Fuse Filesystem Group
Assignment 3

Brenden Owens, Pierce Albert,
Tony Grace, Jeremie Adams, Ian Gaskill
April 23, 2015

1 Problem

The problem that we were given as a group was to create a fuse filesystem that has a raspberry pi hooked up to a Gieger counter as a way to create random bits. The bits would then be used by the filesystem and when ever a read function is called in that directory/filesystem. The read function in our fuse filesystem will then be called and execute according to the instructions that we write for it. We were also to create the write function that would write the average counts per minute to the file that was specified whenever there is a call to the system write function. We were also to write the functions that were needed for the read and write functions to work properly and not return any errors. This would include create, open, seek, unlink, release, flush, and access flags and these functions will be called either by themselves or by one of the class member functions. After we mount the fuse filesystem we will go into it and we will see the files mirrored in the system and can manipulate them the same way as a regular filesystem as long as our functions allow it.

2 Approach

The way our group decided to work on this project was to take it one function at a time and from there we could determine if we needed to change what it did to suit our specific use of it, if we really needed it at all, or if we didn't have a function that we needed. We started with the raspberry pi function that we would need in order to get our random bits in order for any of our filesystem to work correctly. We started with just collecting time stamps, branched to finding intervals, went on to finding the bits, and then ended on writing to a file in bit form. We chose a .bin file since it would only be storing binary information and nothing else, this seemed like a good choice since we are just storing binary data. We also implemented a part in our raspberry pi program that writes the current average events in a file and we will use that when ever a user wants to write to a file. This is the basics of how we went about doing the raspberry pi program part. The fuse functions we took one step at a time till each function worked the way we wanted it to. We started with the create function and went down to the release function and this seemed to help us more more effectively

than just splitting it up for everyone to work on. We were fine tuning everything after three meetings and making changes on how it implemented the functions and used the data that we were collecting. This was a great approach and helped us to finish the project in a very quick pace.

2.1 Assumptions

We made the assumption that the user wanted to see the binary data that we have collected from the raspberry pi. Thus instead of showing it as hex or ASCII values we just give them the byte string that was returned from the actual os.read() call. We also assumed that we should create temporary files that will store our data for the filesystem and then after the user wants the filesystem unmounted then we will just delete the files that were generated. We are also assuming that the Geiger counter will detect enough events as to not run out of random bits.

2.2 Trade Offs

We made a decision to ge with two seperat programs running simultaneously because having the raspberry pi code running as a thread would have been harder to kill than a regular program. Also we chose to have the raspberry pi code started by the fuse filesystem code because when the fuse filesystem is closed then we are done using the binary data.

2.3 Design Choices

We chose to use a .bin file to store our data from the Geiger counter since it would only be binary data and nothing else. We have two seperat programs running because it was easier for us to control and determine which program was running correctly and which one was giving us errors. We have the fuse code killing the raspberry pi code due to the fact that once we unmount the fuse filesystem we have no more need for the binary data to be collected. We also chose to use a dictionary that kept track of all of the open files that the user has open and we used that for all of our functions to make sure that they have the file that they want to work with open and ready to use. If the file is not in the dictionary then we open it for them and then continue with the function like normal.

3 Instructions To User

Open up a terminal and login to a raspberry pi. To have a successful mount of the fuse filesystem you will need to have a directory that will have the raspberry.py program in it along with the inter.py program that will help to demo the fuse filesystem correctly. You will also need an empty directory that will be the mount

point for the fuse filesystem. After those requirements are set up it is time to start up the fuse filesystem. Type in the parent directory of both directories "python fuse/fuse.py -f $fuse $mountpoint" where $fuse is the directory you want to mount and $mountpoint is the directory where you want to on, now open another terminal and login to the raspberry pi and navigate inside the $mountpoint directory and type int "python inter.py" and from there follow what the program asks for. As you follow the inter.py program you will see the actual fuse filesystem calls that are being made. After you are done using the filesystem go one directory up and type in the command "fusermount -u $mountpoint".

4 Assessment of Development Process

4.1 Easy Parts

One of the easy parts of this assignment was the raspberry pi code. The pigpio library that we used was a very helpful tool to use to interface with the Geiger counter. Also the collection of binary data was very easy to do once we got the Geiger counter code working. We found that figuring out what each of our fuse functions should do was very easy and describe once we got to that step of the development.

4.2 Dificult Parts

what we found difficult to do is figure out how we wanted to represent the random bits to the user and store them. We went from just representing them as the bits to hex values to ASCII. We finally settled on the unadulterated bits that we generated through the raspberry.py program. We also found debugging the fuse side of things was a very difficult and tricky thing to figure out. Only after trying a few different ways of accomplishing what we wanted to do did we finally figure out what we could use to make the fuse functions work correctly.