January 11, 2019

A Novel Approach to Measure Code Coverage for a Typical OS Deployment Tool

Debdipta Ghosh
Hewlett Packard Enterprise

Title: A Novel Approach to Measure Code Coverage for a Typical OS Deployment Tool

Abstract:
Code Coverage analysis measures how much of the target code executed during a test. This coverage measurement is used to describe the degree to which the source code of a program is tested by a particular test suite. HP-UX use Bullseye, a code coverage tool which is C/C++ code coverage analyzer used to improve the quality of software in vital systems. This paper will discuss why using bullseye is challenging while finding code coverage of HP-UX binaries during cold installation and how those issues are being addressed through this solution.

Problem Statement:
HP-UX use BullseyeCoverage, a third party C/C++ code coverage tool. In an up and running system, we can easily perform code coverage using bullseye. But when one HP-UX operating system is being installed, we can't use bullseye in the similar way. Understanding of the HP-UX install process and BullseyeCoverage execution steps, will help understanding the problem in a broader way and is descried below:

How Bullseye code coverage works:
Bullseye uses Intrusive approach of code coverage technique. Unlike other tools it inserts probes in source code. When we build source code with bullseye, control passes to the compiler interceptor instead of the real compiler. With Coverage Build enabled, the compiler interceptor invokes covc, which adds probes and then invokes the real compiler on the instrumented source. With Coverage Build disabled, the compiler interceptor simply passes control directly to the real compiler. Below we can see a sample snippet of instrumented code, original code is built with

BullseyeCoverage:

```
static char cov_o_data_1756f3d2[ 11];
unsigned long factorial(unsigned long f)
{if(!cov_o_data_1756f3d2[5])cov_probe_6_4_7(&cov_o_1756f3d2,5);
#line 33
{ if ((((  f == 0)&&((cov_o_data_1756f3d2[7]=1),1))||((cov_o_data_1756f3d2[6]=1),0)) )
        return 1;
        return(f * factorial(f - 1));
}}
```

In above instrumented code snippet, we can observe that in each basic function inside the program have an self-identifying probe inserted defined as *if(!cov_o_data_1756f3d2[]) cov_probe_6_4_7(&cov_o_1756f3d2,);.* While for conditional coverage, it has inserted probe *cov_o_data_1756f3d2[].* When building our project, BullseyeCoverage creates a coverage file

which is defined by the environment variable COVFILE. During execution phase, program built with BullseyeCoverage writes coverage data to file set by environmental variable COVFILE. At last phase, this coverage file is used for generating results of details code coverage. Adding probes in executable, bullseye increases memory footprint that significantly increase the program's execution space. Below table analyses different memory section of HP-UX init binary built with bullseye and aCC:

Build environment	**Total Binary Size**	**Text segment**	**Data segment**	**bss**
aCC build	4156912	3213584	66378	99384
Bullseye build	11354740	6344976	71194	142624

HP-UX install time code coverage challenges while using bullseye:

HP-UX installation is achieved by Ignite-UX tool. Bootloader calls init process of IUX which in turn loads a series of programs responsible for HP-UX installation. From the init to the reboot after the kernel is built, IUX programs whose coverage is required run in the ram filesystem (RAMFS), till the installation switches to the newly created file system on the disk. As in this install file system loaded on ram have very minimal utilities, there is no way to install BullseyeCoverage along with this filesystem. During installation time, on ram only one program can run at a time. This makes it challenging finding code coverage during cold install.

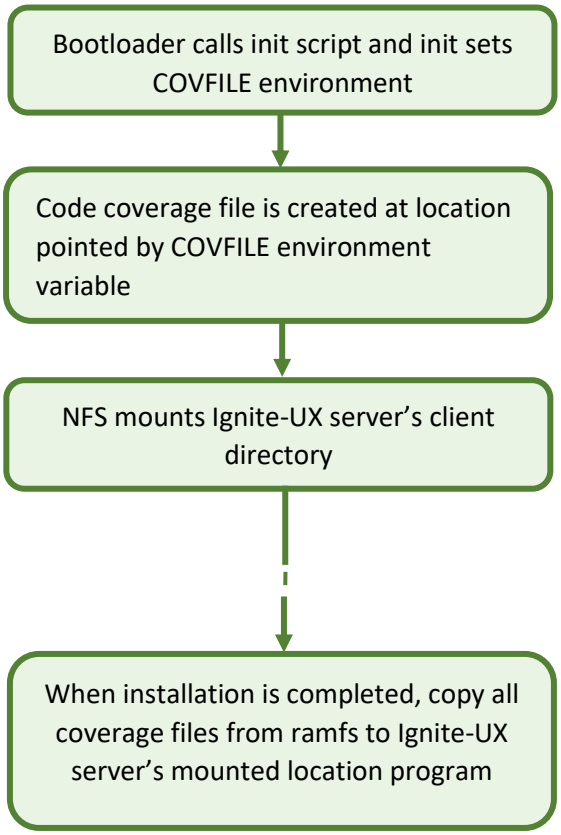
Solution Description:

Solution has three steps:

1. Bundle coverage file that is generated during build of IUX code inside IINSTALLFS. This will copy the coverage file to remote computer's RAMFS.

2. Set COVFILE path in environment variable same as path of build time coverage file which is placed at RAMFS. As init is starting program, we can set this environment variable through putenv system call in init. It is important that the next process that will be called from init also inherit this environment variable. Unix execvp system call inherits parents environment to child's environment. This makes sure that all processes have COVFILE set in env. On execution, every process will now write to build time coverage file which is placed at location pointed by environment variable COVFILE.

3. Collect this covfile before final reboot of the system. net_cfg_preg process nfs mounts clients directory making file copy possible between client and server from RAMFS. We should make sure before final reboot, we should copy coverage file back to server's client directory. Figure 1 describes different states of this solution.


Bootloader calls init script and init sets COVFILE environment
Code coverage file is created at location pointed by COVFILE environment variable
NFS mounts Ignite-UX server's client directory
When installation is completed, copy all coverage files from ramfs to Ignite-UX server's mounted location program