**1. tfs\_read(int fd, void \*p, int n)**

1. Assign f as a pointer to a struct file.
2. Call fd\_to\_file function passing fd and the address of f.
   1. Check if fd is valid, and if it is not, return -1.
   2. Get the file associated with fd and store it in f.
3. Call fileread function passing f, p, and n.
   1. Read n bytes from f into the buffer pointed by p.
   2. Return the number of bytes read.

**2. fd\_to\_file(int fd, struct file \*\*pf)**

1. Declare a pointer f to a struct file.
2. Check if fd is invalid or the file associated with fd is not found. If it is, return -1.
3. Check if pf is not NULL. If it is not, set the value of \*pf to f.
4. Return 0 to indicate success.

**3. fileread(struct file \*f, char \*addr, int n)**

1. Declare an integer variable r.
2. Check if f is readable. If it is not, return -1.
3. Check if the type of file pointed to by f is FD\_INODE. If it is, execute the following steps:
   1. Call the readi function passing the inode associated with f, the buffer pointed to by addr, the offset of the file in the inode, and the number of bytes to read.
   2. Read n bytes from the inode into the buffer pointed by addr.
   3. Return the number of bytes read.
   4. Update the offset of the file by adding r to f->off.
   5. Return r to indicate success.
4. If the type of file pointed to by f is not FD\_INODE, panic.
5. Return -1 to indicate an error.

**4. readi(struct inode \*ip, char \*dst, uint off, uint n)**

1. Declare an unsigned integer variable tot and another unsigned integer variable m.
2. Check if the offset off is greater than the size of the inode or off+n would wrap around. If it is, return -1 to indicate an error.
3. If the sum of off and n is greater than the size of the inode, set n to the difference between the size of the inode and off.
4. For tot initialized to 0, loop while tot is less than n.
   1. Calculate the block number to read by calling bmap function passing the inode associated with ip and off/BSIZE.
   2. Read BSIZE bytes from the block number calculated in the previous step into the buffer buf.
   3. Check if the read is successful. If it is not, panic.
   4. Set m to the minimum value between n-tot and BSIZE - off%BSIZE.
   5. Copy m bytes from buf + off%BSIZE to dst.
   6. Increment tot by m.
   7. Increment off by m.
   8. Increment dst by m.
5. Return n to indicate success.

**5. readfsinfo**()

1. Read the first block of the file system into the buffer buf.
2. Copy the contents of the buffer buf to the superblock structure sb.
3. Read the second block of the file system into the buffer buf.
4. Copy the contents of the buffer buf to the inode bitmap.
5. Read the third block of the file system into the buffer buf.
6. Copy the contents of the buffer buf to the data bitmap.
7. For each of the four blocks containing inodes, do the following:
   1. Read the block into the buffer buf.
   2. If the read is unsuccessful, panic.
   3. For each of the eight inodes in the block, copy the contents of the buffer buf to the corresponding inode in the array inodes.
8. For each inode, copy the value of nlink to ref.
9. Return from the function.

**6. panic(char \*s)**

1. Print the string pointed to by s to the console.
2. Call the exit function passing 1 as an argument to terminate the program with a non-zero exit status.

**7. bmap(struct inode \*ip, uint bn)**

1. Declare an unsigned integer variable addr.
2. Check if bn is less than NDIRECT. If it is, execute the following steps:
   1. Check if the block number bn is allocated for the inode. If it is not, allocate a new block and assign its address to the block number bn for the inode.
   2. Return the address of the block number bn.
3. If bn is greater than or equal to NDIRECT, panic.
4. Return -1 to indicate an error.

**8. balloc()**

1. Declare an unsigned integer variable m.
2. Loop over each bit in the databitmap starting from the ninth bit (index 8) up to the 1024th bit (index 1023).
   1. Calculate the bit mask by shifting 1 left by bi % 32.
   2. Check if the block corresponding to the current bit is free by performing a bitwise AND operation between the bit mask and the corresponding word in databitmap. If the result is 0, the block is free.
   3. If the block is free, mark it as in use by performing a bitwise OR operation between the bit mask and the corresponding word in databitmap.
   4. Write all zeros to the new block by calling bwrite with the block index and a buffer filled with zeros.
   5. Return the index of the newly allocated block.
3. If no free block is found, panic.
4. Return -1 to indicate an error.

**9. bwrite(uint block, char \*buf)**

1. Calculate the offset of the block in the file system by multiplying the block number block by BSIZE.
2. Set the file pointer to the calculated offset by calling lseek function passing the file descriptor fs, the calculated offset, and SEEK\_SET as the third argument.
   1. If the return value of lseek is less than 0, panic.
3. Write BSIZE bytes from the buffer pointed to by buf to the file system by calling write function passing the file descriptor fs, the buffer buf, and BSIZE as the third argument.
   1. If the return value of write is less than 0, panic.
4. Return 0 to indicate success.