File Project – Answer Document

Basic Submission (earns a maximum of 88 points)

bio.c has several test cases. You will build and run TFS to understand the file system and how it is tested. You will use the hexdump tool on tinyfs to further your understanding.

1. Build and run tiny, using all three of the arguments - create, write, read. You can read above and study main in bio.c for the current content of these three arguments. Copy/paste your run log here.

Text

Description automatically generated

1. Update main in bio.c to write data to a file you create. Also update main to read data from your file. For both of these updates, I suggest you retain small sizes for writes and reads for you initial update. Copy/paste your run log here. File named JAMESJOON.

Wrote “JAMESJOON work” into file

Text

Description automatically generated

1. Use hexdump to dump the file tinyfs, which is the TFS TDD to verify that your files are on the TDD. You must locate several aspects of them. Their names, inode numbers, and data block(s) in the root directory. Copy/paste your use of hexdump here. Annotate your copy/paste to demonstrate understanding.
2. Create an annotated diagram of TFS. Show TFS structures and how they interconnect. Include the superblock, the inode bitmap, the data bitmap, the inodes, and two files - one of which is what you added to main in step 2. Also show file descriptors, struct proc, struct file, struct inode and how a file descriptors are converted to inodes. When creating this diagram, use the hex dump from step 3 to demonstrate your understanding. Place your annotated diagram here.
3. Create a function call trace starting with tfs\_read. Create this trace by reading the code. For each function, describe the parameters and a detailed description of what the function does. For example,
   1. tfs\_read’s (in file tfsfile.c) parameters are a file descriptor, a buffer, and a size to read. tfs\_read .
   2. fd\_to\_file (in file tfsfile.c) parameters are an fd and a struct file \*\*. fd\_to\_file ensures the fd is within range and returns the pointer to the struct file in the proc’s ofile[].
   3. fileread (in file.c) parameters are … continue
   4. int tfs\_dup(struct file \*f) parameters are a pointer to a file. Uses the file descriptor to copy the file
   5. int tfs\_write(int fd, void \*p, int n)parameters are a file descriptor, buffer and a size to write. calls fd\_to\_file to convert the file descriptor to a struct file. Then tfs\_read calls filewrite to write to the file
   6. int tfs\_close(int fd) parameter is a file descriptor. Changes the file descriptor to 0 indicating that the file has been closed.
   7. int tfs\_fstat(int fd, struct tfs\_stat \*st)parameters are a file descriptor and a pointer to a file\_stat struct. Converts the file descriptor to a file and calls the filestat function.
   8. int tfs\_link(char \*old, char \*new) parameters are a pointer to an old link and a pointer to a new link. Create the path new as a link to the same inode as old. Creates a char array of size DIRSIZE. Check to see if the old link exists and is not a directory. Increment the nlink. Check if parent directory exists. Checks if new directory was created.
   9. static int isdirempty(struct inode \*dp) parameters is a pointer to a struct inode. Checks if the directory dp empty except for "." and ".."
   10. int tfs\_unlink(char \*path) parameters is a string input of path to be unlinked. Makes 2 pointers to inode structs. Makes a struct dirent. Makes an array of chars the size of DIRSIZ. Call the nameiparent function, check if target directory is parent or current. Call dirlookup function to check inode of target directory. Check the number of links. Check if target file is a directory. Call writei function. Decrease the link if parent directory and file
   11. int tfs\_open(char \*path, int flags, int mode) parameters are a string pointer path, interger for flags integer for mode. Creates a variable to hold a file descriptor, a file struct and an inode struct. Calls the create function and save the output to the inode struct. Uses the file struct to check for the target fie. Returns the file descriptor.
   12. static struct inode\* create(char \*path, short type) parameters are string pointer char and variable of type short. Creates and inode and links it to a directory
   13. int tfs\_mkdir(char \*path) parameters are string pointer char. Creates a new directory by creating an inode and assigning it the type directory
   14. int tfs\_chdir(char \*path) parameters are a sting pointer char. Changes the directory by searching for the target inode and verifying it is a directory
   15. static int fdalloc(struct file \*f) parameters are pointer to a struct file. Allocate a file descriptor for the given file. Takes over file reference from caller on success.
   16. void readfsinfo() no parameters. Read the super block, bitmaps, and inodes.
   17. void print\_inodes()no parameters. print\_inodes can be used for debugging
   18. void writefsinfo() no parameters. Write the super block, bitmaps, and inodes.
   19. uint balloc() no parameters. Allocate a zeroed disk block.
   20. void bfree(uint bi) parameters is a variable of type unit. Free a disk block.
   21. struct inode\* ialloc(short type) parameters is a variable of type short. Allocate a new inode with the given type
   22. struct inode\* iget(uint inum) parameters is a variable of type unit. Find the inode with number inum on device dev and return the in-memory copy.
   23. struct inode\* idup(struct inode \*ip) parameters is struct inode pointer. Increment reference count for ip. Returns ip to enable ip = idup(ip1) idiom.
   24. iput(struct inode \*ip) parameters is a struct inode pointer. Drop a reference to an in-memory inode. If that was the last reference, the inode cache entry can be recycled. If that was the last reference and the inode has no links to it, free the inode (and its content) on disk.
   25. static uint bmap(struct inode \*ip, uint bn) parameters are a struct inode and a variable of type unit. The content (data) associated with each inode is stored in blocks on the disk. The first NDIRECT block numbers are listed in ip->blocks[]. We do not implement NINDIRECT. If so, the next NINDIRECT blocks are listed in block ip->blocks[NDIRECT]. Return the disk block address of the nth block in inode ip. If there is no such block, bmap allocates one.
   26. static void itrunc(struct inode \*ip) parameters are a struct inode pointer. Truncate inode (discard contents). Only called when the inode has no links to it (no directory entries referring to it) and has no in-memory reference to it (is not an open file or current directory).
   27. void stati(struct inode \*ip, struct tfs\_stat \*st) parameters are struct inode pointer, striuct tfs\_stat pointer. Copy stat information from inode.
   28. int readi(struct inode \*ip, char \*dst, uint off, uint n) parameters are struct inode pointer, char pointer, two variables of type unit. Read data from inode.

Place your function call trace here.

1. Update hexdump to include a new flag. You can select the flag. For example, you could add a -i flag that dumps inodes, or you can add a -e flag that has the end block. Copy/paste a run log of hexdump applied to tinyfs with your new flag.
2. Submit your updated bio.c and hexdump.c files.

Advanced Submission (earn 2 points per test)

Update main in bio.c to test additional tfs\_ functions. You will first have to investigate the Linux equivalent to discover what it is supposed to do. Then you can try the tfs\_ version to see if it works. When something does not work, you will be forced into deep study of the underlying code base, which will sharpen your understanding of file system implementation. You can update bio.c with explicit test calls similar to the provided tests. A beautiful update of bio.c would include a shell-like interface that allows you to open, close, link, mkdir, etc. Select from (or do all) the following tests.

1. Create a tinyfs file with over 512 bytes of data. This will cause a second block to be allocated. See that the inode indicates two data blocks. Copy/paste a hexdump of tinyfs showing your large file.
2. Repeat step 6a, but this time write to the file in steps. First write data that fits on a single data block. Then create another file with data. Then write more data to the first file to cause it to grow to two data blocks. This will show a file with two non consecutive data blocks. Copy/paste a hexdump of tinyfs showing your large file with data blocks separated.
3. tfs\_fstat - equivalent to fstat. Copy/paste a log showing your tfs\_fstat test.
4. tfs\_link - equivalent to link. Copy/paste a log showing your tfs\_link test.
5. tfs\_unlink - equivalent to unlink. Copy/paste a log showing your tfs\_unlink test.
6. tfs\_mkdir - equivalent to mkdir. Copy/paste a log showing your tfs\_mkdir test.
7. tfs\_chdir - equivalent to chdir.  Copy/paste a log showing your tfs\_chdir test.
8. tfs\_lseek - equivalent to lseek.  Copy/paste a log showing your tfs\_lseek test.
9. Submit your updated bio.c file that includes your test cases.