Name: Huyen Nguyen

Report for Dumpster Diving

Note: you will want to change environment variables DUMPSTER and PARTITION at the top of test_script.sh before running it. DUMPSTER is the path to the dumpster. PARTITION is the path of the other partition. Also, you want to be in the `test` directory.

Design

a) What programs/scripts you ran and what they did (use pseudo-code)

I used measure_time (compiled from measure_time.c) and test_script.sh.

Pseudocode for test_script.sh

Set dumpster path
Set partition path
Testing ./rm on SAME partition in milliseconds for files
Create files
Call measure_time to remove the files
Testing ./rm on DIFFERENT partition in milliseconds for an empty directory
Create directory named `empty` in partition
Call measure_time to remove `empty`
Testing ./rm on DIFFERENT partition in milliseconds for files
Create files
Call measure_time to remove the files
Testing ./rm on DIFFERENT partition in milliseconds for LARGE directory
Create directory named `large` with structure:

```
large
	sublarge1
		file1.txt
		file2.txt
		file3.txt
		file4.txt
		file5.txt
		file6.txt
		file7.txt
		file8.txt
		file9.txt
		file10.txt
		File11.txt
	sublarge2
	...
	sublarge10
```

Where `sublarge2,..., sublarge10` has the same contents as `sublarge1`.
Call measure_time to remove `large`

Pseudocode for measure_time.c

```
start = get current time
child_pid = Fork child process
If (child_pid == 0)
	Call rm using execvp()
	Print error // If execvp returns, it must have failed
Else
	Wait for child process
	Sync
	end = get current time
	print(start-end)
	Return child status
```

b) How many runs you performed

For each case of same partition and different partitions, I performed 7 runs. Specifically, the file size increases by 10 times for every subsequent run. The program rm was also tested using a folder named `large` with 10 subdirectories, each contained 11 files with size 64KB.

c) How you recorded your data

I used measure_time.c to record time. Specifically, for every file, I recorded the time right before running rm (1). Then, I created a child process to run rm and made the parent process wait for it. Then, when the child process is finished, I used sync(), and then measure the current time again (2). Finally, the time difference between (2)-(1) is the time taken to move the file to the dumpster.

d) What the system conditions were like

The experiment was conducted on MacBook Pro (Retina, 13-inch, Early 2015) with macOS High Sierra. I made sure that the scripts run on the Linux VM for this course. The reason why I decided to run the tests on my actual OS because I experienced problems with gettimeofday() in VMs in OS class.

e) And any other details you think are relevant.

Results

Same Partition

File Size (in KB)	Time Taken (in milliseconds)	Time Taken (in microseconds)
64	145	145213
640	129	128689
6400	127	127042
64000	109	108695
640000	119	118716
6400000	110	110380
64000000	105	105026
Mean	120.5714286	120537.286
Standard deviation	14.14045194	14171.4495

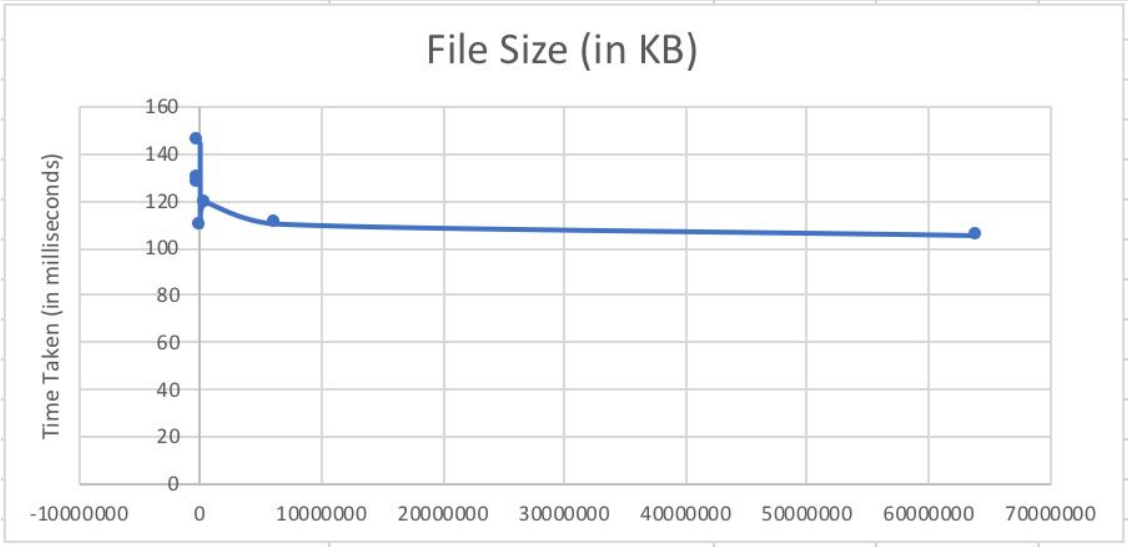

Figure 1: The relationship between file size (in KB) and time taken (in ms) when moving files within the same partition

Different Partitions

File Size (in KB)	Time Taken (in milliseconds)	Time Taken (in microseconds)
64	223	222779
640	120	120011
6400	106	105594
64000	109	109202
640000	158	157795
6400000	822	821828
64000000	6780	6780193
Mean	1188.285714	1188200.3
Standard deviation	2478.982769	2479102.2

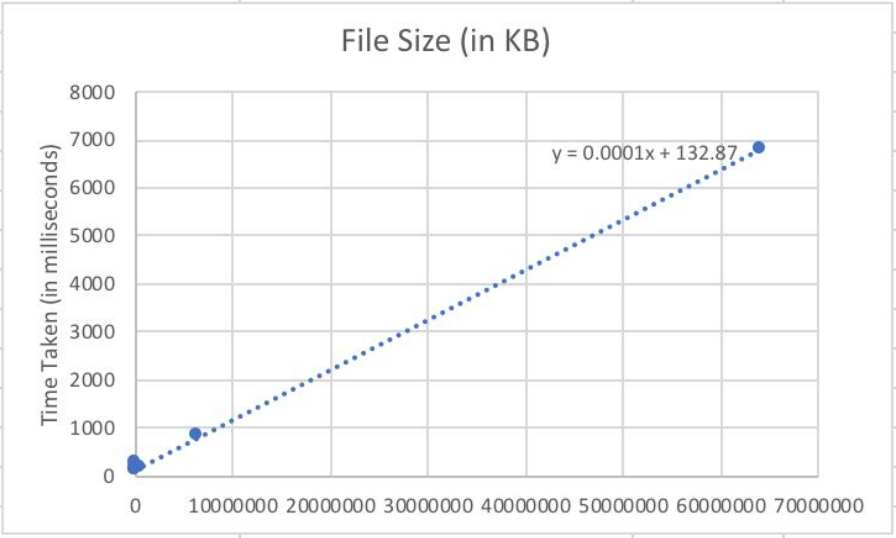

Figure 2: The relationship between file size (in KB) and time taken (in ms) when moving files across different partitions

Large Directory

Time taken is 161020 microseconds, or ~161 milliseconds.

Empty Directory

Time taken is 92868 microseconds, or ~93 milliseconds.

Analysis

Same Partition

The time taken is of the same order of magnitude. In fact, the horizontal line in the graph indicates that the time taken could be a constant. This makes sense because all the OS does is unlinking one directory entry and linking a new one at the destination, so the file size does not have an impact on the time.

Different Partitions

The time taken forms a straight line. This indicates a linear relationship between file size (in KB) and time taken (in ms). This makes sense since the OS has to copy the file to the destination and delete the original. This process involves reading and writing the file contents and its metadata.

Large Directory

First, it is beneficial to compute the expected time token to move `large` directory based on the equation in figure 2, and the time to remove an empty directory.
Expected time taken = Time to remove directories + Time to remove files
Expected time taken = 93 * 11 + (0.0001 * 11 * 10 * 64 + 132.87) (since there are 11 directories, and 11*10 files, each with size 64KB)
Expected time taken = 1156.574 (ms)
The time is a lot shorter than what was actually measured. The reason is the equation does not accurately predict the time for 64KB files. As we see in the table in figure 2, the time taken to move a 64KB file is actually more than moving a 640KB file due to random read (since seek time is more). Therefore, the fact that the actual time is larger than the expected time makes sense.