[Insider Series] Chasing Down the Most Annoying Build System Issue

Devolutions

WHENEVER A DEVELOPER IN THE WAYK TEAM WANTS TO INCORPORATE THEIR CHANGES, A PULL REQUEST IS CREATED!

If you are a software developer, chances are that at some point you have encountered a really annoying problem which has prevented you from building your application. Most of the time, a solution can be found by Googling the error message and applying whatever good answer comes up on StackOverflow. Sometimes, however, the issue is reported by a wide variety of users over a span of multiple years, with no real fix available.

"Ibtool failed with exit code 255."

Those are the six painful words that kept coming up as the reason for the iOS build failure of Wayk Now in Jenkins, our continuous integration environment. For those not familiar with the process, here is how it works: whenever a developer in the Wayk team wants to incorporate their changes, a pull request is created.

This pull request contains the set of changes to be merged into the master branch in our Git version control system. Before accepting the changes, the code has to be reviewed by another developer. It also needs to be built successfully for all our supported platforms and pass a series of automated tests to ensure we don't introduce a regression. The whole automated pipeline is coordinated by Jenkins, which executes the iOS builds on a MacBook Pro with XCode installed.

Feeling the Pain

Every single day, the iOS build would fail with an ibtool or actool "failed with exit code 255" error. Restarting the build would usually "make it work," but sometimes the builds kept failing in a stretch. We tried everything: changing the XCode version, rebooting the machine, and force-closing specific processes or applications. Just like the desperate developers out there who encountered the issue before us, nothing would really make the problem go away for good.

That is, until we decided to take matters in our own hands. We had to fix this silly issue that has been causing problems in automated build environments everywhere since 2012. We managed to fix it, but instead of just giving the answer, we thought we should give a bit of background as to how we found it. After all, it's been causing headaches for 6 years; we think it deserves a good write-up!

TL;DR: The Quick Fix

The solution is simple: set the "IBToolNeverDeque" environment variable to "1" before invoking XCode. This environment variable will get picked up by ibtool and actool, causing a different code path to be taken that doesn't exhibit the exit code 255 problem.

Collecting Evidence

First thing's first: when and how did the problem happen? After all, builds would always work very well on a developer machine, with the failures only occurring in a continuous integration environment such as Jenkins. We believe the problem is not with Jenkins, but simply with the fact that inside an automated build environment, multiple builds can be triggered in parallel. This has the side effect of breaking a lot of tools that don't work well with multiple instances of themselves. We believe this is what happened here.

Actool and ibtool are very small executables, and their goal is to compile assets and image resources into an application. To learn more about how they work, we decided to crack them open in IDA Pro and do some reverse engineering. After a few minutes of investigation, we found that actool and ibtool are almost identical – they even use the same environment variables – so we decided to focus only on ibtool for the next steps. What we discovered next was a way to enable an internal debug log, something that could help clarify the true meaning behind the exit code 255 error. We added the following environment variables in our Jenkins environment and triggered a new build:

IBToolDebugLogLevel=4 IBToolDebugLogFile=/tmp/ibtool.log

Luckily for us, the next build failed because of ibtool, but this time we had it on tap

Finding the Culprit

The complete log was quite verbose, but what we understood from it was that ibtool normally spawns a daemon process, ibtoold, and communicates with it using a combination of shared memory, environment variables and named pipes. The parent process constructs long complicated names for multiple named pipes, creates them, then sets the names in environment variables that will be read by the daemon process, ibtoold. The parent ibtool process sends the resources to be compiled to the child process, ibtoold, and waits for completion status on one of the named pipes.

This is where things can go wrong: sometimes, ibtool fails to read the status from the named pipe and returns with exit code 255. The named pipes created by ibtool are created and destroyed on every execution. Combine that with the usage of shared memory, concurrent builds and a possible conflict in the naming of the named pipes, and you have a recipe for disaster. We didn't have an exhaustive understanding of how ibtool worked, but it was enough to figure out that this was a fragile tool prone to race conditions.

Digging for Answers

We continued our journey into ibtool by searching for all call sites to "getenv" in hopes of finding other interesting environment variables that would affect its behavior. This is where we hit the jackpot:

```
if ( getenv("IBToolNeverDeque") ) { IBExecDirectly(); } else { /* use daemon process */ }
```

This is just a simplified version of the actual code, but it shows that if the "IBToolNeverDeque" environment variable is set, ibtool will "execute ibtool directly" without using the convoluted logic with the shared memory and named pipes that cause it to sometimes fail. The complete if statement also checks for "RC_BUILDIT," "RC_XBS" and "IBCLIServerNeverDequeue," but we chose to go with "IBToolNeverDeque."

The IBExecDirectly() function still uses ibtoold, but in a different way: it creates a copy of the ibtool executable as "ibtoold" and then calls it using command-line parameters instead of using shared memory and named pipes. We don't really know why it does that, but it does look much safer. We modified Jenkins to set the "IBToolNeverDeque" environment variable to 1 and launched a new build.

Now We Are Free

The new build passed, then the next one, and the one after that. We didn't want to call it victory and jinx it, so we ran the build 5 times in a row. All of them passed. With this change, the ibtool log file became much smaller, with a single message: "ibtool executing directly." That's it, nothing else, and most importantly: no errors.

We monitored Jenkins for the next couple of days, just in case, but there was no room for error: the fix actually worked! After seeing so much red in Jenkins, all that green never looked so good.

We hope that this solution can help other developers desperately looking for an answer, and avoid the situation so accurately depicted in the "Wisdom of the Ancients" XKCD comic.