

자기주도프로젝트 중간보고서

- 연구주제 제목: 리눅스 KVM API를 이용한 초소형 가상 머신 모니터 (VMM) 개발
- 학번: 202322071
- 이름: 설규원
- 지도교수: 김상훈
- 목차: 연구계획 / 참고링크 / 주차별 연구 보고서

Current Research Plan

주차	날짜	연구 수행 내용	결과물
1주	9월 1일	문서 수집 및 학습 (KVM API, CPU, 리눅스 커널, QEMU 등)	학습 기록서, 핵심 문서 모음
2주	9월 8일	구조 설계, 개발 환경 구축, Git 저장소 생성	설계도, 환경 정보, Git 원격 저장소
3주	9월 15일	VM 생성 및 메모리 할당 코드 작성	코드 및 설명
4주	9월 22일	계획 점검, 오픈소스 코드 리뷰, 추가 학습	수정된 계획서, 학습 기록서
5주	9월 29일	vCPU 생성, 레지스터 제어, 실행 코드 작성	코드 및 설명
6주	10월 6일	메모리 입출력, VM 종료 코드 작성, 9월 성과 정리	코드 및 설명, 9월 성과보고서
7주	10월 13일	계획/설계 수정, 오픈소스 리뷰, 학습	수정된 계획서/설계도, 학습 기록서
8주	10월 20일	메모리 입출력 및 디버깅 코드 작성	코드 및 설명
중간	10월 24일	중간보고서 작성 및 제출	중간보고서
9주	10월 27일	실행 가능한 바이너리 테스트 및 디버깅	코드 및 설명, 테스트 결과 기록서
10주	11월 3일	리눅스 커널/부트 프로세스 학습, 10월 성과 정리	학습 기록서, 10월 성과보고서
11주	11월 10일	부트로더 실행 코드 작성 및 디버깅	코드 및 설명
12주	11월 17일	리눅스 커널 실행 코드 작성 및 디버깅	코드 및 설명
13주	11월 24일	리눅스 커널 실행 안정화 테스트	코드 및 설명

주차	날짜	연구 수행 내용	결과물
14주	12월 1일	최종 테스트, 시연 영상 녹화, 성과 정리	Git 저장소, 시연 영상, 성과 보고서
15주	12월 8일	문서화 및 프로젝트 마무리	최종 문서, Git 원격 저장소
16주	12월 15일	문서화 및 프로젝트 마무리	최종 문서, Git 원격 저장소
기말	12월 19일	결과 보고서, 연구 노트, 기타 증빙 제출	결과 보고서, 연구 노트, 기타 증빙

Links

Other Repos

- [Fork of xv6-public](#)
- [Copy of bochs-2.2.6](#)

Reference Projects

- [Bochs Repository](#)
- [xv6 Repository](#)
- [QEMU Repository](#)

Docs

- [KVM Documentation](#)
- [Bochs Documentation](#)
- [Slides - Virtualization without direct execution - designing a portable VM](#)
- [Paper - Virtualization Without Direct Execution or Jitting: Designing a Portable Virtual Machine Infrastructure](#)
- [4.3. The configuration file bochsrc](#)

Tutorials

- [How to compile xv6 on Linux](#)
- [Bochs 설정법](#)

Etc.

- [Bochs Releases](#)

1주차 연구내용

목표: 문서 수집 및 학습

연구 내용

이번주 목표는 '문서 수집 및 학습'입니다. 하지만 바로 문서만 읽으면 구체적으로 어떤 부분이 부족한지, 뭘 더 배워야 할 지 모를 것 같다는 생각이 들었습니다.

따라서 교수님께서 제안해주신 xv6 체험을 먼저 진행하며, 과정 중에 생기는 구체적인 궁금증들을 해결하는 과정으로 진행하기로 결정했습니다.

xv6

우선 xv6는 [github](#)에 소스 코드가 공개되어 있습니다. 그러나 README.md를 통해 할 수 있듯, 약 5년 전에 x86 버전은 개발이 중단되었고 현재에는 RISC-V 버전만 개발되고 있습니다. 그럼에도 제 프로젝트의 목표는 KVM을 통해 AMD64 환경에서의 가상화를 구현하는 것이기에, x86 코드를 그대로 활용하기로 결정했습니다.

README.md의 'BUILDING AND RUNNING XV6' 부분을 보면, 컴파일과 부팅을 하는 방법이 명시되어 있습니다. x86 크로스컴파일이 가능한 gcc가 설치되어 있는 환경에서 `make` 명령어를 실행하면 컴파일이 되고, `make qemu` 를 하면 QEMU 환경에서 실행된다고 써 있습니다.

다만, 코드가 너무 방대한 QEMU보다는 교수님께서 제안하신 Bochs를 레퍼런스로 공부하고 싶기에, `make bochs` 로 부팅하면 될 것 같습니다.

컴파일

우선 xv6-public 레포를 로컬로 클론했습니다.

```
git clone https://github.com/mit-pdos/xv6-public
```

이후, 폴더에 들어간 후 `make` 를 실행했으나 다음과 같은 에러가 발생했습니다:

```
In function 'mpconfig',
  inlined from 'mpinit' at mp.c:101:14:
mp.c:83:10: error: array subscript -48806446 is outside array bounds of
'void[2147483647]' [-Werror=array-bounds=]
   83 |     if(conf->version != 1 && conf->version != 4)
       |         ~~~~^~~~~~
mp.c:78:34: note: at offset -2147483648 into object '*<unknown>.physaddr' of
size [0, 2147483647]
```

```

78 |   if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                                     ~~^~~~~~
mp.c:85:28: error: array subscript -48806446 is outside array bounds of
'void[2147483647]' [-Werror=array-bounds=]
85 |   if(sum((uchar*)conf, conf->length) != 0)
    |                                     ~~~~^~~~~~
mp.c:78:34: note: at offset -2147483648 into object '*<unknown>.physaddr' of
size [0, 2147483647]
78 |   if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                                     ~~^~~~~~
mp.c: In function 'mpinit':
mp.c:104:22: error: array subscript -48806446 is outside array bounds of
'struct mpconf[48806446]' [-Werror=array-bounds=]
104 |   lapic = (uint*)conf->lapicaddr;
    |                                     ^~
In function 'mpconfig',
    inlined from 'mpinit' at mp.c:101:14:
mp.c:78:34: note: at offset -2147483648 into object '*<unknown>.physaddr' of
size [0, 2147483647]
78 |   if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                                     ~~^~~~~~
mp.c: In function 'mpinit':
mp.c:105:46: error: array subscript -48806446 is outside array bounds of
'struct mpconf[48806446]' [-Werror=array-bounds=]
105 |   for(p=(uchar*)(conf+1), e=(uchar*)conf+conf->length; p<e; ){
    |                                                     ^~
In function 'mpconfig',
    inlined from 'mpinit' at mp.c:101:14:
mp.c:78:34: note: at offset -2147483648 into object '*<unknown>.physaddr' of
size [0, 2147483647]
78 |   if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                                     ~~^~~~~~
In function 'sum',
    inlined from 'mpconfig' at mp.c:85:6,
    inlined from 'mpinit' at mp.c:101:14:
mp.c:25:16: error: array subscript [-2147483648, -2147418114] is outside
array bounds of 'void[2147483647]' [-Werror=array-bounds=]
25 |   sum += addr[i];
    |           ~~~~^~~
In function 'mpconfig',
    inlined from 'mpinit' at mp.c:101:14:
mp.c:78:34: note: at offset [-2147483648, -2147418114] into object '*
<unknown>.physaddr' of size [0, 2147483647]
78 |   if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                                     ~~^~~~~~

```

```
cc1: all warnings being treated as errors
make: *** [<내장>: mp.o] 오류 1
```

에러 코드를 읽어보니, 자료형 변환 부분에서 경고가 발생하는 것 같았습니다. 그리고 `cc1: all warnings being treated as errors` 라는 라인을 보았을 때, `-werror` 플래그가 모든 경고를 에러로 받아들이고 있다는 사실을 알 수 있었습니다.

(이 부분이 실제 코드의 결함일 수도 있고, 제 컴파일러의 버전이 맞지 않는 것일 수도 있습니다. 실제로 코드에 결함이 있다면, 따로 레포를 포크해 개선할 수 있다면 더 좋을 것 같습니다.)

(제 환경은 Fedora 42이고, GCC 버전은 15.2.1 입니다.)

그래서 Makefile을 수정해 `-werror` 부분을 삭제하고, `make clean` 이후 다시 컴파일했더니, 컴파일 되었습니다. (정상적으로 된 건지는 잘 모르겠습니다.)

부팅

이후 `make bochs` 로 부팅해보았으나, 다음과 같은 에러와 함께 실패했습니다:

```
=====
                        Bochs x86 Emulator 3.0
                Built from GitHub snapshot on February 16, 2025
                Timestamp: Sun Feb 16 10:00:00 CET 2025
=====
000000000000i[      ] BXSHARE not set. using compile time default
'/usr/share/bochs'
000000000000i[      ] reading configuration from .bochsrc
000000000000p[      ] >>PANIC<< .bochsrc:497: directive 'vga_update_interval'
not understood
000000000000e[SIM   ] notify called, but no bxevent_callback function is
registered
000000000000e[SIM   ] notify called, but no bxevent_callback function is
registered
=====
Bochs is exiting with the following message:
[      ] .bochsrc:497: directive 'vga_update_interval' not understood
=====
000000000000i[SIM   ] quit_sim called with exit code 1
make: *** [Makefile:211: bochs] 오류 1
```

다시 확인해보니, 그냥 bochs를 실행해도 다음과 같은 에러가 뜹니다. `.bochsrc` 파일이 설정되지 않아서 생기는 문제인 것 같습니다.

([Notes 파일](#)에 bochs 관련 정보들이 있는데, 아직은 bochs를 잘 몰라서 어떻게 설정해야 하는지 잘 모르겠습니다.)

따라서, 우선 bochs가 제대로 설정되기 전까지는 xv6가 정상적으로 컴파일되었는지 확인하기 위해 QEMU를 사용하기로 결정했습니다.

`make qemu`, `make qemu-nox` 로도 테스트해봤으나, 별다른 에러 메시지 없이, 다음 출력 외에는 몇분 동안 아무 반응이 없었습니다.

```
SeaBIOS (version 1.17.0-5.fc42)
```

```
iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCAEC0+1EF0AEC0 CA00
```

```
Booting from Hard Disk..
```

컴파일 과정의 문제로 의심이 되어, 검색을 통해 정보를 찾아봤습니다. 그 결과, 한 [Youtube 영상](#)을 발견했습니다.

이 영상에서는 lubuntu 12.04 32bit를 사용하지만, 다운로드 링크가 만료되어있습니다.

64bit 환경인 게 이 문제의 원인이라면, 영상처럼 32bit OS를 활용하면 가능할 것이므로, 데스크탑에 데비안 12를 기반으로 하는 LMDE 6 가상환경을 구성했습니다. (CPU: `kvm32` 로 설정해야 virtio 인터넷 연결이 가능합니다.)

이후, 필요한 패키지들을 설치하고 xv6 레포를 클론했습니다.

```
sudo apt install git qemu-system qemu-user bochs gcc gcc-multilib build-essential make vim
git clone https://github.com/mit-pdos/xv6-public
```

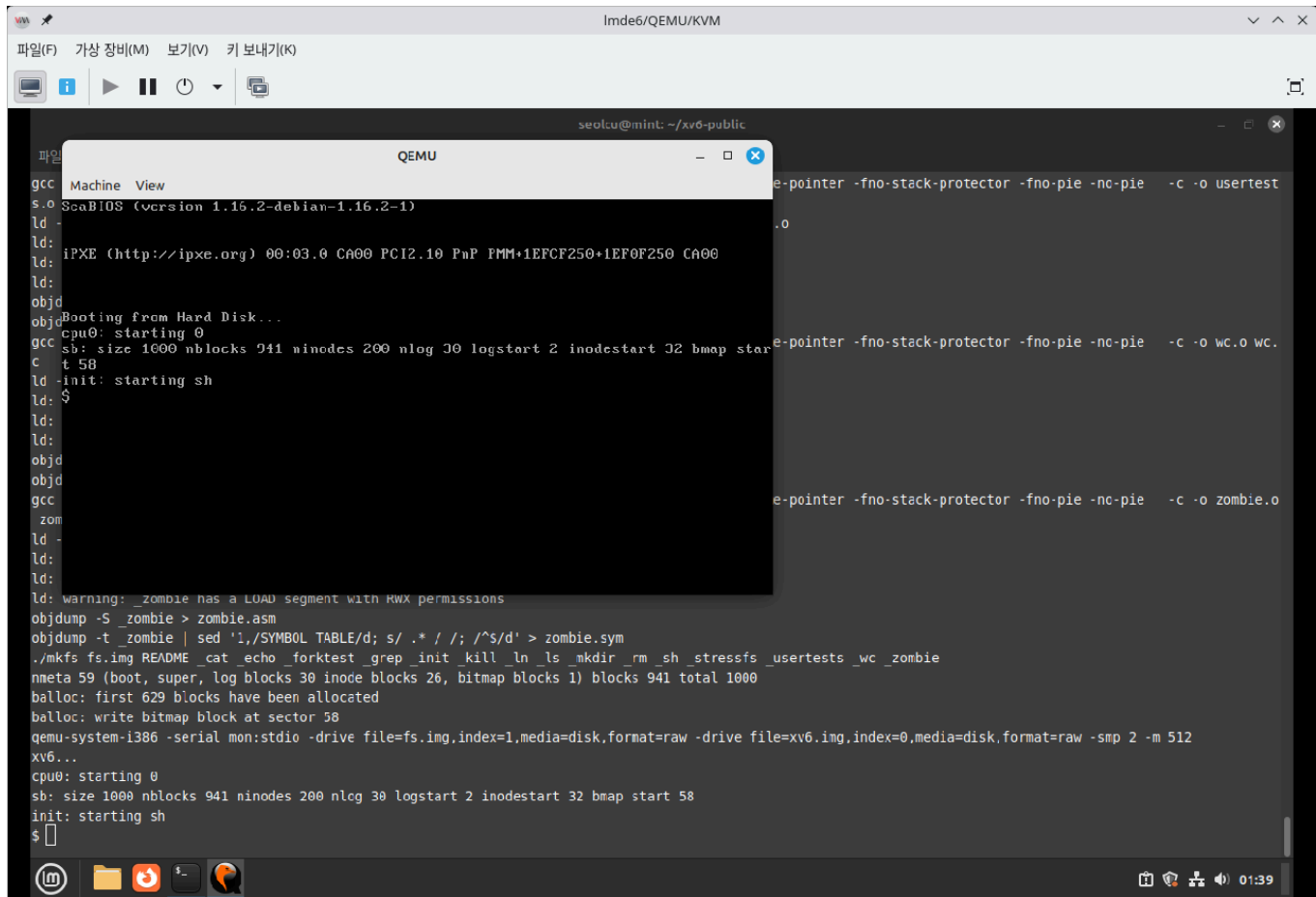
바로 `make` 를 돌려봤습니다.

```
seolcu@mint: ~/xv6-public
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

78 | if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    | ~~~~~
mp.c: In function 'mpinit':
mp.c:104:22: error: array subscript -48866446 is outside array bounds of 'struct mpconf[48806446]' [-Werror=array-bounds]
104 |     lapic = (uint*)conf->lapicaddr;
    |                      ^
In function 'mpconfig',
inlined from 'mpinit' at mp.c:101:14:
mp.c:78:34: note: at offset -2147483648 into object '*_119 = PHI <_133(48), _16(4), _45(6)>.physaddr' of size [0, 2147483647]
78 |     if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                      ~~~~~
mp.c: In function 'mpinit':
mp.c:105:46: error: array subscript -48866446 is outside array bounds of 'struct mpconf[48806446]' [-Werror=array-bounds]
105 |     for(p=(uchar*)(conf+1), e=(uchar*)conf+conf->length; p<e; ){
    |                                     ^
In function 'mpconfig',
inlined from 'mpinit' at mp.c:101:14:
mp.c:78:34: note: at offset [-2147483648, -2147418114] into object '*_119 = PHI <_133(48), _16(4), _45(6)>.physaddr' of size [0, 2147483647]
78 |     if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                      ~~~~~
In function 'sum',
inlined from 'mpconfig' at mp.c:85:6,
inlined from 'mpinit' at mp.c:101:14:
mp.c:25:16: error: array subscript [-2147483648, -2147418114] is outside array bounds of 'void[2147483647]' [-Werror=array-bounds]
25 |     sum += addr[i];
    |             ~~~~~
In function 'mpconfig',
inlined from 'mpinit' at mp.c:101:14:
mp.c:78:34: note: at offset [-2147483648, -2147418114] into object '*_119 = PHI <_133(48), _16(4), _45(6)>.physaddr' of size [0, 2147483647]
78 |     if((mp = mpsearch()) == 0 || mp->physaddr == 0)
    |                      ~~~~~
cc1: all warnings being treated as errors
make: *** [<내장>: mp.o] 오류 1
seolcu@mint:~/xv6-public$
```

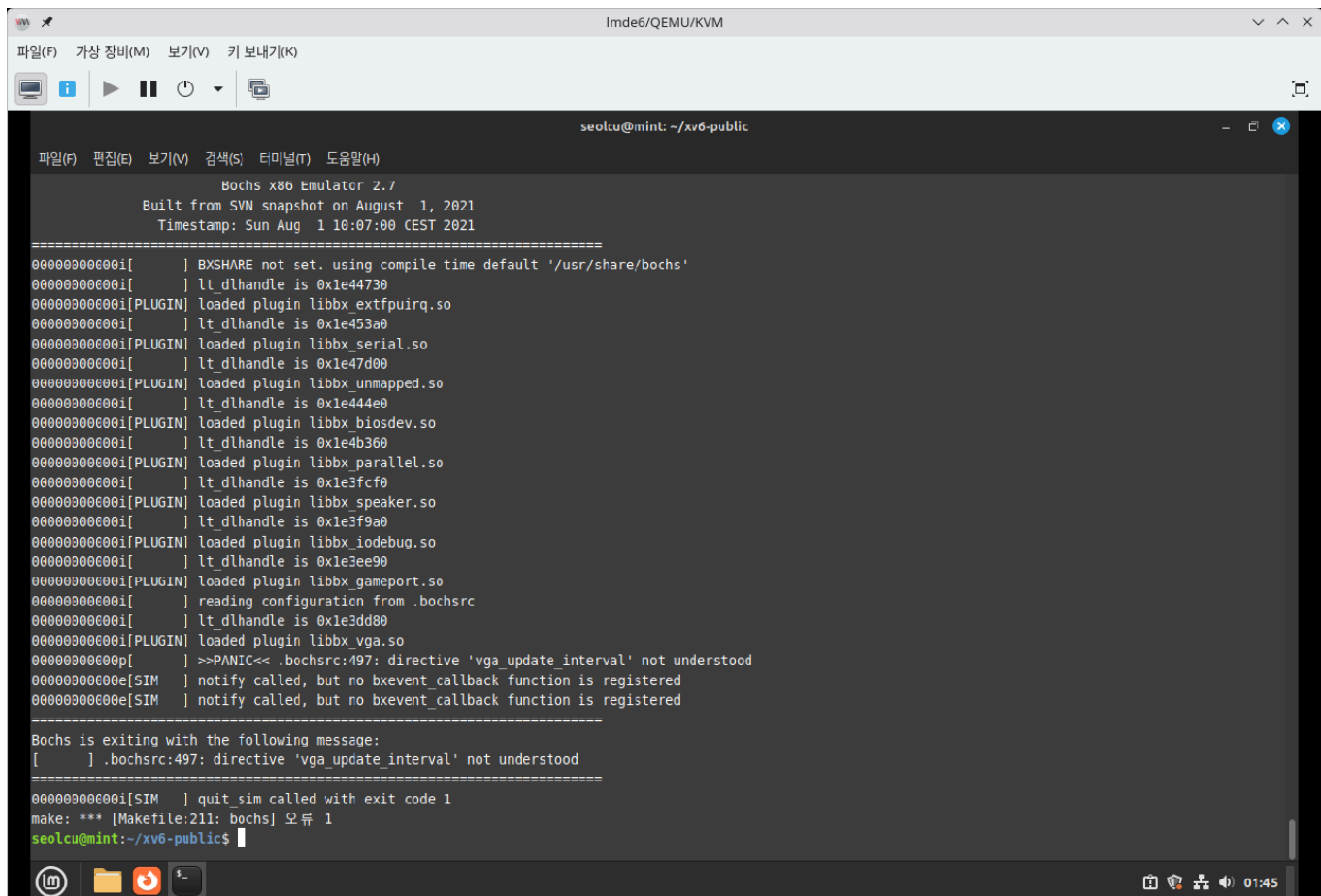
그 결과, 똑같이 make 과정에서 경고(에러)가 발생했습니다. 이를 보아, xv6 코드 자체의 문제인 것 같습니다.

앞선 해결방법과 똑같이, Makefile에서 `-Werror` 플래그를 삭제하고 `make` 로 컴파일한 후, `make qemu` 로 실행해보았습니다.



```
gcc Machine View
s.o ScaBIOS (vcrsion 1.16.2-debian-1.16.2-1)
ld:
ld: iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP FMM+1EFCE250+1EF0F250 CA00
ld:
objd
objd Booting from Hard Disk...
cpu0: starting 0
gcc sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
ld: init: starting sh
ld: $
ld:
ld:
objd
objd
gcc
ld:
ld:
ld: warning: _zombie has a LOAD segment with RWX permissions
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
./mkfs fs,img README_cat_echo_forktest_grep_init_kill_ln_ls_mkdir_rm_sh_stressfs_userstests_wc_zombie
mmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 629 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

잘 실행되는 모습을 볼 수 있었습니다.



```
Bochs x86 Emulator 2.7
Built from SVN snapshot on August 1, 2021
Timestamp: Sun Aug 1 10:07:00 CEST 2021
=====
0000000000i[ ] BXSHARE not set. using compile time default '/usr/share/bochs'
0000000000i[ ] lt_dlhandle is 0x1e44730
0000000000i[PLUGIN] loaded plugin libbx_extfpuirq.so
0000000000i[ ] lt_dlhandle is 0x1e453a0
0000000000i[PLUGIN] loaded plugin libbx_serial.so
0000000000i[ ] lt_dlhandle is 0x1e47d00
0000000000i[PLUGIN] loaded plugin libbx_unmapped.so
0000000000i[ ] lt_dlhandle is 0x1e444e0
0000000000i[PLUGIN] loaded plugin libbx_biosdev.so
0000000000i[ ] lt_dlhandle is 0x1e4b360
0000000000i[PLUGIN] loaded plugin libbx_parallel.so
0000000000i[ ] lt_dlhandle is 0x1e3fc0
0000000000i[PLUGIN] loaded plugin libbx_speaker.so
0000000000i[ ] lt_dlhandle is 0x1e3f9a0
0000000000i[PLUGIN] loaded plugin libbx_iodebug.so
0000000000i[ ] lt_dlhandle is 0x1e3ee90
0000000000i[PLUGIN] loaded plugin libbx_gameport.so
0000000000i[ ] reading configuration from .bochsrc
0000000000i[ ] lt_dlhandle is 0x1e3dd80
0000000000i[PLUGIN] loaded plugin libbx_vga.so
0000000000p[ ] >>PANIC<< .bochsrc:497: directive 'vga_update_interval' not understood
0000000000e[SIM] notify called, but no bxevent_callback function is registered
0000000000e[SIM] notify called, but no bxevent_callback function is registered
=====
Bochs is exiting with the following message:
[ ] .bochsrc:497: directive 'vga_update_interval' not understood
=====
0000000000i[SIM] quit_sim called with exit code 1
make: *** [Makefile:211: bochs] 오류 1
seolcu@mint:~/xv6-public$
```

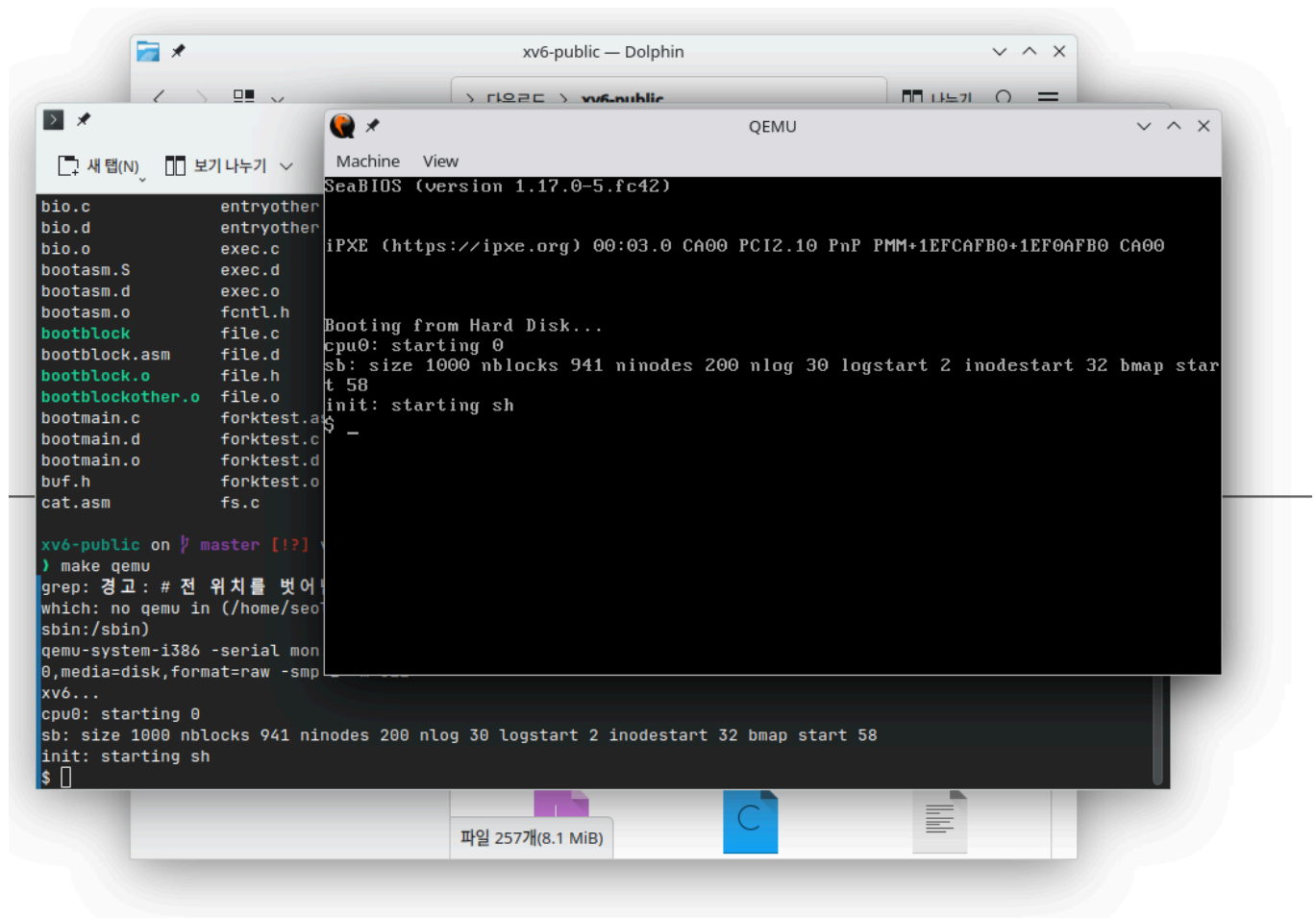
make bochs 는 여전히 작동하지 않았습니다 (앞서 확인했듯이, .bochsrc가 설정되지 않았습니다.)

Fedora 42(x64)에서는 작동하지 않았는데, LMDE 6(x86)에서는 작동한 이유가 무엇인지에 대한 분석

가설 1. 컴파일러의 문제이다.

만약 컴파일러가 문제였다면, LMDE에서 컴파일한 이미지를 Fedora로 넘겨서 qemu로 돌렸을 때 정상적으로 실행되어야 합니다.

따라서, LMDE에서 make 로 컴파일한 파일을 Fedora로 복사해 make qemu 로 부팅해봤습니다.



그 결과, qemu로 정상적으로 부팅되는 모습을 볼 수 있었습니다. 이를 통해, 컴파일 과정에서 문제가 발생했음을 확인할 수 있었습니다. 따라서 가설 1이 맞다는 것을 검증할 수 있었습니다.

그러면 Fedora의 컴파일러는 뭐가 잘못된 것이었을까?

우선, [xv6 레포](#)의 README에는 다음과 같이 쓰여 있습니다.

BUILDING AND RUNNING XV6

To build xv6 on an x86 ELF machine (like Linux or FreeBSD), run

```
"make". On non-x86 or non-ELF machines (like OS X, even on x86), you
will need to install a cross-compiler gcc suite capable of producing
x86 ELF binaries (see https://pdos.csail.mit.edu/6.828/).
Then run "make TOOLPREFIX=i386-jos-elf-". Now install the QEMU PC
simulator and run "make qemu".
```

그러니까 x86 ELF 환경에서는 그냥 make 로 컴파일이 되고, 그게 아닌 경우에는 크로스컴파일러를 설치한 후 make TOOLPREFIX=i386-jos-elf- 를 하면 된다는 것 같습니다. 여기서, "non-x86"에 x86-64가 포함되는건지 아닌건지는 잘 모르겠습니다. 우선 x86-64가 non-x86이 맞다고 가정하고 문제를 해결해보려 했습니다.

Fedora에 gcc-multilib 설치

우선 dnf search gcc로 관련 패키지들을 검색해봤습니다.

그러나 어떻게 gcc-multilib에 해당하는지 알 수 없어, 검색을 통해 정보를 찾던 도중 [스택오버플로우 글](#)을 발견해, 다음과 같은 명령어로 제안하는 패키지들을 설치해보았습니다.

```
sudo dnf install glibc-devel.i686 libstdc++-devel.i686
```

이후 make , make qemu 를 했으나 여전히 결과는 같았습니다. 글에서 -static.i686 접미사 달린 패키지들도 설치하라는 말을 발견했습니다. 그것이 효과가 있을지 보기 위해 Makefile 을 보니 정말 -static 컴파일 옵션이 있는 걸 볼 수 있었습니다. 따라서 "정적 링크를 위한 glibc 파일이 없어서 문제가 발생했다" 라는 추론을 할 수 있었습니다. 따라서 dnf search glibc 로 관련 패키지들을 전부 조회해, glibc-static.i686 이라는 패키지를 발견해 설치했습니다. 그러나, 여전히 문제가 해결되지는 않았습니다.

만약 64비트인게 문제가 아니라 그냥 Fedora인게 문제라면, 추후 distrobox를 활용해 데비안/우분투 기반으로 다시 컴파일 했을 때 될 수도 있겠습니다.

컴파일에 관한 결정

문제가 아직 해결되지 않아서 마음이 불편한 부분이 있지만, 우선 LMDE에서 컴파일한 파일이 있으므로 당분간은 그걸 계속 사용하기로 결정했습니다. 따라서 복사한 레포 + 컴파일된 이미지를 .zip파일로 묶어 이 프로젝트 레포의 archive 폴더에서 관리할 수 있도록 했습니다.

Bochs

앞서 xv6 컴파일 및 부팅 과정에서 파악한 현재의 문제는 대략 두가지입니다.

1. 컴파일 환경의 불안정성 (LMDE에서 컴파일해 해결.)
2. Bochs 사용법에 대한 이해 부족

1번은 어느정도 해결했으므로 큰 문제는 아닙니다. 이 프로젝트의 주제가 xv6같은 운영체제 그 자체가 아닌, qemu나 bochs같은 하이퍼바이저이기 때문입니다. 따라서 xv6가 실행될 수 있도록 컴파일된 현재의 상황에서는 큰 문제가 되지 않습니다.

하지만, 2번은 확실한 문제입니다. 따라서 지금 집중해야 하는 부분은 Bochs에 대한 문서를 수집하고, `.bochsrc` 를 가져오거나 작성하는 것, 그리고 `make bochs` 를 성공시키는 것입니다. 이를 성공하면 2주차 과정에서 구조를 설계하는 데 큰 도움이 될 것입니다.

문서 수집 및 학습

주요 문서들은 메인 README.md에 정리했습니다.

우선 [GitHub](#)의 README.md를 읽어보면, Bochs는 C++로 짜여진 x86 **에뮬레이터**입니다. 즉, VMware나 Virtualbox에서 구현하는 가상화 또는 JVM같은 JIT 컴파일러와 다릅니다. 에뮬레이터라는 특성 덕분에, x86 뿐만 아니라 ARM, MIPS 등의 다양한 아키텍처 위에서도 x86 환경을 에뮬레이팅 할 수 있습니다.

제가 진행할 프로젝트는 KVM을 이용한 가상화에 초점을 맞췄기에, 이런 부분에서 Bochs와 차이점을 가집니다.

README 아래에는 베이징 ISCA-35에서 발표된, Portable VM에 관한 발표의 [PPT 파일](#)이 있습니다. 이 자료에서는 여러 가상화/에뮬레이션 기술들을 소개합니다. 대부분의 내용을 이해하지는 못했지만, 이해한 부분을 정리하자면:

- JIT나 하드웨어 기반 가상화가 아닌, 한줄씩 번역하는 인터프리터 방식을 차용한다.
 - 따라서 디버깅이 쉽고, 이식성이 좋다. 이를 통해 CPU와의 완전한 격리를 이뤄낼 수 있다.
- fetch-decode-dispatch 순서대로 명령을 처리한다.
 - Bochs는 50% 이상의 시간을 이 사이클에 소요한다.
 - decode된 명령어는 i-cache에 저장된다.
- Lazy Flag 방식을 사용한다.

xv6 구동하기

앞선 단계에서, `make bochs` 는 에러를 출력하며 구동되지 않았습니다. 그리고 그 문제는 `.bochsrc` 파일이 없어서 그런 것 같습니다.

QEMU는 구동 옵션을 인라인으로 다 넣기 때문에 따로 설정파일을 만들지 않아도 작동한 것 같지만, Bochs는 그렇지 않아 `.bochsrc` 파일을 만들거나, 또는 누군가 만들어 놓은 설정 파일을 가져와야 할 것 같습니다.

우선 검색을 통해 다음 설정 파일을 찾았습니다: [dot-bochsrc by Thomas Nguyen](#)

해당 파일을 복사해 그대로 `.bochsrc`로 만들어보았으나, 여전히 같은 에러를 출력하며 잘 작동하지 않았습니다.

추가적으로 정보를 더 찾아봤으나 마땅한 설정 파일을 찾지 못하여, 2차시에 Bochs에 대해 공부하면서 .bochsrc 또한 [공식 Documentation](#)과 같이 공부하겠습니다.

다음주 todo:

- .bochsrc 설정 파일 세팅하기
- Bochs 를 레퍼런스로, 실질적인 VMM 구조 설계하기
- 개발 환경 구축 (C)
- (시간이 남는다면,) Fedora에서 xv6이 컴파일되지 않은 이유 조사. (만약 Fedora가 문제라면 Distrobox를 이용해 컨테이너 위 컴파일 진행)
- (시간이 남는다면,) xv6 소스 코드를 포크하고 개선해 `-Werror` 플래그를 지우지 않고도 성공적으로 컴파일되도록 개선

2주차 연구내용

목표: 구조 설계, 개발 환경 구축, Git 저장소 생성

이번주 todo:

- .bochsrc 설정 파일 세팅하기
- Bochs 를 레퍼런스로, 실질적인 VMM 구조 설계하기
- 개발 환경 구축 (C)
- Fedora에서 xv6이 컴파일되지 않은 이유 조사. (만약 Fedora가 문제라면 Distrobox를 이용해 컨테이너 위 컴파일 진행)
- xv6 소스 코드를 포크하고 개선해 `-Werror` 플래그를 지우지 않고도 성공적으로 컴파일되도록 개선

연구 내용

이번주 목표는 '코드 구조 설계 및 개발 환경 구축'입니다. 또한 저번에 해결하지 못한 bochs 부팅까지 해결할 계획입니다.

xv6 컴파일 문제 해결하기

문제점 분석

이번주에 발생한 xv6 컴파일 관련 문제는 다음 두가지입니다.

1. 컴파일 과정 중 생기는 경고가 `-Werror` 플래그에 걸려 컴파일이 실패하는 것.
2. 컴파일 후, `make qemu` 가 정상적으로 실행되지 않는것.

1번 문제의 경우에는, 저번주에 임시로 `-Werror` 플래그를 제거해서 해결했습니다. 그러나, 교수님께서 '컴파일러 버전에 따라 처리 방식이 다르다'고 조언해주셨으니, 다양한 버전으로 테스트해보는 과정이 필요할 것 같습니다.

2번 문제의 경우에는, LMDE 32bit 환경에서 컴파일해 해결했습니다. Fedora의 multilib 컴파일러가 제대로 작동하지 않은 것 같으니, 다른 배포판으로도 테스트를 해봐야겠습니다.

따라서, 1번과 2번을 동시에 해결하기 위해 다양한 배포판들을 [distrobox](#)로 테스트하기로 결정했습니다. distrobox는 도커 기반의 리눅스 컨테이너를 간단하게 생성할 수 있도록 해주는 툴으로, 권한 설정과 홈 폴더 공유와 같은 번거로운 작업을 간단하게 해결해줍니다.

다른 학교의 사례 분석

무작정 배포판들을 테스트해보기보단, 우선 교수님께서 조언해주신 대로 여러 학교들의 사례를 찾아보았습니다.

- 서울대, 연세대: xv6-riscv 버전을 사용중인 것으로 확인됨 (해당없음)
- KAIST: [KAIST OSLAB](#), 데비안 기반 (우분투로 추정됨)의 운영체제를 사용함.
- 아주대: 공개된 정보 없음.

따라서 KAIST 관련 정보로 검색하다, 다음 [블로그 글](#)을 확인했습니다.

해당 글에서는 Ubuntu 22.04 (64bit)에서 컴파일이 성공한 것을 확인할 수 있었습니다.

따라서, distrobox로 Ubuntu 22.04 환경을 먼저 구성해보았습니다.

Ubuntu 22.04 테스트

우선 distrobox로 Ubuntu 22.04 이미지를 다운로드해, 컨테이너를 생성하고 실행했습니다.

```
distrobox create -i quay.io/toolbx/ubuntu-toolbox:22.04
distrobox enter ubuntu-toolbox-22-04
```

이후, 필요한 패키지들을 설치했습니다. `make` 와 `gcc-multilib` 같은 패키지들은 `build-essential` 메타패키지로 한번에 설치됩니다.

```
sudo apt install build-essential qemu-system -y
```

xv6 레포를 git으로 클론해줬습니다. git은 컨테이너에 기본적으로 설치되어있었습니다.

```
git clone https://github.com/mit-pdos/xv6-public
cd xv6-public
```

바로 `make` 로 컴파일해보았습니다.

```
~ : distrobox — Konsole
새 탭(N) 보기 나누기
복사(C) 붙여넣기(P) 찾기(F)...

gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointe
r -fno-stack-protector -fno-pie -no-pie -fno-pic -nostdinc -I. -c entryother.S
ld -m elf_i386 -N -e start -Ttext 0x7000 -o bootblockother.o entryother.o
objcopy -S -O binary -j .text bootblockother.o entryother
objdump -S bootblockother.o > entryother.asm
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointe
r -fno-stack-protector -fno-pie -no-pie -fno-pic -nostdinc -I. -c initcode.S
ld -m elf_i386 -N -e start -Ttext 0 -o initcode.out initcode.o
objcopy -S -O binary initcode.out initcode
objdump -S initcode.o > initcode.asm
ld -m elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o ioapic.o kalloc.o kb
d.o lapic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o switch.o syscall.o sysfile
.o sysproc.o trapasm.o trap.o uart.o vectors.o vm.o -b binary initcode entryother
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.00794035 s, 645 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 5.5641e-05 s, 9.2 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
393+1 records in
393+1 records out
201448 bytes (201 kB, 197 KiB) copied, 0.000367693 s, 548 MB/s
[seolcu@ubuntu-toolbox-22-04 xv6-public]$
```

컴파일 에러 없이 잘 컴파일 된 것을 볼 수 있었습니다. 이로서, 앞선 1번 `-Werror` 문제는 컴파일러 버전(배포판)을 바꿔서 해결할 수 있었습니다.

따라서, 바로 `make qemu` 로 실행도 해보았습니다.

```
QEMU
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ _
```


문제없이 바로 실행되었습니다... 이로서 2번 문제도 해결했습니다.

그냥 처음부터 우분투 기준으로 해볼걸... 하는 생각이 들지만, 이 과정을 컴파일러 버전의 중요성을 배울 수 있었습니다.

xv6 fork

프로젝트 관리를 위해, xv6 레포지토리를 fork했습니다. 그리고 README에 distrobox를 이용한 컴파일 가이드를 작성했습니다. 앞으로 xv6 코드에 변경사항이 생기면 이 레포지토리로 관리할 것입니다.

<https://github.com/seolcu/xv6-public>

Bochs를 이용한 부팅 도전

우선 컨테이너에 bochs를 설치해줬습니다.

```
sudo apt install bochs
```

그리고 `make bochs` 를 실행했습니다. 1주차와 같은 오류가 발생했습니다. 1주차에 교수님께서 .bochsrc 설정법을 설명해 주셨으니, 그에 따라 고쳐보려고 합니다.

```

xv6-public on  master [?] via C v11.4.0-gcc via v5.34.0
[ubuntu-toolbox-22-04] > make bochs
if [ ! -e .bochsrc ]; then ln -s dot-bochsrc .bochsrc; fi
bochs -q
00000000000i[      ] LTDL_LIBRARY_PATH not set. using compile time default '/usr/lib/x86_64-linux-gnu/bochs/plugins'
=====
                Bochs x86 Emulator 2.7
                Built from SVN snapshot on August  1, 2021
                Timestamp: Sun Aug  1 10:07:00 CEST 2021
=====
00000000000i[      ] BXSHARE not set. using compile time default '/usr/share/bochs'
00000000000i[      ] lt_dlopen is 0x56471551b440
00000000000i[PLUGIN] loaded plugin libbx_biosdev.so
00000000000i[      ] lt_dlopen is 0x56471551b6e0
00000000000i[PLUGIN] loaded plugin libbx_extfpvirq.so
00000000000i[      ] lt_dlopen is 0x5647154db5b0
00000000000i[PLUGIN] loaded plugin libbx_gameport.so
00000000000i[      ] lt_dlopen is 0x5647154dbe30
00000000000i[PLUGIN] loaded plugin libbx_iodebug.so
00000000000i[      ] lt_dlopen is 0x5647154dc6a0
00000000000i[PLUGIN] loaded plugin libbx_parallel.so
00000000000i[      ] lt_dlopen is 0x5647154de2b0
00000000000i[PLUGIN] loaded plugin libbx_serial.so
00000000000i[      ] lt_dlopen is 0x5647154e2510
00000000000i[PLUGIN] loaded plugin libbx_speaker.so
00000000000i[      ] lt_dlopen is 0x5647154e35e0
00000000000i[PLUGIN] loaded plugin libbx_unmapped.so
00000000000i[      ] reading configuration from .bochsrc
00000000000i[      ] lt_dlopen is 0x5647155209b0
00000000000i[PLUGIN] loaded plugin libbx_vga.so
00000000000p[      ] >>PANIC<< .bochsrc:497: directive 'vga_update_interval' not understood
00000000000e[SIM  ] notify called, but no bxevent_callback function is registered
00000000000e[SIM  ] notify called, but no bxevent_callback function is registered
=====
Bochs is exiting with the following message:
[      ] .bochsrc:497: directive 'vga_update_interval' not understood
=====
00000000000i[SIM  ] quit_sim called with exit code 1
make: *** [Makefile:211: bochs] Error 1

```

우선 아래의 Makefile을 보면, 아래 부분에서 dot-bochsrc 파일을 가르키는 소프트링크로 .bochsrc 파일을 생성하는 것을 볼 수 있습니다. 따라서 dot-bochsrc 를 수정하려 합니다.

```

bochs : fs.img xv6.img
    if [ ! -e .bochsrc ]; then ln -s dot-bochsrc .bochsrc; fi
    bochs -q

```

환경변수와 bochsbios

dot-bochsrc를 읽어보기 전에, 앞선 make bochs 에서의 출력 메시지에서 환경변수 설정을 볼 수 있었습니다.

```

00000000000i[      ] LTDL_LIBRARY_PATH not set. using compile time default
'/usr/lib/x86_64-linux-gnu/bochs/plugins'
=====
                Bochs x86 Emulator 2.7
                Built from SVN snapshot on August  1, 2021
                Timestamp: Sun Aug  1 10:07:00 CEST 2021

```

```
=====
000000000000i[      ] BXSHARE not set. using compile time default
'/usr/share/bochs'
```

LTDL_LIBRARY_PATH 은 /usr/lib/x86_64-linux-gnu/bochs/plugins 로 설정되고,

BXSHARE 는 /usr/share/bochs 로 설정되는 것을 볼 수 있었습니다.

컨테이너에서 확인하니, 두 경로 모두 정상적으로 존재하는 것을 볼 수 있었습니다.

그러나, dot-bochsrc에서 다음과 같은 라인을 발견할 수 있었습니다.

```
romimage: file=$BXSHARE/BIOS-bochs-latest
```

이를 보면, /usr/share/bochs/BIOS-bochs-latest 라는 파일이 존재해야 하는 것을 알 수 있습니다. 그러나, 실제로는 해당 위치에 파일이 존재하지 않았습니다. 따라서 apt search bochs 를 하니, 몇몇 관련 패키지들이 설치되지 않았다는 사실을 알 수 있었습니다.

따라서 sudo apt install bochsbios 를 진행하니, 해당 위치에 BIOS-bochs-latest 파일이 정상적으로 생겨났습니다.

비슷하게, bochs-term, vgabios 등도 설치가 필요했습니다.

Ubuntu의 Bochs 패키지로 실행 실패

앞선 패키지들을 모두 설치했음에도, 다음과 같은 에러가 계속 발생했습니다.

```
000000000000p[      ] >>PANIC<< .bochsrc:497: directive 'vga_update_interval'
not understood
000000000000e[SIM    ] notify called, but no bxevent_callback function is
registered
000000000000e[SIM    ] notify called, but no bxevent_callback function is
registered
=====
Bochs is exiting with the following message:
[      ] .bochsrc:497: directive 'vga_update_interval' not understood
=====
000000000000i[SIM    ] quit_sim called with exit code 1
make: *** [Makefile:211: bochs] Error 1
```

에러에 해당하는 부분을 전부 주석처리해보았으나, 메모리 부분에서 계속 커널 패닉이 발생하며 실행에 실패했습니다.

따라서, 잠시 멈춰서 문제의 원인을 떠올려봤습니다.

생각해보니, xv6의 코드는 약 15년 전에 작성되었으므로, 그 당시의 Bochs의 설정 파일 형식과 현재의 설정 파일 형식이 많이 다르겠다는 생각이 들었습니다.

그래서 16.04 컨테이너를 새로 만들어 해봤으나, 여전히 같은 문제에 부딪혔습니다.

그러다, 1주차에서 보았던 Notes 파일이 떠올랐습니다.

Bochs 2.2.6

xv6 레포에는 Notes라는 파일이 있고, 그 파일 안에는 이런 내용이 있습니다.

```
bochs 2.2.6:
./configure --enable-smp --enable-disasm --enable-debugger --enable-all-
optimizations --enable-4meg-pages --enable-global-pages --enable-pae --
disable-reset-on-triple-fault
bochs CVS after 2.2.6:
./configure --enable-smp --enable-disasm --enable-debugger --enable-all-
optimizations --enable-4meg-pages --enable-global-pages --enable-pae
```

따라서, 이를 참고해 2.2.6 버전을 사용해보기로 했습니다.

해당 버전은 GitHub에는 존재하지 않아서, [Sourceforge](https://sourceforge.net/project/bochs/bochs/2.2.6/bochs-2.2.6.tar.gz)에서 [Bochs 2.2.6](https://sourceforge.net/project/bochs/bochs/2.2.6/bochs-2.2.6.tar.gz)을 다운받을 수 있었습니다.

```
wget https://master.dl.sourceforge.net/project/bochs/bochs/2.2.6/bochs-
2.2.6.tar.gz
tar -xvf bochs-2.2.6.tar.gz
rm bochs-2.2.6.tar.gz
cd bochs-2.2.6
```

그대로 ./configure ... 를 하니, 다음과 같은 에러가 발생했습니다.

```
ERROR: X windows gui was selected, but X windows libraries were not found.
```

Distrobox 컨테이너는 minimal이기에 xorg는 기본적으로 설치되지 않습니다. 따라서, xorg-dev 패키지를 설치해 라이브러리를 충족해주고, ./configure ... 를 실행했습니다.

```
sudo apt install xorg-dev
./configure --enable-smp --enable-disasm --enable-debugger --enable-all-
optimizations --enable-4meg-pages --enable-global-pages --enable-pae --
disable-reset-on-triple-fault
```

./configure ... 는 성공적으로 진행되었습니다.

```
checking for cc_r... gcc
checking for specified CMOS start time (deprecated)... no
checking for configuration interface (deprecated)... no
checking for control panel (deprecated)... no
checking for VGA emulation (deprecated)... no
checking for number of processors (deprecated)... no
checking for gzip... /usr/bin/gzip
checking for tar... /usr/bin/tar
configure: creating ./config.status
config.status: creating Makefile
config.status: creating iodev/Makefile
config.status: creating bx_debug/Makefile
config.status: creating bios/Makefile
config.status: creating cpu/Makefile
config.status: creating memory/Makefile
config.status: creating gui/Makefile
config.status: creating disasm/Makefile
config.status: creating instrument/stubs/Makefile
config.status: creating misc/Makefile
config.status: creating fpu/Makefile
config.status: creating doc/docbook/Makefile
config.status: creating build/linux/bochs-dlx
config.status: creating bxversion.h
config.status: creating build/macosx/Info.plist
config.status: creating build/win32/nsis/Makefile
config.status: creating build/win32/nsis/bochs.nsi
config.status: creating host/linux/pcidev/Makefile
config.status: creating config.h
config.status: creating ltdlconf.h
[seolcu@ubuntu-toolbox-22-04 bochs-2.2.6]$
```

하지만, `make` 단계에서 컴파일 에러가 발생해 또 막히게 되었습니다.

```

../bochs.h: In member function 'char* iofunctions::getaction(int)':
../bochs.h:307:36: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
307 |         static char *name[] = { "ignore", "report", "ask", "fatal" };
    |                                ^~~~~~
../bochs.h:307:46: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
307 |         static char *name[] = { "ignore", "report", "ask", "fatal" };
    |                                ^~~~~~
../bochs.h:307:56: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
307 |         static char *name[] = { "ignore", "report", "ask", "fatal" };
    |                                ^~~~~~
../bochs.h:307:63: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
307 |         static char *name[] = { "ignore", "report", "ask", "fatal" };
    |                                ^~~~~~

In file included from harddrv.cc:39:
hdimage.h: At global scope:
hdimage.h:277:8: error: extra qualification 'sparse_image_t::' on member 'get_physical_offset' [-fpermissive]
277 |         sparse_image_t::
    |         ^~~~~~
hdimage.h:282:8: error: extra qualification 'sparse_image_t::' on member 'set_virtual_page' [-fpermissive]
282 |         sparse_image_t::
    |         ^~~~~~

In file included from ../bochs.h:447,
                  from iodev.h:32,
                  from harddrv.cc:38:
harddrv.cc: In function 'int libharddrv_LTX_plugin_init(plugin_t*, plugin_type_t, int, char**)':
../plugin.h:33:29: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
33 | #define BX_PLUGIN_HARDDRV "harddrv"
    |                             ^~~~~~
../plugin.h:50:81: note: in definition of macro 'BX_REGISTER_DEVICE_DEVMODEL'
50 | #define BX_REGISTER_DEVICE_DEVMODEL(a,b,c,d) pluginRegisterDeviceDevmodel(a,b,c,d)
    |                                                                 ^
harddrv.cc:97:59: note: in expansion of macro 'BX_PLUGIN_HARDDRV'
97 |     BX_REGISTER_DEVICE_DEVMODEL(plugin, type, theHardDrive, BX_PLUGIN_HARDDRV);
    |                                                             ^~~~~~
harddrv.cc: In constructor 'bx_hard_drive_c::bx_hard_drive_c()':
harddrv.cc:115:11: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
115 |     put("HD");
    |         ^~~~

make[1]: *** [Makefile:120: harddrv.o] Error 1
make[1]: Leaving directory '/home/seolcu/문서/코드/bochs-2.2.6/iodev'
make: *** [Makefile:259: iodev/libiodev.a] Error 2
[seolcu@ubuntu-toolbox-22-04 bochs-2.2.6]$

```

컴파일러 버전 문제를 의심하여, Ubuntu 16.04에서 다시 해 보았으나 여전히 같은 문제가 발생했습니다.

따라서 우선 이 문제는 보류해두고, 오늘의 주 목표인 코드 구조 설계 및 개발 환경 구축을 진행했습니다.

코드 구조 설계

Bochs의 코드를 참고해 코드를 작성할 것이므로, 우선 Bochs 2.2.6의 코드를 읽어보았습니다.

Bochs 코드를 컴파일해 실행해보지는 못해도, 전체적인 로직을 보는 것은 가능하겠다고 생각했습니다.

우선 그 전에, Bochs 2.2.6 코드를 온라인으로 볼 수 있는 곳이 적어, 편한 브라우징과 기록을 위해 코드를 [GitHub](#)에 올렸습니다. (LGPL 라이선스 파일도 같이 올렸으니 괜찮을 것 같습니다.)

main.cc

우선 코드의 가장 중심인 main.cc 를 살펴봤습니다.

그 중 `main` 함수는 아래와 같습니다:

```
int main (int argc, char *argv[])
{
    bx_startup_flags.argc = argc;
    bx_startup_flags.argv = argv;
    #if BX_WITH_SDL && defined(WIN32)
        // if SDL/win32, try to create a console window.
        RedirectIOToConsole ();
    #endif
    #if defined(WIN32)
        SetConsoleTitle("Bochs for Windows - Console");
    #endif
    return bxmain ();
}
```

처음에 입력하는 인자를 `bx_startup_flags` 라는 구조체 인스턴스에 저장하는 것을 볼 수 있습니다. 해당 인스턴스는 전역변수로 선언되어있습니다. 이후 `bxmain` 함수를 실행합니다.

이어지는 `bxmain` 함수는 다음과 같습니다.

```
int bxmain () {
    #ifdef HAVE_LOCALE_H
        // Initialize locale (for isprint() and other functions)
        setlocale (LC_ALL, "");
    #endif
    bx_user_quit = 0;
    bx_init_siminterface (); // create the SIM object
    static jmp_buf context;
    if (setjmp (context) == 0) {
        SIM->set_quit_context (&context);
        if (bx_init_main (bx_startup_flags.argc, bx_startup_flags.argv) < 0)
            return 0;
        // read a param to decide which config interface to start.
        // If one exists, start it. If not, just begin.
        bx_param_enum_c *ci_param = SIM->get_param_enum
(BXP_SEL_CONFIG_INTERFACE);
        char *ci_name = ci_param->get_choice (ci_param->get());
        if (!strcmp(ci_name, "textconfig")) {
    #if BX_USE_TEXTCONFIG
            init_text_config_interface (); // in textconfig.h
    #else
        BX_PANIC(("configuration interface 'textconfig' not present"));
    #endif
        }
}
```

```

#if BX_WITH_WX
    else if (!strcmp(ci_name, "wx")) {
        PLUG_load_plugin(wx, PLUGTYPE_CORE);
    }
#endif
    else {
        BX_PANIC (("unsupported configuration interface '%s'", ci_name));
    }
    int status = SIM->configuration_interface (ci_name, CI_START);
    if (status == CI_ERR_NO_TEXT_CONSOLE)
        BX_PANIC (("Bochs needed the text console, but it was not usable"));
    // user quit the config interface, so just quit
} else {
    // quit via longjmp
}
SIM->set_quit_context (NULL);
#if defined(WIN32)
    if (!bx_user_quit) {
        // ask user to press ENTER before exiting, so that they can read
        messages
        // before the console window is closed. This isn't necessary after
        pressing
        // the power button.
        fprintf (stderr, "\nBochs is exiting. Press ENTER when you're ready to
        close this window.\n");
        char buf[16];
        fgets (buf, sizeof(buf), stdin);
    }
#endif
    return SIM->get_exit_code ();
}

```

초반부에서 `bx_init_siminterface` 함수를 실행해 시뮬레이터를 세팅합니다. 이와 관련한 주석을 `gui/siminterface.cc` 에서 발견할 수 있었습니다.

```

bx_simulator_interface_c *SIM = NULL;
logfunctions *siminterface_log = NULL;
#define LOG_THIS siminterface_log->

// bx_simulator_interface just defines the interface that the Bochs
// simulator
// and the gui will use to talk to each other. None of the methods of
// bx_simulator_interface are implemented; they are all virtual. The
// bx_real_sim_c class is a child of bx_simulator_interface_c, and it
// implements all the methods. The idea is that a gui needs to know only

```



```
// definition of bx_simulator_interface to talk to Bochs. The gui should
// not need to include bochs.h.
//
// I made this separation to ensure that all guis use the siminterface to do
// access bochs internals, instead of accessing things like
// bx_keyboard.s.internal_buffer[4] (or whatever) directly. -Bryce
//
```

이를 읽어보면, `bx_simulator_interface` 라는 구조체는 Bochs와 GUI의 통신을 담당하는 인터페이스라고 합니다. 그래서 기본적으로 `bx_simulator_interface` 구조체에는 실제 메서드가 구현되어 있지 않고, 비어있습니다.

실제 메서드들은 `bx_real_sim_c` 라는 클래스를 통해 구현되는데, 이를 생성하는 과정은 방금 본 `bx_init_siminterface` 함수에서 발견할 수 있습니다. `bx_init_siminterface` 함수는 아래와 같습니다.

```
void bx_init_siminterface ()
{
    siminterface_log = new logfunctions ();
    siminterface_log->put ("CTRL");
    siminterface_log->settype(CTRLLOG);
    if (SIM == NULL)
        SIM = new bx_real_sim_c();
}
```

이를 통해 보면, `bxmain` 함수에서 `bx_init_siminterface` 를 실행하는 순간부터 `SIM` 인스턴스가 실제로 사용 가능한 형태가 된다는 사실을 알 수 있습니다.

그 아래를 보면 `bx_init_main` 함수를 실행하는 것을 볼 수 있습니다.

```
if (bx_init_main (bx_startup_flags.argc, bx_startup_flags.argv) < 0)
    return 0;
```

리턴값이 0보다 작은 경우에 0을 리턴하는 것을 보아, `bx_init_main` 함수에서 에러가 발생하면 즉시 종료하는 것 같습니다.

따라서 `bx_init_main` 함수를 살펴보기로 했습니다.

```
int norcfile = 1;

if (load_rcfile) {
    /* parse configuration file and command line arguments */
#ifdef WIN32
    int length;
```

```

    if (bochsrc_filename != NULL) {
        lstrcpy(bx_startup_flags.initial_dir, bochsrc_filename);
        length = lstrlen(bx_startup_flags.initial_dir);
        while ((length > 1) && (bx_startup_flags.initial_dir[length-1] != 92))
length--;
        bx_startup_flags.initial_dir[length] = 0;
    } else {
        bx_startup_flags.initial_dir[0] = 0;
    }
#endif
    if (bochsrc_filename == NULL) bochsrc_filename = bx_find_bochsrc ();
    if (bochsrc_filename)
        norcfile = bx_read_configuration (bochsrc_filename);
}

if (norcfile) {
    // No configuration was loaded, so the current settings are unusable.
    // Switch off quick start so that we will drop into the configuration
    // interface.
    if (SIM->get_param_enum(BXP_BOCHS_START)->get() == BX_QUICK_START) {
        if (!SIM->test_for_text_console ())
            BX_PANIC(("Unable to start Bochs without a bochsrc.txt and without a
text console"));
        else
            BX_ERROR(("Switching off quick start, because no configuration file
was found."));
    }
    SIM->get_param_enum(BXP_BOCHS_START)->set (BX_LOAD_START);
}

// parse the rest of the command line. This is done after reading the
// configuration file so that the command line arguments can override
// the settings from the file.
if (bx_parse_cmdline (arg, argc, argv)) {
    BX_PANIC(("There were errors while parsing the command line"));
    return -1;
}
// initialize plugin system. This must happen before we attempt to
// load any modules.
plugin_startup();
return 0;
}

```

코드가 길어서 조금씩 나눠서 보기로 했습니다. 앞부분에서는 옵션 설정 등 기본적인 init 과정을 거칩니다.

주목한 점은 끝부분에서 `.bochsrc` 를 처리한다는 점입니다. 이 과정을 담당하는 함수들은 `bx_find_bochsrc`, `bx_read_configuration`, `bx_parse_cmdline` 등으로 보입니다. 그리고 이 모든 함수들은 `config.cc` 에 있습니다. 이 다음으로는 `config.cc` 를 살펴보면 될 것 같습니다.

그러나, `config.cc` 는 3000줄이 넘어갈 정도로 길어서, `main.cc` 에서 했던 것처럼 하나하나 뜯어보며 읽기는 어려울 것 같습니다.

따라서, 3주차부터는 코드를 리뷰하는데 있어서 좀 다른 방법을 생각해봐야 할 것 같습니다.

다음주 todo:

- Bochs 코드 리뷰
- 프로젝트 코드 구조 설계
- (KVM) VM 생성 및 메모리 할당 코드 작성

3주차 연구내용

목표: VM 생성 및 메모리 할당 코드 작성

저번주 todo:

- Bochs 코드 리뷰
- 프로젝트 코드 구조 설계
- (KVM) VM 생성 및 메모리 할당 코드 작성

연구 내용

Bochs 2.2.6 컴파일

Configure

우선 저번주에 이어서 Bochs 2.2.6 컴파일을 이어서 해봤습니다.

저번주에 사용한 configure 명령은 아래와 같았습니다. (xv6 레포의 Notes 파일에서 발견된 명령어입니다.)

```
./configure --enable-smp --enable-disasm --enable-debugger --enable-all-optimizations --enable-4meg-pages --enable-global-pages --enable-pae --disable-reset-on-triple-fault
```

그리고 이를 컴파일하면 다음과 같은 에러가 발생했었습니다.

```
ERROR: X windows gui was selected, but X windows libraries were not found.
```

저번주에는 xorg를 설치해 해결하려 했으나, 교수님과 상담 후 `--with-term` 옵션을 주기로 결정했습니다. 또한, 멀티코어를 활성화하는 `--enable-smp` 옵션과, 최적화 및 페이징과 관련된 옵션들인 `--enable-all-optimizations --enable-4meg-pages --enable-global-pages --enable-pae` 도 모두 삭제하기로 했습니다.

```
./configure --enable-disasm --enable-debugger --disable-reset-on-triple-fault --with-term
```

그랬더니, 다음과 같은 에러가 발생했습니다.

```
Curses library not found: tried curses, ncurses, termlib and pdcurses.
```

Curses 관련 라이브러리가 없다고 떠서, 아래 명령어로 설치해줬습니다. (Ubuntu 22.04 기준)

```
sudo apt install libncurses-dev
```

이후 같은 옵션으로 configure가 정상적으로 진행되었습니다. (X window 관련 문제가 발생하지 않는 걸 보니, term을 켜면 vga는 저절로 꺼지는 것 같습니다.) 따라서 바로 make 를 진행해봤습니다. 그랬더니 수많은 warning과 함께, 아래와 같은 에러가 발생했습니다.

```
307 |         static char *name[] = { "ignore", "report", "ask", "fatal" };
    |
In file included from harddrv.cc:39:
hdiimage.h: At global scope:
hdiimage.h:277:8: error: extra qualification 'sparse_image_t::' on member 'get_physical_offset' [-fpermissive]
 277 |         sparse_image_t::
    |         ^~~~~~
hdiimage.h:282:8: error: extra qualification 'sparse_image_t::' on member 'set_virtual_page' [-fpermissive]
 282 |         sparse_image_t::
    |         ^~~~~~
In file included from ../bochs.h:447,
                from iodev.h:32,
                from harddrv.cc:38:
```

아무래도 GCC 버전이 달라 C++ 표준이 달라져서 생긴 에러인 것 같습니다.

GCC 버전 낮추기

따라서 시스템의 GCC 버전을 더 낮추기로 했습니다. 그래서 더 찾아보던 중, [C++과 GCC에 관한 블로그](#)를 하나 발견했습니다. 이 글에는 이렇게 쓰여있습니다:

1998년에 첫 번째 표준인 C++ 98이 공개된 후 오랜 기간동안 정체기를 거치다가 2011년이 되어서야 새로운 개념들이 추가된 버전이 공개되었습니다. C++ 11부터 Modern C++ 이라고 부릅니다.

- trusty: Ubuntu 14.04, xenial: Ubuntu 16.04, bionic: Ubuntu 18.04
- focal: Ubuntu 20.04,
- jammy: Ubuntu 22.04, kinetic: Ubuntu 22.10
- lunar: Ubuntu 23.04

우선 현재 사용중이던 Ubuntu 22.04에는 gcc 4.8 미만을 설치할 수 없었습니다.

따라서 distrobox에서 제공하는 가장 낮은 Ubuntu 버전인 Ubuntu 16.04를 설치하고 build-essential 을 설치해봤습니다.

```
distrobox create -i quay.io/toolbx/ubuntu-toolbox:16.04
```

```
sudo apt install build-essential
```

이후 gcc 버전을 확인해보았습니다.

```
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

기본 설치된 gcc 버전은 5.4.0이었습니다. 따라서 apt로 gcc-4.7을 직접 설치해보았습니다.

```
sudo apt install gcc-4.7 gcc-4.7-multilib g++-4.7 g++-4.7-multilib
```

그리고 gcc 4.7을 기본 컴파일러로 사용하기 위해, [update-alternatives와 관련된 블로그 글](#)을 통해 gcc 4.7을 기본으로 설정했습니다.

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.7 10
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.7 10
```

이제 gcc 4.7이 설치되었습니다:

```
$ gcc --version
gcc (Ubuntu/Linaro 4.7.4-3ubuntu12) 4.7.4
Copyright (C) 2012 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

따라서 바로 configure 후 make 해보았습니다:

```
../bochs.h: In member function 'char* iofunctions::getaction(int)':
../bochs.h:307:64: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../bochs.h:307:64: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../bochs.h:307:64: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
../bochs.h:307:64: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
In file included from harddrv.cc:39:0:
hdimage.h: At global scope:
hdimage.h:277:8: error: extra qualification 'sparse_image_t::' on member 'get_physical_offset' [-fpermissive]
hdimage.h:282:8: error: extra qualification 'sparse_image_t::' on member 'set_virtual_page' [-fpermissive]
harddrv.cc: In function 'int libharddrv_LTX_plugin_init(plugin_t*, plugin_type_t, int, char**)':
harddrv.cc:97:3: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
harddrv.cc: In constructor 'bx_hard_drive_c::bx_hard_drive_c()':
harddrv.cc:115:15: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
Makefile:120: recipe for target 'harddrv.o' failed
make[1]: *** [harddrv.o] Error 1
make[1]: Leaving directory '/home/seolcu/문서/코드/bochs-2.2.6/iodev'
Makefile:259: recipe for target 'iodev/libiodev.a' failed
make: *** [iodev/libiodev.a] Error 2
```

여전히 같은 문제로 컴파일에 실패했습니다.

himage.h 고치기

컴파일러 버전을 최대한 낮췄음에도 컴파일이 되지 않으니, 직접 코드를 수정해 컴파일되도록 만들기로 했습니다.

컴파일 에러는 다음과 같습니다:

```
himage.h: At global scope:
himage.h:277:8: error: extra qualification 'sparse_image_t::' on member
'get_physical_offset' [-fpermissive]
himage.h:282:8: error: extra qualification 'sparse_image_t::' on member
'set_virtual_page' [-fpermissive]
```

해당하는 부분은 `idev/himage.h` 파일입니다. 문제가 되는 부분은 아래와 같습니다:

```
off_t
#ifdef PARANOID
    sparse_image_t::
#endif

    get_physical_offset();

void
#ifdef PARANOID
    sparse_image_t::
#endif

    set_virtual_page(Bit32u new_virtual_page);
```

`sparse_image_t` 클래스 안에서 그 클래스를 스스로 호출해 생긴 문제인 것 같습니다. 이 코드를 다음과 같이 고쳤습니다:

```
off_t get_physical_offset();
void set_virtual_page(Bit32u new_virtual_page);
```

그랬더니, 다음과 같은 에러가 발생했습니다:

```

../bochs.h:307:64: warning: deprecated conversion from string constant to 'char*' [-Wwrite-strings]
symbols.cc: At global scope:
symbols.cc:135:10: error: 'hash_map' does not name a type
symbols.cc:143:1: error: 'hash_map' does not name a type
symbols.cc: In constructor 'context_t::context_t(Bit32u)':
symbols.cc:150:5: error: 'map' was not declared in this scope
symbols.cc: In static member function 'static context_t* context_t::get_context(Bit32u)':
symbols.cc:171:12: error: 'map' was not declared in this scope
Makefile:72: recipe for target 'symbols.o' failed
make[1]: *** [symbols.o] Error 1
make[1]: Leaving directory '/home/seolcu/문서/코드/bochs-2.2.6/bx_debug'
Makefile:264: recipe for target 'bx_debug/libdebug.a' failed
make: *** [bx_debug/libdebug.a] Error 2

```

symbols.cc 고치기

에러 메시지는 다음과 같았습니다:

```

symbols.cc: At global scope:
symbols.cc:135:10: error: 'hash_map' does not name a type
symbols.cc:143:1: error: 'hash_map' does not name a type
symbols.cc: In constructor 'context_t::context_t(Bit32u)':
symbols.cc:150:5: error: 'map' was not declared in this scope
symbols.cc: In static member function 'static context_t*
context_t::get_context(Bit32u)':
symbols.cc:171:12: error: 'map' was not declared in this scope
Makefile:72: recipe for target 'symbols.o' failed
make[1]: *** [symbols.o] Error 1
make[1]: Leaving directory '/home/seolcu/문서/코드/bochs-2.2.6/bx_debug'
Makefile:264: recipe for target 'bx_debug/libdebug.a' failed
make: *** [bx_debug/libdebug.a] Error 2

```

문제가 되는 부분은 bx_debug/symbols.cc 파일이었습니다. 라인 135 근처를 확인했습니다.

```

private:
    static hash_map<int, context_t*>* map;
    // Forward references (find name by address)
    set<symbol_entry_t*, lt_symbol_entry_t*> syms;
    // Reverse references (find address by name)
    set<symbol_entry_t*, lt_rsymbol_entry_t*> rsyms;
    Bit32u id;
};

hash_map<int, context_t*>* context_t::map = new hash_map<int, context_t*>;

```

hash_map이라는 것을 타입을 사용하지만, 컴파일러가 이를 못 찾는 것 같습니다.

따라서 이 타입을 불러오는 include 부분을 살펴봤습니다.


```

/* Haven't figured out how to port this code to OSF1 cxx compiler.
   Until a more portable solution is found, at least make it easy
   to disable the template code: just set BX_HAVE_HASH_MAP=0
   in config.h */
#ifdef BX_HAVE_HASH_MAP
#include <hash_map>
#elif BX_HAVE_HASH_MAP_H
#include <hash_map.h>
#endif

```

OSF1 cxx 컴파일러와의 호환성을 위해 임시로 사용한 전역변수의 흔적이 남아있습니다. 또한 hash_map 이라는 타입이, unordered_map 이 등장하며 비표준으로 옮겨졌으므로, <ext/hash_map> 으로 바꿔졌습니다.

```

#include <ext/hash_map>
using namespace __gnu_cxx;

```

make 해보니 다음과 같은 에러가 발생했습니다:

```

yacc -p bx -d parser.y
make[1]: yacc: Command not found
Makefile:104: recipe for target 'parser.c' failed
make[1]: *** [parser.c] Error 127
make[1]: Leaving directory '/home/seolcu/문서/코드/bochs-2.2.6/bx_debug'
Makefile:264: recipe for target 'bx_debug/libdebug.a' failed
make: *** [bx_debug/libdebug.a] Error 2

```

yacc가 설치되지 않은 것 같아, 터미널에 입력하니 bison 패키지 설치를 권유받았습니다. 따라서 bison 을 설치하고 make 를 진행해보았더니, 드디어 정상적으로 컴파일되었습니다. 따라서 바로 sudo make install 로 설치까지 진행했습니다.

그리고, 지금까지의 과정을 통해 알아낸 Bochs 2.2.6 컴파일 방법을 [seolcu/bochs-2.2.6](https://seolcu.github.io/bochs-2.2.6/) 레포에 가이드로 작성했습니다.

Bochs 2.2.6으로 xv6 부팅

바로 xv6를 실행하기 위해, make 후 make bochs 해봤습니다.

```
bochs -q
=====
                Bochs x86 Emulator 2.2.6
                Build from CVS snapshot on January 29, 2006
=====
000000000000i[      ] reading configuration from .bochsrc
000000000000p[CTRL ] >>PANIC<< numerical parameter Number of CPUs in SMP mode was set to 2, which is out of range 1 to 1
000000000000i[SYS  ] Last time is 0
=====
Bochs is exiting with the following message:
[CTRL ] numerical parameter Number of CPUs in SMP mode was set to 2, which is out of range 1 to 1
=====
000000000000i[CPU  ] real mode
000000000000i[CPU  ] CS.d_b = 16 bit
000000000000i[CPU  ] SS.d_b = 16 bit
000000000000i[CPU  ] 1-5M: 00000000-5M: 00000000-5M: 00000000-5M: 00000000
```

CPU 개수 관련 에러가 발생했습니다. 역시 `.bochsrc` 파일을 수정할 필요가 있어보입니다.

CPU 설정

원래 설정:

```
cpu: count=2, ips=10000000
```

CPU 개수 1개로 조정:

```
cpu: count=1, ips=10000000
```

ATA 설정

CPU 설정 이후 다음과 같은 에러가 발생했습니다:

```
=====
                Bochs x86 Emulator 2.2.6
                Build from CVS snapshot on January 29, 2006
=====
000000000000i[      ] reading configuration from .bochsrc
000000000000i[      ] installing term module as the Bochs GUI
000000000000i[      ] Warning: no rc file specified.
000000000000i[      ] using log file bochsout.txt
=====
Bochs is exiting with the following message:
[HD   ] ata0/1 image size doesn't match specified geometry
=====
Makefile:210: recipe for target 'bochs' failed
make: *** [bochs] Error 1
```

이 부분과 관련된 설정은 다음 부분인 것 같았습니다:

```
ata0-master: type=disk, mode=flat, path="xv6.img", cylinders=100, heads=10,
spt=10
ata0-slave: type=disk, mode=flat, path="fs.img", cylinders=1024, heads=1,
spt=1
```

뒤에 있는 cylinders, heads, spt 값이 올바르지 않아서 부팅에 문제가 생긴건지 확인하기 위해, [bochsrc 공식 문서](#)와 [위키피디아 CHS 문서](#)를 읽으며 값을 계산했습니다.

디스크 사이즈 계산식은 다음과 같습니다:

```
총 크기 (바이트) = 실린더 수 × 헤드 수 × 트랙당 섹터 수 × 섹터당 바이트 수(512)
```

따라서, 다음과 같이 정리가 가능합니다:

```
총 크기 (바이트) = cylinders * heads * spt(secter per track) * 512
```

ata0-master

따라서 ata0-master의 디스크 사이즈 기대값은, $100 * 10 * 10 * 512 = 5120000$ 입니다. 이 값이 xv6.img의 크기와 일치해야합니다. 확인해보면:

```
$ ls -l xv6.img
-rw-r--r--. 1 seolcu seolcu 5120000  9월 23 12:08 xv6.img
```

이미지 사이즈가 5120000이므로, ata0-master의 디스크 사이즈는 맞습니다.

ata0-slave

다음으로 ata0-slave의 디스크 사이즈 기대값은, $1024 * 1 * 1 * 512 = 524288$ 입니다. fs.img를 확인해 보면:

```
$ ls -l fs.img
-rw-r--r--. 1 seolcu seolcu 512000  9월 23 12:08 fs.img
```

이미지 사이즈가 512000으로, 일치하지 않는 것을 확인할 수 있었습니다.

이를 해결하기 위해 cylinders를 1024에서 1000으로 조정했습니다.:

```
ata0-master: type=disk, mode=flat, path="xv6.img", cylinders=100, heads=10,
spt=10
```

```
ata0-slave: type=disk, mode=flat, path="fs.img", cylinders=1000, heads=1,
spt=1
```

디버거 때문? 인풋 안 들어감

이후 make bochs를 하니, 아래와 같이 뭔가 이상하게 돌아갔습니다. 아무 입력이 들어가지 않았습니
다.

```
Next at t=0
      (0) [0xffffffff] f000:fff0 (unk. ctxt): jmp far f000:e05b          ; ea5be000f0
                                           <bochs:1>
```

bochs 컴파일할 때 `--enable-debugger` 옵션을 켜서 디버거로 들어간건가 하는 생각에, 그 플래그를
빼고 다시 컴파일하고 실행해보았습니다.

SMP

```
Plex86/Bochs VGABios 0.5d 29 Dec 2005
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

NO Bochs VBE Support available!

Bochs BIOS - build: 01/25/06
$Revision: 1.160 $ $Date: 2006/01/25 17:51:49 $
Options: apmbios pcibios eltorito

ata0 master: Generic 1234 ATA-2 Hard-Disk (4 MBytes)
ata0  slave: Generic 1234 ATA-2 Hard-Disk (0 MBytes)

Booting from Hard Disk...
lapicid 0: panic: Expect to run on an SMP
801031a1 80102e80 0 0 0 0 0 0 0 0
```

SMP가 꺼져 있어서 커널 패닉이 발생한 것 같습니다. xv6에서 듀얼코어 이상을 요구하는 것 같습니다.
우선은 다음주로 넘겨줘야겠습니다.

다음주 todo

- 전체 점검

4주차 연구내용

목표: 계획 점검, 오픈소스 코드 리뷰, 추가 학습

저번주 todo:

- 전체 점검

연구내용

Bochs에서 SMP 지원 해제하기

저번주에 Bochs 2.2.6을 컴파일하고 xv6를 디버거 없이 부팅해보았을때, 다음과 같은 에러가 발생했습니다.

```
Plex86/Bochs VGABios 0.5d 29 Dec 2005
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

NO Bochs VBE Support available!

Bochs BIOS - build: 01/25/06
$Revision: 1.160 $ $Date: 2006/01/25 17:51:49 $
Options: apmbios pcibios eltorito

ata0 master: Generic 1234 ATA-2 Hard-Disk (4 MBytes)
ata0 slave: Generic 1234 ATA-2 Hard-Disk (0 MBytes)

Booting from Hard Disk...
lapicid 0: panic: Expect to run on an SMP
801031a1 80102e80 0 0 0 0 0 0 0 0
```

```
Booting from Hard Disk...
lapicid 0: panic: Expect to run on an SMP
801031a1 80102e80 0 0 0 0 0 0 0 0
```

교수님께서 xv6가 듀얼코어를 요구할 것 같지는 않다고 하셨기에, .bochsrc 수정 없이 bochs 2.2.6 컴파일 옵션만 수정해보기로 했습니다.

따라서 `--enable-smp` 옵션을 주어 컴파일해보았습니다.

```
./configure --enable-smp --enable-disasm --disable-reset-on-triple-fault --with-term
```

컴파일 이후 `make bochs` xv6를 부팅해보니, 되는듯 하더니 다음과 같은 에러가 발생했습니다.

```
=====
                        Bochs x86 Emulator 2.2.6
                Build from CVS snapshot on January 29, 2006
=====
000000000000i[      ] reading configuration from .bochsrc
000000000000i[      ] installing term module as the Bochs GUI
000000000000i[      ] using log file bochsout.txt
=====
Bochs is exiting with the following message:
[CPU0 ] exception(): 3rd (14) exception with no resolution
=====
Makefile:210: recipe for target 'bochs' failed
make: *** [bochs] Error 1
```

xv6가 아닌 bochs 에러로 보여 bochs 소스 코드에서 `exception with no resolution` 를 검색해보니, `cpu/exception.cc` 파일에서 다음 코드를 발견할 수 있었습니다:

```
#if BX_RESET_ON_TRIPLE_FAULT
    BX_ERROR(("exception(): 3rd (%d) exception with no resolution, shutdown
status is %02xh, resetting", vector, DEV_cmos_get_reg(0x0f)));
    debug(BX_CPU_THIS_PTR prev_eip);
    bx_pc_system.Reset(BX_RESET_SOFTWARE);
#else
    BX_PANIC(("exception(): 3rd (%d) exception with no resolution",
vector));
    BX_ERROR(("WARNING: Any simulation after this point is completely
bogus."));
#endif
#if BX_DEBUGGER
    bx_guard.special_unwind_stack = true;
#endif
    longjmp(BX_CPU_THIS_PTR jmp_buf_env, 1); // go back to main decode loop
}
```

BX_RESET_ON_TRIPLE_FAULT 이라는 상수값에 따라 명령이 분기되는데, 이 옵션은 bochs configure 과정에서 사용한 옵션인 --disable-reset-on-triple-fault 옵션과 관련이 있어 보였습니다. 뭔가 CPU에 문제가 발생하는 상황을 가정한 것 같은데, 이 옵션을 넣으면 무조건 else문으로 빠져 작동을 멈추도록 설계된 것 같습니다.

그래서 우선 configure에서 해당 플래그를 삭제하고 다시 컴파일해보았습니다.

```
./configure --enable-smp --enable-disasm --with-term
```

컴파일 이후 xv6를 부팅해보니, 부팅 과정에서 막혔습니다.

```
Plex86/Bochs VGABios 0.5d 29 Dec 2005
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

NO Bochs VBE Support available!

Bochs BIOS - build: 01/25/06
$Revision: 1.160 $ $Date: 2006/01/25 17:51:49 $
Options: apmbios pcibios eltorito

ata0 master: Generic 1234 ATA-2 Hard-Disk (4 MBytes)
ata0 slave: Generic 1234 ATA-2 Hard-Disk (0 MBytes)

Booting from Hard Disk...
```

무슨 상황인가 싶어 configure 옵션에 디버거를 추가해 컴파일하고 xv6를 부팅해보았습니다.

```
./configure --enable-smp --enable-disasm --with-term --enable-debugger
```

```
Next at t=0
(0) [0xffffffff] f000:fff0 (unk. ctxt): jmp far f000:e05b ; ea5be00f0
<bochs:1> |
```


저번에 디버거 붙였을 때랑 같은 에러가 발생했습니다. 아무래도 configure 옵션을 더 살펴봐야겠습니다.

Bochs gdb stub 활성화하기

`./configure --help` 로 `--enable-debugger` 의 설명을 읽어보았습니다.

```
--enable-debugger          compile in support for Bochs internal
debugger
```

Bochs의 내부 디버거를 활용한다고 되어있습니다. 저번에 교수님께서 디버거 붙일 때 다른 프로세스에서 gdb 붙이는 방식을 제안하셨으므로, 다른 옵션을 써야할 것 같았습니다. 따라서 검색 중, bochs 문서에서 [GDB Stub 관련 항목](#)을 발견했습니다.

해당 문서에 따르면, configure에서 `--enable-gdb-stub` 옵션을 주고 bochsrc에 [gdbstub 관련 옵션](#)을 주면 설정한 포트로 gdb를 대기한다고 합니다. 따라서 바로 시도해보았습니다.

```
./configure --enable-smp --enable-disasm --with-term --enable-gdb-stub
```

이후 make를 했더니, 다음과 같은 에러가 발생했습니다.

```
$ make
cd iodev && \
make libiodev.a
make[1]: Entering directory '/home/seolcu/문서/코드/bochs-2.2.6/iodev'
g++ -c -I.. -I../.. -I../instrument/stubs -I../instrument/stubs -g -O2 -
D_FILE_OFFSET_BITS=64 -D_LARGE_FILES devices.cc -o devices.o
In file included from iodev.h:32:0,
      from devices.cc:30:
../bochs.h:381:2: error: #error GDB stub was written for single processor
support. If multiprocessor support is added, then we can remove this check.
  #error GDB stub was written for single processor support. If
multiprocessor support is added, then we can remove this check.
  ^
In file included from iodev.h:32:0,
      from devices.cc:30:
../bochs.h: In member function 'char* iofunctions::getaction(int)':
../bochs.h:307:64: warning: deprecated conversion from string constant to
'char*' [-Wwrite-strings]
    static char *name[] = { "ignore", "report", "ask", "fatal" };
                                                                ^
../bochs.h:307:64: warning: deprecated conversion from string constant to
'char*' [-Wwrite-strings]
../bochs.h:307:64: warning: deprecated conversion from string constant to
```

```

'char*' [-Wwrite-strings]
../bochs.h:307:64: warning: deprecated conversion from string constant to
'char*' [-Wwrite-strings]
devices.cc: In constructor 'bx_devices_c::bx_devices_c()':
devices.cc:50:12: warning: deprecated conversion from string constant to
'char*' [-Wwrite-strings]
    put("DEV");
        ^
Makefile:120: recipe for target 'devices.o' failed
make[1]: *** [devices.o] Error 1
make[1]: Leaving directory '/home/seolcu/문서/코드/bochs-2.2.6/iodev'
Makefile:259: recipe for target 'iodev/libiodev.a' failed
make: *** [iodev/libiodev.a] Error 2

```

멀티코어에서는 GDB stub이 작동하지 않는 것 같습니다. 따라서 SMP를 꺼야 할 것 같습니다. SMP를 켜고 있을 때 또 xv6에서 에러가 발생할까봐 걱정되지만, 우선 SMP를 다시 끄고 컴파일하기로 했습니다.

```
./configure --enable-disasm --with-term --enable-gdb-stub
```

역시 SMP를 빼고 나니 정상적으로 컴파일되었습니다. 이제 bochsrc에 다음과 같은 gdbstub 옵션을 넣어보았습니다.

```
gdbstub: enabled=1, port=1234, text_base=0, data_base=0, bss_base=0
```

이후 바로 make bochs 로 xv6를 부팅해보았습니다. 다음과 같이 뜨며 1234 포트에서 gdb 연결 대기가 시작되었습니다.

```
Waiting for gdb connection on localhost:1234
```

다른 터미널에서 바로 gdb를 켜서 target remote localhost:1234 명령어로 연결해보았습니다.

```

(gdb) target remote localhost:1234
Remote debugging using localhost:1234
warning: unrecognized item "ENN" in "qSupported" response
qTStatus: Target returns error code 'NN'.
0x0000ffff in ?? ()
qTStatus: Target returns error code 'NN'.

```

뭔가 경고가 뜨긴 하지만 되는 것 같습니다. c 명령어로 계속 실행하도록 했습니다.

```
Plex86/Bochs VBA Bios 0.5d 29 Dec 2005
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

NO Bochs VBE Support available!

Bochs BIOS - build: 01/25/06
$Revision: 1.160 $ $Date: 2006/01/25 17:51:49 $
Options: apmbios pcibios eltorito

ata0 master: Generic 1234 ATA-2 Hard-Disk (4 MBytes)
ata0 slave: Generic 1234 ATA-2 Hard-Disk (0 MBytes)

Booting from Hard Disk...
lapicid 0: panic: Expect to run on an SMP
801031a1 80102e80 0 0 0 0 0 0 0

[seolcu@msi xv6-public]$
[seolcu@msi xv6-public]$
[seolcu@msi xv6-public]$
[seolcu@msi xv6-public]$
[seolcu@msi xv6-public]$ gdb
GNU gdb (Ubuntu 7.11.1-8ubuntu1-16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
warning: unrecognized item "ENM" in "qSupported" response
qTStatus: Target returns error code 'NN'.
0x0000ffff in ?? ()
qTStatus: Target returns error code 'NN'.
(gdb) c
Continuing.
█
```

```
Booting from Hard Disk...
lapicid 0: panic: Expect to run on an SMP
801031a1 80102e80 0 0 0 0 0 0 0
```

여전히 xv6에서 SMP 관련 에러가 발생합니다. 아무래도 xv6에서 SMP 요구 자체를 끌 필요가 있을 것 같습니다.

멀티코어 요구 제거를 위한 xv6 코드 수정

main.c 파일에서 멀티코어와 관련되어보이는 함수 `mpinit()` 와 `startothers()` 등의 함수를 비활성화하면서 멀티코어 처리를 끄려고 해 보았으나, bochs에서 잘 작동하지 않았습니다. 따라서 코드는 다시 되돌렸습니다.

qemu에서 gdb 붙여보기

qemu 기준으로 xv6를 다시 보도록 했습니다.

xv6 부팅에서 qemu는 다음과 같은 옵션을 사용합니다.

```
ifndef CPUS
CPUS := 2
endif
QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive
file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA)
```

CPUS로 CPU 개수를 설정한 후, qemu 옵션으로 바로 넣는 모습입니다. 여기서 사용된 CPUS 변수는 qemu에서만 사용됩니다. 여기서 우선 CPUS를 1로 바꾸고 돌려보았습니다.

```
QEMU
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF92460+1FED2460 C980

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

아무 문제 없이 부팅되었습니다.

make qemu-gdb 옵션도 있어서, 시도해보았습니다.

```
$ make qemu-gdb
sed "s/localhost:1234/localhost:26000/" < .gdbinit.tmpl > .gdbinit
*** Now run 'gdb'.
qemu-system-i386 -serial mon:stdio -drive
file=fs.img,index=1,media=disk,format=raw -drive
file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512 -S -gdb tcp::26000
```

포트 26000을 열어주어서, gdb로 attach 해보았습니다.

```
teador@teador-pis:~/no-pis$ cd ~/zombie.o
ld -m elf_i386 -N -e main -Ttext 0 -o _zombie.o ulib.o usys.o printf.o umalloc.o
objdump -t _zombie.o | sed '1,/SYMBOL TABLE/d; s/ / /; /%d' > zombie.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _ustestfs _wc _zombie
rmmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
ballocc: first 367 blocks have been allocated
ballocc: write bitmap block at sector 58
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,for
mat=raw -smp 1 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ [seolcu@seolcu xv6-public]$ make qemu-gdb
sed "s/localhost:1234/localhost:26000/" < .gdbinit.tmpl > .gdbinit
*** Now run 'gdb'.
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,for
mat=raw -smp 1 -m 512 -S -gdb tcp::26000
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
Warning: File "/home/seolcu/문서/코드/xv6-public/.gdbinit" auto-loading has been declined by your `auto-load safe-path' set to
`$debugdir:$datadir/auto-load'.
---Type <return> to continue, or q <return> to quit---
To enable execution of this file add
add-auto-load-safe-path /home/seolcu/문서/코드/xv6-public/.gdbinit
line to your configuration file "/home/seolcu/.gdbinit".
To completely disable this security protection add
set auto-load safe-path /
line to your configuration file "/home/seolcu/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
info "(gdb) auto-loading safe-path"
(gdb) target remote localhost:26000
Remote debugging using localhost:26000
0x00000000 in ?? ()
(gdb) c
Continuing.
(gdb)
```

바로 gdb가 붙었습니다. c 를 입력하니 정상적으로 작동합니다.

gdb에 편의성 기능들을 추가해주는 pwndbg 를 사용하니 gdbinit을 자동으로 읽어 attach 해주고 symbol도 가져와줬습니다.

Bochs에서 안 돼서 아쉽지만, 우선 이 방식이라도 [xv6-public 레포](#)의 README.md에 정리해줬습니다.

프로젝트 되돌아보기

프로젝트에서 너무 문제 해결에 몰두하다 보면, 원래의 목적에서 벗어나 문제 해결에만 몰두하는, 주객이 전도되는 상황이 일어나곤 합니다. 이를 방지하기 위해, 4주차에는 프로젝트를 되돌아보는 시간을 가졌습니다.

프로젝트의 목표는 무엇이었을까

이 프로젝트의 최종 목표는 리눅스 KVM API를 이용한 초소형 가상 머신 모니터 (VMM) 개발 입니다.

OS

프로젝트를 위해선 VMM 위에서 돌아갈 OS가 필요했습니다. 처음에는 리눅스를 생각했지만, 리눅스는 디바이스 드라이버 요구사항이 너무 많아 조금 더 원시적인 OS가 필요했습니다.

따라서 교육용 OS인 xv6 를 선택했습니다. 현재 xv6 는 RISC-V로 개발중이기에, x86 기반으로 개발된 이전 버전을 사용하기로 했습니다.

바로 로컬 환경에서 xv6 를 컴파일해보았는데, 컴파일러 버전이 너무 높아 문제가 발생했습니다. 따라서 distrobox로 Ubuntu 16.04 환경을 만들어 컴파일에 성공했습니다.

VMM

아무런 레퍼런스 없이 VMM을 스스로 만들기는 어렵기에, 다른 오픈소스 VMM을 참고하기로 했습니다.

대표적인 VMM으로는 QEMU와 Bochs가 있었습니다. 여기에서 QEMU는 너무 방대하기에, Bochs를 틀로 사용하기로 했습니다. 다만 Bochs는 KVM 등의 하드웨어 지원을 사용하지 않고, 명령어를 하나하나 번역하는 인터프리터 방식이었기에 KVM과 관련된 부분은 QEMU의 코드를 참고하기로 했습니다.

그러나 Ubuntu 16.04의 Bochs 버전이 너무 높아, xv6가 잘 작동하지 않았습니다. 따라서 xv6 메모에 명시된 대로 Bochs 2.2.6를 직접 컴파일해 쓰기로 했습니다. 이 과정에서 SMP를 활성화하면 GDB를 못 붙이고, SMP를 비활성화하면 xv6가 안 돌아가는 진퇴양난의 상황을 마주하게 됩니다.

반면, QEMU는 코어 설정과 상관없이 잘 실행되었고, GDB도 잘 붙었습니다.

앞으로 뭘 하면 좋을까?

교수님과 상담을 통해, 앞으로의 계획을 더 탄탄히 하고 싶습니다.

5-6주차 연구내용

목표: vCPU 생성 및 제어, 메모리 입출력, VM 종료 코드 작성

저번주 todo:

- 전체 점검

연구 내용

4주차까지 Bochs를 이용해 xv6를 부팅하려는 시도를 했으나, 여러 문제로 인해 큰 진전이 없었습니다. 교수님과의 상담을 통해, Bochs 접근 방식을 포기하고 새로운 전략을 시도하기로 했습니다.

교수님께서 x86은 너무 복잡하니, RISC-V로 하이퍼바이저 개념을 먼저 배우고 나중에 x86 KVM으로 전환하는 방법을 추천해 주셨습니다. 그래서 [RISC-V 기반 VMM 개발 튜토리얼](#)을 따라가기로 했습니다.

튜토리얼 구성 확인

해당 튜토리얼을 훑어보니, RISC-V 아키텍처 기반으로 Rust를 사용해 bare-metal 하이퍼바이저를 만드는 방법을 설명하고 있습니다. 목표는 약 1000줄 정도의 코드로 Linux 커널을 부팅하는 것이고, QEMU 에뮬레이터를 사용한다고 합니다.

튜토리얼은 총 13개 챕터로 구성되어 있습니다:

1. Getting Started - 개발 환경 구성
2. Boot - 하이퍼바이저 부팅
3. Hello World - 기본 출력
4. Memory Allocation - 메모리 할당
5. Guest Mode - 게스트 모드 진입
6. Guest Page Table - 페이지 테이블
7. Hello from Guest - 게스트 실행
8. Build Linux Kernel - 리눅스 빌드
9. Boot Linux - 리눅스 부팅
10. Supervisor Binary Interface - SBI
11. Memory Mapped I/O - MMIO
12. Interrupt Injection - 인터럽트
13. Outro - 마무리

이 중 챕터 8부터는 미완성인 듯 합니다.

Chapter 1-2: 개발 환경 및 부팅 구조 만들기

Rust 툴체인 설치

패키지 매니저 대신 튜토리얼에서 권장하는 Rustup으로 설치했습니다:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

QEMU는 이미 설치되어 있었기 때문에 넘어갔습니다.

Rust 프로젝트 생성

새 프로젝트를 만들었습니다:

```
cargo init --bin hypervisor
```

rust-toolchain.toml 파일을 만들어서 툴체인을 지정했습니다:

```
[toolchain]
channel = "stable"
targets = ["riscv64gc-unknown-none-elf"]
```

최소 부팅 코드 작성

src/main.rs 를 작성했습니다. `#![no_std]` 와 `#![no_main]` 속성을 사용해 표준 라이브러리 없이 bare-metal 환경에서 실행되도록 했습니다:

```
#![no_std]
#![no_main]

use core::arch::asm;

#[unsafe(no_mangle)]
#[unsafe(link_section = ".text.boot")]
pub extern "C" fn boot() -> ! {
    unsafe {
        asm!(
            "la sp, __stack_top",
            "j {main}",
            main = sym main,
            options(noreturn)
        );
    }
}
```

```

}

fn main() -> ! {
    // BSS 초기화
    unsafe {
        let bss_start = &raw mut __bss;
        let bss_size = (&raw mut __bss_end as usize) - (&raw mut __bss as
        usize);
        core::ptr::write_bytes(bss_start, 0, bss_size);
    }

    loop {}
}

```

링커 스크립트 작성

hypervisor.ld 를 만들어 메모리 레이아웃을 정의했습니다:

```

ENTRY(boot)

SECTIONS {
    . = 0x80200000; /* OpenSBI가 커널을 로드하는 주소 */

    .text :{
        KEEP(*(.text.boot));
        *(.text .text.*);
    }

    .rodata : ALIGN(8) {
        *(.rodata .rodata.*);
    }

    .data : ALIGN(8) {
        *(.data .data.*);
    }

    .bss : ALIGN(8) {
        __bss = .;
        *(.bss .bss.* .sbss .sbss.*);
        __bss_end = .;
    }

    . = ALIGN(16);
    . += 1024 * 1024; /* 1MB 스택 */
}

```



```
__stack_top = .;
}
```

Panic Handler 추가

빌드하니 panic handler가 없다는 에러가 발생했습니다:

```
error: `[panic_handler]` function required, but not found
```

no_std 환경에서는 panic handler를 직접 구현해야 합니다:

```
use core::panic::PanicInfo;

#[panic_handler]
pub fn panic_handler(_info: &PanicInfo) -> ! {
    loop {
        unsafe {
            core::arch::asm!("wfi");
        }
    }
}
```

빌드 및 실행

run.sh 스크립트를 작성했습니다:

```
#!/bin/sh
set -ev

RUSTFLAGS="-C link-arg=-Thypervisor.ld -C linker=rust-ld" \
    cargo build --bin hypervisor --target riscv64gc-unknown-none-elf

cp target/riscv64gc-unknown-none-elf/debug/hypervisor hypervisor.elf

qemu-system-riscv64 \
    -machine virt \
    -cpu rv64 \
    -bios default \
    -smp 1 \
    -m 128M \
    -nographic \
    -d cpu_reset,unimp,guest_errors,int -D qemu.log \
    -serial mon:stdio \
```

```
--no-reboot \  
-kernel hypervisor.elf
```

실행해봤습니다:

```
chmod +x run.sh  
./run.sh
```

OpenSBI 부팅 메시지가 출력된 후 멈췄습니다. 아직 아무것도 출력하지 않지만, `main()` 함수의 무한 루프가 실행되고 있는 듯 합니다.

Chapter 3: Hello World

SBI를 이용한 콘솔 출력

튜토리얼을 보니, OpenSBI 펌웨어가 제공하는 SBI(Supervisor Binary Interface)를 사용할 수 있다고 합니다. `src/print.rs` 를 만들어서 SBI `putchar` 함수를 구현했습니다:

```
use core::arch::asm;  
  
pub fn sbi_putchar(ch: u8) {  
    unsafe {  
        asm!(  
            "ecall",  
            in("a6") 0,  
            in("a7") 1,  
            inout("a0") ch as usize => _,  
            out("a1") _  
        );  
    }  
}
```

`ecall` 명령을 inline assembly로 호출합니다. `a7 = 1`은 Console Putchar extension입니다.

println! 매크로 구현

Rust의 `fmt::Write` 트레이트를 구현해서 `println!` 매크로를 만들었습니다:

```
pub struct Printer;  
  
impl core::fmt::Write for Printer {  
    fn write_str(&mut self, s: &str) -> core::fmt::Result {  
        for byte in s.bytes() {  
            sbi_putchar(byte);  
        }  
    }  
}
```

```

    }
    Ok(())
}
}

#[macro_export]
macro_rules! println {
    ($($arg:tt)*) => {{
        use core::fmt::Write;
        let _ = writeln!($crate::print::Printer, $($arg)*);
    }};
}

```

Trap Handler 구현

CPU가 exception이나 interrupt를 만나면 trap handler가 호출됩니다. src/trap.rs 를 만들었습니다:

```

#[macro_export]
macro_rules! read_csr {
    ($csr:expr) => {{
        let mut value: u64;
        unsafe {
            ::core::arch::asm!(concat!("csrr {}, ", $csr), out(reg) value);
        }
        value
    }};
}

#[unsafe(link_section = ".text.stvec")]
pub fn trap_handler() -> ! {
    let scause = read_csr!("scause");
    let sepc = read_csr!("sepc");
    let stval = read_csr!("stval");

    panic!("trap: scause={:#x} at {:#x} (stval={:#x})", scause, sepc,
stval);
}

```

Panic handler도 수정해서 panic 정보를 출력하도록 했습니다:

```

#[panic_handler]
pub fn panic_handler(info: &PanicInfo) -> ! {
    println!("panic: {}", info);
}

```

```

    loop {
        unsafe {
            core::arch::asm!("wfi");
        }
    }
}

```

main() 수정

src/main.rs 를 수정해서 trap handler를 등록하고 부팅 메시지를 출력했습니다:

```

#[macro_use]
mod print;
mod trap;

fn main() -> ! {
    unsafe {
        let bss_start = &raw mut __bss;
        let bss_size = (&raw mut __bss_end as usize) - (&raw mut __bss as
        usize);
        core::ptr::write_bytes(bss_start, 0, bss_size);

        asm!("csrw stvec, {}", in(reg) trap::trap_handler as usize);
    }

    println!("\nBooting hypervisor...");
    loop {}
}

```

실행 결과

실행해보니:

```
Booting hypervisor...
```

드디어 출력에 성공했습니다.

Chapter 4: Memory Allocation

Bump Allocator 구현

no_std 환경에서 Vec, Box 같은 자료구조를 쓰려면 메모리 할당자를 직접 구현해야 합니다. 가장 단순한 bump allocator를 구현했습니다.

spin 크레이트를 추가합니다.

```
cargo add spin
```

src/allocator.rs 를 만듭니다.

```
use core::alloc::{GlobalAlloc, Layout};
use spin::Mutex;

pub struct BumpAllocator {
    next: usize,
    end: usize,
}

unsafe impl GlobalAlloc for BumpAllocator {
    unsafe fn alloc(&self, layout: Layout) -> *mut u8 {
        let mut next = self.next;
        let align = layout.align();
        let size = layout.size();

        // Align
        next = (next + align - 1) & !(align - 1);

        let new_next = next + size;
        if new_next > self.end {
            return core::ptr::null_mut();
        }

        self.next = new_next;
        next as *mut u8
    }

    unsafe fn dealloc(&self, _ptr: *mut u8, _layout: Layout) {
        // Bump allocator는 해제를 지원하지 않음
    }
}

#[global_allocator]
static ALLOCATOR: Mutex<BumpAllocator> = Mutex::new(BumpAllocator {
    next: 0,
    end: 0,
});
```

링커 스크립트에 힙 영역을 추가합니다.

```

    . = ALIGN(16);
    __heap = .;
    . += 8 * 1024 * 1024; /* 8MB 힙 */
    __heap_end = .;

```

main() 에서 할당자를 초기화했습니다:

```

unsafe extern "C" {
    static mut __heap: u8;
    static mut __heap_end: u8;
}

fn main() -> ! {
    // ... BSS 초기화 ...

    unsafe {
        let heap_start = &raw mut __heap as usize;
        let heap_size = (&raw mut __heap_end as usize) - heap_start;
        allocator::init(heap_start, heap_size);
    }

    // ...
}

```

Page Allocator 구현

Bump allocator 위에 4KB 단위로 메모리를 할당하는 page allocator를 만들었습니다:

```

extern crate alloc;
use alloc::vec::Vec;

const PAGE_SIZE: usize = 4096;

pub fn alloc_pages(size: usize) -> *mut u8 {
    let pages = (size + PAGE_SIZE - 1) / PAGE_SIZE;
    let mut v = Vec::with_capacity(pages * PAGE_SIZE);
    v.resize(pages * PAGE_SIZE, 0);
    let ptr = v.as_mut_ptr();
    core::mem::forget(v);
    ptr
}

```

테스트

vec 을 테스트해봤습니다

```
extern crate alloc;
use alloc::vec::Vec;

fn main() -> ! {
    // ... 초기화 ...

    println!("\nBooting hypervisor...");

    let mut v = Vec::new();
    for i in 0..10 {
        v.push(i);
    }
    println!("vec test: {:?}", v);

    loop {}
}
```

다음과 같은 실행 결과가 출력되었습니다:

```
Booting hypervisor...
vec test: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

잘 작동합니다.

Chapter 5-6: Guest Mode 및 Page Table

게스트 모드 진입 시도

RISC-V 가상화 모드를 활성화하려면 H-extension을 켜야 합니다. `run.sh` 에 `-cpu rv64,h=true` 를 추가했습니다:

```
qemu-system-riscv64 \
    -machine virt \
    -cpu rv64,h=true \
    ...
```

게스트 모드 진입 코드를 작성했습니다:

```
fn main() -> ! {
    // ... 초기화 ...
```

```
println!("\nBooting hypervisor...");

let guest_code: [u8; 4] = [0x73, 0x00, 0x00, 0x00]; // wfi 명령

unsafe {
    // hstatus 설정
    let mut hstatus: u64 = 0;
    hstatus |= 2 << 32; // VSXL=2 (64-bit mode)
    hstatus |= 1 << 7; // SPV=1 (virtualization mode)
    asm!("csrw hstatus, {}", in(reg) hstatus);

    // sstatus 설정
    let mut sstatus: u64 = 0;
    sstatus |= 1 << 8; // SPP=1 (return to supervisor mode)
    asm!("csrw sstatus, {}", in(reg) sstatus);

    // sepc 설정
    asm!("csrw sepc, {}", in(reg) guest_code.as_ptr() as u64);

    // 게스트로 진입
    asm!("sret");
}

loop {}
}
```

실행하니 바로 트랩이 발생했습니다:

```
panic: trap: scause=0xc at 0x80200b7a (stval=0x80200b7a)
```

scause=0xc 는 instruction page fault입니다. 게스트 코드가 직접 실행되려 했지만 주소 변환이 안 돼서 발생한 에러입니다.

Guest Page Table 구현

2-stage address translation을 구현해야 합니다. `src/guest_page_table.rs` 를 만듭니다.

```
use crate::allocator::alloc_pages;

pub const PTE_V: u64 = 1 << 0;
pub const PTE_R: u64 = 1 << 1;
pub const PTE_W: u64 = 1 << 2;
pub const PTE_X: u64 = 1 << 3;

pub struct GuestPageTable {
```



```

    root: *mut u64,
}

impl GuestPageTable {
    pub fn new() -> Self {
        let root = alloc_pages(4096) as *mut u64;
        unsafe {
            core::ptr::write_bytes(root, 0, 4096);
        }
        Self { root }
    }

    pub fn map(&mut self, guest_pa: u64, host_pa: u64, flags: u64) {
        // 4-level page table (Sv48x4) 구현
        // ...
    }

    pub fn hgatp(&self) -> u64 {
        let mode = 9u64 << 60; // Sv48x4 mode
        mode | ((self.root as u64) >> 12)
    }
}

```

Page table 구현은 4단계로 나뉩니다:

1. VPN[3]으로 L3 테이블 인덱스
2. VPN[2]로 L2 테이블 인덱스
3. VPN[1]로 L1 테이블 인덱스
4. VPN[0]으로 L0 테이블 인덱스

각 단계에서 PTE가 없으면 새 페이지를 할당하고, 마지막 단계에서 host PA를 매핑합니다.

게스트 코드 컴파일

간단한 어셈블리 코드를 작성했습니다:

```

.section .text
.global guest_boot
guest_boot:
    j guest_boot

```

GCC로 컴파일하고 바이너리로 변환했습니다:

```
riscv64-linux-gnu-as -o guest.o guest.S
riscv64-linux-gnu-ld -Ttext=0x1000000 -o guest.elf guest.o
riscv64-linux-gnu-objcopy -O binary guest.elf guest.bin
```

메모리 로딩 및 매핑

게스트 코드를 메모리에 로드하고 page table에 매핑합니다.

```
fn main() -> ! {
    // ... 초기화 ...

    println!("\nBooting hypervisor...");

    // 게스트 코드 로드
    let guest_bin = include_bytes!("../guest.bin");
    let kernel_memory = alloc_pages(guest_bin.len());
    unsafe {
        core::ptr::copy_nonoverlapping(
            guest_bin.as_ptr(),
            kernel_memory,
            guest_bin.len()
        );
    }

    // Page table 설정
    let guest_entry = 0x1000000u64;
    let mut table = GuestPageTable::new();
    table.map(guest_entry, kernel_memory as u64, PTE_R | PTE_W | PTE_X);

    println!("map: {:08x} -> {:08x}", guest_entry, kernel_memory as u64);

    // 게스트 모드 진입
    unsafe {
        let mut hstatus: u64 = 0;
        hstatus |= 2 << 32;
        hstatus |= 1 << 7;
        asm!("csrw hstatus, {}", in(reg) hstatus);

        let sstatus: u64 = 1 << 8;
        asm!("csrw sstatus, {}", in(reg) sstatus);

        asm!("csrw hgatp, {}", in(reg) table.hgatp());
        asm!("csrw sepc, {}", in(reg) guest_entry);

        asm!("sret");
    }
}
```

```

    }

    loop {}
}

```

실행 결과

실행해보니:

```

Booting hypervisor...
map: 00100000 -> 80305000

```

멈췄습니다. 게스트 코드가 무한 루프를 실행하고 있기 때문으로 보입니다.

문제: alloc_pages 길이 처리

튜토리얼과 다르게 한 가지 문제를 발견했습니다. 튜토리얼 코드는 `alloc_pages(1)` 을 호출해 1페이지를 할당하는데, 제 구현은 `alloc_pages(size)` 가 바이트 단위입니다.

따라서 `alloc_pages(4096)` 로 수정해야 했습니다. 아니면 `alloc_pages` 의 시그니처를 바꿔서 페이지 수를 받도록 할 수도 있지만, 우선은 그대로 두기로 했습니다.

Chapter 7: Hello from Guest (Hypercall)

Chapter 7의 목표는 게스트에서 하이퍼바이저로 통신하는 것입니다.

게스트 코드 수정

게스트가 SBI의 "Console Putchar"를 호출하도록 `guest.s` 를 수정했습니다:

```

.section .text
.global guest_boot
guest_boot:
    li a7, 1          # EID=1 (legacy console)
    li a6, 0          # FID=0 (putchar)
    li a0, 'A'        # Parameter: 'A'
    ecall             # Call SBI

    li a7, 1
    li a6, 0
    li a0, 'B'
    ecall

    li a7, 1
    li a6, 0

```

```
    li a0, 'C'
    ecall

halt:
    j halt
```

빌드하고 실행하면 이렇게 트랩이 발생합니다.

```
panic: trap: scause=0xa at 0x1000008 (stval=0x0)
```

scause=0xa 는 "environment call from VS-mode"입니다. 게스트가 `ecall` 을 호출한 거라 정확히 우리가 원하는 트랩입니다.

VCpu 구조체 도입

Hypercall을 구현하기 전에, 게스트 상태를 관리할 `VCpu` 구조체를 만들겠습니다. 새 파일 `src/vcpu.rs` 를 만듭니다.

```
use core::{arch::asm, mem::offset_of};
use crate::{allocator::alloc_pages, guest_page_table::GuestPageTable};

#[derive(Debug, Default)]
pub struct VCpu {
    pub hstatus: u64,
    pub hcatp: u64,
    pub sstatus: u64,
    pub sepc: u64,
    pub host_sp: u64,
    pub ra: u64,
    pub sp: u64,
    pub gp: u64,
    pub tp: u64,
    pub t0: u64,
    pub t1: u64,
    pub t2: u64,
    pub s0: u64,
    pub s1: u64,
    pub a0: u64,
    pub a1: u64,
    pub a2: u64,
    pub a3: u64,
    pub a4: u64,
    pub a5: u64,
    pub a6: u64,
```

```

    pub a7: u64,
    pub s2: u64,
    pub s3: u64,
    pub s4: u64,
    pub s5: u64,
    pub s6: u64,
    pub s7: u64,
    pub s8: u64,
    pub s9: u64,
    pub s10: u64,
    pub s11: u64,
    pub t3: u64,
    pub t4: u64,
    pub t5: u64,
    pub t6: u64,
}

impl VCpu {
    pub fn new(table: &GuestPageTable, guest_entry: u64) -> Self {
        let mut hstatus: u64 = 0;
        hstatus |= 2 << 32; // VSXL=2
        hstatus |= 1 << 7;  // SPV=1

        let sstatus: u64 = 1 << 8; // SPP=1

        let stack_size = 512 * 1024;
        let host_sp = alloc_pages(stack_size) as u64 + stack_size as u64;

        Self {
            hstatus,
            hgap: table.hgap(),
            sstatus,
            sepc: guest_entry,
            host_sp,
            ..Default::default()
        }
    }

    pub fn run(&mut self) -> ! {
        unsafe {
            asm!(
                "csrw hstatus, {hstatus}",
                "csrw sstatus, {sstatus}",
                "csrw sscratch, {sscratch}",
                "csrw hgap, {hgap}",
                "csrw sepc, {sepc}",

```

```

"mv a0, {sscratch}",
"ld ra, {ra_offset}(a0)",
"ld sp, {sp_offset}(a0)",
"ld gp, {gp_offset}(a0)",
"ld tp, {tp_offset}(a0)",
"ld t0, {t0_offset}(a0)",
"ld t1, {t1_offset}(a0)",
"ld t2, {t2_offset}(a0)",
"ld s0, {s0_offset}(a0)",
"ld s1, {s1_offset}(a0)",
"ld a1, {a1_offset}(a0)",
"ld a2, {a2_offset}(a0)",
"ld a3, {a3_offset}(a0)",
"ld a4, {a4_offset}(a0)",
"ld a5, {a5_offset}(a0)",
"ld a6, {a6_offset}(a0)",
"ld a7, {a7_offset}(a0)",
"ld s2, {s2_offset}(a0)",
"ld s3, {s3_offset}(a0)",
"ld s4, {s4_offset}(a0)",
"ld s5, {s5_offset}(a0)",
"ld s6, {s6_offset}(a0)",
"ld s7, {s7_offset}(a0)",
"ld s8, {s8_offset}(a0)",
"ld s9, {s9_offset}(a0)",
"ld s10, {s10_offset}(a0)",
"ld s11, {s11_offset}(a0)",
"ld t3, {t3_offset}(a0)",
"ld t4, {t4_offset}(a0)",
"ld t5, {t5_offset}(a0)",
"ld t6, {t6_offset}(a0)",
"ld a0, {a0_offset}(a0)",

"sret",
hstatus = in(reg) self.hstatus,
sstatus = in(reg) self.sstatus,
hgap = in(reg) self.hgap,
sepc = in(reg) self.sepc,
sscratch = in(reg) (self as *mut VCpu as usize),
ra_offset = const offset_of!(VCpu, ra),
sp_offset = const offset_of!(VCpu, sp),
gp_offset = const offset_of!(VCpu, gp),
tp_offset = const offset_of!(VCpu, tp),
t0_offset = const offset_of!(VCpu, t0),
t1_offset = const offset_of!(VCpu, t1),

```

```

        t2_offset = const offset_of!(VCpu, t2),
        s0_offset = const offset_of!(VCpu, s0),
        s1_offset = const offset_of!(VCpu, s1),
        a0_offset = const offset_of!(VCpu, a0),
        a1_offset = const offset_of!(VCpu, a1),
        a2_offset = const offset_of!(VCpu, a2),
        a3_offset = const offset_of!(VCpu, a3),
        a4_offset = const offset_of!(VCpu, a4),
        a5_offset = const offset_of!(VCpu, a5),
        a6_offset = const offset_of!(VCpu, a6),
        a7_offset = const offset_of!(VCpu, a7),
        s2_offset = const offset_of!(VCpu, s2),
        s3_offset = const offset_of!(VCpu, s3),
        s4_offset = const offset_of!(VCpu, s4),
        s5_offset = const offset_of!(VCpu, s5),
        s6_offset = const offset_of!(VCpu, s6),
        s7_offset = const offset_of!(VCpu, s7),
        s8_offset = const offset_of!(VCpu, s8),
        s9_offset = const offset_of!(VCpu, s9),
        s10_offset = const offset_of!(VCpu, s10),
        s11_offset = const offset_of!(VCpu, s11),
        t3_offset = const offset_of!(VCpu, t3),
        t4_offset = const offset_of!(VCpu, t4),
        t5_offset = const offset_of!(VCpu, t5),
        t6_offset = const offset_of!(VCpu, t6),
    );
}

    unreachable!();
}
}

```

sscratch 에 VCpu 구조체 포인터를 저장합니다. Trap handler에서 이 포인터로 게스트 상태를 저장/복원합니다.

main() 수정

src/main.rs 에서 VCpu 를 사용하도록 수정합니다.

```

mod vcpu;

use crate::{
    allocator::alloc_pages,
    guest_page_table::{GuestPageTable, PTE_R, PTE_W, PTE_X},
    vcpu::VCpu,
}

```

```
};

fn main() -> ! {
    // ... 초기화 및 게스트 코드 로드 ...

    let mut vcpu = VCpu::new(&table, guest_entry);
    vcpu.run();
}
```

Trap Handler 수정

src/trap.rs 를 #[naked] 함수로 수정해서 게스트 상태를 저장하고 복원하도록 했습니다:

```
use core::{arch::naked_asm, mem::offset_of};
use crate::vcpu::VCpu;

#[unsafe(link_section = ".text.stvec")]
#[naked]
pub unsafe extern "C" fn trap_handler() {
    naked_asm!(
        // sscratch와 a0 교환 (VCpu 포인터 가져오기)
        "csrrw a0, sscratch, a0",

        // 모든 레지스터 저장
        "sd ra, {ra_offset}(a0)",
        "sd sp, {sp_offset}(a0)",
        // ... (생략) ...
        "sd t6, {t6_offset}(a0)",

        // CSR 저장
        "csrr t0, sepc",
        "sd t0, {sepc_offset}(a0)",

        // sscratch 복구 (원래 a0 값 저장)
        "csrr t0, sscratch",
        "sd t0, {a0_offset}(a0)",

        // 하이퍼바이저 스택으로 전환
        "ld sp, {host_sp_offset}(a0)",

        // handle_trap 호출
        "call {handle_trap}",

        ra_offset = const offset_of!(VCpu, ra),
        sp_offset = const offset_of!(VCpu, sp),
```



```

    // ... (생략) ...
    sepc_offset = const offset_of!(VCpu, sepc),
    a0_offset = const offset_of!(VCpu, a0),
    host_sp_offset = const offset_of!(VCpu, host_sp),
    handle_trap = sym handle_trap,
);
}

fn handle_trap(vcpu: &mut VCpu) -> ! {
    let scause = read_csr!("scause");

    if scause == 10 {
        // VS-mode ecall
        println!("SBI call: eid={:x}, fid={:x}, a0={:x} ('{}')",
            vcpu.a7, vcpu.a6, vcpu.a0, vcpu.a0 as u8 as char);

        // sepc += 4 (ecall 명령 건너뛰기)
        vcpu.sepc += 4;

        vcpu.run();
    }

    panic!("trap: scause={:x} at {:x}", scause, vcpu.sepc);
}

```

중요한 부분은:

- csrrw a0, sscratch, a0 로 VCpu 포인터와 a0 교환
- 모든 레지스터를 VCpu 구조체에 저장
- host_sp 로 스택 전환
- handle_trap() 호출
- sepc += 4 로 ecall 명령 건너뛰기
- vcpu.run() 으로 게스트 복귀

실행 결과

실행해보니:

```

Booting hypervisor...
map: 00100000 -> 80306000
SBI call: eid=0x1, fid=0x0, a0=0x41 ('A')
SBI call: eid=0x1, fid=0x0, a0=0x42 ('B')
SBI call: eid=0x1, fid=0x0, a0=0x43 ('C')

```

성공했습니다. 게스트에서 하이퍼바이저로 문자를 전달해서 출력되는 모습입니다.

튜토리얼 8-10챕터 확인

Chapter 7까지 끝내고, 남은 챕터들을 봤습니다.

나머지 챕터

챕터 8~10까지는 튜토리얼에는 미완성이라고 되어있지만, 튜토리얼 저장소를 보니까 Chapter 8-10 구현이 어느정도 되어있는 것 같았습니다.

그래서 코드를 복사해서 실행해봤는데, Linux 부팅 시도하니까 초기 커널 메시지 몇 개 나오다가 guest page fault 나면서 멈췄습니다.

Chapter 11(MMIO), 12(Interrupt), 13(Outro)는 아예 작업이 되지 않은 것 같습니다.

따라서 우선은 여기까지 하기로 했습니다.

x86 KVM으로 전환 준비

RISC-V 튜토리얼로 하이퍼바이저의 핵심 구조를 어느 정도 이해했으니, 이제 x86 KVM으로 전환할 계획을 세웠습니다.

RISC-V에서 배운 걸 정리하면:

- OpenSBI가 하이퍼바이저 `boot()` 호출 → 스택/BSS 초기화 → `main()`
- Heap 위에 page allocator (4KB 단위)
- Guest memory는 page allocator에서 할당 → guest page table에 매핑
- 2-stage translation: guest PA → host PA
- Guest 실행: CSR 설정 (`hgatp`, `hstatus`, `sepc`) → `sret`
- VM Exit: trap handler → 레지스터 저장 → Rust 코드 처리 → 레지스터 복원 → `sret`

이걸 x86 KVM에 대응하는 코드로 바꾸면 될 것 같습니다.

다음주 todo

- KVM: 간단한 게스트 코드 실행

참고 자료

- [RISC-V Hypervisor Tutorial](#)
- [Tutorial GitHub Repository](#)

구현한 코드는 `hypervisor/` 폴더에 있습니다.