Memory Management (5) DXE memory management process

原创 Pedro Posted on 2016-12-30 23:10:28 Read 2.7k Collection 6 Likes 6 Copyright CC 4.0 BY-SA
Category Column: UEFI-MemoryManagement Article Tags: Memory Management

UEFI-MemoryMana… This column includes this content 8 articles Subscribe to our column

摘要 This article discusses the memory management strategy in UEFI firmware, including the management of key address segments such as reserved memory, ACPINVS, ACPIReclaim, etc. It introduces how PEIMRC manages these reserved memories through the Hob structure, and explains in detail how the DXE stage uses this information to allocate memory.

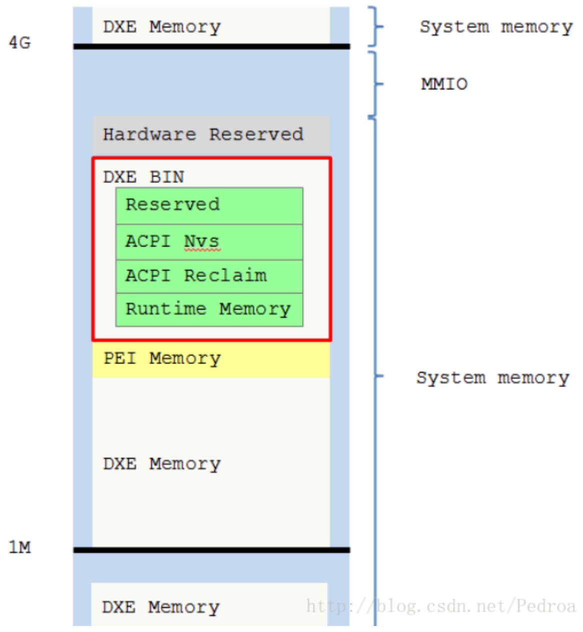
The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

Whether it is OS or UEFI Firmware, they all have their own application areas. In order to better fulfill their roles, they will use strategies and methods that suit them. Since UEFI Firmware is modern software, some of its ideas are similar to those of current mainstream software, so some people will compare UEFI Firmware and OS. Comparison is possible, but first of all, you must be very accurate in the application scenarios, technical limitations, etc. of the content you compare. For example, Firmware is the rough house before your house is renovated. It will provide you with water, electricity, and the basic resources you need. The specific decoration design is the work of the household owner OS. There are connections and similarities between rough houses and renovated houses.

UEFI memory management strategy
Because the OS has requirements, when S4 resume returns, some contents in the memory must be restored from the disk. Therefore, when the OS asks the firmware to report the memory, some contents cannot be changed. If they are changed, S4 resume fails. So which address segments cannot be changed?

- Reserved memory
- ACPI NVS
- ACPI Reclaim
- Runtime data
- RUntime code

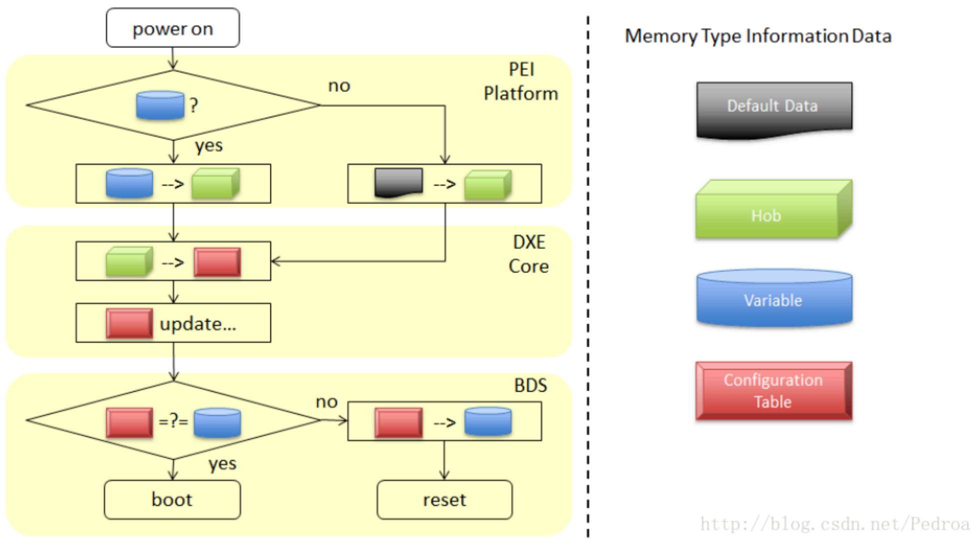
The above addresses are set by the OS and cannot be changed after S4. The strategy given by UEFI firmware is to simply allocate a large block to each of the above memory types. The name given is bin. As shown below.



This figure is a screenshot of Intel white paper BIOS memory map.

Look at the picture carefully, and look at the code with this picture, you will find that there is a reason for the code to be written like this, of course, this picture is just a reference. First of all, each IBV will reserve the size and type of each reserved memory according to the different board designs. Some BIN files will have BS data/code. The reason is to allocate as much as possible in this bin to reduce memory fragmentation. The following is a brief introduction to the general process:

1. PEI MRC will create a Hob based on the information of each reserved memory.
2. The DXE stage obtains this Hob when Dxe Core initializes memory services, and then allocates memory to the bin files of the relevant type based on the information in the Hob (default value).
3. When other codes in the firmware need to allocate memory resources, they will first allocate memory to the corresponding bin.
4. In fact, at this time, the firmware will also maintain a global table. This table is placed on the system table and its name is Efi memory type information. It records the information of each memory actually used by the firmware this time.
5. In the ready to boot event of the bds stage, there is a function
   (BdsSetMemoryTypeInformationVariable)
   in which it will compare the actual size of each memory used with the default system allocation before. If it is found that the actual size of each memory type used is smaller than the default, then continue to boot. If it is found that the actual size of the memory type used is larger than the default size, the reset system function will be called. At the next reset, the firmware will set the bin file size of the corresponding memory type according to the actual usage last time.



This picture is a screenshot of the Intel white paper BIOS memory map.

You may understand the picture above. Once you understand the process, the code will not be so obscure.

Records (the following are personal records):

- GetPlatformMemorySize ()/Memory.c builds the memory information hob
- CoreInitializeMemoryServices ()/DxeMain.c records the size of each memory type bin file and determines the base information when allocating it and puts it in the mMemoryTypeStatistics table.
- BdsSetMemoryTypeInformationVariable()/BdsMisc.c compares the current actual memory usage with the default memory type size. If the default reserved size can meet the actual usage, then boot directly. Otherwise, reset the system and set the memory bin file size to the size of the last actual memory used next time.