شناسنامه‌ی گزارش	
عنوان گزارش	راهنمای راه اندازی Coreboot و Slimbootloader روی Qemu + اجرای محیط UEFI
چکیده	

شماره نسخه	تاریخ تهیه	نگارش/ویرایش کننده	تغییرات اعمال شده	تاریخ تاییدیه	مدیر تاییدکننده
۱	۱۴۰۱/۰۶/۰۶	علی حسین قربان	بررسی SlimBootloader بررسی راه اندازی Coreboot روی Qemu تهیه مستند		
۲		کیارش کرکی	رفع باگ بالا نیامدن گرافیک EDK2 روی Qemu		
۳					
۴					
۵					

فهرست

فصل ۱ مقدمه

در فصل دوم این سند ابتدا نحوه نصب و راه اندازی coreboot روی شبیه ساز Qemu را بررسی کرده ایم. سپس آن را با بار کاری ساده Coreinfo که صرفا ویژگی های پردازنده را نمایش می دهد اجرا کردیم. و در بخش پایانی فصل دوم، بارکاری Tianocore یا EDK2 که محیط UEFI را فراهم میکند را جایگزین Coreinfo کرده و ر.ی محیط Qemu شبیه سازی را انجام دادیم.

در فصل سوم این سند، در نظر داشتیم که کارهای مشابه را برای ثابت افزار SlimBootloader بحای coreboot نیز تکرار کنیم. اما یکی از مشکلاتی که به آن برخوردیم و تا زمان تهیه این گزارش هنوز رفع نشده است، مواجه شدن با خطایی است که هنگام تغییر معماری سیستم به X64 رخ می دهد. این خطا هنگامی که از معماری IA32 استفاده می کنیم وجود ندارد.

فصل ۲ راه اندازی EDK2 با coreboot در شبیه ساز Qemu

بیشتر اطلاعات این بخش از راهنمایی که در سایت زیر قرار دارد استفاده شده است.

https://doc.coreboot.org/tutorial/part1.html

سند زیر روی سیستمی با سیستم عامل ubuntu 2018.04 تست شده است

۲-۱ نصب نرم افزار های پیش نیاز و دانلود و کامپایل سورس coreboot

۱. در ابتدا مطمین شوید که نرم افزارهای زیر نصب شده باشند

```
sudo apt-get update

sudo apt-get install build-essential git autopoint uuid-dev cmake libfreetype6-dev
libfontconfig1-dev xclip unifont autotools-dev autoconf automake iasl python3-distutils
qemu bison curl flex git gnat libncurses5-dev m4 zlib1g-dev -y

sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.6 10
```

۲. همچنین پکیج nasm ورژن ۲.۱۵.۰۵ را به روش زیر نصب نمایید. اگر nasm را با دستور apt-get نصب نمایید برای ubuntu 2018.04 ممکن است ورژن های قدیمی تر مثل ۲.۱۳.۰۲ نصب شود.

```
mkdir nasm && cd nasm
wget https://www.nasm.us/pub/nasm/releasebuilds/2.15.05/nasm-2.15.05.tar.xz
tar -xf nasm-2.15.05.tar.xz --strip-components=1
/.configure --prefix=/usr && make
sudo make install
nasm --version
        //NASM version 2.15.05 compiled on DATE...
```

۳. درنهایت سیستم را ریست نمایید.

٤. مرحله بعد دریافت کد coreboot از سورس github است.

```
git clone https://github.com/coreboot/coreboot.git
cd coreboot
git submodule update --init --checkout
```

۵. در مرحله بعد نیاز است تا coreboot toolchain را بسازیم. دو نکته مهم در این بخش وجود دارد. نکته اول آن است که در این مرحله باید VPN روشن باشد. برای VPN در لینوکس می توانید از WIndscribe استفاده کنید (https://windscribe.com/guides/linux). نکته دوم آن است که نمیتوان Toolchain را برای معماری x64 ساخت و خطا می دهد، و همانطور که در سایت (https://doc.coreboot.org/tutorial/part1.html) توضیح داده شده است باید برای معماری i386 ساخت که این معماری پلتفرم های x64 را نیز ساپورت می کند.

```
make crossgcc-i386 CPUS=3   # build i386 toolchain
```

۶. بعد از اجرای این دستور می توانید VPN را خاموش نمایید.

۲-۲ ساخت و اجرای payload پیش فرض (coreinfo)

در این بخش مراحل مورد نیاز برای اینکه coreboot بار کاری ساده coreinfo را روی شبیه ساز qemu اجرا کند بررسی شده است. لطفا در نظر داشته باشید که این بخش، پیشنیاز بخش بعد نیست و می توان از آن عبور کرد.

۷. برای ساخت بار کاری coreinfo دستورات زیر را اجرا می کنیم

```
make -C payloads/coreinfo olddefconfig
make -C payloads/coreinfo
```

۸. سپس menuconfig را اجرا کرده تا بتوانیم تنظیمات پلتفرم را انجام دهیم

```
make menuconfig
```

۹. در پنجره باز شده تغییرات زیر را اعمال می کنیم

- گزینه Mainborad را انتخاب کرده
  - مقدار Mainboard vendor باید Emulation باشد.
  - مقدار Mainboard model باید QEMU x86 i440fx/piix4 باشد.
- گزینه General Setup را انتخاب کرده
  - گزینه Allow building with any toolchain باید انتخاب شده باشد
- گزینه Payload را انتخاب کرده
  - گزینه Add a Payload را انتخاب کرده و در آن گزینه An Elf executable payload را انتخاب کنید
  - گزینه Payload path and filename را انتخاب کرده و مقدار payloads/coreinfo/build/coreinfo.elf را وارد نمایید.

۱۰. از menuconfig خارج شوید و تغییرات را ذخیره کنید

```
make savedefconfig
```

۱۱. در نهایت با دستور زیر پروژه را بسازید

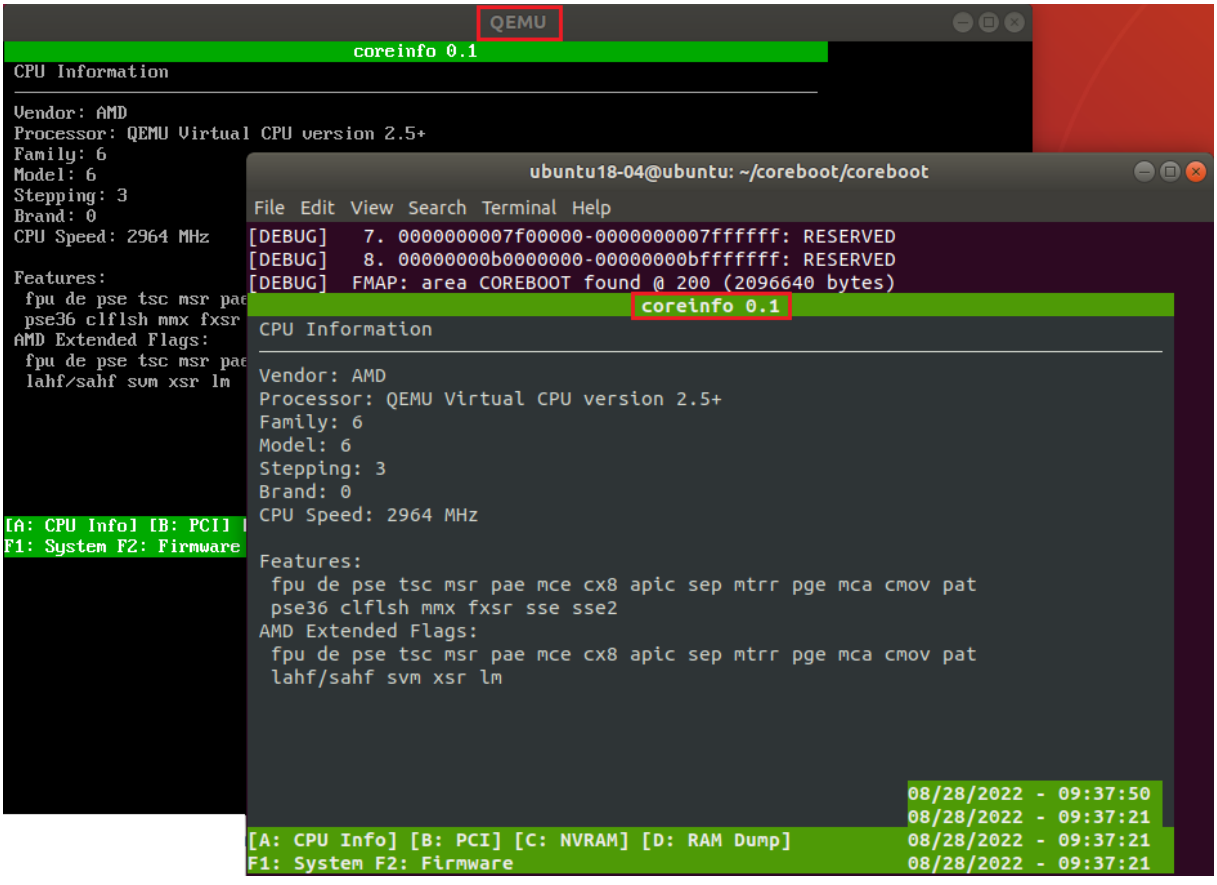
```
make all V=1
```

پس از ساخت، باید متن زیر را در ترمینال مشاهده نمایید و فایل coreboot.rom در پوشه build قرار می گیرد که firmware bios سامانه خواهد بود.

```
Build emulation/qemu-i440fx (QEMU x86 i440fx/piix4)
```

۱۲. در نهایت با اجرای دستور زیر شبیه ساز qemu را با بایوس تولید شده اجرا میکنیم

```
qemu-system-x86_64 -bios build/coreboot.rom -serial stdio
```



۲-۳ ساخت و اجرای payload های EDK2

همانطور که گفتیم این بخش مستقل از بخش ۲-۲ است. به همین دلیل شماره گذاری ها از ۶ شروع می شود. یعنی بعد از انجام عملیات توضیح داده شده در بخش ۲-۱ نیز می توان این بخش را انجام داد.

۶. ابتدا menuconfig را اجرا کرده تا بتوانیم تنظیمات پلتفرم را انجام دهیم

```
make menuconfig
```

۷. در پنجره باز شده تغییرات زیر را اعمال می کنیم

- گزینه Mainborad را انتخاب کرده
  - مقدار Mainboard vendor باید Emulation باشد.
  - **مقدار Mainboard model باید QEMU x86 q35/ich9 باشد.**
- گزینه General Setup را انتخاب کرده
  - گزینه Allow building with any toolchain باید انتخاب شده باشد
- گزینه Payload را انتخاب کرده
  - گزینه Add a Payload را انتخاب کرده و در آن گزینه edk2 payload را انتخاب کنید
  - با انتخاب گزینه بالا باید فایل باینری edk2 با نام UEFIPAYLOAD.fd را در پوشه Build قرار گیرد. همچنین در این حالت پروژه edk2 از آدرس https://github.com/mrchromebox/edk2 دانلود می شود.

۸. از menuconfig خارج شوید و تغییرات را ذخیره کنید

```
make savedefconfig
```

۹. فایل UEFIPAYLOD.fd را در پوشه build کپی می کنیم. برای ساخت این فایل می توانید آموزش راه اندازی EDK2 را مطالعه نمایید.

۱۰. در نهایت با دستور زیر پروژه را بسازید

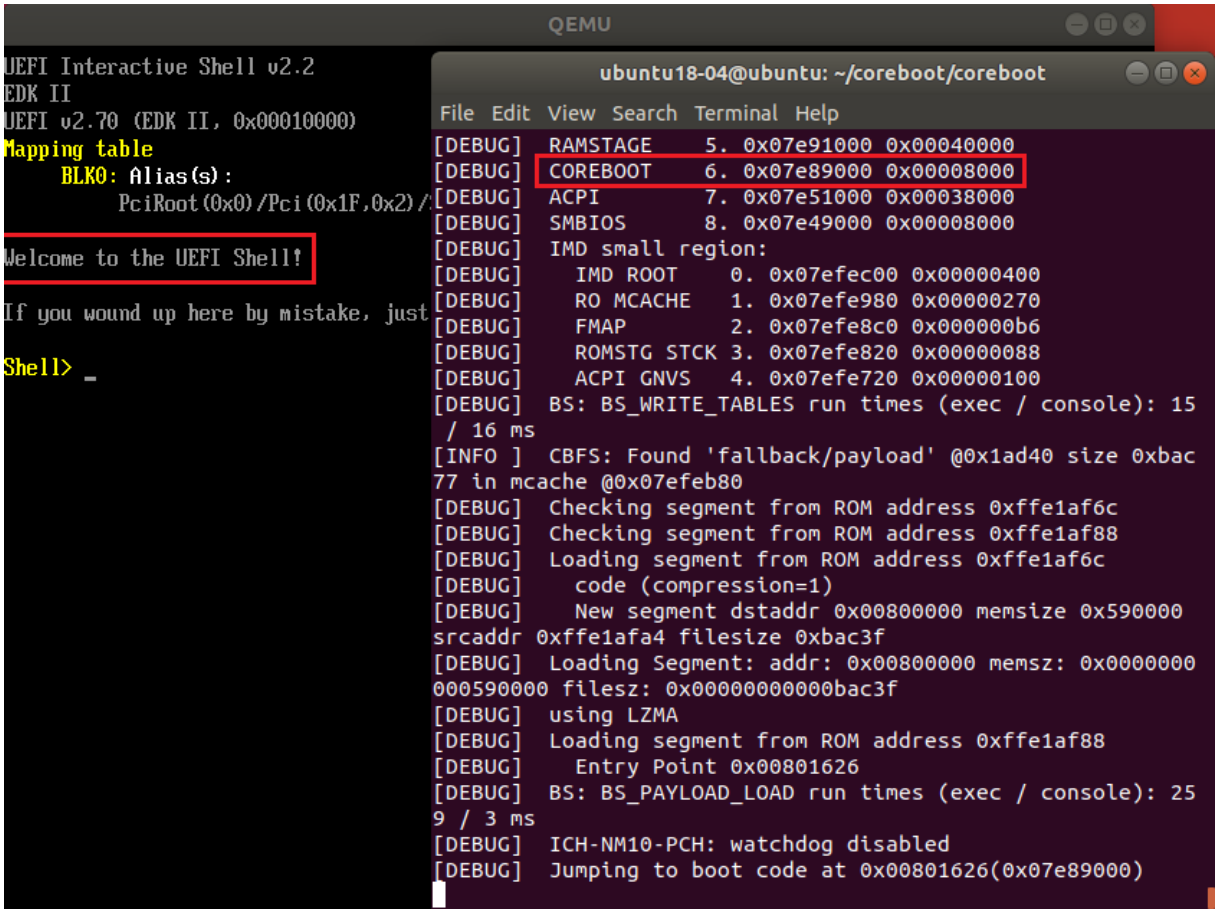
```
make all V=1
```

۱۱. پس از ساخت، باید متن زیر را در ترمینال مشاهده نمایید و فایل coreboot.rom در پوشه build قرار می گیرد که firmware bios سامانه خواهد بود.

```
Build emulation/qemu-q35 (QEMU x86 q35/ich9)
```

۱۲. پس از ساخت پروژه، با دستور زیر Qemu را اجرا کنید. در نظر داشته باشید که در اینجا باید نوع ماشین را q35 انتخاب نمایید.

```
qemu-system-x86_64 -M q35 -bios build/coreboot.rom -serial stdio
```


QEMU
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
BLK0: Alias(s):
PciRoot(0x0)/Pci(0x1F,0x2)/
Welcome to the UEFI Shell!
If you wound up here by mistake, just
Shell>
ubuntu18-04@ubuntu: ~/coreboot/coreboot
File Edit View Search Terminal Help
[DEBUG] RAMSTAGE 5. 0x07e91000 0x00040000
[DEBUG] COREBOOT 6. 0x07e89000 0x00008000
[DEBUG] ACPI 7. 0x07e51000 0x00038000
[DEBUG] SMBIOS 8. 0x07e49000 0x00008000
[DEBUG] IMD small region:
[DEBUG] IMD ROOT 0. 0x07efec00 0x00000400
[DEBUG] RO MCACHE 1. 0x07efe980 0x00000270
[DEBUG] FMAP 2. 0x07efe8c0 0x000000b6
[DEBUG] ROMSTG STCK 3. 0x07efe820 0x00000088
[DEBUG] ACPI GNVS 4. 0x07efe720 0x00000100
[DEBUG] BS: BS_WRITE_TABLES run times (exec / console): 15 / 16 ms
[INFO] CBFS: Found 'fallback/payload' @0x1ad40 size 0xbac77 in mcache @0x07efeb80
[DEBUG] Checking segment from ROM address 0xffe1af6c
[DEBUG] Checking segment from ROM address 0xffe1af88
[DEBUG] Loading segment from ROM address 0xffe1af6c
[DEBUG] code (compression=1)
[DEBUG] New segment dstaddr 0x00800000 memsize 0x590000 srcaddr 0xffe1afa4 filesize 0xbac3f
[DEBUG] Loading Segment: addr: 0x00800000 memsz: 0x0000000000590000 filesz: 0x00000000000bac3f
[DEBUG] using LZMA
[DEBUG] Loading segment from ROM address 0xffe1af88
[DEBUG] Entry Point 0x00801626
[DEBUG] BS: BS_PAYLOAD_LOAD run times (exec / console): 259 / 3 ms
[DEBUG] ICH-NM10-PCH: watchdog disabled
[DEBUG] Jumping to boot code at 0x00801626(0x07e89000)

فصل ۳ راه اندازی Slim Bootloader روی Qemu

۳-۱ نصب SlimBootloader

ابتدا نرم افزارهای زیر را نصب نمایید

```
Python 3.6
NASM 2.12.02
IASL 20190509 (wget http://archive.ubuntu.com/ubuntu/pool/universe/a/acpica-
unix/acpica-tools_20190509-1_amd64.deb)
OpenSSL
Git
```

سپس دستورات زیر را به ترتیب در ترمینال اجرا نمایید. در این دستورات ابتدا پروژه slimbootloader را از گیت دریافت می کنیم. سپس چند کلید توسط برنامه GenerateKeys.py تولید می شود.

```
git clone https://github.com/slimbootloader/slimbootloader.git
cd slimbootloader
mkdir sbl_key_dir
python BootloaderCorePkg/Tools/GenerateKeys.py -k sbl_key_dir/
```

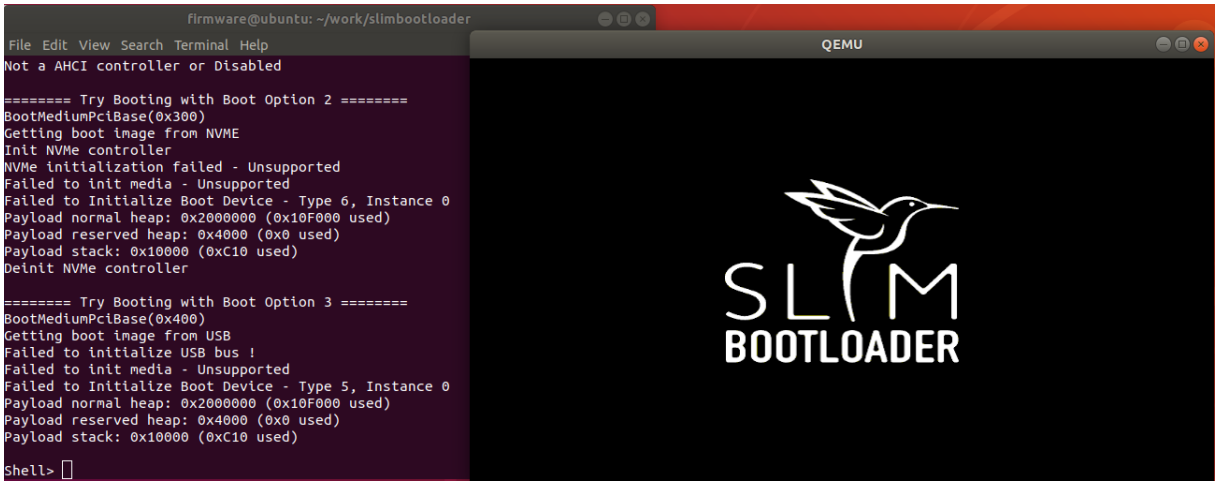
۳-۲ راه اندازی payload پیش فرض (معماری IA32)

به دایرکتوری slimbootloader بروید و برنامه Buildloader.py اجرا کرده تا فایل SlimBootloader.bin در مسیر Outputs/qemu ساخته شود. لازم به ذکر است که در ابتدا باید متغیر SBL_KEY_DIR را با آدرس پوشه ای که کلید ها در آن ساخته شده اند مقداردهی کنیم.

```
export SBL_KEY_DIR=$(pwd)/sbl_key_dir
/.BuildLoader.py build qemu
```

در نهایت با اجرای دستور زیر برنامه Qemu را با بایوس ساخته شده اجرا می کنیم.

```
qemu-system-x86_64 -machine q35 -serial mon:stdio -pflash
Outputs/qemu/SlimBootloader.bin
```

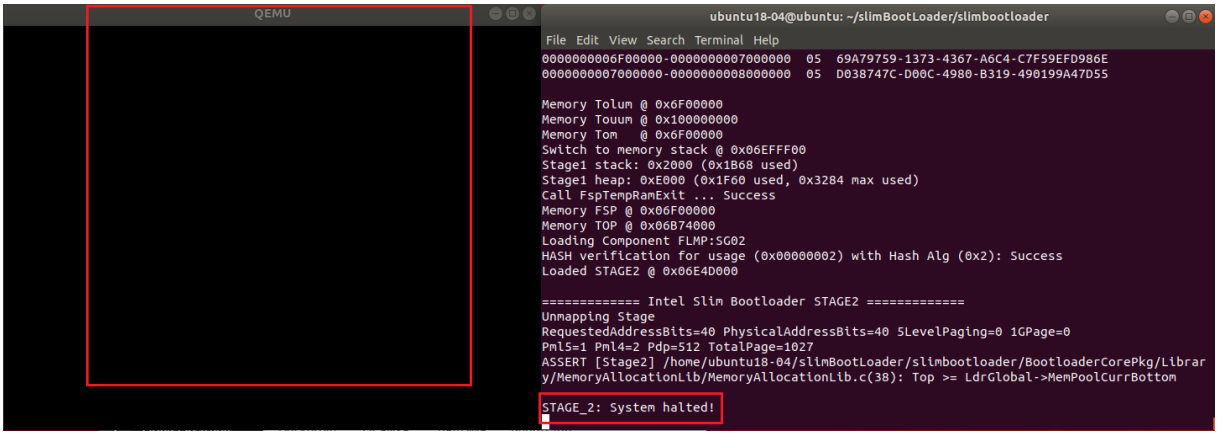


شکل بالا خروجی اجرای SlimBootloader است که در نهایت یک shell در اختیار کاربر قرار می دهد.

۳-۳ نکته بسیار بسیار مهم

یکی از مهمترین مشکلاتی که در SlimBootloader با آن مواجه شدیم (و تا زمان نگارش این سند حل نشده است) آن است که دستوری که در بخش فصل۲۱-۳ اجرا کردیم بایوس را برای معماری IA32 میسازد و در صورت تغییر آن دستور برای معماری X64، با آنکه بایوس ساخته می شود اجرای آن روی Qemu با مشکل مواجه می شود. به همین دلیل در حال حاضر SlimBootloader گزینه مناسبی نیست.

```
/.BuildLoader.py build qemu  -a x64
```



۳-۴ ساخت و اجرای برنامه uefi چاپ در کنسول (معماری IA32)

در مسیر slimbootloader/PayloadPkg فایلی با نام PayloadPkg.dsc و پوشه ای با نام HelloWorld وجود دارد که شامل یک فایل h. یک فایل c. و یک فایل inf. است. این ساختار بسیار شبیه ساختار بسته های پروژه edk2 می باشد.

به دایرکتوری slimbootloader بروید. ابتدا باید متغیر SBL_KEY_DIR را با آدرس پوشه ای که کلید ها در آن ساخته شده اند مقداردهی کنید. سپس برنامه Buildloader.py را با آرگومان های مشخص شده اجرا می کنیم. آرگومان a- مشخص می کند که فایل efi خروجی برای معماریx64 یا ia32 است. کلید t- مشخص می کند که از چه toolchain استفاده شود که در اینجا از GCC5 استفاده کردیم. کلید r- مشخص می کند که در مد Release هستیم و نگذاشتن آن در حالت Defug ساخته می شود. و در نهایت سوییچ p- پکیجی که قرار است کامپایل شود را مشخص می کند. در نهایت فایل HelloWorld.efi در مسیر slimbootloader/Build/PayloadPkg/RELEASE_GCC5/X64 ساخته شود.

```
export SBL_KEY_DIR=$(pwd)/sbl_key_dir
python BuildLoader.py build_dsc -p PayloadPkg/PayloadPkg.dsc
```

پوشه PayloadPkg/PayloadBins را می سازیم و فایل efi تولید شده را در آن کپی می کنیم.

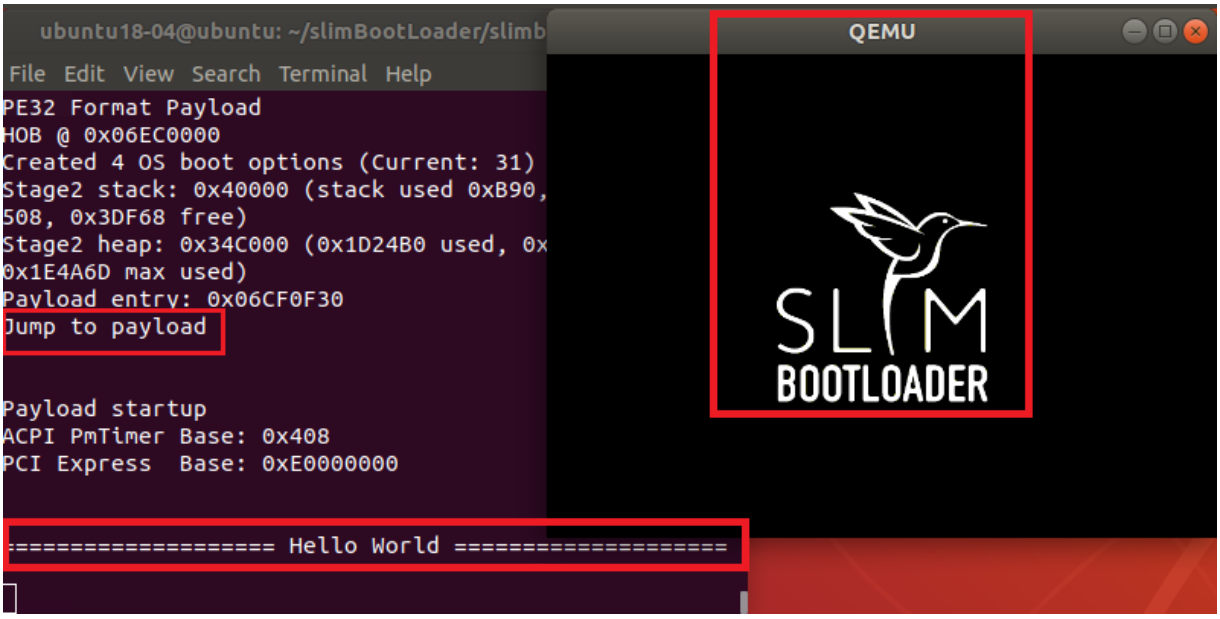
```
mkdir -p PayloadPkg/PayloadBins
cp Build/PayloadPkg/DEBUG_GCC5/IA32/HelloWorld.efi PayloadPkg/PayloadBins
```

برای ساخت فایل Bios که برنامه HelloWorld.efi را به عنوان Payload اجرا کند دستور زیر را اجرا می کنیم.

```
python BuildLoader.py build qemu -p HelloWorld.efi:HLWD:Lz4
```

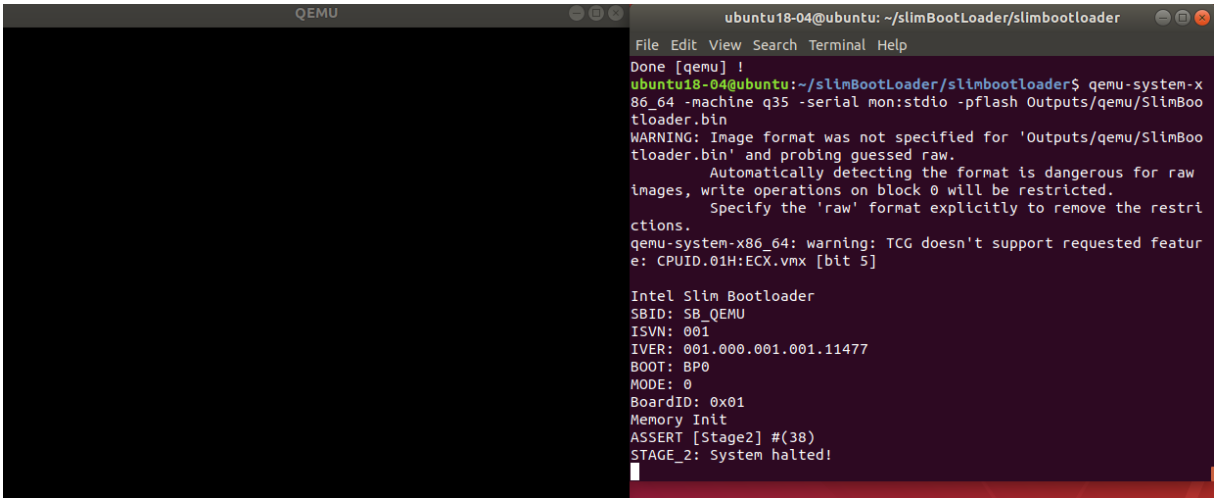
در نهایت با اجرای دستور زیر، برنامه را در شبیه ساز Qemu اجرا می کنیم.

```
qemu-system-x86_64 -machine q35 -serial mon:stdio -pflash
Outputs/qemu/SlimBootloader.bin
```



برای معماری X64 باید دستورات را به شکل زیر وارد کنیم که متاسفانه فعلا کار نکرده است.

```
export SBL_KEY_DIR=$(pwd)/sbl_key_dir
python BuildLoader.py build_dsc -r -a x64 -p PayloadPkg/PayloadPkg.dsc
mkdir -p PayloadPkg/PayloadBins
rm PayloadPkg/PayloadBins*/
cp Build/PayloadPkg/RELEASE_GCC5/X64/HelloWorld.efi PayloadPkg/PayloadBins/
python BuildLoader.py build qemu -a x64 -p HelloWorld.efi:HLWD:Lz4
qemu-system-x86_64 -machine q35 -serial mon:stdio -pflash
Outputs/qemu/SlimBootloader.bin
```



فصل ۴ مراجع

[1] Link: https://doc.coreboot.org/tutorial/part1.html
[2] Link: https://windscribe.com/guides/linux
[3] Link: https://github.com/tianocore/edk2/blob/master/UefiPayloadPkg/BuildAndIntegrationInstructions.txt
[4] https://slimbootloader.github.io/introduction/index.html