1. Linux에 RISC-V 크로스 컴파일 환경 구축

1. RISC-V 가상 환경 실행을 위한 QEMU 설치

- `sudo apt-get install qemu-system` 명령으로 qemu 설치가 가능했지만, 해당 패키지 설치 방식으로 설치한 qemu의 버전은 6.x 버전으로 설치된다.
- 따라서, 테스트가 필요하긴 하지만 문제가 없다면 패키지 설치 방식도 사용 가능하며 여기서는 항상 시도하는 시점의 최신 버전 QEMU를 설치하기 위해 소스 컴파일 방식을 진행하였다.

1-1. QEMU 빌드를 위한 의존성 패키지 설치

- 아래의 명령을 사용하여 시스템 패키지들을 최신으로 업데이트하고, qemu 빌드 시 필요한 의존성 패키지들을 설치한다.

```
$ sudo apt-get update
$ sudo apt-get upgrade

$ sudo apt-get install git libglib2.0-dev libfdt-dev libpixman-1-dev zlib1g-dev ni
nja-build acpica-tools libsdl2-dev libgcrypt-dev libvde-dev libvdeplug-dev libvte-
dev libtasn1-6-dev libssh-dev liblzo2-dev libsnappy-dev sparse libaio-dev libzstd-
dev libslirp-dev
```

1-2. 빌드 작업을 진행하기 위한 작업 디렉토리 생성

- 빌드 시 사용되는 경로를 일치시켜 설명하기 편하기 위한 부분이기 때문에 이미 사용중인 작업 디렉토리가 있다면 해당 디렉토리를 사용해도 된다.

```
$ mkdir ~/workdir
$ cd ~/workdir
```

1-3. QEMU Github Repository 복제

- QEMU 프로젝트는 Github이 아닌 Gitlab에서 관리되는 것으로 보이지만, github에 존재하는 저장소 또한 공식 QEMU Mirror이기 때문에 가져와서 사용하는데는 문제가 없다.

```
$ git clone https://github.com/qemu/qemu
$ cd qemu
```

1-4. RISC-V 64bit QEMU 빌드

- `./configure` 명령 실행 시 `--target-list`에 사용하고자 하는 타겟 아키텍처 시스템을 적으면 윈도우에서 설치하는 것과 같이 여러 타겟 아키텍처 시스템을사용 가능하지만, 여기서는 risc-v 64bit만 필요하기 때문에 risc-v 64bit만 사용하였다.
  - `make`의 `-j`옵션은 빌드 시 사용할 코어 수를 의미하며, `$(nproc)`는 현재 시스템에 할당된 CPU의 논리적인 코어 수를 반환하기 때문에 최대로 사용 가능한 코어를 사용해 빠르게 빌드할 수 있다.
  - 다만, 의존성으로 인해 순차적으로 빌드가 필요하는 등의 이유로 항상 유용하게 사용할 수 있는 옵션은 아닌듯 하여 먼저 `-j` 옵션을 주고 실행해보고 에러가 나면 `-j` 옵션을 제외하고 `make` 명령을 실행해보긴 한다.

```
$ ./configure --target-list="riscv64-softmmu" --prefix=$HOME/workdir/riscv64-qemu
$ make -j$(nproc) && make install
```

1-5. RISC-V 64bit QEMU 설치 확인

- `./configure` 명령 실행 시 지정한 `--prefix` 경로에 RISC-V 64bit QEMU 실행 파일과 실행에 필요한 파일들이 설치된다.
- 따라서, 해당 경로로 이동하여 RISC-V 64bit QEMU 실행파일의 버전이 잘 출력되는지 확인한다.
  - 2023년 8월 12일 기준으로는 8.0.93 (v8.1.0-rc3)으로 출력된다.
- 설치가 정상적으로 완료되었다면, 기존에 사용한 QEMU 소스코드 디렉토리는 용량이 큰 편이기 때문에 제거하여도 된다.

```
$ cd $HOME/workdir/riscv64-qemu/bin
$ ./qemu-system-riscv64 --version
```

1-6. 어디서나 실행을 위한 환경변수 등록

- 아래 명령어를 실행하여 RISC-V 64bit QEMU 실행파일이 위치한 경로를 PATH 환경변수에 등록해 준다.

```
# 텍스트 에디터면 모두 상관없음
$ vi ~/.bashrc

# 파일의 제일 아래로 이동한 뒤 아래 문자열 입력 후 저장
export PATH=$PATH:$HOME/workdir/riscv64-qemu/bin

# 추가한 결과가 바로 반영될 수 있도록 아래 명령어 실행
$ source $HOME/.bashrc

# 아래 명령을 실행하여 위에서 추가한 문자열이 존재하는지 확인
$ echo $PATH
```

2. RISC-V 아키텍처를 위한 GCC 컴파일러 설치

2-1. 의존성 패키지 설치

2-1-1. Ubuntu

```
$ sudo apt-get install autoconf automake autotools-dev curl python3 python3-pip libmpc
-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patc
hutils bc zlib1g-dev libexpat-dev ninja-build git cmake libglib2.0-dev
```

2-1-2. Fedora/CentOS/RHEL OS

```
$ sudo yum install autoconf automake python3 libmpc-devel mpfr-devel gmp-devel gawk  b
ison flex texinfo patchutils gcc gcc-c++ zlib-devel expat-devel
```

2-2. 빌드 작업을 진행하기 위한 작업 디렉토리 생성

- 빌드 시 사용되는 경로를 일치시켜 설명하기 편하기 위한 부분이기 때문에 이미 사용중인 작업 디렉토리가 있다면 해당 디렉토리를 사용해도 된다.
    - 앞서 QEMU 빌드 시 아래의 작업 디렉토리를 생성하였다면, cd 명령어만 실행한다.

```
$ mkdir ~/workdir
$ cd ~/workdir
```

2-3. 소스 컴파일을 위한 RISC-V GCC github clone하기

2-3-1. RISC-V GCC 소스 코드 복제

- 해당 github 저장소의 readme.md에 따르면 소스코드는 약 6.65GB의 디스크를 사용한다고 되어있기 때문에 복제에는 꽤 오랜 시간이 걸린다.

```
$ git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
```

2-3-2. RISC-V GCC 소스코드 빌드

- 아래와 같이 `./configure` 실행 시 `--prefix` 를 사용하여 설치할 경로를 지정해주는 것이 나중에 사용하기에 편하다.

```
$ cd riscv-gnu-toolchain
$ ./configure --prefix=$HOME/workdir/riscv64-gcc
$ make linux
```

2-3-3. 빌드된 RISC-V GCC 툴 체인 확인

- prefix로 입력한 경로로 이동하여 아래 폴더 목록들이 존재하는지 확인한다.

```
$ cd $HOME/workdir/riscv64-gcc
$ ls
bin  include  lib  libexec  riscv64-unknown-linux-gnu  share  sysroot
```

- bin 폴더로 이동하면 아래와 같이 gcc 툴 체인이 존재하는 것을 확인할 수 있다.

```
$ cd bin
$ ls
riscv64-unknown-linux-gnu-addr2line   riscv64-unknown-linux-gnu-gdb
riscv64-unknown-linux-gnu-ar          riscv64-unknown-linux-gnu-gdb-add-index
riscv64-unknown-linux-gnu-as          riscv64-unknown-linux-gnu-gfortran
riscv64-unknown-linux-gnu-c++         riscv64-unknown-linux-gnu-gprof
riscv64-unknown-linux-gnu-c++filt     riscv64-unknown-linux-gnu-ld
riscv64-unknown-linux-gnu-cpp         riscv64-unknown-linux-gnu-ld.bfd
riscv64-unknown-linux-gnu-elfedit     riscv64-unknown-linux-gnu-lto-dump
riscv64-unknown-linux-gnu-g++         riscv64-unknown-linux-gnu-nm
riscv64-unknown-linux-gnu-gcc         riscv64-unknown-linux-gnu-objcopy
riscv64-unknown-linux-gnu-gcc-12.2.0  riscv64-unknown-linux-gnu-objdump
riscv64-unknown-linux-gnu-gcc-ar      riscv64-unknown-linux-gnu-ranlib
riscv64-unknown-linux-gnu-gcc-nm      riscv64-unknown-linux-gnu-readelf
riscv64-unknown-linux-gnu-gcc-ranlib  riscv64-unknown-linux-gnu-run
riscv64-unknown-linux-gnu-gcov        riscv64-unknown-linux-gnu-size
riscv64-unknown-linux-gnu-gcov-dump   riscv64-unknown-linux-gnu-strings
riscv64-unknown-linux-gnu-gcov-tool   riscv64-unknown-linux-gnu-strip
```

- 아래 명령을 실행하여 RISC-V GCC 컴파일러 버전을 확인한다.
  - 2023년 8월 12일 시점으로는 12.2.0 버전으로 출력된다.

```
$ ./riscv64-unknown-linux-gnu-gcc --version
```

2-3. 어디서나 실행을 위한 환경변수 등록

- 아래 명령어를 실행하여 risc-v gcc 툴 체인들의 실행파일이 위치한 경로를 PATH 환경변수에 등록해 준다.

```
# 텍스트 에디터면 모두 상관없음
$ vi ~/.bashrc

# 파일의 제일 아래로 이동한 뒤 아래 문자열 입력 후 저장
export PATH=$PATH:$HOME/workdir/riscv64-gcc/bin

# 추가한 결과가 바로 반영될 수 있도록 아래 명령어 실행
$ source $HOME/.bashrc

# 아래 명령을 실행하여 위에서 추가한 문자열이 존재하는지 확인
$ echo $PATH
```

3. QEMU로 RISC-V 가상 환경 구축

3-1. 빌드 작업을 진행하기 위한 작업 디렉토리 생성

- 빌드 시 사용되는 경로를 일치시켜 설명하기 편하기 위한 부분이기 때문에 이미 사용중인 작업 디렉토리가 있다면 해당 디렉토리를 사용해도 된다.
    - 앞서 QEMU 빌드 시 아래의 작업 디렉토리를 생성하였다면, 아래 가상 환경이 위치할 폴더 생성 부분의 명령어만 실행한다.

```
$ mkdir ~/workdir
$ cd ~/workdir

# 가상 환경이 위치할 폴더 생성
$ mkdir -p ~/workdir/riscv64-virtual-env
$ cd ~/workdir/riscv64-virtual-env
```

3-2. openSBI, u-boot-qemu 패키지 설치

- 윈도우에서 파일을 다운로드하고 복잡하게 추출했던 방식과는 다르게, 패키지 설치를 사용하여 openSBI와 u-boot-qemu를 다운로드 한다.

```
$ sudo apt-get install opensbi u-boot-qemu
```

3-3. Preinstalled Ubuntu Image 다운로드

- 아래 명령을 사용하여 RISC-V 64bit 아키텍처의 Preinstalled Ubuntu 22.04 이미지 파일을 다운로드 한다.

```
$ wget https://cdimage.ubuntu.com/releases/22.04/release/ubuntu-22.04.3-preinstall
ed-server-riscv64+unmatched.img.xz
```

- 만약 위의 명령어 실행 시 **ERROR 404: Not Found 에러가 뜬다면** 버전 번호에서 패치 번호가 변경되어 파일 이름이 바뀌었기 때문이므로, 아래 링크로 접속하여 Preinstalled server image 항목 중 **"RISC-V for SiFive HiFive Unmatched preinstalled server image" 링크를 클릭**하여 다운로드 한다.
  - 링크 : https://cdimage.ubuntu.com/releases/22.04.2/release/

- 다운로드 받은 Preinstalled Ubuntu Image 파일은 아래 명령어를 실행하여 압축을 해제한다.

```
# xz -d [xz로 압축된 파일 명]
$ xz -d ubuntu-22.04.3-preinstalled-server-riscv64+unmatched.img.xz
```

3-4. RISC-V 64bit Ubuntu 가상 환경의 디스크 용량 확장

3-4-1. 가상 환경의 디스크 용량을 확장해야 하는 이유

- Preinstalled Image는 앞서 서술한 것과 같이 우분투가 미리 설치된 디스크 이미지이며, 해당 파일의 크기는 4.5G로 되어있다.
- 해당 파일은 컴퓨터의 SSD나 하드디스크 용량과도 같은데 첫 부팅 후 apt update를 수행하게 되면 사용할 수 있는 여유 공간이 부족한 것을 확인하였고, 추후 새로운 패키지를 설치하거나 파일을 생성해야 하면 용량이 부족할 것이다.
- 따라서, 이에 대한 해결 법은 디스크 이미지 파일의 용량을 확장하는 것이다.

3-4-2. 가상 환경의 디스크 용량 확장 방법

- 다행히 `qemu-img resize [디스크 이미지 파일명] [원하는 용량]` 명령을 사용하여 간단하게 디스크 용량을 확장할 수 있다.
  - 단, 디스크 용량 확장은 가상머신이 종료된 상태에서 해야하며 명령을 실행한 결과 성공하였으면 아래와 같이 “Image resized.” 라는 문자열이 출력된다.
  - `ls -alh` 명령으로 파일의 크기를 확인해봐도 크기가 설정한 용량에 맞게 증가한 것을 확인할 수 있다.

3-5. RISC-V 64bit 가상환경 실행을 위한 스크립트 파일 생성

- QEMU로 RISC-V 64bit 가상 환경을 실행하기 위해 사용되는 명령어는 매우 길고 외우기 어렵기 때문에 스크립트 파일로 명령어를 미리 만들어 두어야 한다.
  - 다만, 3-3에서 압축 해제한 파일의 파일명과 `-drive if=none,file=` 이후의 파일명이 일치하는지 확인하여야 한다.

```
$ echo "qemu-system-riscv64 -M virt -smp 2 -m 4096 -nographic -bios /usr/lib/
riscv64-linux-gnu/opensbi/generic/fw_jump.elf -kernel /usr/lib/u-boot/qemu-ri
scv64_smode/uboot.elf -drive if=none,file=ubuntu-22.04.3-preinstalled-server-
riscv64+unmatched.img,format=raw,id=hd0 -device virtio-blk-device,drive=hd0 -
device virtio-net-device,netdev=net0 -netdev user,hostfwd=tcp:127.0.0.1:2222
-:22,id=net0" >> riscv64-ubuntu-22.04.sh

$ chmod +x riscv64-ubuntu-22.04.sh
```

3-6. RISC-V 64bit 가상환경 실행

- 아래 명령을 사용하여 가상 환경을 실행한다.

```
$ ./riscv64-ubuntu-22.04.sh
```

- 만약 정상적인 부팅이 진행된다면, 몇 초간 기다린 뒤 아래와 같은 메시지가 출력되어야 한다.

```
Starting kernel ...

[    0.000000] Linux version 5.19.0-1021-generic (buildd@riscv64-qemu-lgw01-082)
 (riscv64-linux-gnu-gcc-12 (Ubuntu 12.1.0-2ubuntu1~22.04) 12.1.0, GNU ld (GNU Binu
tils for Ubuntu) 2.38) #23~22.04.1-Ubuntu SMP Thu Jun 22 12:49:35 UTC 2023 (Ubuntu
5.19.0-1021.23~22.04.1-generic 5.19.17)
[    0.000000] random: crng init done
[    0.000000] OF: fdt: Ignoring memory range 0x80000000 - 0x80200000
[    0.000000] Machine model: riscv-virtio,qemu
[    0.000000] earlycon: ns16550a0 at MMIO 0x0000000010000000 (options '')
[    0.000000] printk: bootconsole [ns16550a0] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] cma: Reserved 32 MiB at 0x00000000fd600000
[    0.000000] NUMA: No NUMA configuration found
[    0.000000] NUMA: Faking a node at [mem 0x0000000080200000-0x000000017fffffff]
[    0.000000] NUMA: NODE_DATA [mem 0x17fff0740-0x17fff1fff]
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000080200000-0x00000000ffffffff]
[    0.000000]   Normal   [mem 0x0000000100000000-0x000000017fffffff]
```

```
[    0.000000] Movable zone start for each node


......

[  OK  ] Started Dispatcher daemon for systemd-networkd.
[  OK  ] Finished Pollinate to seed…seudo random number generator.
         Starting OpenBSD Secure Shell server...
[  OK  ] Started OpenBSD Secure Shell server.

Ubuntu 22.04.3 LTS ubuntu ttyS0

......

# 이후 엔터를 2~3회 누르면 로그인 프롬프트가 뜨는 것을 확인할 수 있음
```

3-7. ssh 를 사용하여 RISC-V 64bit 가상 환경에 접속

- 아래 명령어를 실행해서 접속이 잘 되어야 를 진행할 수 있다.

```
# -p는 22번 포트를 제외한 다른 포트를 사용할 때 사용하는 옵션으로, 3-4에서 설정한 2222를 변경하지
 않았다면 아래와 같이 입력한다.

$ ssh ubuntu@127.0.0.1 -p 2222
```

4. 크로스 컴파일 한 뒤 원격에서 실행하기

4-1. RISC-V 가상 환경에서 실행할 c언어 파일 생성

- 아래와 같이 테스트를 위한 test.c 파일 생성 후 저장

```
# 텍스트 에디터면 모두 상관없음
$ vi ~/.bashrc
```

```
#include <stdio.h>

int main(void) {
  printf("hello world!\n");
}
```

4-2. RISC-V 가상 환경에서 실행할 수 있도록 크로스 컴파일

4-2-1. RISC-V 리눅스 가상 환경에서 동작시킬 코드 컴파일

- 아래 명령을 실행하여 RISC-V GCC 크로스 컴파일 도구로 코드 컴파일 진행

```
$ riscv64-unknown-linux-gnu-gcc -o riscv-test test.c
```

4-2-2. 크로스 컴파일 결과가 맞는지 실행 파일 확인

- RISC-V GCC 크로스 컴파일 도구에 의해 생성된 실행 파일이 맞는지 확인하기 위해 `file` 명령을 사용하여 확인
  - 실행 결과의 중간에 RISC-V 라는 문자열이 존재하면 해당 파일은 RISC-V 아키텍처에서 실행 가능한 실행파일

```
$ file riscv-test
riscv-test: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float ABI, ver
sion 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-riscv64-lp64d.s
o.1, for GNU/Linux 4.15.0, not stripped
```

  - x86-64 아키텍처에서 실행 가능한 실행 파일의 예시는 아래와 같음

```
$ file t-fibonacci
t-fibonacci: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamic
ally linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=50ecd5aa8
64f19a59998b05ddcfc661bf7b716ad, for GNU/Linux 3.2.0, not stripped
```

4-2-3. RISC-V 가상 환경으로 컴파일 된 실행파일 전송

- 리눅스의 SCP 명령을 사용하여 RISC-V 가상 환경으로 실행파일을 전송하기 위해 아래 명령어를 실행한다.

```
# -P는 22번 포트를 제외한 다른 포트를 사용할 때 사용하는 옵션으로, 3-4에서 설정한 2222를 변경하지
 않았다면 아래와 같이 입력한다.

# 실행 파일명에 컴파일 후 생성된 실행파일의 이름을 입력한다.
```

```
$ scp -P 2222 [실행파일명] ubuntu@127.0.0.1:/home/ubuntu
```

4-2-4. RISC-V 가상 환경에서 실행파일 실행

- ssh로 접속하거나, 에서 실행한 쉘을 사용해 RISC-V 64bit 가상 환경의 쉘에서 아래 명령을 실행한다.

```
# scp와 같이 외부에서 실행 파일을 전송해오는 경우 실행 권한이 사라지게 되기 때문에 실행 권한을 다시 부여해야 한다.
$ chmod +x [실행파일명]

$./[실행파일명]
```

- “hello world!” 문자열을 출력하는 프로그램이었기 때문에 정상적으로 동작하는 경우 해당 문자열이 출력되어야 한다.