UEFI Development Exploration 38 – Miscellaneous Discussions on Compiling AppPkg in Ubuntu

原创 luobing4365 Posted on 2019-10-19 09:55:48 Read 1.2k Collection 2 Likes

Category Column: UEFI Development Article Tags: UEFI Programming UEFI Linux Low-level programming EDK Ubuntu compiles AppPkg

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

In the previous blog, I encountered a problem when compiling AppPkg. An error occurred during compilation. The error message was posted in the previous blog, so I will not post it here. In response to this problem, I searched for some information and did some experiments. I will record it in the form of a random talk.

1 EADK

In order to facilitate the use of standard C libraries, EDKII provides a development kit: EDK II Application Development Kit, referred to as EADK. It was first derived from EFI_ToolKit and is a sample program package for Intel to promote EFI.

As UEFI develops, the original architecture cannot be unified, and the code of EFI_ToolKit is integrated into different packages. Python and Application Development Kit are replaced and maintained by EADK. However, judging from the date on github, the last maintenance time is 2015. Is it abandoned later?

In any case, it can be used to build Uefi programs very conveniently, and can also directly use familiar C library functions. A lot of codes in previous blogs use it. It contains three packages:

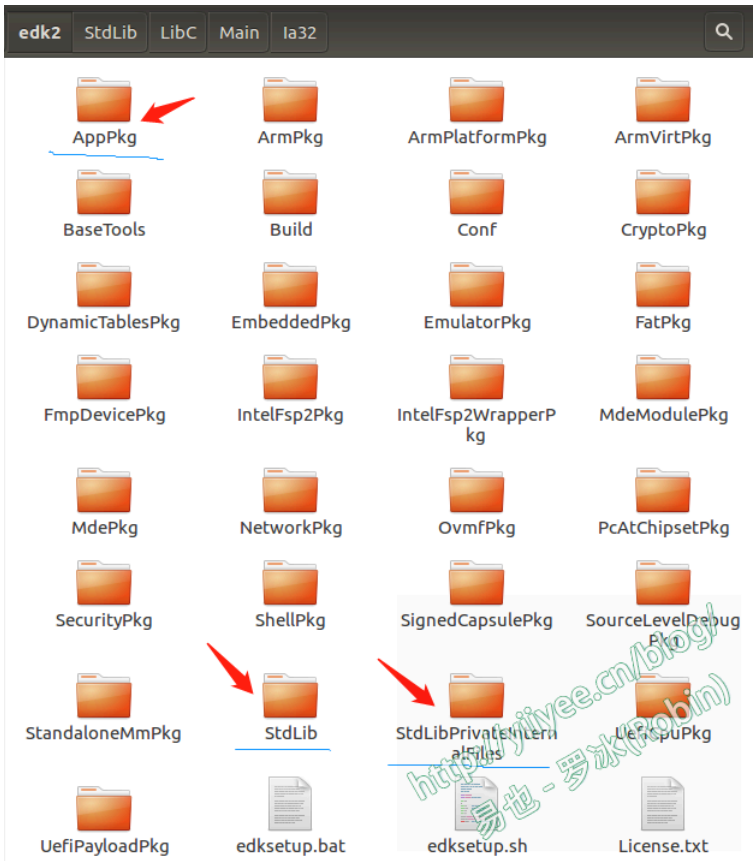

Figure 1 EADK

They are AppPkg, StdLib and StdLibPrivateInternalFiles. Among them, the functions in StdLibPrivateInternalFiles are best not to be used, and it is very likely to be cancelled in the future.

In the previous blog, the programs with main() as the main function were compiled based on AppPkg and used the StdLib library. I really wanted to compile the original code in the Linux development environment, including IA32 and X64, but unfortunately I encountered some obstacles.

The following are the EADK wiki and github download addresses:

https://github.com/tianocore/tianocore.github.io/wiki/EDKII-EADK
https://github.com/tianocore/edk2-libc

2 EDK2 compilation process

When looking for the reason why AppPkg could not be compiled, I read the compilation process of EDk2 carefully, hoping to find some clues from it.

When compiling normally, I usually follow two steps to compile:
1) Set up the compilation environment. Use edksetup.bat or edksetup.sh to set the working environment and tool path and configuration file path, etc.;
2) Compile the specified package or program. Use the build tool to compile the specified package or program. The relevant parameters can be given through the command line or set in target.txt.

The compilation process involves the processing of various files, including DEC files, DSC files, INF files, etc. The compilation flowchart is as follows:

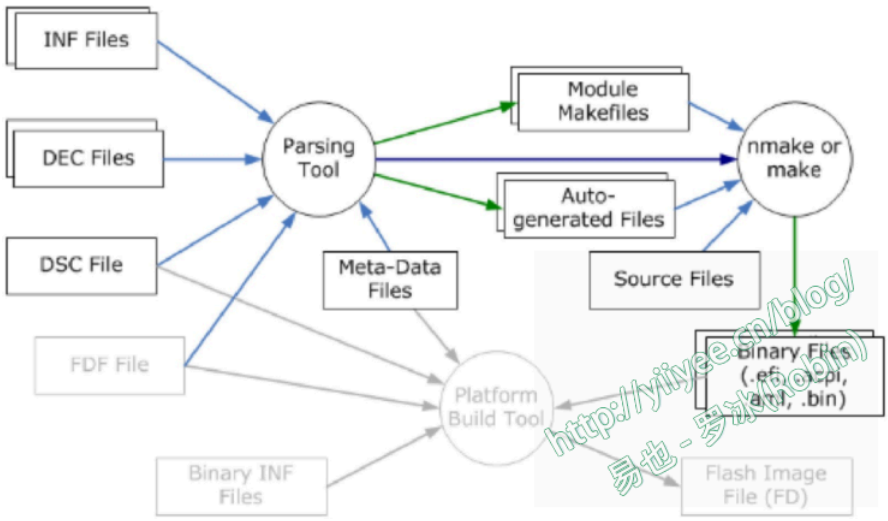

Figure 2 EDK2 platform compilation process

INF file, Module Information File, module description file. Module can be an executable file or a library file. For details, please refer to the Inf file description. There are too many details, so I will not list the contents.

DEC file, Package Declaration File, component package description file, used to support package compilation and module publishing. In this file, you can specify the header file path under [Includes] and the dependent library and path under [LibraryClasses].

DSC file, Platform Description File, platform description file.

EDK Build Tools is one of the EDK II common components. In order to use EDK II common components including EDK II Build Tools in your own module, you must write DSC and FDF configuration files in your own module.

The DSC file is used with FDF/DEC/INF and other files to generate PE32/PE32+/Coff binary files.

DSC files include:
1) EDK II Module INF Files
2) EDK Components
3) EDK Libraries (only used by EDK Components)
4) EDK II Library Class Instance Mappings (only used by EDK II Modules)
5) EDK II PCD Entries

The LibraryClasses field indicates which libraries the current Package depends on and the paths of the library description files. The Components field indicates which libraries the current Package provides externally and the paths of the library description files.

FDF file, Flash Definition File, Flash layout description file, similar to the lds link script. This file can be used to manage when writing multiple Option ROMs.

3 Why can't AppPkg be compiled?

To be precise, AppPkg fails to compile under Ubutnu 16.04 using the latest EDK2 (201908 git clone) when the target architecture is IA32.

I originally thought that some compile options in the build process caused the compilation to fail. From the error information, it seems that a library file is compiled into a PE structure. Can Gcc compile an intermediate file with a PE structure?

With doubts, I checked the compilation-related options and various dsc, dec files, and various configuration files under /conf over and over again, but I couldn't find the problem.

Back to the error message:

Relocatable linking with relocations from format pe-i386 (/home/robin/src/edk2/Build/AppPkg/DEBUG_GCC5/IA32/StdLib/LibC/LibC/OUTPUT/LibC.lib(ftol2.obj)) to format elf32-i386 (/home/robin/src/edk2/Build/AppPkg/DEBUG_GCC5/IA32/AppPkg/Applications/Sockets/SetHostName/SetHostName/DEBUG/SetHostName.dll) is not supported I read LibC.inf carefully and finally found the problem:

I read LibC.inf carefully and finally found the problem:

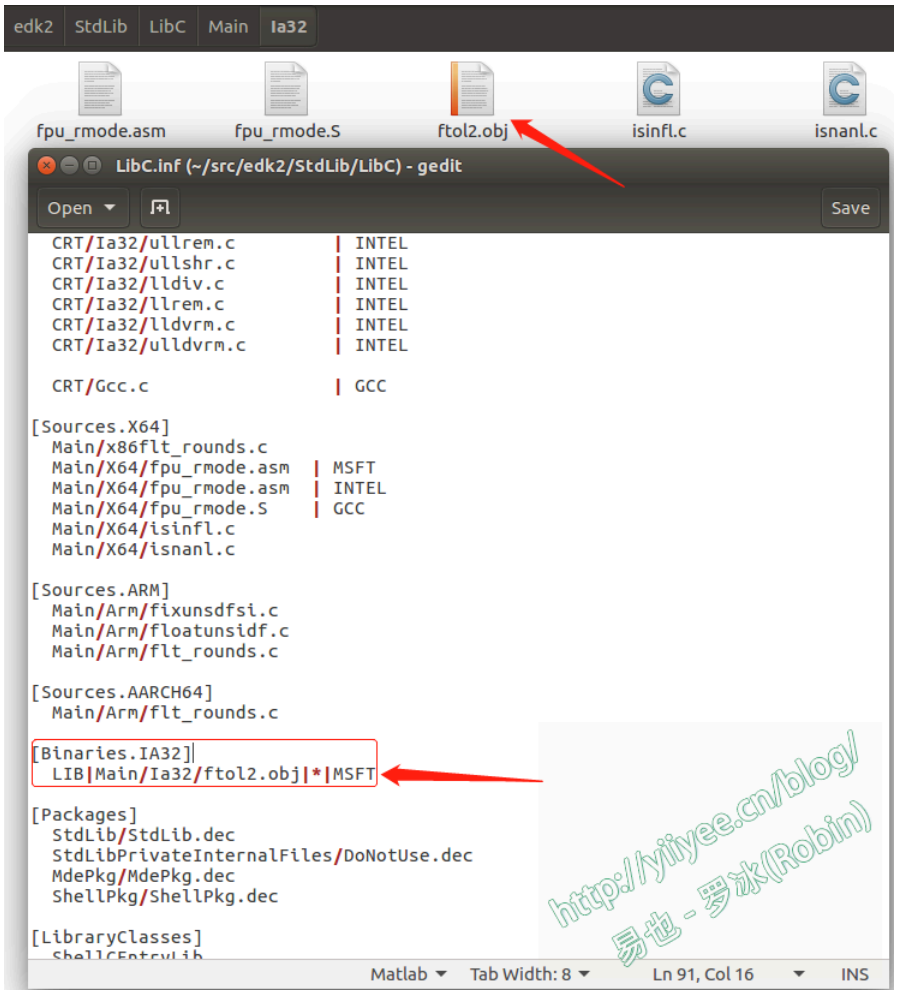

Figure 3 The culprit

In the spec file of Inf, there is a detailed description of the [Binaries] field. In this field, the parameter $(MAKE) given by the platform will not be used for compilation, but the specified tool will be used for image generation.

The fields marked in Figure 3 mean that ftol2.boj is compiled using the MSFT tool, namely Microsoft's Vs Studio. In other words, ftol2.obj is a PE structure, and can be viewed using dumpbin.exe:

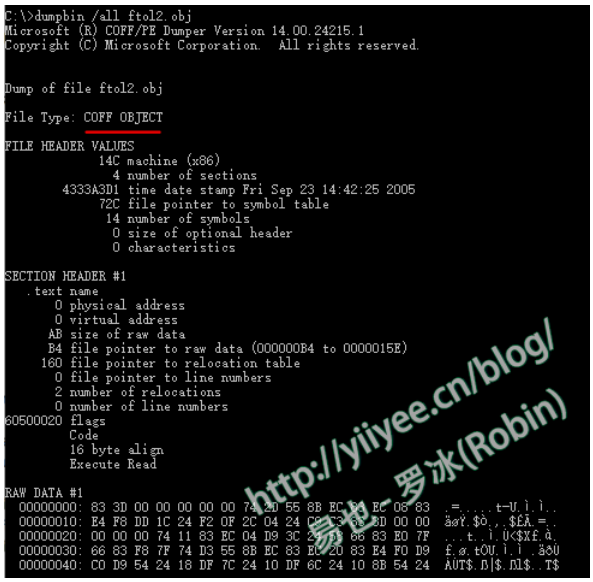

Figure 4 dumpbin reads ftol2.obj

This means that if AppPkg wants to compile for IA32, it can only be compiled under Windows using Vs studio.

At the same time, it means that the codes I wrote before, as long as they use the Stdlib library, cannot be compiled into the IA32 architecture (20200520 robin: There is now a 64-bit emulator).

However, the X64 architecture can still be compiled. It seems that I can't run the program in the emulation environment of EmulatorPkg (the emulation environment seems to only start IA32, not X64), unless I abandon the use of StdLib library.

Or using Qemu and OvmfPkg to create a virtual machine environment to test X64 programs is also a good choice. Let's use this as the title of the next blog.

4 Build the Program

When the target architecture is IA32, it seems that StdLib cannot be used. Therefore, the only two ways left are to build the entry function as UefiMain and ShellAppMain.

I have to complain that UEFI programming under Linux is stricter than that under Windows. Including the initialization of structure arrays, uppercase and lowercase file names, and the judgment of unused variables, all need to be rewritten, otherwise it will not compile.

In fact, in the previous blog code, there are already programs with UefiMian and ShellAppMain as the entry point. I found them and made some modifications in the Ubuntu compilation environment, mainly removing the parts that would prompt warnings.

For easy management, the codes I wrote are all placed under _LuoApp:

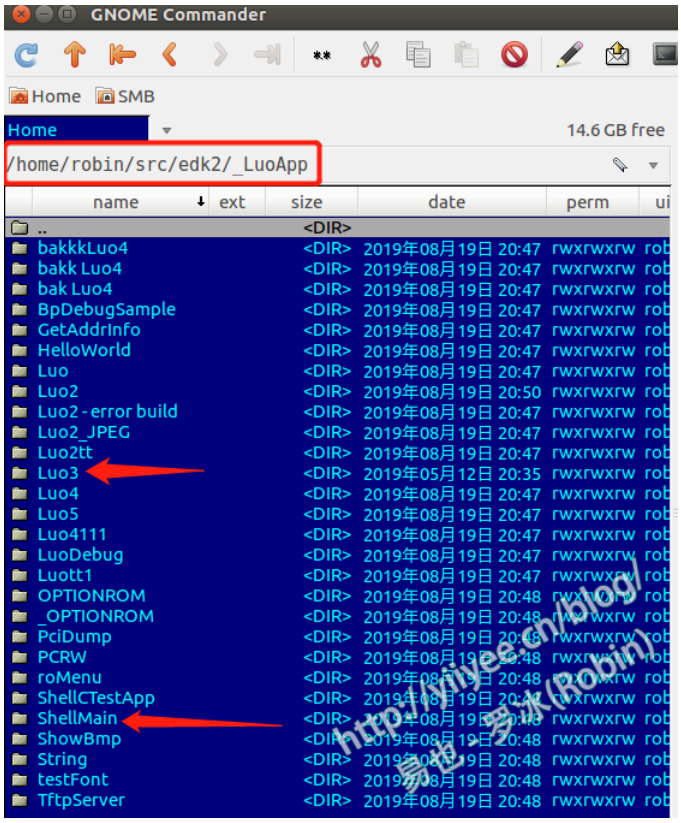

Figure 5 Example code of two types of entry functions

Under /Luo3 is the sample code with UefiMain as the entry point, and under /ShellMain is the sample code with ShellAppMain as the entry point.

Compile ShellMian process:
1) Add a line in [Components] of ShellPkg.dsc: LuoApp/ShellMain/ShellMain.inf;
2) Enter the UEFI development directory ~/src/edk2, and execute edksetup.sh in Shell;
3) Compile command: build -m _LuoApp/ShellMain/ShellMain.inf -p ShellPkg/ShellPkg.dsc -a IA32

Compile Luo3 process:
1) Add a line in [Components] of EmulatorPkg.dsc: _LuoApp/Luo3/Luo3.inf;
2) Enter the UEFI development directory ~/src/edk2, and execute edksetup.sh in Shell; |
3) Compile command: build -m _LuoApp/ Luo3/ Luo3.inf -p EmulatorPkg/EmulatorPkg.dsc -a IA32

The compiled efi files can all be run in the emulation environment of EmulatorPkg. Take Luo3 as an example:

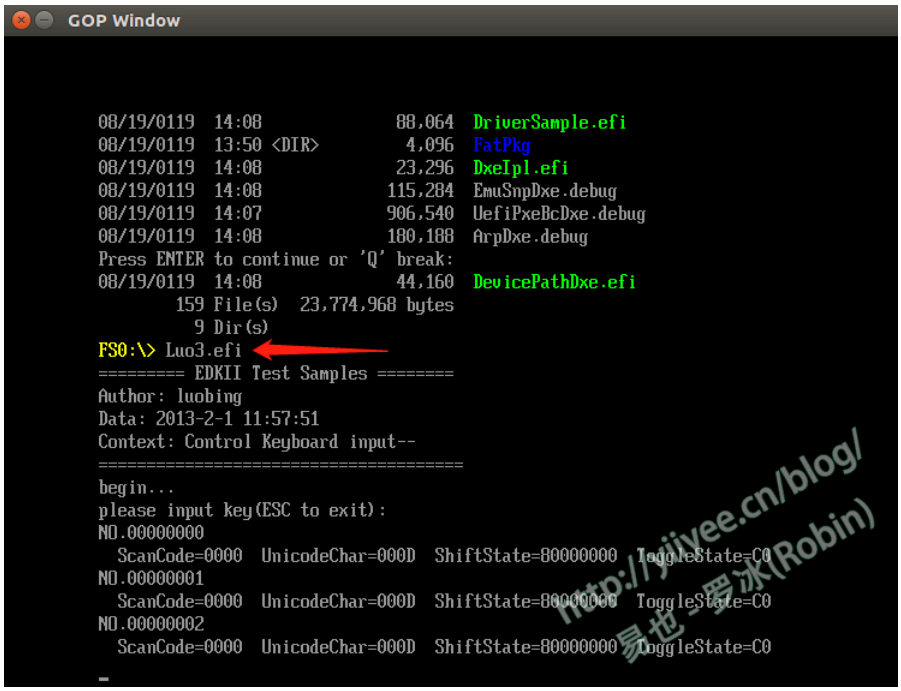

Figure 6 Demonstration

During the writing process, I found that there were always problems with the Protocol for reading and writing files, but the same code had no problems when compiled and run under UDK2018 on Windows 10.

This is the joy of exploration: constantly discovering new problems, solving them, and having fun!

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located at: / 24 Ubuntu-UEFI EntryMain