operating system is that how we allocate the spaces to these files so that the utilization of disk is efficient and the quick access to the file is possible.

LITERATURE SURVEY

1) File Systems in Linux and FreeBSD: A Comparative Study

Kuo-pao Yang , Katie Wallace Computer Science and Industrial Technology Department Southeastern Louisiana University

This paper compares file systems in Ubuntu Linux and FreeBSD and then analyzes the best utilization. The generic file systems, Extended File System (EXT2) of Linux and Fast File System (FFS) of FreeBSD operating systems, are evaluated using benchmark tests.Benchmarks using Bonnie++ and Iozone have been performed on Ubuntu8.10 with ext2 filesystem and FreeBSD7.1 with FFS file system and the results shows that Linux performs better than FreeBSD in these tests.

2)Ext4 file system in Linux Environment:Features and Performance Analysis

Borislav Djordjevic, Valentina Timcenko

This paper considers the characteristics and behavior of the modern 64-bit ext4 file system of Linux operating system, kernel version 2.6.This paper also provides the performance comparison of ext4 file system with earlier ext3 and ext2 file systems. The performance is measured using the Postmark benchmarking software that simulates the workload of Internet mail server. We have defined three types of workloads, generally dominated by relatively small objects. Postmark benchmarking software results have shown superiority of ext4 file system compared to its predecessors, ext2 and ext3 file systems.

3) Comparative study of File systems (NTFS, FAT, FAT32, EXT2, EXT3, EXT4)

Akash Bundele1, Prof. Dr. S. E. Yedey2 PG Department of Computer Science & Technology, Hanuman Vyayam Prasarak Mandal, Amravati, Maharashtra

There are several Windows file systems and Linux file system. Each of them have advantages and disadvantages. In this paper the file systems that are taken into consideration for comparison are FAT ,FAT32,NTFS,EXT2,EXT3,EXT4.This paper focuses on the why the filesystems are introduced, what are the limitations covered by the introduced filesystem over current filesystem and what are the features added to new filesystem. Also this paper describes the comparitive analysis of both Windows and Linux filesystems by taking certain parameters like Maximum file size, in which version it is been introduced, Filesystem size etc.

4)XFS:the big storage file system for Linux

XFS is a file system that was designed for computer systems with large number of CPUs and large disk arrays. XFS file system mainly focuses on supporting large files and good I/O performance. XFS has some interesting adminstrative features which are not supported by other Linux file sytems. This paper provides quick overview of XFS features and also explains why Linux needs a different file system that differs from the default file system and also the benefits of the file system that is designed for large storage.

5)Journaling the Linux ext2fs Filesystem

This paper describes a work-in-progress to design and implement a transactional metadata journal for the Linux ext2 file system by considering the problem of recovering file system after a crash. It describes a design which is intended to increase ext2 file systems speed and reliability of crash recovery by adding a transactional journal to the filesystem. The filesystem design outlined in this paper offers some advantages over the existing ext2 file system. This new file system design requires minimum changes in the existing ext2 code base.

6) File Systems for Various Operating Systems: A Review

Isma Irum, Mudassar Raza, Muhammad Sharif Comsats Institute of Information Technology Wah Cantt, Pakistan

Modern personal computers must have an operating system to run programs like application programs. The Microsoft Windows, Mac OS, UNIX and LINUX are the examples of operating system for a personal computer. An operating system is behaved like an interface between the computer hardware and software. The functions of an operating system are processor management, memory management, security, device management, and control over system performance, job accounting, and error detecting aids, coordination between other software and users and file management. Now we will discuss about the one of important function which named as file management. The file system in OS is usually organized into or efficient usage.

7)A Five-Year Study of File-System Metadata

Nitin Agrawal University of Wisconsin, Madison nitina@cs.wisc.edu

William J. Bolosky, John R. Douceur, Jacob R. Lorch Microsoft Research {bolosky,johndo,lorch}@microsoft.com

This project was a longitudinal extension of an earlier study we performed in 1998 , which was an order of magnitude larger than any prior study of file-system metadata. Our earlier study involved a single capture of file-system metadata, and it focused on lateral variation among file systems at a moment in time. By contrast, the present study focuses on longitudinal changes in file systems over a five-year time span.

8)A Research Paper On Comparison Between Windows And Linux: A Survey

Deepa Mewara1,Aditi Jain2

1Research scholar, Dept. of Computer Science & Engineering, JVWU Jaipur
2assistant professor, Dept. of Computer Science & Engineering, JVWU Jaipur

This project was a longitudinal extension of an earlier study we performed in 1998 , which was an order of magnitude larger than any prior study of file-system metadata. Our earlier study involved a single capture of file-system metadata, and it focused on lateral variation among file systems at a moment in time. By contrast, the present study focuses on longitudinal changes in file systems over a five-year time span.

9) A Study of Linux File System Evolution

LANYUE LU, ANDREA C. ARPACI-DUSSEAU, REMZI H. ARPACI-DUSSEAU, and SHAN LU, University of Wisconsin, Madison

Open-sourcelocalfilesystems,suchasLinuxExt4 ,XFS[Sweeney et al. 1996], and Btrfs [Mason 2007; Rodeh et al. 2012], remain a critical component in the world of modern storage. For example, many recent distributed file systems, such as Google GFS [Ghemawat et al. 2003] replicate data objects(and associated metadata)across local file systems. On smartphones, most user data is managed by a local file system; for example, Google Android phones use Ext4 [Kim et al. 2012; Simons 2011] and Apple's iOS devices use HFSX [Morrissey 2010]. Finally, many desktop users still do not backup their data regularly [Jobs et al. 2006; Marshall 2008]; in this case, the local file system clearly plays a critical role as sole manager of user data.

10.A Study of Linux File System Evolution Provided by: Association for Computing Machinery

Open-source local file systems, such as Linux Ext4, XFS, and Btrfs, remain a critical component in the world of modern storage. For example, many recent distributed file systems, such as Google GFS and Hadoop DFS, replicate data objects (and associated metadata) across local file systems. The authors conduct a comprehensive study of file-system code evolution. By analyzing eight years of Linux filesystem changes across 5079 patches, they derive numerous new (and sometimes surprising) insights into the file-system development process; their results should be useful for both the development of file systems themselves as well as the improvement of bug-finding tools.

11)Comparison and Review of Memory Allocation and File Access Techniques and Techniques preferred for Distributed Systems

Muazzam A. Khan, Nauman Nisar, Department of Computer Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology,Islamabad, Pakistan.

Memory Allocation is a process in which operating system manages the Primary Memory and allocates user programs in spaces in the Main Memory. File Access is a process in which the files required to execute are accessed within in the Main Memory and brought to the CPU. Memory Allocation and file access methods play an important role in optimizing the CPU performance and primary memory performance. The paper focuses on the techniques that are applied for memory allocation and file retrieval and comparison of the techniques which performs better is Distributed Systems environment. This paper will also enlist some of the good features that should be included within techniques for memory allocation and file access within Distributed Systems.

12) Directory Structure and File Allocation Methods

Mandeep Kaur, Sofia Singh, Rupinder Kaur

Assistant Professor, PG Department of Computer Science and Applications, GHG Khalsa College Gurusar Sadhar, Ludhiana, Punjab, India

The storage of large amount of data permanently in computer system files is used. In this research paper we discuss the file that is a collection of records or information stored on secondary storage such as hard disk. In computing a file system is used to control how data is stored and retrieved. File system control the files starting and ending locations. The information present in the file can be accessed using access methods. In file any time data is failure with hardware problem for solution file system provide protection with access privileges of users. In files secondary storage space is allocated using file allocation methods. These allocated space in such a manner so that disk space is utilized effectively and files can be accessed quickly. Directory structure is use symbol table of files that stores all the related information about the file it holds with the contents.

13)<u>An Empirical Study of File Systems on NVM(Non-Volatile memory)</u>

Priya Sehgal, Sourav Basu, Kiran Srinivasan, Kaladhar Voruganti,NetApp Inc

Based on our study we found that traditional file systems can be tuned to perform better than their default setting on NVM. In some cases these fine-tuned file systems perform at par with PMFS. Further, we found that features that help improve CPU and memory utilizations turn out to be better performing than others on NVM. PMFS, which is an NVMaware file system, has most of the features that leverage the byte-addressability and low latency characteristics of the media. But, if one wishes to use a traditional file system with minor modifications or reconfigurations, we recommend few file system features that can help improve its performance on NVM. Few of the features include:. In-place update layout,Execute-in-place (XIP),Simple and parallel allocation strategy,. Fixed sized data blocks.

<u>14)Understanding Manycore Scalability of File Systems</u>

Changwoo Min Sanidhya Kashyap Steffen Maass Woonhak Kang Taesoo Kim

Georgia Institute of Technology

In this paper we performed a comprehensive analysis of the manycore scalability of five widely-deployed file systems using our FXMARK benchmark suite. We observed many unexpected scalability behaviors of file systems. Some of them lead us to revisit the core design of traditional file systems; in addition to well-known scalability techniques, scalable consistency guarantee mechanisms and optimizing for storage devices based on their performance characteristics will be critical. We believe that our analysis results and insights can be a starting point toward designing scalable file systems for manycore systems.

COMPARISON OF FILE SYSTEMS AND IMPLEMENTATION

1) Windows File Systems :

[2]Microsoft Windows OS use two major file systems: FAT, inherited from old DOS with its later extension FAT32, and widely-used NTFS file systems. Recently released ReFS file system was developed by Microsoft as a new generation file system for Windows 8 Servers.

It serves as a database for all kinds of data like structured, unstructured and semi structured. It helps developers to transfer data in many different ways. It is developed by Microsoft to provide a link between windows vista's application layer and NTFS-New Technology File System. It does not serve as alternative to NTFS of Microsoft's.

Microsoft Windows had 2 widely used file systems. They are

1) NTFS- New Technology File System:- This is the present file system which is given as default to the Windows Operating Systems.
2) FAT-File Allocation Table :- This file system is inherited and developed from old DOS and has been extended to several later versions like exFAT which is also called extended File systems.[2]

FAT:

[3]FAT-File allocation Table was introduced by Microsoft in 1980's. At that time it was one of the simplest file systems types and versions. It consists of (i)File System Descriptor Sector(boot sector), (ii)File System Block Allocation Table and (iii)Plane storage space for storing data in files, folders etc. FAT use directories to store the files and folders. These arrays are of 32-byte size which define a file or any external attribute of that file. It also makes a record of first block of the file. The next blocks are found easily through the Block Allocation Table by using it as a linked list.[3]

[4]The Block Allocation Table contain an array of block descriptors. Zero value indicates that the block is not used and a non-zero one relates to the next block of a file or a special value for the file end.

The numbers 12, 16 and 32 in FAT12, FAT16, FAT32 represent the number of bits used to enumerate a file system block. This means that FAT12 can use up to 2^12=4096 different block references, where as FAT16 and FAT32 can use up to 2^16=65536 and 2^32=4294967296 respectively. The actual maximum count of blocks is actually less and depends file system driver.

FAT12 and FAT16 are used in old floppy disks and they are been out-dated now. FAT32 is still widely used in memory cards and USB's. FAT32 is supported in smartphones, digital camera and other portable devices.

FAT32 can be used on Windows compatible external storages or disk partitions with the size less than or equal to <= 32 GB. Windows cannot create a FAT32 file system which would be larger than 32 GB, whereas Linux supports the size up to 2 TB and which further doesn't allow to create files the size of which exceeds 4 GB. To overcome this problem, exFAT was introduced by Microsoft to overcome limitations in sizing of files or partitions.[4]

NTFS:

[5]NTFS-New Technology File System came into existence in 1993 with Windows NT OS and is currently the most common file system available for end user computers on Windows. Almost every windows server use this format only.

Files in NTFS are stored as a file descriptor in Master File Table. All the details containing entries of file size, name, allocation, etc are stored in this Master File Table. First 16 entries of the master file table are retained for bitmap. This file table also keeps a record of all clusters which are free and used. It also detects the bad clusters called Badclus. The first sector and last sector of the file system contains system settings. This fie system uses 48 bit and 64 bit values as reference files, which makes them supportable to store large data and have high capacity.[5]

UDF:

[8]UDF(Universal Disk Format) - file system is the industrial based file format for storing information on the DVD (Digital Versatile Disc or Digital Video Disc) optical media.

The UDF file system is provided as dynamically loadable 32–bit and 64–bit modules, with system administration utilities for creating, mounting, and checking the file system on both SPARC and IA platforms. The Solaris UDF file system works with supported ATAPI and SCSI DVD drives, CD-ROM devices, and disk and diskette drives. In addition, the Solaris UDF file system is fully compliant with the UDF 1.50 specification.[8]

ReFS:

ReFS - Resilient File System is the latest development of Microsoft introduced with Windows 8 OS and now available for Windows 10 OS. This file system architecture absolutely differs from other Windows file systems and is mainly organized in a form of the B+-tree. ReFS has high tolerance to failures due to new features included into the system. And, namely, Copy-on-Write (CoW): no metadata is modified without being copied; data is not written over the existing data, but into new disk space. With any file modifications, a new copy of metadata is stored into free storage space, and then the system creates a link from older metadata to the newer one. Thus, the system stores significant quantity of older backups in different places providing easy file recovery unless this storage space is overwritten.

NOTE :

(COMPARISON OF WINDOWS FILE SYSTEM ACCORDING TO THEIR SPEED)

[6]If we need the Windows-only environment, NTFS is the best choice. If we had to exchange files (even occasionally) with a non-Windows system like a Mac or Linux box, then FAT32 will give you fine result and with no data loss, where your file size should be less than 4GB.

While file transfer speed and maximum throughput is limited by the slowest link (usually the hard drive interface to the PC like SATA or a network interface like 3G WWAN), NTFS formatted hard drives have tested faster on benchmark tests than FAT32 formatted drives

Generally speaking, exFAT drives are faster at writing and reading data than FAT32 drives. All benchmarks show that NTFS is much faster than exFAT. The bottom line is that unless you are 100 percent sure that you will never have a file smaller than 4 GB, format the drive as exFAT[6]

[7]Comparison of different windows file systems based on their features:

Features	NTSE	FAT32	ReFS	ExFAT
Operating systems	Windows NT Windows 2000 Windows XP Windows Vista Windows 7	Windows 98 Windows ME Windows 2000 Windows XP Windows Vista Windows 7	Windows 8 Windows 8.1 Windows 10	Windows XP (upgrade) Windows Vista Windows 7
Maximum file name strength	255 Unicode characters	255 Unicode characters	255 Unicode characters	127 Unicode characters
Max. path name length	32760 Unicode characters	32760 Unicode characters	32760 Unicode characters	32760 Unicode characters
Max. file size	256 TB for 64 KB cluster	4 GB	16 EB(Exabyte)	512 TB
Max. volume size	256TB or 16TB depends on the cluster size	2 TB	16 EB	512 TB
Short file name support	Yes	Yes	No	Yes
Security and permissions	Yes	No	Yes	No
Tracking file owner	YES	NO	YES	NO
Encryption in file system level	Yes	No	No	No
Access control lists	Yes	No	YES	No
Checksum and ECC	No	No	No	Metadata
POSIX file permissions	No (available in POSIX subsystem feature)	No	Yes	No
Built-in compression software.	Yes	No	No	No
User level disk spacing	Yes	No	No	No

2) LINUX FILE SYSTEMS

Ext

[9]Ext(extended file system) was the first file system in the series of extended file systems. Extended file system(ext) was implemented in April 1992 and it is the first file system created specially for the Linuxkernel. It has metadata structure inspired by the traditional Unix File System (UFS) and was designed and implemented by Remy Card to overcome the disadvantages of the MINIX file system.

Ext2

The ext2 or second extended file system is a file system for the Linux kernel. It was initially designed by Remy Card as a replacement for the extended file system (ext).

ext2 was the default filesystem in several Linux distributions, such as Debian and Red Hat Linux, until ext2 is supplanted more recently by ext3, which is completely compatible with ext2 and ext3 is a journaling file system. ext2 is still used as a filesystem for flash-based storage media (such as SD cards and USB flash drives etc.)

Ext3

Third extended file system(ext3) is the commonly used Linux file system. Ext3 is introduced in 2001 which is developed by Stephen Tweedie. Ext3 was available starting from Linux kernel(v2.4.15). Maximum file size of ext3 ranges from 16GB to 2TB. There are three types of journaling available in ext3 file system.The main advantage of ext3 over ext2 is journaling, which improves reliability and eliminates the need to check the file system after an unclean shutdown.

Ext4

Ext4 was developed to extend storage limits and to add other performance improvements.It supports the maximum file size as comparative to other file systems.Maximum file size of ext4 ranges from 16 GB to 16 TB. Some new features are added in Ext4 such as it supports multiblock allocation, delayed allocation, journal checksum.These new features have improved the performance and reliability of the file system when compared to ext3.

XFS

XFS is a high-performance 64bit journaling file system created by Silicon Graphics,Inc(SGI) in 1993.It is supported by most Linux distributions,some of them uses XFS as the default file system.SGI's IRIX operating system(starting from v5.3) uses XFS as the default file system.

XFS excels in the execution of parallel input/output (I/O) operations.XFS enables extreme scalability of I/O threads, file system bandwidth, and size of files when spanning multiple physical storage devices.

XFS features include striped allocation of RAID arrays,file system journaling,variable block sizes,direct I/O,guaranteed-rate I/O,snapshots,online defragmentation,online resizing. A unique feature to XFS is the pre-allocation of I/O bandwidth at a pre-determined rate,this is suitable for many real-time applications. However, this unique feature was supported only on IRIX operating system and only with specialized hardware.

JFS

Journaled File System(JFS) is a 64-bit journaling file system created by IBM. There are versions for AIX, eComStation, OS/2, and Linux operating systems. The latter is available as free software under the terms of the GNU General Public License (GPL).

In the AIX operating system, there are two generations of JFS filesystem namely JFS (JFS1) and JFS2 respectively. The second generation exists in operating systems such as OS/2 and Linux and is simply called as JFS.W**e** should not be confused with JFS in AIX that refers to JFS1.

ReiserFS

ReiserFS is a general-purpose journaled file system formerly designed and implemented by a team at Namesys led by Hans Reiser..ReiserFs was introduced in Linux kernel(v2.4.1) ReiserFS is used as the default file system on Elive, Xandros, Linspire, and YOPER Linux distributions. ReiserFS is used as default file system in Novell's SUSE Linux Enterprise until Novell decided to move to ext3 on October 12, 2006 for future releases.

Namesys considered ReiserFS as stable and feature-complete and, with the exception of security updates and critical bug fixes, ceased development on it to concentrate on its successor, Reiser4.

Reiser4

Reiser4 is a file system created and developed by Namesys and it is successor to ReiserFS. The creation of Reiser4 was backed by the Linspire project as well as DARPA. What makes Reiser4 special is its multitude of transaction models.[9]

COMPARISON BETWEEN DIFFERENT LINUX FILE SYSTEMS:

Features	ext	ext2	ext3	ext4
Operating Systems	Linux until Linux 2.1.20	Linux, BeOS, Free BSD	Linux, Mac OS with Paragon ExtFS, Solaris	Linux, Mac OS with Paragon ExtFS
Maximum file name length	255 bytes	255 bytes	255 bytes	255 bytes
Maximum path name length	No limit defined	No limit defined	No limit defined	No limit defined
Maximum file size	2 GiB	16 GiB to 2 TiB	16 GiB to 2 TiB	16 GiB to 16 TiB
Maximum volume size	2 GiB	2 TiB to 32 TiB	2 TiB to 32 TiB	1 EiB
Stores file owner	Yes	Yes	Yes	Yes
POSIX file permissions	Yes	Yes	Yes	Yes
Creation Time stamps	No	No	No	Yes
Access control lists	No	Yes	Yes	Yes
Security labels	No	Yes	Yes	Yes
Check sum/ ECC	No	No	No	Partial
Block journaling	No	No	Yes	Yes
Metadata only journaling	No	No	Yes	Yes
File change log	No	No	No	No
Internal branching	No	No	No	No
File system level encryption	No	No	No	Yes

Features	XFS	JFS	ReiserFS	Reiser4
Operating Systems	Linux, BeOS with addon(read only)	Linux, OS/2	Linux, BeOS with addon	Linux with kernel patch
Maximum file name length	255 bytes	255 bytes	4,032 bytes	3,976 bytes
Maximum path name length	No limit defined	No limit defined	No limit defined	No limit defined
Maximum file size	8 EiB	4 PiB	8 TiB(v3.6), 4 GiB(v3.5)	8 TiB on x86
Maximum volume size	8 EiB	32 PiB	16 TiB	Not known
Stores file owner	Yes	Yes	Yes	Yes
POSIX file permissions	Yes	Yes	Yes	Yes
Creation Time stamps	Partial	Yes	No	No
Access control lists	Yes	Yes	Yes	No
Security labels	Yes	Yes	Yes	No
Check sum/ ECC	Partial	No	No	No
Block journaling	Yes	Yes	Yes	Yes
Metadata only journaling	Yes	Yes	Yes	No
File change log	Yes	No	No	No
Internal branching	No	Not known	No	Not known
File system level encryption	No	No	No	Yes
Data deduplication	Yes	Not known	No	Not known

[10]Ext4 file system in Linux Environment:Features and Performance Analysis

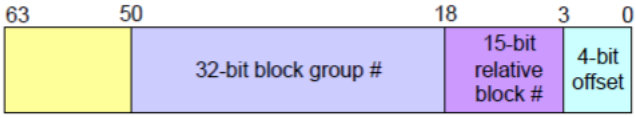
Ext4 was developed to extend storage limits and to add other performance improvements.It supports the maximum file size as comparative to other file systems.Maximum file size of ext4 ranges from 16 GB to 16 TB. Some new features are added in Ext4 such as it supports multiblock allocation, delayed allocation, journal checksum.These new features have improved the performance and reliability of the file system when compared to ext3.

New features added to ext4 are

1) 64 bit file system

Ext4 is 64bit filesystem allowing file size upto 16TB

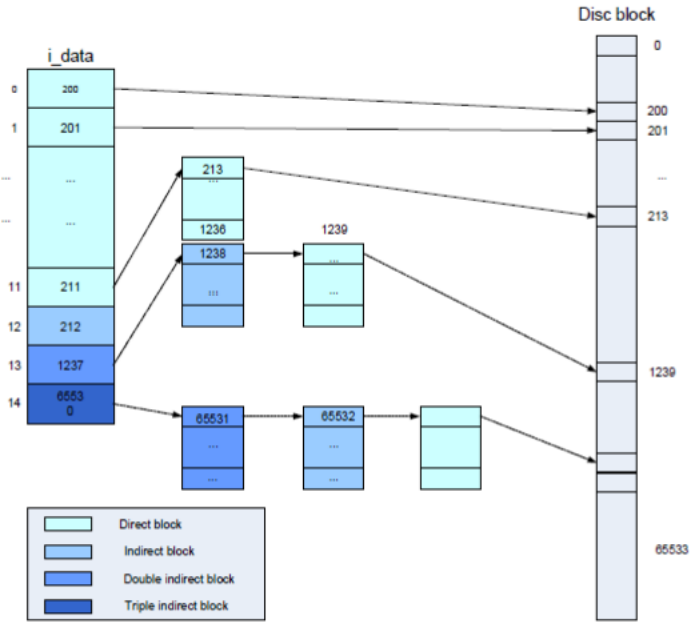
2) Inode size of 256 bytes



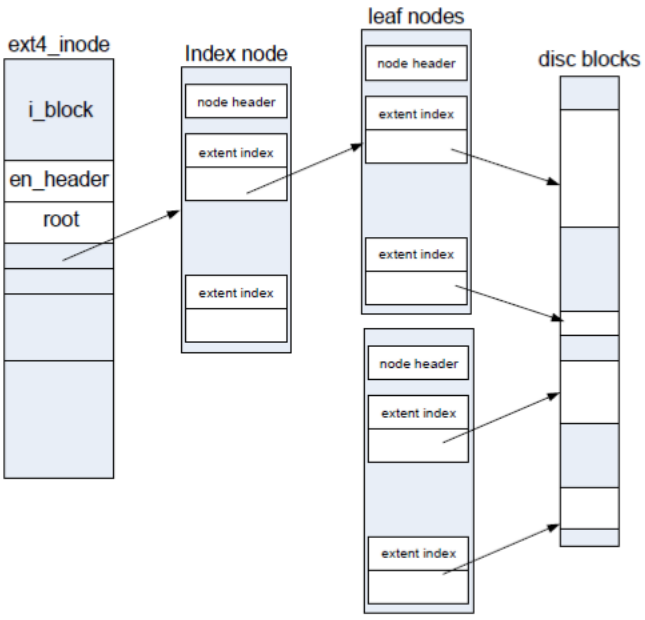
64-bit inode layout

3) Extents

Major difference between ext3 and ext4 file system is the way that the block numbers are stored with data files.Ext3 uses indirect mapping blocks.ext4 remembers the number of blocks in extent (descriptor that represents a continuous series of physical blocks)



ext2/ext3 mapping



Map of ext4 extents tree

4) Directory Scalability

In Ext3 directory entries are stored in a linked list,which is very inefficient for directories with large number of enries.The directory indexing feature addresses this scalability issue by storing directory entries in Htree data structure,which is specialized BTree like structure using 32bit hashes.Fast lookup time of Htree improves performance on large directories.

5) Block allocation enhancment

Higher fragmentation rates causes greater disk access time affecting overall throughput.It has impact on increased metadata overhead causing less efficient mapping.By improving block allocation techniques we can reduce file system fragmentation.

6) Online defragmentation

There is a possibility that the ext4 file system can still become quite fragmented.Ext4 online defragmentation tool,e4defrag can defragment individual files or entire file system and by this way it helps in avoiding file fragmentation caused with file system aging.

7) Reliabilty

Ext3 is one of the most reliable file systems,therefore developers are putting much effort to make it even more reliable.[10]

BENCHMARKTESTS:

1) Bonnie++ and Iozone test:

[11]Benchmark tests using Bonnie++ and Iozone have been performed for Ubuntu and FreeBSD

The versions for Ubuntu 8.10(x86_64) with the Linux 2.6.27 kernel,X server 1.5.2,GCC 4.3.2,GNOME 2.24,the ext2 file system and FreeBSD 7.1 Beta2 (AMD64) with X server 1.5.2,GCC 4.3.2,GNOME 2.24,the FFS file system and Java 1.6.0_07_02.Both the operating systems were left in their default configuration.For testing both operating systems comparison was completed with AMD hardware,the strongest performance is exhibited with Ubuntu 8.10.

1.1 Bonnie++

Bonnie++ is a program o test filesystems for performance.Bonnie++ tests file system operations for different applications used to unlike degrees.It gives result of the amount of workdone per second and the percentage of CPU time for this took.For performance results,higher numbers are better.The step used in this test arecreation,reading and deleting many small files.Test results are shown in fig 1a and fig 1b.

Version 1.93d		Sequential Output						Sequential Input				Random Seeks	
Concurrency	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
1	2G	329	86	10050	24	7946	26	500	85	19318	30	87.5	33

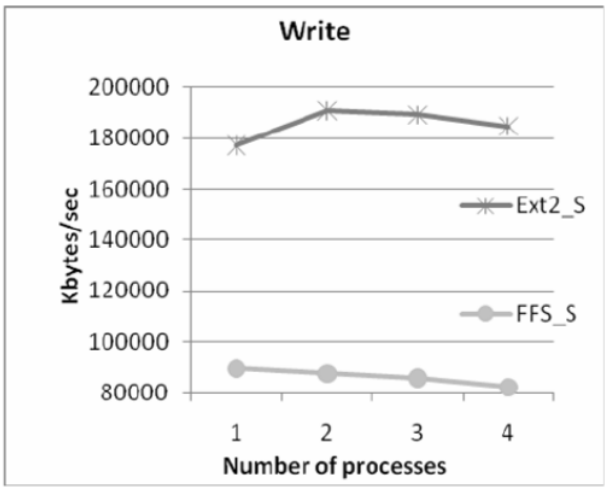
1a.Bonnie++ using Linux

Version 1.93d		Sequential Output						Sequential Input				Random Seeks	
Concurrency	Size	Per Char		Block		Rewrite		Per Char		Block			
		K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	K/sec	% CPU	/sec	% CPU
1	2G	10468	90	25075	43	9711	13	13019	78	19812	17	86.9	0

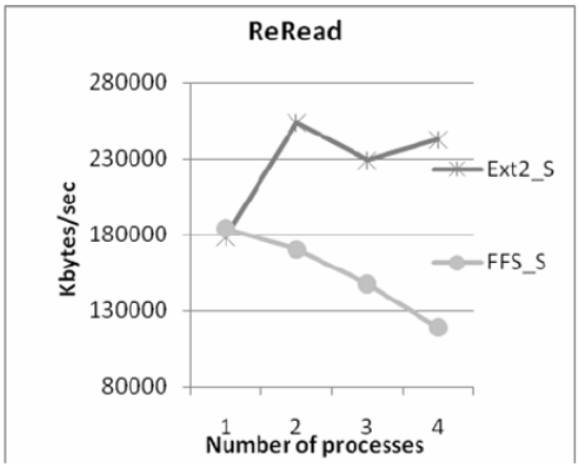
1b.Bonnie++ using Ubuntu

1.2 Iozone

The benchmark generates and measures a variety of file operations.Iozone has been ported to many machines and runs under many operating systems.Iozone performs broader file system analysis than Bonnie++.The benchmark tests file I/O performance for the following operations such as read,write,re-read,random write,random read.Iozone enables the administrator to receive a broad file system performance analysis to optimize his operating system for the best performance.Test results are shown in fig 2a and fig 2b.



2a.Iozone write for ext2 and FFS



2b.Iozone reread for ext2 and FFS

Observations:

After performing the Bonnie and Iozone performance tests ext2 executes more efficiently than ffs.Thus Linux performs better than FreeBSD in these Bonnie++ and Iozone tests.[11]

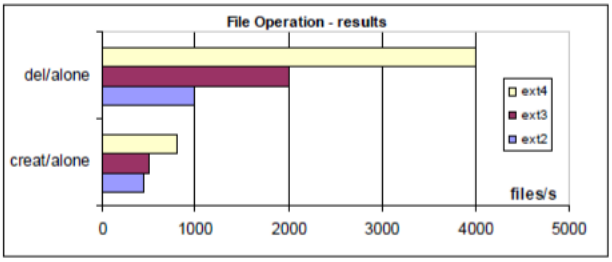
2)POSTMARK BENCHMARKTEST:

[13]Postmark benchmark technique is used for testing the chosen file systems.It simuates the Internet Mail server load.Postmark creates large set of randomly generated files and saves them in any one location in the filesystem.Over this set of fies Postmark and the operating system performs operations like creation,reading,registration and deletion of files and determine the time required to perform these operations.These operations are performed randomly inorder to provide the credibility of the simulation. Number of files, their size range and number of transactions are configurable. To eliminate the cache effect it is recommendable to create large set of files(minimum 10,000) and execution of large number of transactions over the generated files.We have presented the results of three different test procedures.

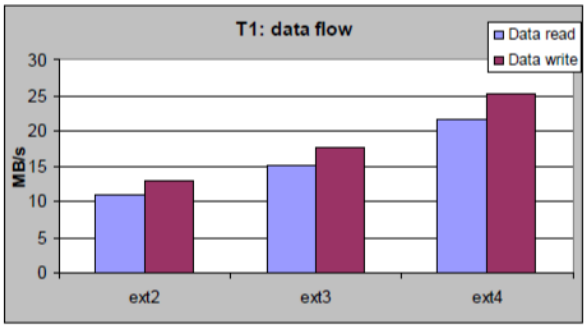
2.1.Postmark test-1(small files)

Files used for performing this test are relatively small,ranging from 1K to 100K.Postmark configuration used for this test was:file size ranges from 1000 to 100000,number of generated files are 4000 and number of transactions performed are 50000.Performance results for each file operation such as creation,creation/alone, creation/with transactions, reading, appending,deleting, deleting/alone, deleting/with transactions are given on the Figures 1a and 1b. Operations delete/alone and create/alone are performed in special benchmark phases, and do not suppose transactions. The file sytems chosen for performing this Postmark test are ext2,ext3 and ext4.

1a.File operations with transactions



1b.File operations without transactions



Test-1 data flow results

Observations:

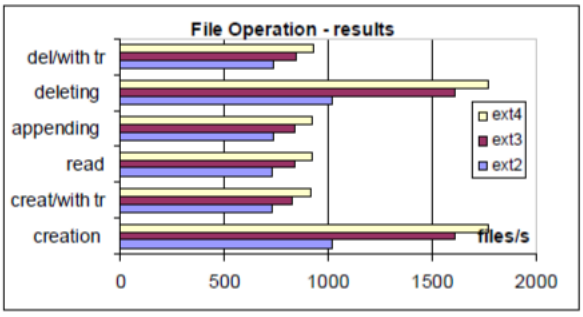
1)In this test ext4 file system has shown higher performance when compared to ext2 and ext3.

2)Ext4 is 40% more faster than ext3 and ext4 is twice as fast as ext2.

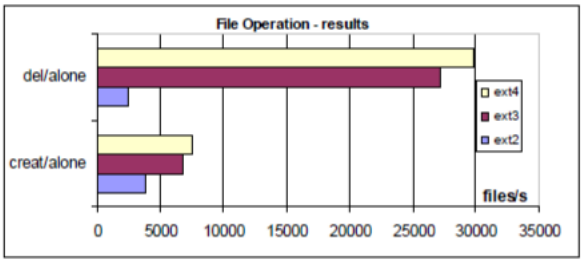
3) ext3 is 35% more faster than ext

2.2.Postmark test-2(ultra small files)

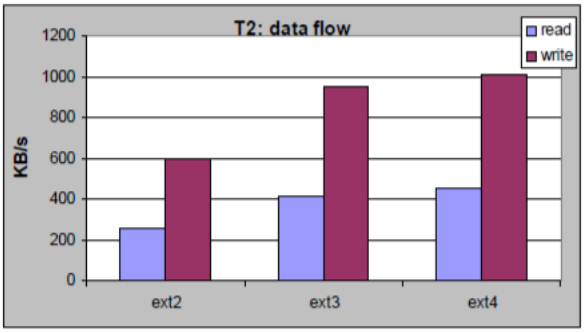
Files used for performing this test are ultra small,ranging from 1byte to 1K.Postmark configuration used for this test was:file size ranges from 1byte to 1K,number of generated files are 30000 and number of transactions performed are 50000.Performance results for each file operation are given on the Figures 2a and 2b.



2a.File operations with transactions



2b.File operations without transactions



Test-2 data flow results.

Oservations:

1) In this test procedure with ultra smal files,Ext4 filesystem has higher performances over ext2 and ext3.

2) Ext4 is 10% faster than ext3.

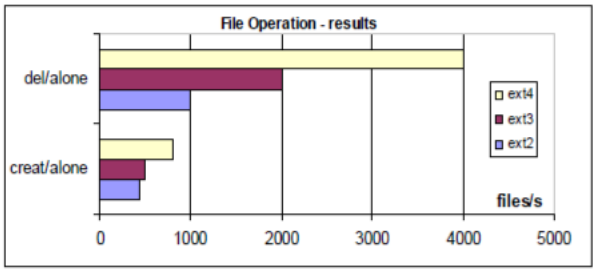
3) Ext4 is 70% more faster than ext2.

4) Ext3 is 60% faster than ext2.

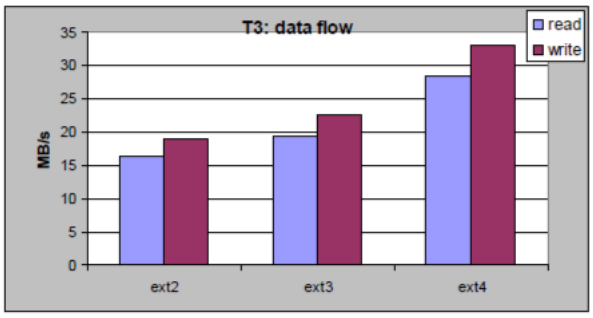
2.3.Postmark test-3(large files)

Files used for performing this test are large,ranging from 1K to 300K.Postmark configuration used for this test was:file size ranges from 1000 t0 300000,number of generated files are 4000 and number of transactions performed are 50000.Performance results for each file operation are given on the Figures 3a and 3b.

3a..File operations with transactions



3b..File operations without transactions



Test-3 data flow results

Observations:

1)In this test procedure with large files,Ext4 filesystem has higher performances over ext2 and ext3.

2) Ext4 is 46% faster than ext3.

3) Ext4 is 73% more faster than ext2.

4) Ext3 is 18% faster than ext2.[13]

3.FILE ALLOCATION METHODS

[12]File Allocation method is used to effectively utilize disk space available by accommodating space for files. Operating system allocates the disk space for the files by using different approaches. It's divided into three broad categories: Contiguous Allocation, Indexed Allocation and Linked allocation. The main objective to adapt these 3 approaches is to make sure there efficient utilization of disk space and speedy access to the files. These three methods use distinct methods to effectively use disk space which has its own pros and cons.

Comparitive Study Of Three Approaches

1. **Contiguous Allocation Method –**

   By this approach, continuous blocks of disk spaces is assigned to each and every file. It uses the concept of first fit or best fit. It's the simplest allocation method.
   Location of file is identified by the starting address of the block followed by the length of file. These two details are crucial to work with file.

File	Starting Address	Length
A	2	3
B	6	2

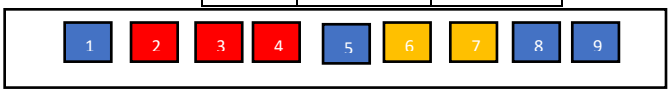


Fig 1:continuous Allocation Method

Advantages –For contiguous memory allocation we can easily access the file Sequential Access and Direct Access. Multiple reading is not required. Can be read by a single operation since it's allocated in continuous blocks resulting in excellent operation.
Number of disk seeks for accessing the file is minimal so it's extremely fast.

Disadvantages-The major con is that the maximum size of the file has to be limited at the starting of creation itself. Most of the disk space is wasted due to external fragmentation leading to inefficient use of disk space. Compaction can be applied as a solution to external fragmentation. It's a very expensive solution to the problem.

2)Linked Allocation Method-

For File allocation method, each file is not allocated in continuous memory blocks. The disk blocks are placed anywhere on the disk. Each file is a linked list of blocks.In this approach, the directory contains the starting address and ending address of the file which are pointers. Then each block contains a pointer that points to the next block.

File	Start block	Length
A	2	3

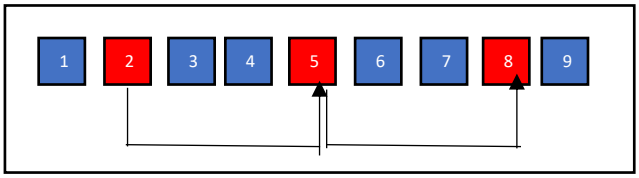


Fig 2:Linked Allocation Method

Advantages – The waste of memory is minimized and maximum disk space utilization is implemented.External fragmentation is not present in this method of allocation. Only internal fragmentation is possible in the last block.It's not necessary to specify the file size during creation.

Disadvantages-Space is occupied for storing pointers leading to extra overhead.Access time is much greater for linked list allocation since the files are located in random order.This method doesn't support direct access only supports sequential access. For finding a block of file it has to implement sequential access from beginning of the file and follow the pointers until the block of the file is identified.

3.Indexed Allocation Method-

All shortcomings of linked allocation method and contiguous allocation method is overcome by indexed allocation method.In this approach, all the pointers are brought together into one location known as the indexed block.All the pointers in index block is set to null.

> **Advantages** –This method supports direct access resulting in faster access of the files also supports sequential method of accessing the file.In this method external fragmentation is absent.Directory just keeps track of the starting block address.Free blocks in the disk is used efficiently.

Disadvantages-The index table should be present in the memory.The index block should be large to hold pointers.Searching for entry in index table is tedious procedure.[12]

File	Index block
A	8

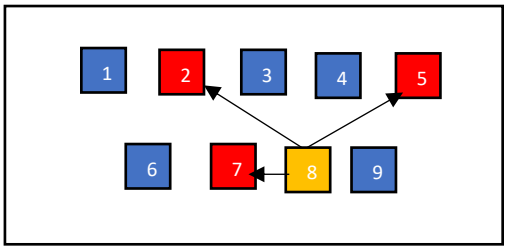


Fig 3:Indexed Allocation Method

IMPLEMENTATION AND ANALYSIS

LINUX

FUSE: SIMPLE FILE SYSTEM

FUSE (Filesystem in User space) is an interface that lets us write our own filesystem for Linux in the user space.

INSTALLING FUSE

```
sabxfab@19BCI0105:~$ sudo apt-get install -y fuse
Reading package lists... Done
Building dependency tree
Reading state information... Done
fuse is already the newest version (2.9.9-3).
fuse set to manually installed.
The following packages were automatically installed and are no longer required:
  python3-click python3-colorama
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 212 not upgraded.
sabxfab@19BCI0105:~$
```

COMPLETE SOURCE CODE SOURCE CODE:

//SIMPLE FILE SYSTEM USING FUSE

//FUSE allows us to create filesystem in user defined

virtual space #define FUSE_USE_VERSION 30

#include <fuse.h>

#include <stdio.h>

#include <unistd.h>

#include

<sys/types.h>

#include <time.h>

#include <string.h>

#include <stdlib.h>

```
#include
<sys/stat.h>
#include <errno.h>

char dir_list[ 256 ][ 256 ]; //maintains the names of directories that have been created by
the user int curr_dir_idx = -1;

char files_list[ 256 ][ 256 ]; //maintains the names of files that have been created by
the user int curr_file_idx = -1;

char files_content[ 256 ][ 256 ]; //maintains the contents of
the files int curr_file_content_idx = -1;

void add_dir( const char *dir_name )
{
        curr_dir_idx++;
        strcpy( dir_list[ curr_dir_idx ], dir_name );
}

int is_dir( const char *path )
{
        path++; // Eliminating "/" in the path

        for ( int curr_idx = 0; curr_idx <= curr_dir_idx;
                curr_idx++ ) if ( strcmp( path, dir_list[
                curr_idx ] ) == 0 )
                        return 1;

        return 0;
}

void add_file( const char *filename )
{
```

```
        curr_file_idx++;
        strcpy( files_list[ curr_file_idx ], filename );

        curr_file_content_idx++;
        strcpy( files_content[ curr_file_content_idx ], "" );
}

int is_file( const char *path )
{
        path++; // Eliminating "/" in the path

        for ( int curr_idx = 0; curr_idx <= curr_file_idx;
                curr_idx++ ) if ( strcmp( path, files_list[
                curr_idx ] ) == 0 )
                        return 1;

        return 0;
}

int get_file_index( const char *path )
{
        path++; // Eliminating "/" in the path

        for ( int curr_idx = 0; curr_idx <= curr_file_idx;
                curr_idx++ ) if ( strcmp( path, files_list[
                curr_idx ] ) == 0 )
                        return curr_idx;

        return -1;
}

void write_to_file( const char *path, const char *new_content )
{
```

```
	int file_idx = get_file_index( path );

	if ( file_idx == -1 ) // No
		such file return;

	strcpy( files_content[ file_idx ], new_content );
}
// -------------------------------------------------------------------------------------------------



// function called when Operating System asks SFS for the attributes of a
specific file static int do_getattr( const char *path, struct stat *st )
{
	printf( "[getattr] Called\n" );
	printf( "\tAttributes of %s requested\n", path );


	st->st_uid = getuid(); // The owner of the file/directory is the user who mounted the
	filesystem st->st_gid = getgid(); // The group of the file/directory is the same as the
	group of the user who
mounted the filesystem
	st->st_atime = time( NULL ); // The last "a"ccess of the file/directory is right now
	st->st_mtime = time( NULL ); // The last "m"odification of the file/directory is right now

	if ( strcmp( path, "/" ) == 0 ) // check if root dir
	{
		st->st_mode = S_IFDIR | 0755; //mode --> directory || Permission: Owner-->RWX
		| User-->
RX
		st->st_nlink = 2; // linking itself and parent
	}
	else if (is_file(path) == 1)
	{
```

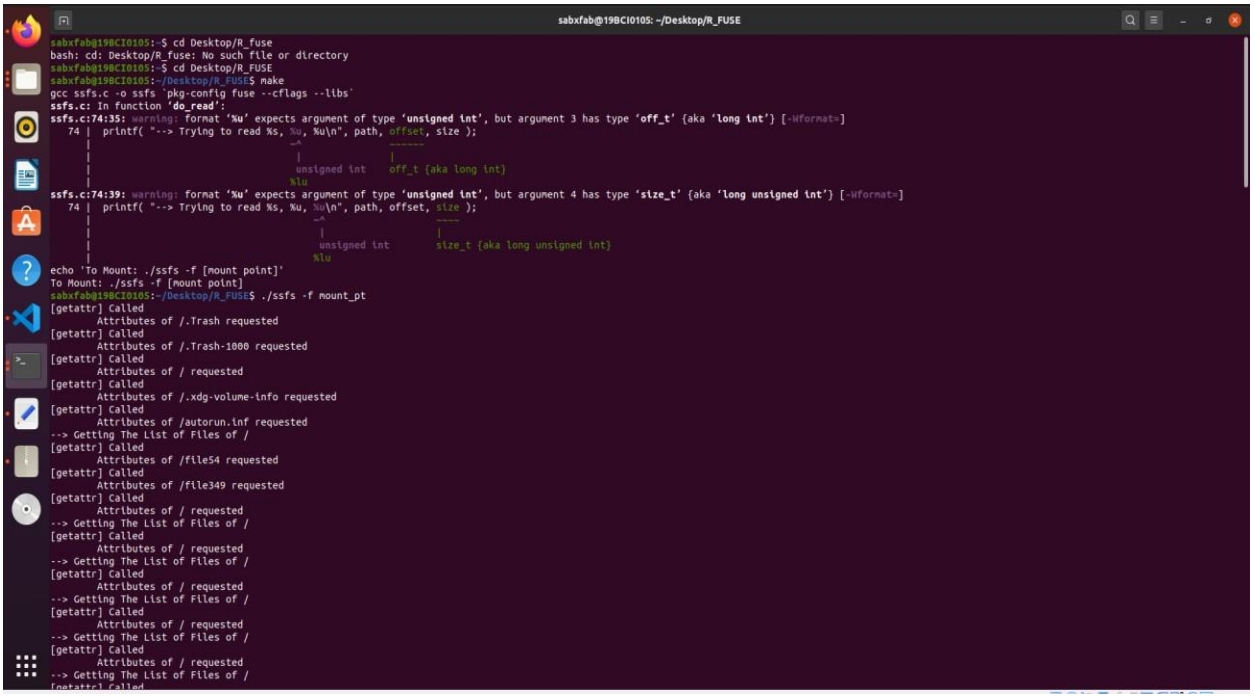
```
RX
                                                                                    }
                                                                                else
        {
        st->st_mode = S_IFREG | 0644; //mode --> regular || Permission: Owner-->RWX | User-->

        st->st_nlink = 1;
        st->st_size = 1024; // file size allocation




return -ENOENT;
        }

            return 0;

 }



 typedef int (*fuse_fill_dir_t) (void *buf, const char *name,
                                    const struct stat *stbuf, off_t off);

 // ROOT LEVEL OBJECTS ONLY( SFS RESTRICTION )
 //provides the operating system with the list of files/directories that reside in a given directory



 static int do_readdir( const char *path, void *buffer, fuse_fill_dir_t filler, off_t offset, struct
 fuse_file_info
 *fi )
 {
            filler( buffer, ".", NULL, 0 ); // Current
            Directory filler( buffer, "..", NULL, 0 ); //
            Parent Directory
```

```
	if ( strcmp( path, "/" ) == 0 ) // If the user is trying to show the files/directories of
the root directory show the following
	{
		for ( int curr_idx = 0; curr_idx <= curr_dir_idx;
			curr_idx++ ) filler( buffer, dir_list[ curr_idx
			], NULL, 0 );

		for ( int curr_idx = 0; curr_idx <= curr_file_idx;
			curr_idx++ ) filler( buffer, files_list[ curr_idx
			], NULL, 0 );
	}

	return 0;
}
// The function of the event read is called when the operating system wants to read the
content of a specific file.

static int do_read( const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi
)
{
	int file_idx = get_file_index( path );

	if ( file_idx == -1 )
		return -1;

	char *content = files_content[ file_idx ];

	memcpy( buffer, content + offset, size );

	return strlen( content ) - offset;
}

// mkdir called when the user wishes to create a new directory.
```

```
static int do_mkdir( const char *path, mode_t mode )
{
	path++;
	add_dir( path
	);

	return 0;
}

//function of mknod event is called when the user wishes to create a new file -- (Regular files only restriction)
static int do_mknod( const char *path, mode_t mode, dev_t rdev )
{
	path++;
	add_file( path
	);

	return 0;
}

// function will be called when the user wishes to write content the a specific file.
static int do_write( const char *path, const char *buffer, size_t size, off_t offset, struct fuse_file_info *info
)
{
	write_to_file( path, buffer );

	return size;
}

//-------------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------
static struct fuse_operations operations = {
```

```
    .getattr    = do_getattr,
    .readdir    = do_readdir,
    .read       = do_read,
    .mkdir      = do_mkdir,
    .mknod      = do_mknod,
    .write      = do_write,
};

int main( int argc, char *argv[] )
{
    return fuse_main( argc, argv, &operations, NULL );
}
```

EXECECUTING THE FILESYSTEM AND MOUNTING IT TO A DIRECTORY:



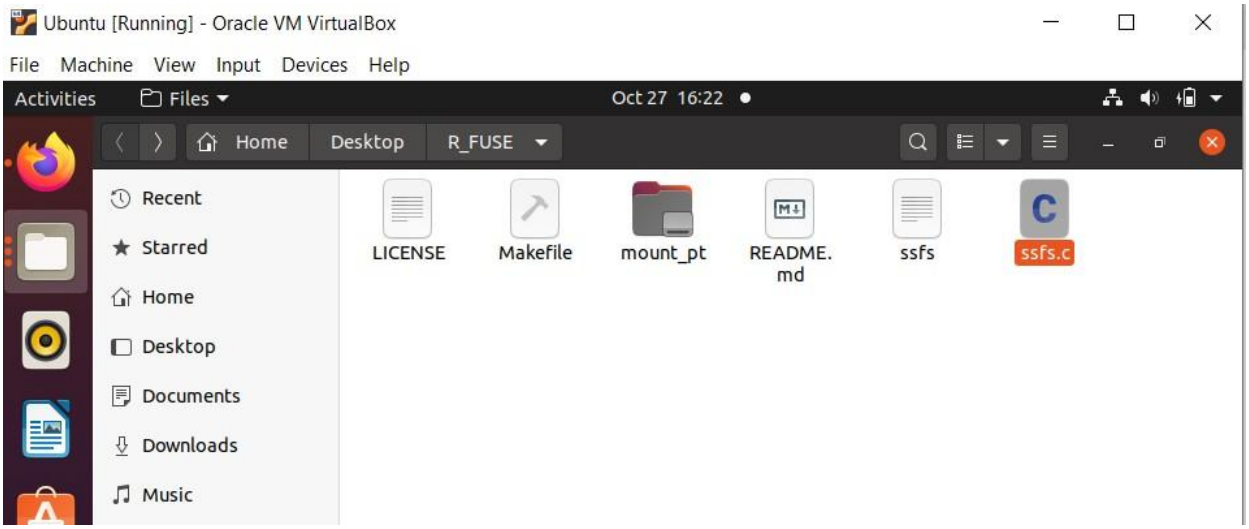
```
echo 'To Mount: ./ssfs -f [mount point]'
To Mount: ./ssfs -f [mount point]
sabxfab@19BCI0105:~/Desktop/R_FUSE$ ./ssfs -f mount_pt
[getattr] Called
        Attributes of /.Trash requested
[getattr] Called
        Attributes of /.Trash-1000 requested
[getattr] Called
        Attributes of / requested
[getattr] Called
        Attributes of /.xdg-volume-info requested
[getattr] Called
        Attributes of /autorun.inf requested
--> Getting The List of Files of /
[getattr] Called
        Attributes of /file54 requested
[getattr] Called
        Attributes of /file349 requested
[getattr] Called
        Attributes of / requested
--> Getting The List of Files of /
[getattr] Called
        Attributes of / requested
--> Getting The List of Files of /
[getattr] Called
        Attributes of / requested
--> Getting The List of Files of /
[getattr] Called
        Attributes of / requested
--> Getting The List of Files of /
[getattr] Called
        Attributes of / requested
--> Getting The List of Files of /
[getattr] Called
```

READ FILES:

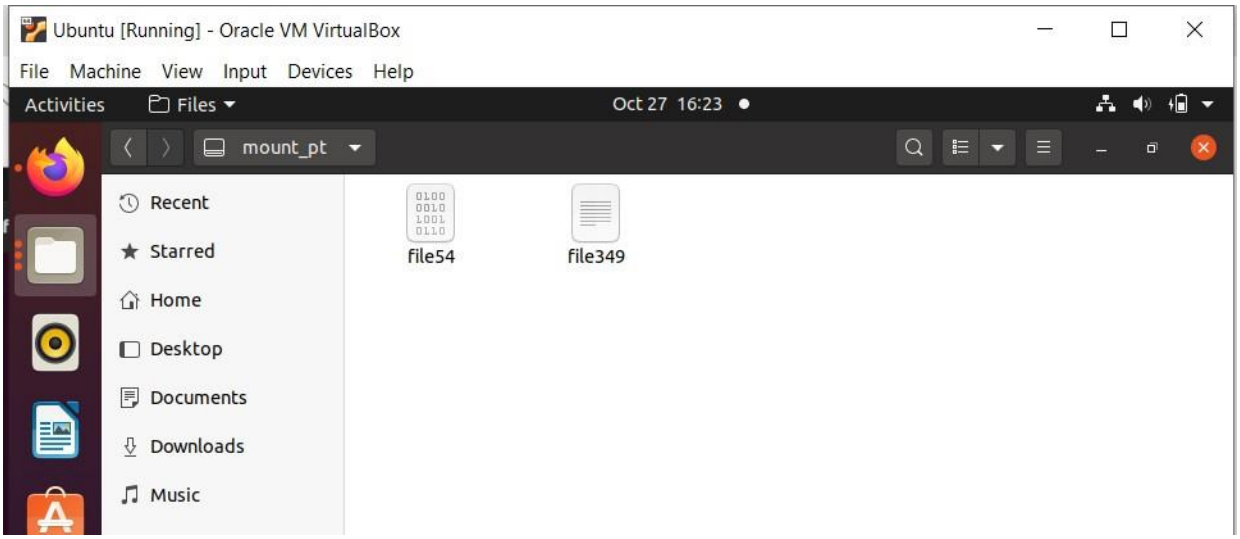
```c
static int do_read( const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi )
{
    printf( "--> Trying to read %s, %u, %u\n", path, offset, size );

    char file54Text[] = "Hello World From File54!";
    char file349Text[] = "Hello World From File349!";
    char *selectedText = NULL;

    // ... //

    if ( strcmp( path, "/file54" ) == 0 )
        selectedText = file54Text;
    else if ( strcmp( path, "/file349" ) == 0 )
        selectedText = file349Text;
    else
        return -1;

    // ... //

    memcpy( buffer, selectedText + offset, size );

    return strlen( selectedText ) - offset;
}
```

```
--> Getting The List of Files of /
[getattr] Called
        Attributes of / requested
--> Getting The List of Files of /
--> Trying to read /.xdg-volume-info, 0, 4096
--> Trying to read /.xdg-volume-info, 0, 4096
--> Trying to read /autorun.inf, 0, 4096
--> Trying to read /autorun.inf, 0, 4096
[getattr] Called
        Attributes of / requested
[getattr] Called
        Attributes of /.Trash requested
[getattr] Called
        Attributes of /.Trash-1000 requested
--> Getting The List of Files of /
[getattr] Called
        Attributes of /file54 requested
[getattr] Called
        Attributes of /.hidden requested
--> Trying to read /.hidden, 0, 4096
--> Trying to read /.hidden, 0, 4096
--> Trying to read /file54, 0, 4096
[getattr] Called
        Attributes of / requested
[getattr] Called
        Attributes of /file349 requested
--> Trying to read /file349, 0, 4096
[getattr] Called
        Attributes of / requested
[getattr] Called
        Attributes of / requested
[getattr] Called
        Attributes of /.Trash requested
[getattr] Called
        Attributes of /.Trash-1000 requested
--> Getting The List of Files of /
[getattr] Called
        Attributes of /file54 requested
[getattr] Called
        Attributes of /.hidden requested
--> Trying to read /.hidden, 0, 4096
--> Trying to read /.hidden, 0, 4096
--> Trying to read /file54, 0, 4096
[getattr] Called
        Attributes of / requested
[getattr] Called
        Attributes of /file349 requested
--> Trying to read /file349, 0, 4096
[getattr] Called
        Attributes of / requested
```

SCREENSHOT AFTER MOUNTING:

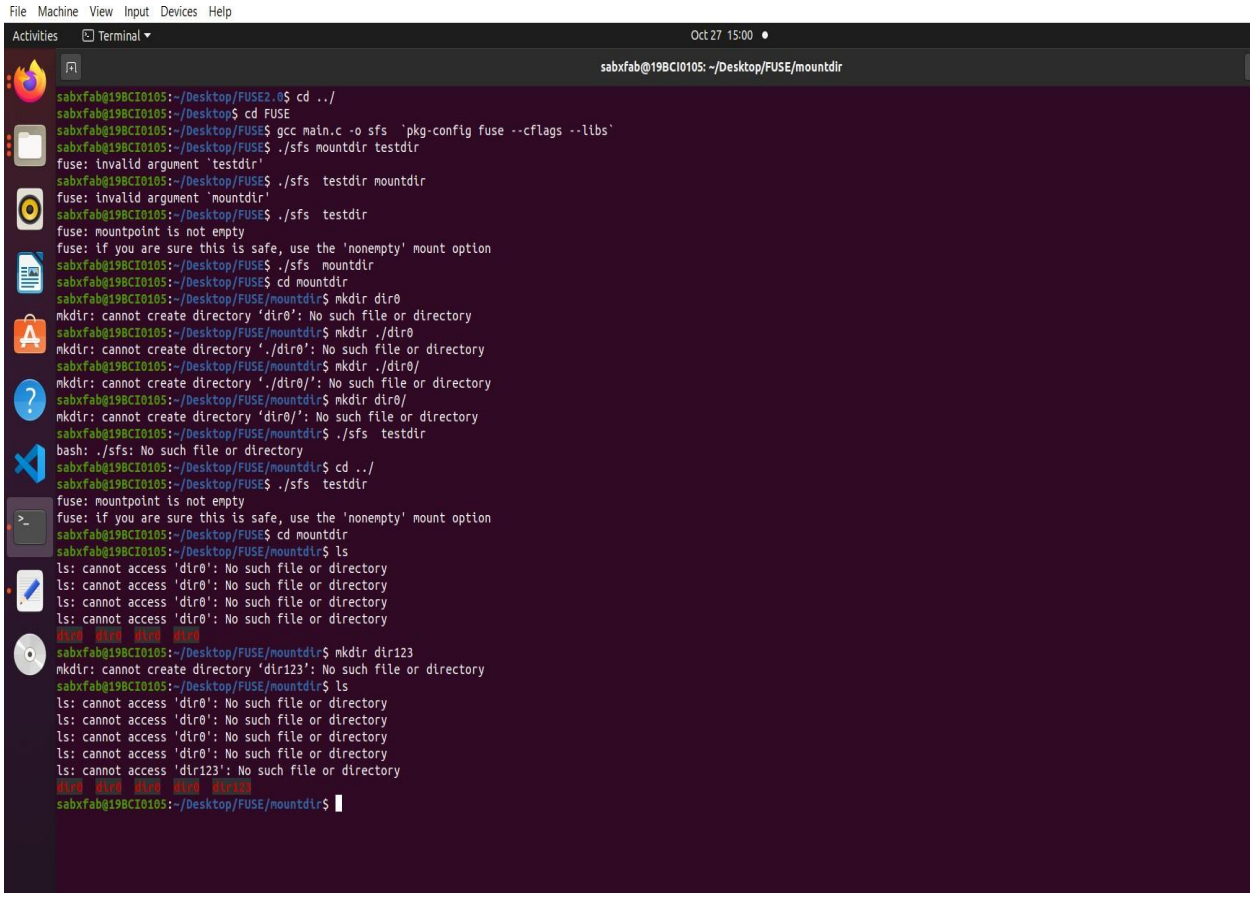


File54 and file349 created by the program on empty directory



CREATE FILES:

```c
// mkdir called when the user wishes to create a new directory.

static int do_mkdir( const char *path, mode_t mode )
{
    path++;
    add_dir( path );

    return 0;
}

//function of mknod event is called when the user wishes to create a new file -- (Regular files only restriction)
static int do_mknod( const char *path, mode_t mode, dev_t rdev )
{
    path++;
    add_file( path );

    return 0;
}

// function will be called when the user wishes to write content the a specific file.
static int do_write( const char *path, const char *buffer, size_t size, off_t offset, struct fuse_file_info *info )
{
    write_to_file( path, buffer );

    return size;
}

//-------------------------------------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------------------------------
```



THERE ARE STILL ISSUES REGARDING FILE PERMISSIONS IN THE SIMPLE FILE SYSTEM.

```
sabxfab@19BCI0105:~/Desktop/FUSE/mountdir$ ls
ls: cannot access 'dir0': No such file or directory
ls: cannot access 'dir0': No such file or directory
ls: cannot access 'dir0': No such file or directory
ls: cannot access 'dir0': No such file or directory
dir0  dir0  dir0  dir0
sabxfab@19BCI0105:~/Desktop/FUSE/mountdir$ mkdir dir123
mkdir: cannot create directory ‘dir123’: No such file or directory
sabxfab@19BCI0105:~/Desktop/FUSE/mountdir$ ls
ls: cannot access 'dir0': No such file or directory
ls: cannot access 'dir0': No such file or directory
ls: cannot access 'dir0': No such file or directory
ls: cannot access 'dir0': No such file or directory
ls: cannot access 'dir123': No such file or directory
dir0  dir0  dir0  dir0  dir123
sabxfab@19BCI0105:~/Desktop/FUSE/mountdir$ cd ../
sabxfab@19BCI0105:~/Desktop/FUSE$ ls
main.c  mountdir  sfs  testdir
sabxfab@19BCI0105:~/Desktop/FUSE$ ls -l  mountdir
ls: cannot access 'mountdir/dir0': No such file or directory
ls: cannot access 'mountdir/dir0': No such file or directory
ls: cannot access 'mountdir/dir0': No such file or directory
ls: cannot access 'mountdir/dir0': No such file or directory
ls: cannot access 'mountdir/dir123': No such file or directory
total 0
?????????? ? ? ? ?            ? dir0
?????????? ? ? ? ?            ? dir0
?????????? ? ? ? ?            ? dir0
?????????? ? ? ? ?            ? dir0
?????????? ? ? ? ?            ? dir123
sabxfab@19BCI0105:~/Desktop/FUSE$
```

FAT Files (WINDOWS)

Source code for creating, opening, displaying FAT details and block details:

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include<conio.h>
#include <string.h>

int s_b[10][2],empty[10]; //Stores size and number of blocks
int mem=500; //keeps track of free memory
char *block[10]; //store address of the blocks
int *address[10]; //stores address of files
char name[10][30]; //stores name of the file
void open() //to create a file
{
 char fname[30],c;
 int size=0,i;
 printf("Enter .txt file name\n");
 scanf("%s",fname);;
 FILE *inputf;
 inputf = fopen(fname,"r");
 if (inputf == NULL)
 {
 printf("\nFile unable to open ");
 exit(0);
 }
 rewind(inputf);
 while(c!=EOF)
 {
 c=fgetc(inputf);
 size=size+1;
```

```
}
printf("The size of given file is : %d\n", size-2);
if(mem>=size)
{
int n=1,parts=0,m=1;
while(address[n]!=0)
n++;
strcpy(name[n],fname);
s_b[n][1]=size;
int bnum=size/50;
if(size%50!=0)
bnum=bnum+1;
s_b[n][2]=bnum;
mem=mem-(bnum*50);
int *bfile=(int*)malloc(bnum*(sizeof(int)));
address[n]=bfile;
printf("Number of blocks required: %d\n",bnum);
rewind(inputf);
c = fgetc(inputf);
while(parts!=bnum && c!=EOF)
{
int k=0;
if(empty[m]==0)
{
char *temp=block[m];
while(k!=50)
{
*temp=c;
c=fgetc(inputf);
temp++;
k=k+1;
}
*(bfile+parts)=m;
parts=parts+1;
```

```
 empty[m]=1;
 }
 else
 m=m+1;
 }
 printf("File created\n");
 printf("\n");
 fclose(inputf);
}
else
 printf("Not enough memory\n");
}
int filenum(char fname[30])
{
int i=1,fnum=0;
while(name[i])
{
 if(strcmp(name[i], fname) == 0)
{
fnum=i;
 break;
 }
 i++;
 }
 return fnum;
}
void blocks()
{
int i;
printf(" Block adddress empty/free\n");
for(i=1;i<=10;i++)
 printf("%d. %d - %d\n",i,block[i],empty[i]);
 printf("\n");
}
```

```
void file(){
int i=1;
printf("File name size address\n");
for(i=1;i<=10;i++){
 if(address[i]!=0)
 printf("%s %d %d\n",name[i],s_b[i][1],address[i]);
}
printf("\n");
}
void print()
{
char fname[30];
int i=1,j,k,fnum=0;
printf("Enter the file name: ");
scanf("%s",fname);
fnum=filenum(fname);
 if(fnum!=0&& address[fnum]!=0)
 {
 int *temp;
 temp=address[fnum];
 printf("Content of the file %s is:\n",name[fnum]);
 int b=(s_b[fnum][2]);
 for(j=0;j<b;j++)
{
 int s=*(temp+j);
 char *prt=block[s];
 for(k=0;k<50;k++)
 {
 printf("%c",*prt);
 prt++;
 }
 }
 printf("\n");
 printf("\n");
```

```
 }
 else
 printf("File not available:/n");
}
void change()
{
char fname[30];
int i=1,j,k,fnum=0;
printf("Enter the file name: ");
scanf("%s",fname);
fnum=filenum(fname);
 if(fnum==0)
 printf("File not available:/n");
 else{
 int *temp=address[fnum];
 int b=(s_b[fnum][2]);
 mem=mem+b*50;
 for(j=0;j<b;j++){
 int s=*(temp+j);
 empty[s]=0;
 }
 address[fnum]=0;
 }
 printf("\n");
}
void create() //to create a file
{
 FILE *inputf;
 char fname[30],ch;
 int a;
 printf("Enter the name of the file to be created : ");
 scanf("%s",fname);
 inputf=fopen(fname,"w");
 printf("File-%s is successfully created\n",fname);
```

```
 printf("Do you want to write into the file\n1.Yes\n2.No\n");
 scanf("%d",&a);
 printf("\n");
 if(a==1)
 {
    printf("Enter the text that u want to insert into the file(Press tab to end) :");
   while((ch=getchar())!='\t')
   {
      fputc(ch,inputf);
   }
   printf("Text successfully inserted\n");
}
fclose(inputf);
}
int main()
{
   char*buffer =(char*)malloc(500); //Memory created-500 bytes
int choice,i;
char *temp;
 if (buffer == NULL)
 {
 fputs ("Memory error",stderr);
 exit (2);
}
 temp=buffer;
 block[1]=buffer;
 empty[1]=0;
 for(i=2;i<=10;i++)
 {
 block[i]=block[i-1]+50;
 empty[i]=0;
}
while(1)
{
```

```
printf("1.Open a file\n");
printf("2.Delete a file\n");
printf("3.Print a file \n");
printf("4.Display FAT table\n");
printf("5.Display Block Details\n");
printf("6.Create a new file\n");
 printf("7.Exit.\n");
 printf("Enter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:
 open();
 break;
 case 2:
change();
 break;
 case 3:
 print();
 break;
 case 4:
 file();
 break;
 case 5:
 blocks();
 break;
 case 6:
create();
break;
case 7:
 exit(1);
 }
 }
 return 0;}
```

OUTPUT COMMAND WINDOW:

Creating a file:

```
C:\Users\sengo\Documents\programming\programs\19BCI0059_OS\bin\Debug\19BCI0059_OS.exe
1.Open a file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Create a new file
7.Exit.
Enter your choice: 6
Enter the name of the file to be created : OS_PROJECT.txt
File-OS_PROJECT.txt is successfully created
Do you want to write into the file
1.Yes
2.No
1

Enter the text that u want to insert into the file(Press tab to end) :This project is done by Sabareshwar,Goutham Reddy,Ankita,Devansh and Sengosharan.


Text successfully inserted
```

CREATED FILE:

OS_PROJECT - Notepad

File Edit Format View Help

This project is done by Sabareshwar,Goutham Reddy,Ankita,Devansh and Sengosharan.

Printing the contents of a file:

```
1.Open a file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Create a new file
7.Exit.
Enter your choice: 3
Enter the file name: OS_PROJECT.txt
Content of the file OS_PROJECT.txt is:

This project is done by Sabareshwar,Goutham Reddy,Ankita,Devansh and Sengosharan.
```

Displaying the block details:

C:\Users\sengo\Documents\programming\programs\19BCI0059_OS\bin\Debug\19BCI0059_OS.exe

```
File name size address

1.Open a file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Create a new file
7.Exit.
Enter your choice: 5
 Block adddress empty/free
1. 1774688 - 0
2. 1774738 - 0
3. 1774788 - 0
4. 1774838 - 0
5. 1774888 - 0
6. 1774938 - 0
7. 1774988 - 0
8. 1775038 - 0
9. 1775088 - 0
10. 1775138 - 0
```

TO EXIT:

C:\Users\sengo\Documents\programming\programs\19BCI0059_OS\bin\Debug\19BCI0059_OS.exe

```
1.Open a file
2.Delete a file
3.Print a file
4.Display FAT table
5.Display Block Details
6.Create a new file
7.Exit.
Enter your choice: 7

Process returned 1 (0x1)   execution time : 2.103 s
Press any key to continue.
```

CONCLUSION

If we need the Windows-only environment, NTFS is the best choice. If we had to exchange files (even occasionally) with a non-Windows system like a Mac or Linux box, then FAT32 will give you fine result and with no data loss, where your file size should be less than 4GB.

While file transfer speed and maximum throughput is limited by the slowest link (usually the hard drive interface to the PC like SATA or a network interface like 3G WWAN), NTFS formatted hard drives have tested faster on benchmark tests than FAT32 formatted drives

Generally speaking, exFAT drives are faster at writing and reading data than FAT32 drives. All benchmarks show that NTFS is much faster than exFAT. The bottom line is that unless you are 100 percent sure that you will never have a file smaller than 4 GB, format the drive as exFAT.

There are many file systems available on Linux. Each serves a unique purpose for unique users looking to solve different problems.Most popularly used are ext4,XFS,Reiser4.

Ext4 is the file system of choice for most of the Linux distributions.ext4 has all the goodness that we have come to expect from the past file systems(ext2/ext3) but with enhancements.ext4 has good features such as file system journaling,journal check sums,backward compatibility support for ext2 and ext3,persistent pre-allocation of free space,improved file system checking,support for large files.

XFS is a high-end file system that specializes in speed and performance. XFS does extremely well when it comes to parallel input and output because of its focus on performance. The XFS file system can handle massive amounts of data, so much in fact that some users of XFS have close to 300+ terabytes of data.If you have a home server and you're perplexed on where you should go with storage, consider XFS. A lot of the features the file system comes with (like snapshots) could aid in your file storage system. It's not just for servers, though. If you're a more advanced user and you're interested in a lot of what was promised in BtrFS, check out XFS. It does a lot of the same stuff and doesn't have stability issues.

Reiser4 has the unique ability to use different transaction models. It can use the copy-on-write model (like BtrFS), write-anywhere, journaling, and the hybrid transaction model. It has a lot of improvements upon ReiserFS, including better file system journaling via wandering logs, better support for smaller files, and faster handling of directories.Resier4 is for those looking to stretch one file system across multiple use-cases. Maybe you want to set up one machine with copy-on-write, another with write-anywhere, and another with hybrid transaction, and you don't want to use different types of file systems to accomplish this task. Reiser4 is perfect for this type of use-case.

REFERENCES

Websites:

1) https://en.wikipedia.org/wiki/File_Allocation_Table

2) http://searchexchange.techtarget.com/definition/file-allocation-table

3)https://www.techopedia.com/definition/1369/file-allocation-table-fat

4)https://www.lifewire.com/what-is-file-allocation-table-fat-2625877

5)https://en.wikipedia.org/wiki/NTFS

6)https://www.ufsexplorer.com/und_fs.php

7)https://searchstorage.techtarget.com/definition/file-system

8)https://en.wikipedia.org/wiki/Universal_Disk_Format

9)https://en.wikipedia.org/wiki/FAT_filesystem_and_Linux

10)https://en.wikipedia.org/wiki/File_Allocation_Table

11)https://www.tutorialspoint.com/operating_system/os_file_system.htm

12)https://www.geeksforgeeks.org/file-allocation-methods

13)http://www.ques10.com/p/14746/explain-file-systems-of-windows-and-linux-operating

Journals:

1)Comparative study of File systems (NTFS, FAT, FAT32, EXT2, EXT3, EXT4)

Akash Bundele1, Prof. Dr. S. E. Yedey2 PG Department of Computer Science & Technology, Hanuman Vyayam Prasarak Mandal, Amravati, Maharashtra[3]

2) File Systems in Linux and FreeBSD: A Comparative Study

Kuo-pao Yang , Katie Wallace Computer Science and Industrial Technology Department Southeastern Louisiana University[1]

3) File Systems for Various Operating Systems: A Review

Isma Irum, Mudassar Raza, Muhammad Sharif Comsats Institute of Information Technology Wah Cantt, Pakistan[6]

4) Ext4 file system in Linux Environment:Features and Performance Analysis

Borislav Djordjevic, Valentina Timcenko[2]

5) Comparison and Review of Memory Allocation and File Access Techniques and Techniques preferred for Distributed Systems

Muazzam A. Khan, Nauman Nisar, Department of Computer Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology,Islamabad, Pakistan.[11]