__cxa_atexit launch time regression with lots of non-POD globals

Fixed Bug P3 + Platform adexe s nau

STATUS UPDATE No update yet. Edit

DESCRIPTION ok...@cocone.co.jp created issue #1

When I switched to Android 11, the app started slow.
It takes an awful long time for the first splash screen to end.
I debugged it with ver.11 and it took about 30 seconds for System.loadLibrary. The library(libPocketColony.so) is our app based on cocos2dx.ver2.
On Android 10, it was about 3 seconds!

Environment.
########
Pixel3a
ndk13
target SDK:28
cocos2dx ver2
########
I changed it to ndk17, targetSDK29 to try it, but it was the same.
It was the same in both debug and release mode.

Have you hit on any possible causes of the problem?
Please help me. Thank you.

SourceCode
########
```
 public class JNIInterface {
   static {
              DebugManager.printError("======== versionup] JNIInterface  loadlibrary start ");
             System.loadLibrary("PocketColony");
              DebugManager.printError("======== versionup] JNIInterface  loadlibrary end ");
   }
    ~~
```
########

Log
########
```
 2020-09-09 18:40:11.288 14295-14295/ ======== versionup] Application onCreate
 ~~
 2020-09-09 18:40:11.388 14295-14295/ ======== versionup] JNIInterface  loadlibrary start
 2020-09-09 18:40:37.949 14295-14295/ ======== versionup] JNIInterface  loadlibrary end
 2020-09-09 18:40:37.957 14295-14295/ ======== versionup] AvatarActivity onCreate 1
 2020-09-09 18:40:37.974 14295-14295/ ======== versionup] AvatarActivity onCreateView 1
 2020-09-09 18:40:39.092 14295-14420/ ======== versionup] AvatarActivity createdSurface 1
```
########
↑ It takes 26 seconds from 「JNIInterface loadlibrary start」 to 「JNIInterface loadlibrary end」

✓ Links (10)

Links (10)

" ...urn on extra logging to see where the time is going: https://android.googlesource.com/platform/bionic/+/refs/heads/master/android-changes-for-ndk-developers.md#enable-logging-of-dlopen_dls
"https://drive.google.com/file/d/1GZoCkMZ3GrWiFGYiaZDXyDbNkFa5fxnz/view?usp=drive_web (proj.android-debug.apk)"
"http://debug.firebase.analytics.app"
"i think we'll need the bug submitter to profile their app to see where the time is going: https://developer.android.com/studio/profile/cpu-profiler"
"The problem reproduces with the app from the Play Store (jp.cocone.pocketcolony). I also downloaded the APK on #4 but haven't tried it yet."

See all related links

COMMENTS

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

Assigned to vi...@google.com.

Thank you for reporting this issue. We have shared this with our product and engineering team and will update this issue with more information as it becomes available.

en...@google.com <en...@google.com> #3

we'll need to see an apk that reproduces the problem to be able to help.

note that you can turn on extra logging to see where the time is going: https://android.googlesource.com/platform/bionic/+/refs/heads/master/android-changes-for-ndk-developers.md#ena

---

ok...@cocone.co.jp <ok...@cocone.co.jp> #4

Thank you for your reply.
I have attached the apk.
I used the following command and logged it again.
"adb shell setprop debug.ld.all dlerror,dlopen"

●Apk
https://drive.google.com/file/d/1GZoCkMZ3GrWiFGYiaZDXyDbNkFa5fxnz/view?usp=drive_web (proj.android-debug.apk)

●SourceCode
```
public class JNIInterface {
   static {
            DebugManager.printLog("======== versionup] JNIInterface  CoconePocketColony loadlibrary start ");
      System.loadLibrary("CoconePocketColony");
            DebugManager.printLog("======== versionup] JNIInterface  CoconePocketColony loadlibrary end ");
   }

   ~~
```

●Log
```
2020-09-11 16:38:06.994 8381-8381/jp.cocone.pocketcolony D/COLONY: ======== versionup] JNIInterface  CoconePocketColony loadlibrary start
      at jp.cocone.pocketcolony.jni.JNIInterface.<clinit>(JNIInterface.java:32)
2020-09-11 16:38:06.995 8381-8381/jp.cocone.pocketcolony D/linker: dlopen(name="/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-NGJzQsMFgG3RDuWHT2Pyog==
reserved_addr=0x0, reserved_size=0x0, relro_fd=0, library_fd=0, library_fd_offset=0x0, library_namespace=classloader-namespace@0x714cbe9a60], caller="/apex/com.android.art/lib64/libn
2020-09-11 16:38:06.995 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony
is_dt_needed=0
2020-09-11 16:38:06.996 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
search_linked_namespaces=1): calling open_library with realpath=
2020-09-11 16:38:06.996 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
realpath=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-NGJzQsMFgG3RDuWHT2Pyog==/lib/arm64/libCoconePocketColony.so, search_linked_namespaces=1)
2020-09-11 16:38:06.999 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: libcapsdk.so
2020-09-11 16:38:06.999 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: libGLESv2.so
2020-09-11 16:38:06.999 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: liblog.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: libz.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: libdl.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: libc.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: libm.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: load_library(ns=classloader-namespace, task=/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-N
DT_NEEDED task: libstdc++.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=libcapsdk.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=libcapsdk.so): Already loaded (by soname): /data/app/~~aHH2
NGJzQsMFgG3RDuWHT2Pyog==/lib/arm64/libcapsdk.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=libGLESv2.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=libGLESv2.so): Already loaded (by soname): /system/lib64/libGL
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=liblog.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=liblog.so): Already loaded (by soname): /system/lib64/liblog.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=libz.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=libz.so): Already loaded (by soname): /system/lib64/libz.so
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=libdl.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=libdl.so): Already loaded (by soname): /apex/com.android.runtim
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=libc.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=libc.so): Already loaded (by soname): /apex/com.android.runtim
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=libm.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=libm.so): Already loaded (by soname): /apex/com.android.runtim
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_libraries(ns=classloader-namespace): task=libstdc++.so, is_dt_needed=1
2020-09-11 16:38:07.000 8381-8381/jp.cocone.pocketcolony D/linker: find_library_internal(ns=classloader-namespace, task=libstdc++.so): Already loaded (by soname): /system/lib64/libstd
2020-09-11 16:38:07.099 8381-8464/jp.cocone.pocketcolony V/FA: App measurement collection enabled
2020-09-11 16:38:07.104 8381-8464/jp.cocone.pocketcolony V/FA: App measurement enabled for app package, google app id: jp.cocone.pocketcolony, 1:233129504622:android:735a652bc
2020-09-11 16:38:07.108 8381-8464/jp.cocone.pocketcolony I/FA: App measurement initialized, version: 31049
2020-09-11 16:38:07.108 8381-8464/jp.cocone.pocketcolony I/FA: To enable debug logging run: adb shell setprop log.tag.FA VERBOSE
2020-09-11 16:38:07.108 8381-8464/jp.cocone.pocketcolony I/FA: To enable faster debug mode event logging run:
    adb shell setprop debug.firebase.analytics.app jp.cocone.pocketcolony
2020-09-11 16:38:07.108 8381-8464/jp.cocone.pocketcolony D/FA: Debug-level message logging enabled
2020-09-11 16:38:07.308 8381-8464/jp.cocone.pocketcolony V/FA: Connecting to remote service
2020-09-11 16:38:07.481 8381-8464/jp.cocone.pocketcolony V/FA: Connection attempt already in progress
2020-09-11 16:38:07.876 8381-8464/jp.cocone.pocketcolony V/FA: Connection attempt already in progress
2020-09-11 16:38:11.042 8381-8381/jp.cocone.pocketcolony D/linker: ... dlopen calling constructors: realpath="/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-NGJzQ
soname="libCoconePocketColony.so", handle=0xd23ab5e9bbec1fe9
2020-09-11 16:38:36.899 8381-8381/jp.cocone.pocketcolony D/linker: ... dlopen successful: realpath="/data/app/~~aHH2v0EJMem1ZNindgkotQ==/jp.cocone.pocketcolony-NGJzQsMFgG3
soname="libCoconePocketColony.so", handle=0xd23ab5e9bbec1fe9
2020-09-11 16:38:36.903 8381-8381/jp.cocone.pocketcolony D/COLONY: ======== versionup] JNIInterface  CoconePocketColony loadlibrary end
      at jp.cocone.pocketcolony.jni.JNIInterface.<clinit>(JNIInterface.java:35)
```
---

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

Reassigned to rp...@google.com.

---

ok...@cocone.co.jp <ok...@cocone.co.jp> #5

Hello.
Do you have any update on this issue?

---

ok...@grenge.co.jp <ok...@grenge.co.jp> #6

Hello.

Our project faces the same issue.

I logged using the following command:

"adb shell setprop debug.ld.all dlerror, dlopen"

```
MainActivity.java
  static{
    Log.d (TAG, "System.loadLibrary start");
    System.loadLibrary ("cocos2dcpp");
    Log.d (TAG, "System.loadLibrary end");
  }
```

◆Android11 (Picel3a)

excerpt_log_android11_Picel3a_20200928

＊ This issue did not occur on Android 10 (Picel 3a).

```
2020-09-28 15: 19: 53.362 23281-23281 /? D / MainActivity: System.loadLibrary start
2020-09-28 15: 20: 32.993 23281-23281 /? D / MainActivity: System.loadLibrary end
```

System.loadLibrary took about 39 seconds.

◆Android8.1.0 (VerneeV2Pro)

excerpt_log_android8_1_VerneeV2Pro_20200928

＊ I updated the Pixel 3a to Android 11, so I output the log on the above device instead.

```
2020-09-28 15: 23: 02.126 29457-29457 /? D / MainActivity: System.loadLibrary start
2020-09-28 15: 23: 02.761 29457-29457 /? D / MainActivity: System.loadLibrary end
```

System.loadLibrary took less than a second.

◆environment.

```
ndk-r13b

cocos2dx ver3.2
    compileSdkVersion 28
    buildToolsVersion '28 .0.3'

Our application (jp.grenge.pocolondungeons)
    compileSdkVersion 29
    buildToolsVersion '29 .0.3'
```

I suspect it has something to do with `2020-09-28 15:19:58.306 23281-23301/? I/ondungeons.tes: Waiting for a blocking GC ProfileSaver`, but I don't have a clear clue.

If you need an apk, please tell me the email address you want to receive.

I will email the apk for development and all the logs.

We hope it helps you solve the problem quickly.

Thank you.

---

deleted
0 B

deleted
0 B

---

en...@google.com <en...@google.com> #7

Reassigned to ca...@google.com.

+calin for "Waiting for a blocking GC ProfileSaver"...

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

Reassigned to ma...@google.com.

I think that's a red-herring. Mathieu can you confirm?

ma...@google.com <ma...@google.com> #9

Reassigned to rp...@google.com.

That output means that the profile saver thread is blocked some 283ms waiting for GC to complete. All of the interactions between the application threads and the profile saver should be asy
reported 30s loadlibrary time.

en...@google.com <en...@google.com> #10

i think we'll need the bug submitter to profile their app to see where the time is going: https://developer.android.com/studio/profile/cpu-profiler

rp...@google.com <rp...@google.com> #11

I suspect the issue is with __cxa_atexit, but I need to study it a bit more.

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

do you already have a repro case, or do you need the bug submitter to send you an apk?

rp...@google.com <rp...@google.com> #13

The problem reproduces with the app from the Play Store (jp.cocone.pocketcolony). I also downloaded the APK on #4 but haven't tried it yet.

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

I see the app calling __cxa_atexit about 92000 times at startup. Each __cxa_atexit call registers a destructor for a single C++ global variable, to be called when the process exits cleanly (e.g. b

Unfortunately, my https://android-review.googlesource.com/c/platform/bionic/+/1231458/ change regressed the performance of __cxa_atexit when there are many C++ objects:

- In Q, __cxa_atexit uses a linked list of chunks, and calls mprotect twice on the single chunk to be modified.
- In R, __cxa_atexit calls mprotect twice on a single contiguous array of handlers. Each array entry is 2 pointers. For arm64, 92000 entries is 1.4MiB (360 pages).

Some options for a platform fix:

- mprotect only the page being modified.
- Go back to a linked list design.
- Stop using mprotect on the table. Maybe XOR a secret value into the entries instead?

Perhaps the app can work around the problem by removing or skipping the destructors on global variables:

- For a particular global variable, the `[[clang::no_destroy]]` attribute skips the destructor call.
- Pass `-fno-c++-static-destructors` to the compiler to skip the destructors for all static variables. This flag also skips destructors for `thread_local` variables. If there are `thread_l`
  `[[clang::always_destroy]]` to override the compiler flag.
- Pass `-Wexit-time-destructors` to the compiler to make it warn on every instance of an exit-time destructor, to highlight where the __cxa_atexit registrations are coming from.

For some context on the compiler flags:

- https://reviews.llvm.org/D50994
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1247r0.html

en...@google.com <en...@google.com> #15

wasn't the problem with the cocos2d middleware library? that might make it harder to work around...

i thought we had a benchmark for this? or is that just "lots of libraries" and "lots of functions", not "lots of non-POD variables"?

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

i assume we get relatively little value from write-protecting the atexit handler list because exiting isn't very common? +nnk for thoughts.

(but XOR like with setjmp buffers might be good enough and cheap enough anyway?)

rp...@google.com <rp...@google.com> #17

> i thought we had a benchmark for this? or is that just "lots of libraries" and "lots of functions", not "lots of non-POD variables"?

I added a benchmark for linker relocation. I don't know if we have anything for __cxa_atexit.

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

ah. i assume that -- since atexit() just ends up as __cxa_atexit anyway -- we don't need to do any of the annoying code generation that we do in other cases, and can just call atexit() in a loop?

---

nn...@google.com <nn...@google.com> #19

Attacks against the `atexit()` handler are some of the oldest exploitation strategies, and having read-only memory has gone a long way towards driving attackers away from `atexit()` poin information on how often this exploit strategy would be used **on Android**. `atexit()` read-only page protections have been in place since prior to the first version of Android.

Intentional exiting may not be very common from Android apps, but:

1. These protections are intended for the entire system, not just Android apps. For example, there have been privilege escalation attacks against things like /system/bin/run-as executed fr classes of processes don't exit.
2. While exiting doesn't occur in normal operation, a common error handling idiom is to write code which looks like:

```
if (something_unexpected_happens) {
  printf("OMG!");
  exit(1);
}
```

If an attacker can force the code to go down one of these error routes, an attacker may be able to force an exit() call. We shouldn't look just at the normal state of the system, but the state of

Vishwath: Do you know the current state of CFI and Android? Perhaps the XOR code and making the pages read-only is obsolete and duplicated by CFI protections? In theory CFI should prote remove all this XOR / mprotect complexity?

Elliott: If we can't find a clean way to fix the performance issues, I'm reluctantly OK with removing this protection. By the time an attacker is at the point where they can overwrite function poi

---

ok...@cocone.co.jp <ok...@cocone.co.jp> #20

Hello!
How about this issue?
The number of adnroid11 users of our app is also increasing little by little, and we are troubled.
Please help us.

---

en...@google.com <en...@google.com> #21

https://android-review.googlesource.com/c/platform/bionic/+/1464716 is the fix, but this won't be in Android 11 until the March QPR at the earliest, and since this isn't a security issue it wor

if these non-POD globals are in *your* code, see the workarounds in #14 (which will make your code even faster than it used to be). if they're in cocos2d, though, that's going to be hard to work latency).

---

rp...@google.com <rp...@google.com> #22

I verified that the platform change fixed the jp.cocone.pocketcolony app. With the fix, the splash screen appears for a little over a second. Previously, it appeared for about 19 seconds.

I don't currently have an apk for jp.grenge.pocolondungeons to test.

---

rp...@google.com <rp...@google.com> #23

This change is merged into rvc-qpr-dev now, https://googleplex-android-review.git.corp.google.com/c/platform/bionic/+/12980665.

---

rp...@google.com <rp...@google.com>

Marked as fixed.