nent Exploration 99 – Screen capture tool under UEFI Shell

ed on 2021-08-26 15:47:32 Read 2.7k Collection 10 Likes 1

FI Development Article Tags: uefi UEFI Programming Practice bios Low-level application development

nt This column includes this content 503 Subscribe

ıg https://blog.csdn.net/luobing4365)

can only be tested in the UEFI Shell of the actual machine. For example, the diskdump program in the previous article cannot be run in the simulator.

lator, you can directly use various screenshot software to take screenshots, and the images are still relatively clear. When running on the actual machi take photos. I am still very clear about how bad my photography skills are. The photos taken can only barely show the running status of the program. The later.

ens a lot, and we have to find a way to solve it.

orning, an idea came to my mind: why not write a software to take screenshots under UEFI Shell?

ms that need to be solved, mainly including the following:
ı must be able to reside in the memory to allow other programs to call it out while running;
call out the screenshot program running in the background at any time;
n the format of a bmp image on the hard disk or USB flash drive.

e TSR program in the previous DOS system. There were a large number of such programs in the early DOS.

olementation method, I think this screenshot software should be implemented under UEFI. The resident memory should be implemented through the UEF s BMP images on the screen. The image processing code written before can be slightly modified.

ht about it, the more I felt like I had seen this idea before.

pment logs and found that Microsoft's Github library provided software with the same function. I had seen it a long time ago, but I had never compiled an

n the mu_plus library on Github, the address of the library is: https://github.com/microsoft/mu_plus.git .

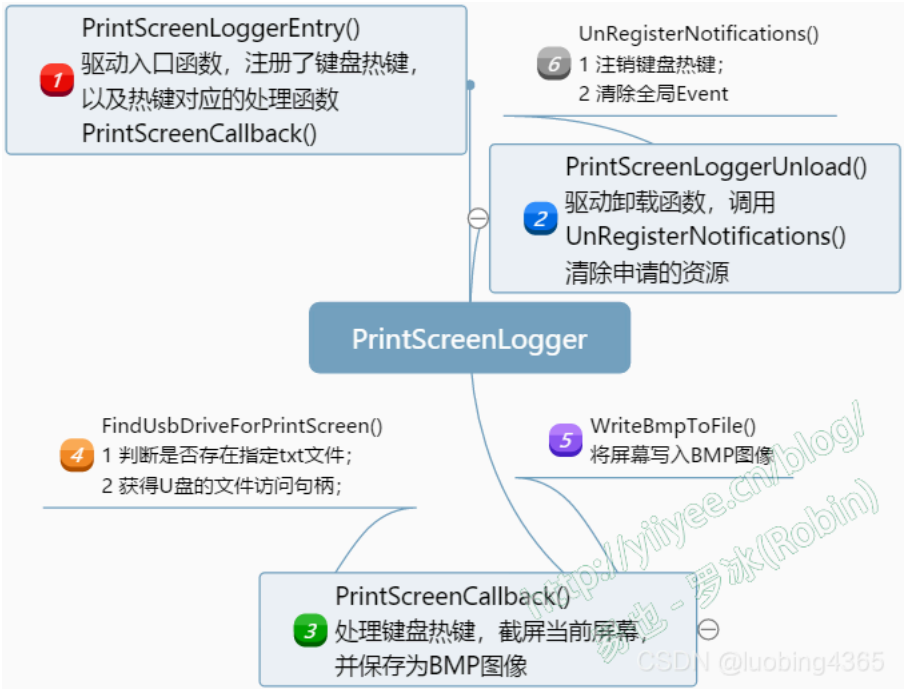
is located in mu_plus's MsGraphicsPkg and is named PrintScreenLogger.

ere is no need to write more. This article tries to understand its implementation principle and test it in a real environment.

r code structure

int of view, it is pretty much what you would expect. PrintScreenLogger uses the UEFI driver to allow the program to reside in memory.

n in Figure 1.



er program structure

hree functions : PrintScreenLoggerEntry(), PrintScreenLoggerUnload(), and PrintScreenCallback(). The global event gTimerEvent is used for synchron
ge to disk.

Entry()

f the driver. In this function, two hotkeys are registered: left Ctrl+PrtScn and right Ctrl+PrtScn, as well as the corresponding hotkey processing function Pr

TIMER type event gTimerEvent is created in the function. The implementation code of the function is as follows:

```
 nt for this driver.

 eHandle     Image handle of this driver.
 emTable     Pointer to the system table.

 STATUS      Always returns EFI_SUCCESS.



rEntry (
          ImageHandle,
TABLE     *SystemTable


    Status = EFI_NOT_FOUND;
    i;

_LOAD, "%a: enter...\n", __FUNCTION__));


 cess to ConSplitter's TextInputEx protocol

soleInHandle != NULL) {
 gBS->OpenProtocol (
        gST->ConsoleInHandle,
        &gEfiSimpleTextInputExProtocolGuid,
        (VOID **) &gTxtInEx,
        ImageHandle,
        NULL,
        EFI_OPEN_PROTOCOL_BY_HANDLE_PROTOCOL);

R(Status)) {
EBUG_ERROR, "%a: Unable to access TextInputEx protocol. Code = %r\n", __FUNCTION__, Status));
```

```
Register for PrtScn callbacks

: 0; i < NUMBER_KEY_NOTIFIES; i++) {
tus = gTxtInEx->RegisterKeyNotify (
          gTxtInEx,
          &gPrtScnKeys[i].KeyData,
          PrintScreenCallback,
          &gPrtScnKeys[i].NotifyHandle);
EFI_ERROR (Status)) {
 DEBUG ((DEBUG_ERROR, "%a: Error registering key %d. Code = %r\n", __FUNCTION__, i, Status));
 break;



_ERROR(Status)) {

. Create the PrtScn hold off timer

us = gBS->CreateEvent(
                EVT_TIMER,
                0,
                NULL,
                NULL,
                &gTimerEvent);
!EFI_ERROR(Status)) {
//
// 4. Place event into the signaled state indicating PrtScn is active.
//
Status = gBS->SignalEvent (gTimerEvent);



_ERROR(Status)) {
G((DEBUG_INFO, "%a: exit. Ready for Ctl-PrtScn operation\n", __FUNCTION__));

gisterNotifications ();
G((DEBUG_ERROR, "%a: exit with errors. Ctl-PrtScn not operational. Code=%r\n", __FUNCTION__, Status));



UCCESS;
```

Jnload()

) is the opposite of PrintScreenLoggerEntry(). It is the driver uninstallation function. It unregisters the previously applied keyboard hotkey and deletes the ntation code is as follows:

```
eanup the driver on unload.

t           Not Used.
ext         Not Used.



rUnload (
   ImageHandle


tifications ();
UCCESS;
```

```
 In callbacks and end the timer



.cations (


.;
status;

 i < NUMBER_KEY_NOTIFIES; i++) {
ScnKeys[i].NotifyHandle != NULL) {
us = gTxtInEx->UnregisterKeyNotify (gTxtInEx,  gPrtScnKeys[i].NotifyHandle);
 EFI_ERROR(Status)) {
 DEBUG((DEBUG_ERROR, "%a: Unable to uninstall TxtIn Notify. Code = %r\n", __FUNCTION__, Status));




ent != NULL) {
Timer (gTimerEvent, TimerCancel, 0);
seEvent (gTimerEvent);
```

k()

enshot are concentrated in this function. First post the implementation of the function:

```
 t key notification

          A pointer to a buffer that is filled in with the keystroke
          information for the key that was pressed.

 UCCESS   Always - Return code is not used by SimpleText providers.



ack (
A     *KeyData


OTOCOL *FileHandle;
        Index;
        PrtScrnFileName[] = L"PrtScreen####.bmp";
        Status;
        Status2;
OTOCOL *VolumeHandle;

egister two keys - LeftCtrl-PrtScn and RightCtrl-PrtScn.
int screen function if this function is called.
_INFO,"%a: Starting PrintScreen capture. Sc=%x, Uc=%x, Sh=%x, Ts=%x\n",
ON__,
 >Key.ScanCode,
 >Key.UnicodeChar,
 >KeyState.KeyShiftState,
 >KeyState.KeyToggleState));

->CheckEvent (gTimerEvent);

= EFI_NOT_READY) {
EBUG_INFO,"Print Screen request ignored\n"));
FI_SUCCESS;



 suitable USB drive - one that has PrintScreenEnable.txt on it.
```

```
ıdUsbDriveForPrintScreen(&VolumeHandle);

ıOR(Status)) {

 nd the first value of PrtScreen#### that is available

 0;


ıx++;
 Index > MAX_PRINT_SCREEN_FILES) {
 goto Exit;


ıodeSPrint (PrtScrnFileName, sizeof (PrtScrnFileName), L"PrtScreen%04d.bmp", Index);
ıus = VolumeHandle->Open (VolumeHandle, &FileHandle, PrtScrnFileName, EFI_FILE_MODE_READ, 0);
 !EFI_ERROR(Status)) {
 if (Index % PRINT_SCREEN_DEBUG_WARNING == 0) {
     DEBUG((DEBUG_INFO,"%a: File %s exists.  Trying again\n", __FUNCTION__, PrtScrnFileName));
 }
 Status2 = FileHandle->Close (FileHandle);
 if (EFI_ERROR(Status2)) {
     DEBUG((DEBUG_ERROR,"%a: Error closing File Handle. Code = %r\n", __FUNCTION__, Status2));
 }
 continue;

 Status == EFI_NOT_FOUND) {
 break;

 (TRUE);


ıeate the new file that will contain the bitmap

ı VolumeHandle->Open (VolumeHandle, &FileHandle, PrtScrnFileName, EFI_FILE_MODE_READ | EFI_FILE_MODE_WRITE | EFI_FILE_MODE_CREATE
_ERROR(Status)) {
IG((DEBUG_ERROR,"%a: Unable to create file %s. Code = %r\n", __FUNCTION__, PrtScrnFileName, Status));
ı Exit;



ıite the contents of the display to the new file

ı WriteBmpToFile (FileHandle);
_ERROR(Status)) {
IG((DEBUG_INFO,"%a: Screen captured to file %s.\n", __FUNCTION__, PrtScrnFileName));


 ose the bitmap file

 = FileHandle->Close (FileHandle);
_ERROR(Status2)) {
IG((DEBUG_ERROR,"%a: Error closing bit map file %s. Code = %r\n", __FUNCTION__, PrtScrnFileName, Status2));



 ose the USB volume

 = VolumeHandle->Close (VolumeHandle);
_ERROR(Status2)) {
IG((DEBUG_ERROR,"%a: Error closing Vol Handle. Code = %r\n", __FUNCTION__, Status2));



 ture PrtScn requests for some period.  This is due to the make
ı of PrtScn being identical, and it takes a few seconds to complete
 screen capture.
ı->SetTimer (gTimerEvent, TimerRelative, PRINT_SCREEN_DELAY);
```

```
;UCCESS;
```

hether the current storage device is a USB drive and whether there is a file PrintScreenEnable.txt in its root directory. This is achieved through the functic en().

JsbDriveForPrintScreen() function will return a pointer variable of type EFI_FILE_PROTOCOL, which will be used as the file Protocol instance for subseq

ıe root directory of the USB drive to see if PrtScreen####.bmp exists (#### value range is 0000 to 0512). This is a traversal search process, and during tl ated (sequential search, for example, if PrtScreen0000.bmp to PrtScreen0015.bmp exist, then PrtScreen0016.bmp is created).

:all WriteBmpToFile() to save the current screenshot into the created bmp file.

ıe end of the function, a 3-second trigger time is set for gTimerEvent. This time is used to allow the device to complete the storage of the BMP file to prev ·e the file is saved.

'rintScreen() and WriteBmpToFile() called in the function, please view the source code in the project given at the end of the article . The implementatior

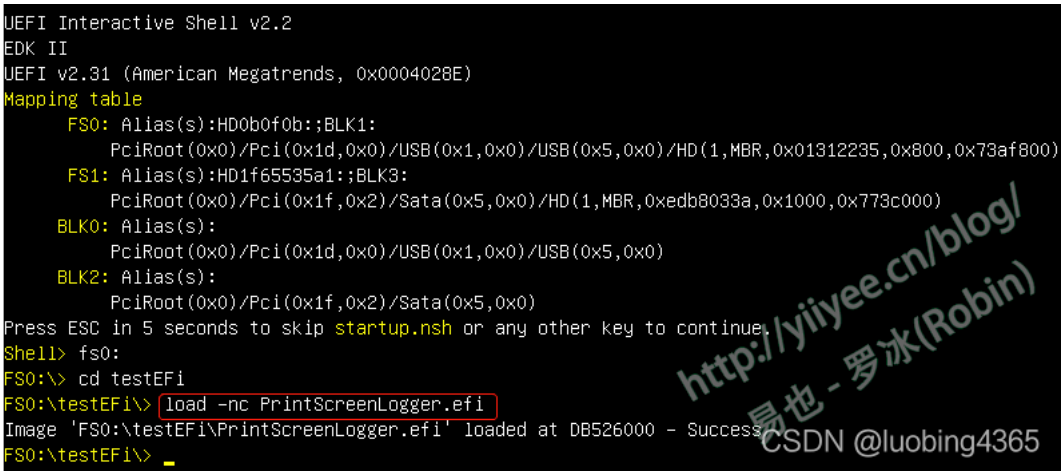
out hard disk access Diskdump, and the pictures I took were very bad, which I was not satisfied with (Figure 2 of the previous article UEFI Development a ınt to experiment with a new way of taking screenshots.

sues in the original PrintScreenLogger that prevented it from compiling. These were mainly header file inclusions and a few cast issues. I have now modif · it using the following command:

```
!>build -p RobinPkg\RobinPkg.dsc -m RobinPkg\Drivers\PrintScreenLogger\PrintScreenLogger.inf -a X64
```

e with UEFI Shell, and create PrintScreenEnable.txt in the root directory of the USB flash drive. The file content should be empty.

PrintScreenLogger.efi, as shown in Figure 2.



·nshot tool

you can use Ctrl+PstScn to take a screenshot. The image will be stored in the root directory of the USB drive with the name PrtScreen####.bmp (#### ra red using this method.

n in the previous article, and the screenshot is shown in Figure 3:

kdump running

sult chart at the end of the previous blog, this chart is obviously clearer. Figure 3 is composed of two pictures spliced together, mainly because one screer ng, I did not do any beautification, just deleted the redundant content.

ol to take screenshots under UEFI.

ds, I think I can make a simple screen recording software to record the operations under UEFI Shell. Of course, it can also be simply processed to collect econd). The pictures are then integrated and processed frame by frame using software to get the operation process.

e some minor changes to meet this requirement.

of this article is as follows:

e.com/luobing4365/uefi-explorer
/FF RobinPkg/RobinPkg/Drivers/PrintScreenLogger

about Us Careers Business Cooperation Seeking coverage 400-660-0108 kefu@csdn.net Online Customer Service Working hours 8:30-22:00
Public Security Registration Number 11010502030143 Beijing ICP No. 19004658 Beijing Internet Publishing House [2020] No. 1039-165
Commercial website registration information Beijing Internet Illegal and Harmful Information Reporting Center Parental Control
Online 110 Alarm Service China Internet Reporting Center Chrome Store Download Account Management Specifications
Copyright and Disclaimer Copyright Complaints Publication License Business license
©1999-2025 Beijing Innovation Lezhi Network Technology Co., Ltd.