Using HookLogger

This article provides a simple and short introduction to HookLogger, a tool for finding memory and resource leaks in Symbian C++ code running on the Symbian Emulator.

Symbian devices are expected to be able to run for months without rebooting. One thing that can prevent this is memory leaks. As the application size grows pin pointing the source of memory leak manually becomes an arduous task, if not impossible. Fortunately there is at least one good tool - HookLogger for making this task easier.

Installing HookLogger

First of all you need to download the latest version of this tool form Symbian Developer Network Tools Section. Unzip it in some folder and install it.

Now while installing you should remember couple of points. Ideally you should install in the same drive where your SDK is installed with no whitespace character in the installation path.

You can install to any other but then you have to append the installation path at the end of your system PATH variable.

Also you can install in a path which contains blank spaces but then you have to modify `HookEUSER.pl` (located in the HookLogger installation directory) in so that all the variables that refer to path with blank space has to be put under a `\" \"`(line 74 and 77), which is asking for unnecessary trouble when you have simple work around. (Most of the Symbian tool chain has this limitation of whitespace character. Considering the phenomenal growth of Symbian ecosystem over the last few years it is really a odd limitation on the part of a developer ...)

Hooklogger may be dowloaded from: File:HookLogger.zip

Hooking euser.dll

Now to be able to track memory leaks you must "hook" the `euser.dll` (located in `Epoc32\release\winscw\udeb`). "Hooking" is simply replacing the original euser.dll with a "parasitic" one which enables the attachment of hooks so that memory allocations, leaves, thread creation etc can be logged. HookLogger tool is a really nice front end for showing these logs in more readable format.

Attachment of hook is done by `HookEUSER.cmd` batch file. But before attaching the hook the `HookEUSER.cmd` (or to be exact `HookEUSER.pl`) must know the location of `euser.dll`.

There are two ways of achieving this. The easiest way is to define an environment variable EPOCROOT and point it to the target SDK epoc32 path.

In my case the S60 3rd Edition SDK was installed at `C:\Symbian\9.2\S60_3rd_FP1` . So I have set the EPOCROOT as shown below

Figure 1: Setting EPOCROOT

The other way for defining path of `euser.dll` is to modify the `HookEUSER.pl` located in the HookLogger installation directory.

Open `HookEUSER.pl` using any text editor and replace line # 53 which is

```
my $path = "$ENV
Unknown macro: {'epocroot'}

epoc32/release/$platform/$release";
```

with

```
my $path = " C:/Symbian/9.2/S60_3rd_FP1/epoc32/release/$Platform/$release";
```

One small caveat!!! If you have more than one Symbian SDK installed and you have set EPOCROOT to point to one of them, then building form command line for other SDK(s) will fail. You should remove or modify EPOCROOT before building for other SDK.

Now open command prompt and run the command - `HookEUser WINSCW`

If you are using an EKA1 based SDK with CodeWarrior then if you have to type - `HookEUser WINSCW UDEB EKA1`

If the command is successful you will see confirmation - Target path is `\Symbian\9.2\S60_3rd_FP1\epoc32/release/WINSCW/UDEB` 1 file(s) copied. Modified `euser.dll` to hook `EUserParasite_eka2.dll`, original is `euser.orig.dll`. `Run HookEUSER` with -r to restore

Note: There is no file HookEUSER.pl in the newest edition, the menthod of hooking euser.dll is to run command "SetupHooks.cmd S60_5th_Edition_SDK_v1.0:com.nokia.s60" (S60_5th_Edition_SDK_v1.0:com.nokia.s60 is the sdk i'm using) under the hooklogger installation location. If succeed, you will see confirmation shown as Figure 1.1.

Figure 1.1: Hooking euser.dll

Hooking is complete now.

Locating the Memory Leak

Before looking for memory leak make sure that heap allocations are monitored in the HookLogger.

Run HookLogger by typing **HookLogger** in the command prompt or run it from the start menu (**Start Menu > Programs > Symbian OS Tools > HookLogger > HookLogger**).

Ensure that heap allocations are monitored in the general tab.

For producing memory leak I will use the HelloWorld application from S60 SDK. I have included the following line in the `CHelloWorldBasicAppView::ConstructL()` to produce a memory leak

```
CPeriodic* localTimerObj =
CPeriodic::NewL(CActive::EPriorityStandard);
```

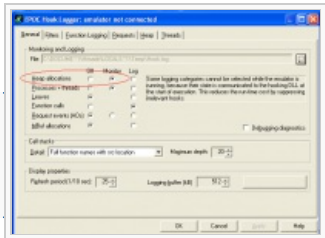

Figure 2: General Tab

If you close the application memory leak confirmation happens.

For tracing this memory leak run HookLogger. As HookLogger logs all the memory allocations for the emulator we may filter memory allocations for the application to be debugged.

For this open the Filters tab and search for the .exe file of the application. You can also add application UID. Select "Include only checked" to log memory allocations for the HelloWorld application only.

Now run the S60 emulator. HookLogger will be connected to the emulator and you will get the confirmation on the Title bar.

Before opening the HelloWorld application go to the Heap tab and press "Mark". Open the application and close it. You will get memory leak confirmation. (Please refer fig. 3). Now press "Check" in the HookLogger Heap tab.

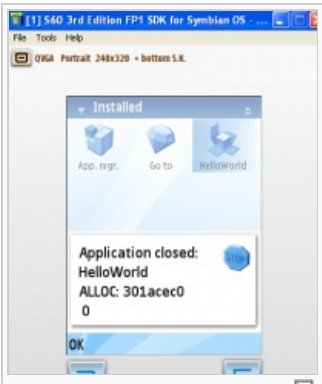

Figure 3: Emulator Memory Leak

You will get listing of memory allocations for HelloWorld application.

Search for the memory leaking pointer(in my case it is `301acec0`) from the list. Double click on the pointer or press "Alloc Details". You will get the stack. From the stack you can find source file locations for the memory allocations. If you click on a particular memory allocation it will point to the source line of allocation.

If the number of listings is too much to find the pointer, you can save the log using "Save" button from the Heap tab. Using any text editor you can find the pointer and see the stack for it.

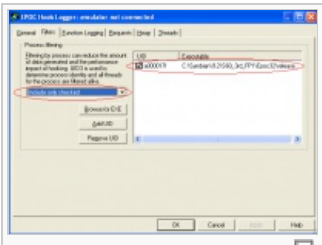

Figure 5: Emulator Connected to HookLogger

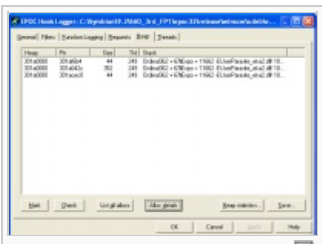

Figure 6: Heap Listings

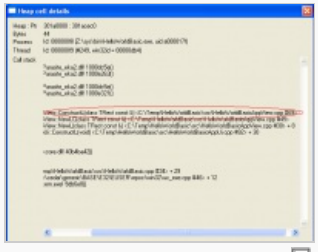

Figure 7: Stack