Gears OS のBootに関する研究

- 奥田光希
  - 琉球大学工学部工学科知能情報コース
- 河野 真治
  - 琉球大学工学部

OSとアプリケーションの信頼性を高めたい

- Meta計算を用いて信頼性を高めるGearsOSを提案している
- x.v6を元にRaspberry Pi上で動くGearsOSを実装中
- BIOSからBootしているのでUEFIに移行したい

UEFI採用の利点

- CPUなどの機種依存性を避けることができる
- GearsOSはCbC(Continuation based C)で記述されていて、CPUやデバイスに影響されない
- 様々な組み込みシステムに対してGearsOSを応用できる様になる

CbC(Continuation based C)

- 並列信頼研究で開発されているプログラミング言語
- C言語の下位言語
- 関数呼び出しではなく継続(goto)
- 関数の代わりにCodeGearという単位でプログラミングを行う。

GearsOS

- 並列信頼研究で開発されているOS
- 信頼性と拡張性がテーマ
- CbCによって記述されている
- x.v6をCbCで書き直して実装している

UEFI

- Unified Extensible Firmware Interfaceの略
- OSとプラットフォームファームウェアの間のソフトウェアインタフェースを定義する仕様
- Intel、AMD、Apple、Microsoftなどの企業からなるUnified EFI　Forumの元で開発
- BIOSの後継

UEFIのここがすごい

- CPUやドライバに依存しない
- 2TBを超える大きなディスクからBootできる
- ネットワークにつながる
- メモリも64bitなら理論上16EB
- 高速でBoot
- 仕様だから開発が簡単

UEFI 開発環境

- edk2
- gnu-efi
- qemu
- singularity

gnu-efi

- システムのネイティブGCCでUEFIアプリケーションをコンパイルできる
- UEFI　Applicationをリンクするためのライブラリがある
- UEFI Applicationの開発に特化している
- EDK2のファームウェアがベース

qemu

- 異なるアーキテクチャのプログラムを動かすエミュレータ
- 本開発ではX86上でARMを動かした
- gnu-efiで実装したUEFIを動かした

singularity

- ユーザーが自身の計算環境を完全再現し、保持できる様にしたLinuxコンテナ
- 学科の新システムで利用できる
- CbC GCC ARM CrossCompile環境を作った
- UEFIの開発環境を作った

UEFI Application

- UEFI Boot Managreがロード、実行するプログラムのこと
- C言語で記述可能
- OSがなくてもプログラムを書ける
- CPUやドライバに依存しない

UEFI Applicationの例

Hello.c

```
#include <efi.h>
#include <efilib.h>

EFI_STATUS
EFIAPI
efi_main (EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  InitializeLib(ImageHandle, SystemTable);
  Print(L"Hello, world!\n");
  return EFI_SUCCESS;
}
```

BootLoader.c

- efi_mainの引数設定

```
#include<efi.h>
#include<efilib.h>

EFI_STATUS
efi_main(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
    EFI_DEVICE_PATH *Path;
    EFI_LOADED_IMAGE *LoadedImageParent;
    EFI_LOADED_IMAGE *LoadedImage;
    EFI_HANDLE Image;
    CHAR16 *Options = L"root=/dev/sda2 rootfstype=btrfs rw quiet splash";
    EFI_STATUS Status=EFI_SUCCESS;

    InitializeLib(ImageHandle, SystemTable);
    Print(L"Hello, EFI!\n");
```

- OSファイルのファイルパスを代入している

```
Status = uefi_call_wrapper(BS->OpenProtocol, 6, ImageHandle, &
LoadedImageProtocol,(void**)&LoadedImageParent,
ImageHandle, NULL, EFI_OPEN_PROTOCOL_GET_PROTOCOL);
if (EFI_ERROR(Status)) {
    Print(L"Could not get LoadedImageProtocol handler %r\n", Status);
    return Status;
}
Path = FileDevicePath(LoadedImageParent->DeviceHandle, L"\\xv6.img");
if (Path == NULL) {
    Print(L"Could not get device path.");
    return EFI_INVALID_PARAMETER;
}
```

- KernelをLoadしている

```
    Status = uefi_call_wrapper(BS->LoadImage, 6, FALSE, ImageHandle, Path, NULL, 0, &Image);
    if (EFI_ERROR(Status)) {
        Print(L"Could not load %r", Status);
        FreePool(Path);
        return Status;
    }
```

- ImageをLoadしてKernelを起動している

```
    Status = uefi_call_wrapper(BS->OpenProtocol, 6, Image, &LoadedImageProtocol,
    (void**)&LoadedImage, ImageHandle, NULL, EFI_OPEN_PROTOCOL_GET_PROTOCOL);
    if (EFI_ERROR(Status)) {
        Print(L"Could not get LoadedImageProtocol handler %r\n", Status);
        uefi_call_wrapper(BS->UnloadImage, 1, Image);
        FreePool(Path);
        return Status;
    }
    LoadedImage->LoadOptions = Options;
    LoadedImage->LoadOptionsSize = (StrLen(LoadedImage->LoadOptions)+1) * sizeof(CHAR16);
    Print(L"Hello,6!\n");


    Status = uefi_call_wrapper(BS->StartImage, 3, Image, NULL, NULL);
    uefi_call_wrapper(BS->UnloadImage, 1, Image);
    FreePool(Path);

    return EFI_SUCCESS;
}
```

Boot Loader

- UEFIが起動
- Boot Managerから起動される
- OSのKernelをメモリにLoadさせる
- Kernelがinitプロセスを起動
- initプロセスがOSのBootプロセスを起動
- OSがBootされる

大変だったこと

- EDK2は汎用的だがARMのConfigなどの書き換えが困難
- UEFI開発の情報が少なく、偏りがあった
- UEFI独特のプログラムの書き方があった
- 低レベルの開発に慣れていなかった

今後の課題

- Singularity上のqemu-system-armにgdbを接続する
- そのgdbでKernel Loaderをでバックする
- xv6 KernelにUEFIからBootするコードを入れる
- xv6を書き換えたGearsOSを実装する
- USB Driverを実装し、キーボードやマウスを使える様にする

研究の成果

- UEFIの開発環境をSingularityで作成した
- gnu-efiで作成したUEFI ApplicationをQEMU-ARMで動かすことができた
- RaspberryPiにUEFIをファームウェアとして設定し、実行することができた
- ミニマムなKernel Loaderを調査しARM xv6用に書き直した