Exercise 1 : the linux page cache

Sébastien Ponce

May 22, 2015

1 Foreword

All the exercises of the "Strategies and technologies for efficient I/O" track are build on the same ideas :

- they provide working code and you only have to use and look around. No programming skills required :-)
- they are mostly written in python, as python code is more readable than C/C^{++} for non experts. And they are as simple as possible so that the code is very readable
- their only purpose is to demonstrate something. They are in no way supposed to be efficient or well structured. DO NOT REUSE that code in real software ! It is bad, inefficient and more importantly very unsecure.
- they are designed to mostly work locally (yes, even the networking ones) on a crappy old machine, from a live image on a USB stick. So they can run anywhere, including on your laptops. But take care that you may have to install some tools.

The exercises are using the linux command line and the many tools provided in a standard linux distribution. For the ones you do not know, please open the man page and read at least the description. Also feel free to ask any questions.

2 Goals of the exercise

- Exercise the linux page cache
- See effect of read ahead
- Use memory mapped file

3 Setup

3.1 Basics

- open a terminal
- go to the exercise1 directory

```
cd /data/io/exercise1
```

3.2 Create a data file

- Use the generateTemperatureFile.py in the tools directory to create a data file. Use 10000 captors and 100 points for a 40MB file.

```
python ../tools/generateTemperatureFile.py 10000 100 datafile
```

- Be patient, it should take around 2mn
- check that the file is in the kernel cache

```
linux-fincore datafile
```

4 Understanding the page cache

4.1 Efficient access and cache usage

- Have a look at the code of countOverheated.py
  - see that there is a single seek and a single read for the main part
  - note the small read of 4 bytes at the start of the file for nbCaptors
- Drop the datafile from the page cache

```
linux-fadvise datafile POSIX_FADV_DONTNEED
```

- check the cache

```
linux-fincore datafile
```

- start iostat in another shell in order to monitor I/O. "-d 2" will display the amount of I/O every 2 seconds

```
iostat -d 2 -p sdb
```

- run the code to find overheated devices at a given time

```
python countOverheated.py datafile 40 45
```

- check the amount of I/O done from disk and the status of the cache, including the list of cached pages

```
linux-fincore datafile
linux-fincore -p datafile
```

- compute what was the minimal set of pages that needed to be in the cache
- draw some conclusions on the behavior of the cache, in particular concerning read-ahead
- keep iostat running and run the code again for the next point in time

```
python countOverheated.py datafile 41 45
```

- see in iostat output that nothing was read from disk
- see with that the cache did not change

```
linux-fincore -p datafile
```

4.2 scattered reads

- Have a look at the code of plotCaptor.py
  - see the number of small reads of size 4
  - note the small read of 4 bytes at the start of the file for nbCaptors
- Drop the datafile from the page cache

```
linux-fadvise datafile POSIX_FADV_DONTNEED
```

- check the cache

```
linux-fincore datafile
```

- start iostat in another shell in order to monitor I/O

```
iostat -d 2 -p sdb
```

- run the plotCaptor program for one captor

```
python plotCaptor.py datafile 40
```

- check the amount of I/O done from disk and the status of the cache, including the list of cached pages

```
linux-fincore datafile
linux-fincore -p datafile
```

- note that behavior of the page cache : it "understood" that we were doing scattered reads. We have still read 10% if the file for getting 0.01% of the data
- try again with 2 captors not close to each other. Do not forget to clean up the cache

```
linux-fadvise datafile POSIX_FADV_DONTNEED
python plotCaptor.py datafile 40,2040
linux-fincore datafile
linux-fincore -p datafile
```

- no big surprise here : 20% of the file was read
- try again with 3 captors in 3 contiguous pages. Again do not forget to clean up the cache

```
linux-fadvise datafile POSIX_FADV_DONTNEED
python plotCaptor.py datafile 40,1040,2040
linux-fincore datafile
linux-fincore -p datafile
```

- see that we've read 66% of the file rather that 30%. Obviously the read-ahead has been activated by the fact that we read contiguous pages.
- also see that we did not spend much more user time although we've read 3 times more data. Reading continuous blocks without seeking is efficient.

5 Mapped files

The goal here is to read big portions of a big file and see the improvement brought by mapped memory. We will use the temperature file but extend it to over 1GB by adding zeroes and we will count the number of captors exceeding a certain temperature.

5.1 Prepare a big data file

- copy datafile into bigdatafile and extend it with zeroes using dd

```
cp datafile bigdatafile
dd if=/dev/zero of=bigdatafile bs=40000 count=1 seek=40000 conv=notrunc
```

- the file still has 10000 captors, but has now 40000 points (and a bit more), out of which only the first 100 are non zero
- note by the way how the file system optimizes such sparse file

```
ls -l bigdatafile
du -h bigdatafile
du -h --apparent-size bigdatafile
```

5.2 Reading the file the C way

- have a look at the code in countOverHeated.c. This is C code but should still be quite readable. See how the file is read into a local buffer that is then used for counting the number of overheated captors
- make sure that bigdatafile is not in the page cache

```
linux-fadvise bigdatafile POSIX_FADV_DONTNEED
linux-fincore bigdatafile
```

- compile countOverHeated - you will also compile the Mapped version actually

```
make
```

- run countOverHeated for 10000 captors and see how much time it takes

```
./countOverheated bigdatafile 4-10004 45
```

- check that bigdatafile is now partially in the linux page cache. Check that all what we read is there (compute how much we read)

```
linux-fincore bigdatafile
```

- run again the same command and see the improvement due to the page cache. Deduce the amount of time it took to fill the cache versus how much the memory allocation + memory copy took

```
./countOverheated bigdatafile 4-10004 45
```

5.3 Memory mapping the file in C

- have a look at the code in countOverHeatedMapped.c. See how the file is mapped into memory and never explicitely read.
- make sure that bigdatafile is not in the page cache

```
linux-fadvise bigdatafile POSIX_FADV_DONTNEED
linux-fincore bigdatafile
```

- run the mapped version a first time with the same 10000 captors and see how much time it takes

```
./countOverheatedMapped bigdatafile 4-10004 45
```

- see the difference with the non mapped version
- check that bigdatafile is now partially in the linux page cache. Check that all what we read is there (compute how much we read)

```
linux-fincore bigdatafile
```

- run again countOverheatedMapped over the same captors

```
./countOverheatedMapped bigdatafile 4-10004 45
```

- see the improvement due to the page cache

5.4 Reading the file in python

- go back to the code of countOverheated.py, the one we've used at the beginning. It's the python version of the countOverheated.c file. No memory mapping code there
- cleanup cache for bigdatafile

```
linux-fadvise bigdatafile POSIX_FADV_DONTNEED
linux-fincore bigdatafile
```

- run the python version and check timing

```
python countOverheated.py bigdatafile 4-10004 45
```

- see how python is different from C at computing if you do not take care !
- also see that the timing for extraction is close to the non mapped C case
- check page cache status for bigdatafile

```
linux-fincore bigdatafile
```

- run the python version again and check benefit of the page cache

```
python countOverheated.py bigdatafile 4-10004 45
```

- you should see again performance close to C for data extraction (if not slightly better)
- checking the python code more carefully, you should actually be surprised. Indeed, we have an extra data copy that C does not have to do : creating a python string from the C array of bytes.

- let's understand why we do not seem to pay this copy : run again the python version with strace, looking for file handling calls

```
strace -e open,read,write,close,mmap,munmap -s 0 \
  python countOverheated.py bigdatafile 4-10004 45
```

- the output is big, but most of it is python starting, you only have to look at the last 20 lines or so. Identify the 'open("bigdatafile", O_RDONLY)' entry and read on.
- what do you notice about the file access ? mmap is used ! So we do pay the copy to the python string. But we do not pay the copy to memory as we actually use mmap.
- we can improve and avoid the copy to python by using the mmap module of python. Look at the code of countOverheatedMapped.py. This time, we map the memory directly to python.
- run countOverheatedMapped.py and check benefit compared to the initial version

```
python countOverheatedMapped.py bigdatafile 4-10004 45
```