

# Lab0.环境配置

## 安装Linux

对于本实验，我们需要使用Linux系统，在这里，推荐使用的方案是在Windows系统内使用WSL，在[这里](#)可以找到官方的安装教程，根据教程安装完WSL之后，在Microsoft Store中安装Ubuntu 20.04。

当然，上面的只是我们推荐的方案，如果你想安装真机或者其他的发行版，也可以自行安装，但是遇到问题请自行解决。

### 💡 关于Mac用户

本实验也是可以在Mac上操作的，但是我们并没有尝试过Mac的方案。

## 尝试在Linux中编程

### ✎ 写一个“Hello World”

在Linux下写一个“Hello World”程序并编译运行它，如果你不知道怎么做，你需要向互联网寻找答案。

接下来，如果你此前并没有接触过gdb，你还需要使用gdb去调试这个“Hello World”程序，因为此后的实验需要你使用gdb。如果没有安装gdb，请安装它，如何安装？互联网会告诉你的。

### ✎ 学习使用gdb

使用gdb调试你的“Hello World”程序，在主函数处打断点，并单步调试观察主函数的行为，此外你还可以在该程序中探索更多gdb的功能。

### 💡 别偷懒！

上面叫你写一个“Hello World”程序并且学习一下gdb，可不要偷懒啊，特别是此前还没有接触过Linux下编程和gdb调试的同学。

如果是此前已经具有Linux下编程经验的同学，你当然也可以选择性跳过上面的任务，选择下面的思考题，我们也非常欢迎其他同学在完成上述任务后选择做下面的思考题。

### ✎ 调试一个Segmentation fault的小程序

将下面两个文件中的代码放置于同一文件夹下，输入make指令，GNU make会根据Makefile中的依赖和构建关系自动生成a.out文件。

运行a.out文件，你会发现出现了Segmentation fault，那么为什么会出现Segmentation fault？

使用gdb从第一条指令开始执行，观察寄存器值的变化，你会找到答案的。既然出现了Segmentation fault，要如何修改代码才能使程序正常运行呢？把答案写在你的报告中。

```
// hello.c
int main(){
}
```

```
# Makefile
a.out:
    gcc -c hello.c
    ld hello.o -e main
```

### 💡 关于实验报告

请将本实验的报告和Lab1的报告放在一起。

## 安装工具

接下来我们需要安装一些实验必备的工具，对于不同的系统，我们提供了不同的安装方案。

### WSL

在WSL中运行以下命令：

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install git build-essential gdb-multiarch qemu-
system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu
```

### macOS

[这里](#)是macOS下进行环境配置的教程。

### 其他Linux

Debian 或者 Ubuntu：

```
sudo apt-get install git build-essential gdb-multiarch qemu-
system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu
```

Arch Linux：

```
sudo pacman -S riscv64-linux-gnu-binutils riscv64-linux-gnu-gcc  
riscv64-linux-gnu-gdb qemu-arch-extra
```

## 测试环境配置

为了测试安装成功与否，我们需要编译和运行xv6。

接下来你需要获取实验代码：

```
git clone https://gitlab.eduxiji.net/202310574111123/proj0.git
```

然后打开克隆下来的文件，需要切换到分支Lab01，这里有实验0和实验1的xv6代码：

```
git checkout lab01
```

在当前目录下的就是xv6的代码，紧接着输入：

```
make qemu
```

你会看到一系列的编译信息输出，最后可以看到如下输出：

```
xv6 kernel is booting  
  
hart 1 starting  
hart 2 starting  
init: starting sh  
$
```

此时，你的环境已经配置完成，输入 `Ctrl-a x` 就可以退出xv6。

如果失败了，你需要一个个模块进行测试。

QEMU：

```
qemu-system-riscv64 --version
```

以及至少一个可用的RISC-V的GCC：

```
riscv64-linux-gnu-gcc --version
```

```
riscv64-unknown-elf-gcc --version
```

```
riscv64-unknown-linux-gnu-gcc --version
```

