CS380L: Lab 2

Dinesh Reddy Chinta

Feb 2023

1 User-level File System

1.1 Implementation

1.1.1 System Architecture

I used two Ubuntu 20.04.5 LTS guest Virtual Machines – one as a client on which the remote NFS is mounted and the other as a server in which both the FUSE and NFS file systems are backed. Both virtual machines are connected to the internet through a virtual bridge on the host. The host is running on a Ubuntu 20.04.5 LTS OS. While this is not a recommended system architecture, as the lab documents suggest running the experiments on two physically separate machines, I simulated network delays as discussed in the Miscellaneous section.

The implemented FUSE file system is a modification of the Big Brother File System (BBFS), written for version 2.6. As specified in the lab doc, the server hosts the file system on Ramfs—more on this in the Miscellaneous section.

The NFS version is 4.2.

Please find the code at my git, (https://github.com/chintadinesh/adv_os-lab2) repo. Feel free to request for permissions.

1.1.2 FUSE Calls Implemented

I only implemented three calls: "getattr", "open", and "release".

In the first call, bb_getattr, I used libssh to connect to the server and execute "stat" command on the corresponding remote file. The output of the command is then parsed to extract individual file attributes. The actual implementation is a bit inefficient, if you inspect the submitted code, since we are again executing "lstat" command on the corresponding local file. This was due to parsing issues associated with the remote ssh stat command. Apart from the slight inefficiency, the implementation works without any major issues.

In the second function, bb_open, I copy the file from the remote node if it is not already present locally. While this implementation doesn't work when there are multiple clients, in our usecase, we only have one client. Hence, the local file must have been the most upto date file. I used remote copy ("rcp") to copy the files.

In the third function, bb_release, I simply update the remote file using another remote copy command to the server.

For efficiency purposes, I disabled the logging facility provided that comes with the BBFS.

The test code provided in the lab document, which reads and writes some bytes into "foo" file works for the implemented file system.

1.1.3 Miscellaneous

While debugging and analyzing the results of the experiments, I noticed a few issues with my virtual machines and system architecture. The first one was that both of my virtual machines were on the same host; hence there was an unrealistic network delay. The second issue was the "/tmp" file system. Instead of mounting on RAM, my VM used disk for "/tmp".

To simulate network delays I used Linux's "traffic control" subsystem. For realistic delays I used "ping" command to measure the network delay between two nodes in my subnet. I used half of those mean and standard deviation delays for each of the server and client virtual interfaces in my host OS.

I used Linux's "traffic control" subsystem to simulate network delays. I used the "ping" command to measure the network delay between two nodes in my subnet for realistic network delays. I used half of those mean and standard deviation delays for each server and client virtual network interface in my host OS.

Listing 1: network delay simulation

```
$ ping $(ubuntu)
...
--- 192.168.1.228 ping statistics ---
42 packets transmitted, 42 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 2.996/7.291/15.759/3.704 ms


$ sudo tc qdisc add dev vnet0 root netem delay 4ms 1ms
```

To resolve the disk issues associated with "/tmp", I created a Ramfs to host the FUSE file system in the server. After this, the read delay during remote file copy stabilized to realistic values.

Listing 2: RAM file system for server end FUSE file system

```
$ mkdir share
$ sudo mount -t ramfs ramfs share
```

1.2 Evaluation

The experiments were designed to expose the benefits and limitations of the FUSE filesystem.

In the case of the FUSE file system, it is intuitive to expect that copying the entire file is very inefficient if we only read a few bytes from the file. Again, the same feature is beneficial if the workload accesses all the files eventually.

1.2.1 Experiment Design

In the first experiment, we focus on the benefits of caching the file locally. Copying the entire file is efficient for large files if the workload will read the whole file. However, NFS has some prediction capabilities; you will see the same performance if we access the file sequentially. To work around this, we do random access to the file. I reused the random number generator from the previous lab.

Listing 3: Random file access

```
void large_sequential_rd_wr() {
    const uint32_t FILE_SIZE = opts.opt_file_size;
    int fd1 = open("./fusemnt/foo", O_RDONLY);
    char *rdbuf = (char *)malloc(PAGE * sizeof(char));

    for(int total_read = 0, curr_read = 0; total_read < FILE_SIZE; total_read += curr_read) {
        if(opts.opt_random_access ) {
            long r = (simplerand() % FILE_SIZE) & (~(PAGE - 1)); // random block in the file
            lseek(fd1, r, SEEK_SET);
        }
        curr_read = read(fd1, rdbuf, PAGE); // first read the block
    }

    close(fd1);
}
```

The design of the second experiment is to expose the limitation of the same feature of copying the entire file locally when opening the file. While it is beneficial if the whole file is accessed, it is a waste of resources if the workload accesses only a small portion of the file. As shown in the code, we only read the first 4096 bytes from the file.

Listing 4: fsync

```
void large_single_rd() {
    const uint32_t FILE_SIZE = opts.opt_file_size;
    int fd1 = open("./fusemnt/foo", O_RDONLY);
    char *rdbuf = (char *)malloc(PAGE * sizeof(char));

    int curr_read = read(fd1, rdbuf, PAGE); // read only one block

    close(fd1);
}
```

Linux's block caching is an essential element influencing the performance of file systems. In our experiments, we would like to avoid the effects of caching.

NFS caches the blocks both in the server and the client. To remove this in the client, I unmounted the file system in the client before each experiment. In the server, I delete an existing file and create a new file. While it is debatable whether or not server-side caching is avoided, it could be noted that caching happens for both file systems. Hence, server-side caching should not be the reason for the difference in performance.

Listing 5: Reduce the impact of block caching

```
ssh dinesh@192.168.122.5 "cd /.../nfs_share; rm foo ; dd if=/dev/zero of=foo bs=$file_size count=1"
sudo umount fusemnt
sudo mount -t nfs -o sync 192.168.122.5:/.../nfs_share fusemnt
$executable -f $x -r -o | tee $log_dir/nfs.random_read.$x.$i
```

1.2.2 Results

The time taken to complete experiment 1 (random read access) for NFS is higher than for the FUSE file system. This is represented in the green line in the graph (Fig. 1). Having hamstrung NFS' prediction capabilities, every block access goes through the network. Hence the performance degradation is much higher, which aligns with our expectations. Furthermore, the performance decreases linearly with the file size.

However, in the case of FUSE, which is represented by a blue line in the plot, it is interesting to observe that the performance remains constant with the file size. While at first glance, it appears counterintuitive since the disk accesses time should increase linearly with size. A closer look would clarify the discrepancy. There are two important factors involved. The first one is that the remote file system is on RAM. Hence it is faster than random disk access in the case of NFS. The second factor is that we immediately access the files in the client after copying. The OS dedicates a large amount of RAM for holding disk blocks. In our use case, the 32MB files easily fit the recently copied disk blocks in the RAM. Hence, the impact of random access, or the file size, is barely observed in the case of FUSE.

In the second experiment, where we only access a small portion of a file (4KB), the performance of FUSE is much worse than NFS. It can be observed from the figure that FUSE is an order of magnitude higher than NFS, which is represented by the brown line. There can be two contributing factors, network delay and disk access delay. With further experiments, evaluating which of those two contributed to the significant performance difference is easier.

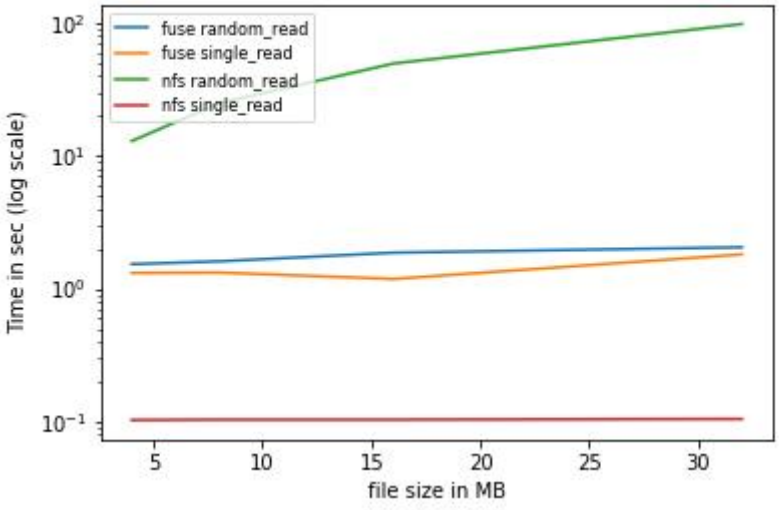

Figure 1: A graph showing the comparison of mean time taken to run the 4 experiments for 5 times. 1: random access of a file backed by FUSE (fuse random_read); 2: single page access of a large FUSE backed file (fuse single_read); 3: random access of a file backed by NFS (nfs random_read); 4: single page access of a large NFS backed file (nfs single_read)

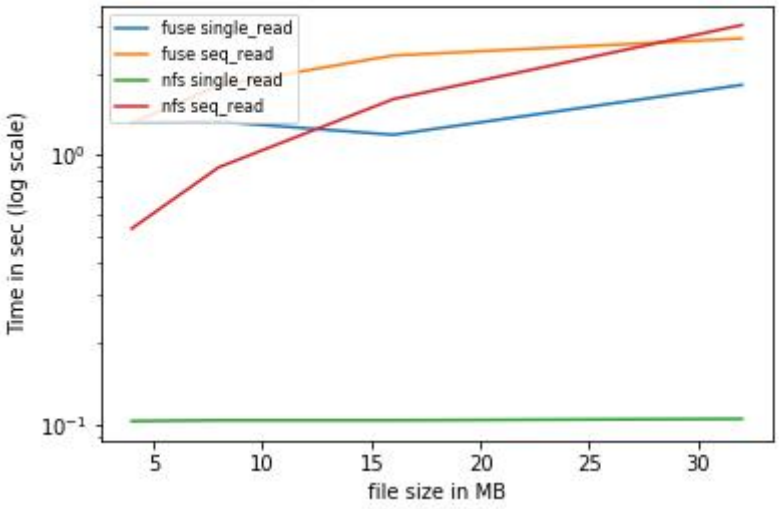

Figure 2: A graph showing the comparison of mean time taken to run the 4 experiments for 5 times. 1: single page access of a large FUSE backed file (fuse single_read); 2: sequential access of a file backed by FUSE (fuse seq_read); 3: single page access of a large NFS backed file (nfs single_read); 4: sequential access of a file backed by NFS (nfs seq_read)

Figure 2. shows the performance comparison of sequential access of a large file backed by NFS and FUSE vs the performance of a single page read of the same large files. The brown line, corresponding to sequential access of NFS, shows that the delay scales linearly with file size. This is expected since the client requests block one by one instead of all simultaneously, as in FUSE. Though it is not tested, the trend shows that even for sequential access, the performance of NFS degrades for large files.

2 System Tools Exercises

A strace

Include the file session_record, and describe what you learned from this exercise.

First of all, I find it interesting to learn about the session_record utility. While there exists sophisticated desktop recording utilities, it could be very convenient to record the activity in a session though this inexpensive utility on embedded devices that have limited functionality. This also takes significantly less storage, which is another advantage for using the low overhead utility.

The strace command exposes the sequence of system calls related to files: fstat, read, mmap, close etc.

Listing 6: session record

```
$ strace cat -> new_file
execve("/usr/bin/cat", ["cat", "-"], 0x7ffc627f4138 /* 33 vars */) = 0
...
mmap(NULL, 5698848, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2245f17000
close(3)                                = 0
fstat(1, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x3), ...}) = 0
fadvise64(0, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2245ef5000
read(0, hi mom
"hi_mom\n", 131072)             = 7
write(1, "hi_mom\n", 7)                 = 7
read(0, "", 131072)                     = 0
munmap(0x7f2245ef5000, 139264)          = 0
close(0)                                = 0
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
dinesh@ubuntu:~/adv_os-lab2$ exit
```

```
Script done on 2023-03-01 10:45:14-06:00 [COMMAND_EXIT_CODE="0"]
```

B lsof

What kind of devices do you see listed? Sometimes you can use this trick to debug a lack of audio output. DO NOT include the output of this command.

The lsof command stands for "list open files," and it is used to display information about files that are currently open on a system. Specifically, it displays information about files that are opened by processes.

The output of the lsof command includes the following information:

- The process ID (PID) of the process that has the file open
- The user ID (UID) of the user who owns the process
- The file descriptor (FD) field contains text instead of a number, it indicates that the file is not a regular file, but instead is a special type of file or resource. It's text indicates, cwd: the current working directory of the process, txt: the executable code of a program, mem: memory-mapped file or shared library, CHR: a character device, PIPE: a named or anonymous pipe, SOCK: a network socket, FIFO: a named pipe or FIFO file
- The type of file (e.g., regular file, directory, socket, etc.)
- The device number (e.g., the disk or partition the file resides on)
- The size of the file
- The file mode (e.g., read-only, write-only, read-write)
- The path to the file
- The lsof command can be used with various options to filter and refine the output, such as displaying only certain types of files, only files opened by a particular process, or files opened by a particular user.

C Network Tools

Include the name of the interface your machine is using to communicate externally.

Listing 7: Network interface

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP>
...
2: enp1s0: ...
...
```

Questions:

1. Are DHCP messages sent over UDP or TCP?

   DHCP (Dynamic Host Configuration Protocol) messages are typically sent over UDP (User Datagram Protocol) rather than TCP (Transmission Control Protocol).

   UDP is a lightweight protocol that provides a simple mechanism for sending datagrams (packets) over a network. DHCP messages are typically small and do not require the reliability and error-checking provided by TCP. Instead, they can be sent using UDP, which provides a faster and more efficient transport mechanism.

   DHCP uses port 67 for server messages and port 68 for client messages. When a DHCP client sends a request for configuration information, it sends a DHCPDISCOVER message to the broadcast address

(255.255.255.255) on port 67. The DHCP server responds with a DHCPOFFER message sent to the client's IP address on port 68. The client then sends a DHCPREQUEST message to the server to accept the offered configuration, and the server responds with a DHCPACK message.

So, in summary, DHCP messages are sent over UDP on port 67 and 68.

2. What is the link-layer (e.g., Ethernet) address of your host? (Feel free to obscure the last couple bytes for privacy's sake)

Listing 8: ethernet address

```
$ ip a show enp1s0
2: enp1s0: ...
    link/ether 52:54:00:c6:1e:** brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.175/24 ...
```

3. What is the IP address of your DHCP server?

Listing 9: ethernet address

```
$ sudo dhclient -r
$ sudo dhclient -v enp1s0
...
DHCPOFFER of 192.168.122.175 from 192.168.122.1
DHCPREQUEST for 192.168.122.175 on enp1s0 to 255.255.255.255 port 67 (xid=0x33625321)
DHCPACK of 192.168.122.175 from 192.168.122.1 (xid=0x21536233)
...
```

4. What is the purpose of the DHCP release message?

If the client no longer needs the assigned IP address, it can send a DHCP release message to the DHCP server, indicating that it is relinquishing the lease. The DHCP server can then update its lease database and make the released IP address available for assignment to other clients.

Sending a DHCP release message can be useful in situations where a client device is leaving a network and will not be returning for some time, or when a device is being decommissioned and the IP address is no longer needed.

5. Does the DHCP server issue an acknowledgment of receipt of the client's DHCP request?

Yes, the DHCP (Dynamic Host Configuration Protocol) server issues an acknowledgment of receipt of the client's DHCP request.

Listing 10: ethernet address

```
$ sudo dhclient -r
$ sudo dhclient -v enp1s0
...
DHCPOFFER of ...
DHCPREQUEST for ...
DHCPACK of ...
...
```

6. What would happen if the client's DHCP release message is lost? If the client's DHCP (Dynamic Host Configuration Protocol) release message is lost, the client's IP address lease will expire based on its original lease time, but the DHCP server may not be aware that the IP address is no longer in use.

   This can lead to several potential issues:

   - IP address depletion: If the DHCP server does not know that the IP address has been released, it will not be able to reuse the address for another client. Over time, this can lead to IP address depletion, which can cause problems for other clients trying to obtain IP addresses from the DHCP server.
   - Configuration conflicts: If the released IP address is reassigned to another client before the original lease time has expired, both clients may end up using the same IP address at the same time, which can cause configuration conflicts and network connectivity issues.
   - Security risks: If the released IP address is not properly released, the client may still have access to network resources that it should no longer have access to, which can pose security risks.

D Time Spent

While it took less time to code. It took a lot of time to debug both the FUSE code issues and the system issues related to network and disk performance.