Direct I/O for smartSD communication on Android 11 not working

Fixed Bug P2 + Add Hotlist

STATUS UPDATE No update yet. Edit

DESCRIPTION vu...@gmail.com created issue #1

Hi,

We have for a long time been using O_DIRECT with native file operations to implement a communication channel between our app and a smart card integrated in an external microSD card, a so c

I do not know if I have understood the history correctly. But it seems like this is just as in Android 4.4 when FUSE was introduced the last time, breaking O_DIRECT and causing the external secur

So far, we have understood that Google removed the O_DIRECT flag due to several file corruption issues regarding aligned I/O.

What we want to know is how to allow apps to read data from the actual card instead of from the page cache. Ensuring that writes are flushed seems quite easy, however we have not been able t

Is this currently feasible? If not, will direct I/O communication ever be possible again?
We are aware that the external microSD card slot is more and more rare, however its use is still relevant in several high security solutions to enable the use of external hardware-based secure ele

Best regards,
Johan Uppman Bruce

✓ Mentioned issues (1) ✓ Links (3)

Mentioned issues (1)

P3 O_DIRECT for hardware communication (microSD-Card) on Android 4.4 not implemented "https://issuetracker.google.com/36992007"

Links (3)

" ...y correctly. But it seems like this is just as in Android 4.4 when FUSE was introduced the last time, breaking O_DIRECT and causing the external secure elements in the form of smartSDs to be unus
"In order to try to understand if the behavior we see is expected or unintended, I have tried to dive down in the AOSP source code (even though I am not familiar with it, especially not with the filesyste
"Got it. Here's what's happening. We currently don't have the O_DIRECT check in pf_create: https://cs.android.com/android/platform/superproject/+/master:packages/providers/MediaProvider/jni/Fus

COMMENTS

en...@google.com <en...@google.com>

Reassigned to an...@google.com.

ch...@google.com <ch...@google.com> #2

Reassigned to ze...@google.com.

Hi zezeozue@,

Could you help this? Thanks.

cl...@gmail.com <cl...@gmail.com> #3

Hi,

we have the same problem with android 11.

Many of our customers use our smartSD card product on android for a fiscalization solution that is mandatory by law.
At the moment we have to tell them not to upgrade to android 11 which of course excludes them from future security updates.

We would really appreciate it if you could fix the O_DIRECT behavior on android 11 or provide us with another way to talk directly with the sdcard.

Greetings
Claus Steuer

ma...@google.com <ma...@google.com> #4

To invalidate read pages (without O_DIRECT), you could try something like:

```
posix_fadvise(fd, 0, 0, POSIX_FADV_DONTNEED);
```

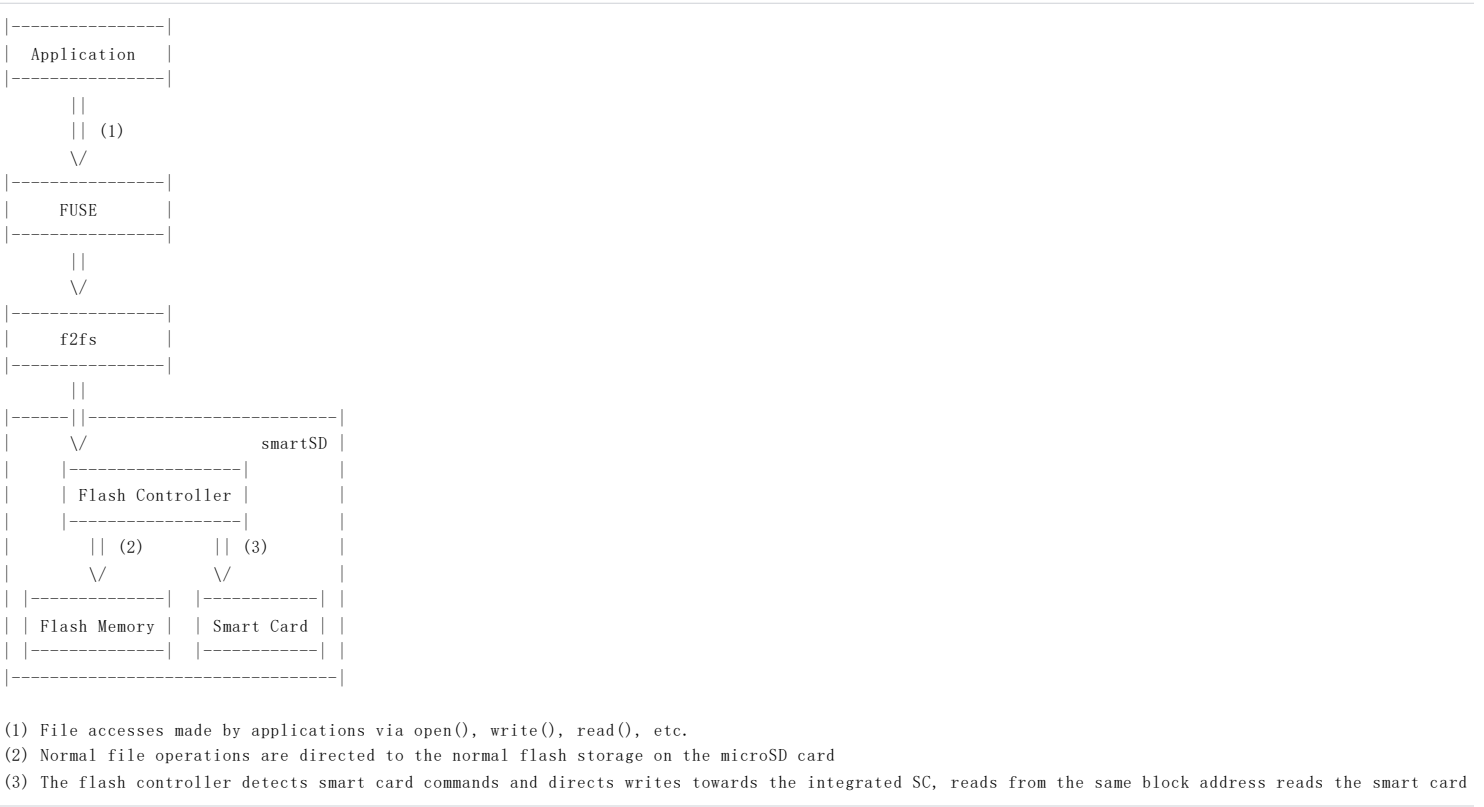
but I'm not sure Linux guarantees this to work; need to look into it more.

vu...@gmail.com <vu...@gmail.com> #5

Thanks for the tip of using `posix_fadvise` with `POSIX_FADV_DONTNEED`. However, we have already tried it and it does not work. As far as I understand Linux gives no guarantees, it should ra

As a side not, we have also tried `fdatasync()`, `O_SYNC`/`O_DSYNC`, and `mmap()` without any success regarding the read operations.

The following diagram shows how I have interpreted the situation so far.



(1) File accesses made by applications via open(), write(), read(), etc.
(2) Normal file operations are directed to the normal flash storage on the microSD card
(3) The flash controller detects smart card commands and directs writes towards the integrated SC, reads from the same block address reads the smart card

As I have understood it, both FUSE and f2fs (the lower level filesystem) has its own page cache.

Thus it could be the case that `posix_fadvise` actually helps in "getting past" the FUSE page cache. But that is just a wild guess.

We have found that using `O_DIRECT|O_EXCL|O_CREAT` when opening the file enables us to `write()` once as well as `read()` multiple times (to poll for the actual smart card response). I.e. w

If you are able to explain why we see this behavior, then maybe we could create a workaround solution based on that. But if it cannot be explained, then I guess it is "unexpected/buggy" beha

As Claus wrote, we would really appreciate if `O_DIRECT` could be fixed in Android 11. But if that is not an option, or if that will take considerable time before being usable, then finding a worka

ma...@google.com <ma...@google.com> #6

I think it's quite likely posix_fadvise() only invalidates the pages in the upper fs (FUSE) page cache. We'll think about other workarounds. One question I had is: is the file that you use to talk to

vu...@gmail.com <vu...@gmail.com> #7

From the smartSDs point of view, it does not matter what the path to the file is (except that it has to be on the card of course). However, since the applications need to be allowed to access t

In order to lower flash wear, I believe it would be preferable if it is possible for each application to keep its "communication file" open during "longer periods". I.e. it would be good if multiple a

Note that there are also scenarios when a single applications can have multiple communication files open. I.e. certain smartSD cards can have several integrated circuits (where the smart ca

However, if a specific path to a single file which is allowed O_DIRET access is what would be possible. Then that would be miles better than no such file at all =)

ze...@google.com <ze...@google.com> #8

Does your solution definitely have to be on a physical sdcard (secondary storage)? I'm asking because on primary (emulated) storage, the /sdcard/Android/{data, obb} paths are bind mounte

vu...@gmail.com <vu...@gmail.com> #9

Yes, since the solution is based on an external smartSD, the communication file has to be opened on the physical sdcard.

As I indicated in the "diagram" above, the flash controller in the physical card detects when smart card commands are written instead of normal file operations. Thus the communication file r

vu...@gmail.com <vu...@gmail.com> #10

Have anyone reflected on my earlier question on why the below described file operations seems to achieve direct I/O? (I have added some more details regarding the scenario than in the ear

```
fd = open(path, O_RDWR | O_CREAT | O_EXCL | O_DIRECT, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
struct flock lock;
lock.l_type = F_WRLCK | F_RDLCK;
lock.l_whence = SEEK_SET;
lock.l_start = 0;
lock.l_len = BLOCK_SIZE;
fcntl(fd, F_SETLK, &lock);
lseek(fd, 0, SEEK_SET);
write(fd, cmdBuf, BLOCK_SIZE);
lseek(fd, 0, SEEK_SET);
do {
  read(fd, rspBuf, BLOCK_SIZE);
} while (rspBuf[0] != CMD_IN_PROGRESS_INDICATION);
close(fd);
remove(path);
```

We have tried implementing the above logic both in a command line program running in an ADB shell as well as in an app. When using the program in the shell the path is */storage/<external*

We have inspected systrace logs which also indicate that writes and reads are actually made towards the physical card. At least as far as we can tell.

We still cannot understand how the above approach enable us to communicate with the smart card on Android 11 devices. How come we do not end up writing and reading towards the f2fs

(Previously, we opened a file as described above, but used the same file for several smart card commands. But on Android 11, that causes `EINVAL` when we try to write the second commanc

In order to try to understand if the behavior we see is expected or unintended, I have tried to dive down in the AOSP source code (even though I am not familiar with it, especially not with the

Can anyone provide any insights regarding this?

---

vu...@gmail.com <vu...@gmail.com> #11

We have found a way to enable writing multiple commands to the smart card using the same file without getting errors.

If we perform `ftruncate(fd, 0)` before each write, we avoid getting `EINVAL` while we still get direct I/O, thus enabling smart card communication.

Thus, it seems like we need zero-sized files when writing to a file which have been opened with O_DIRECT. Otherwise, we get `EINVAL`.

However, I still do not understand why we do not read or write from f2fs cache. Can it be that I have misunderstood how things work and that f2fs does not have a page cache? Or could it be

---

ze...@google.com <ze...@google.com> #12

I'm not sure why that works either. What filesystem is your sdcard? It can't be f2fs since we only support FAT based file systems on physical sdcards

---

vu...@gmail.com <vu...@gmail.com> #13

We are using FAT32 on our SD cards.

Thanks for the clarification regarding lower filesystem, I had clearly misunderstood things =)

---

ze...@google.com <ze...@google.com> #14

Based on comment #10, are you saying that grabbing the file lock (on the FUSE file) has an effect to force direct IO on the FAT file? In other words, do you get negative results without the file

---

vu...@gmail.com <vu...@gmail.com> #15

Sorry for taking so long to answer. The file lock is not required to get a direct I/O working. It is simply there to ensure no other process interferes with the communication. I.e. smart card com

---

ze...@google.com <ze...@google.com> #16

Got it. Here's what's happening. We currently don't have the O_DIRECT check in `pf_create`: https://cs.android.com/android/platform/superproject/+/master:packages/providers/MediaProvi

So during file creation the O_DIRECT flag is honoured but subsequent opens discard it as you pointed out earlier.

We are leaving this workaround as-is for now while we investigate these usecases further and explore alternatives

---

vu...@gmail.com <vu...@gmail.com> #17

Ok, thanks for the information, now we understand why it works.

Also, thanks for keeping the behavior while investigating future solutions!

---

ze...@google.com <ze...@google.com> #18

Marked as fixed.

I'll close this in the meantime, since you have a workaround.

Thanks for the patience

cl...@gmail.com <cl...@gmail.com> #19

Hi,
sorry if this is the wrong place but I do not know where else to ask.

For one of our microSD products it is not possible to use the workaround mentioned above.
The file into which we write our commands is created by the firmware of the microSD controller and mapped to a specific FAT cluster. If the host deletes and then recreates the file it chooses

A possible solution mentioned above was an allowlist for O_DIRECT access.
This would be perfect for us.

Can we expect a solution like this in the future?
If yes where would we hear about that?

Thank you

wr...@gmail.com <wr...@gmail.com> #20

I have tried this fix on a Samsung Galaxy S20+ 5G Android 11 and it does not work. Is this fix limited to certain devices?

wr...@gmail.com <wr...@gmail.com> #21

Update: the first write and read works as described in the workaround above, however, subsequent writes fail to reach the security controller. Without the ftruncate() call the EINVAL errors o

vu...@gmail.com <vu...@gmail.com> #22

I made a quick check with our implementation of the workaround on an S20+ 5G with Android 11. For me, all seemed to work.

Some details on device used:

```
Model: SM-G986B/DS
Android version: 11
Build number: RP1A.200720.012.G986BXXS8DUE4
Service provider software version: SAOMC_SM-G986B_OXM_NEE_RR_0010 NEE/NEE,NEE/EEX/NEE
Android security patch level: 1 June 2021
```

wr...@gmail.com <wr...@gmail.com> #23

Thank you for checking. What smart sd are you using?

Reid

wr...@gmail.com <wr...@gmail.com> #24

Is there more in your source than what is posted above, with the addition of the ftruncate before the write command?

vu...@gmail.com <vu...@gmail.com> #25

I have tested with a SmartSD from Swissbit.

There is more error handling and similar in my source, but the essence is the same. Have you made sure that you create a new file when opening? I.e. are you sure you are using `O_CREAT | O`

wr...@gmail.com <wr...@gmail.com> #26

Yes I am using the open() command exactly as it is in the code above. What i find is that the first command succeeds and the correct response is read. The read after the second command

hu...@gmail.com <hu...@gmail.com> #27

@all: Please consider the ftruncate() work around causes significantly higher wear for a high rate of commands (e.g. key stream generation by the smartSD during voice encryption). So a wo

I furthermore suggest to keep implementation problems for the work around out of the scope of this ticket.

The original function not requiring any flaky trick is still highly appreciated...!

hu...@gmail.com <hu...@gmail.com> #28

Did someone look already at Harmony OS? Maybe this OS is at least sticking to long time best practises and standards, thus a less infatilizing alternative in the future compared to Google/A

hu...@gmail.com <hu...@gmail.com> #29

I tested the the very latest Harmony OS 2. This works totally fine. Absolutely no modifications are required on the apk! Also Cyanogen-based forks are fine.

Is the assignee willing to repoen the ticket? The problem is still not solved on Android 11...