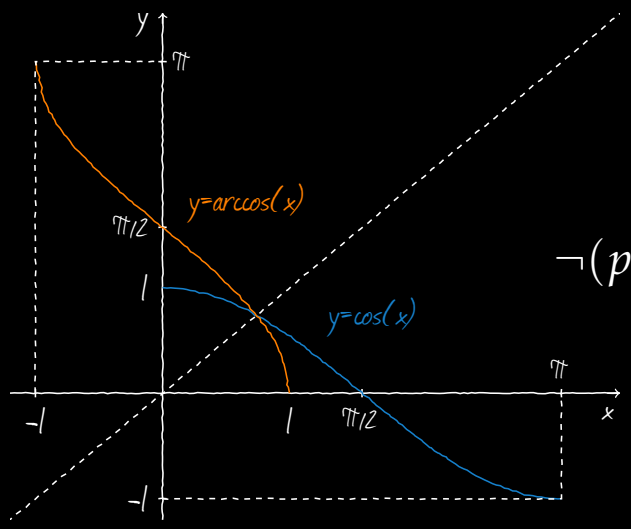
ZJUNIX

实验操作流程

浙江大学

2017.08.20

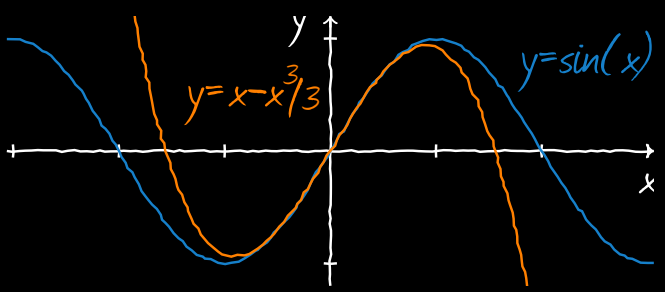
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$



$$\zeta_k = |a|^{1/n} \mathrm{e}^{i(\arg(a)+2k\pi)/n}$$

$$\mathrm{e}^{i\pi} + 1 = 0$$

$$\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$$



$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

目录

1.1 实验目的

1. 安装交叉编译工具链
2. 编译一个简单工程
3. 综合使用编译工具

1.2 安装交叉编译工具链

1.2.1 所需工具

用途	工具	版本	说明
交叉编译	MIPS-SDK	V1.4	用于编译操作系统内核
Linux 工具包	MSYS	MIPS-SDK V1.4 自带	用于在 Windows 下调用 make 等工具

表 1.1: 所需工具列表与版本

注：以上工具在培训时会以压缩包的形式提供给大家

1.2.2 交叉编译工具安装

Windows 安装

1. 下载交叉编译工具链MIPS SDK
2. 安装 MIPS SDK，请注意以下要点
   (a) 因为安装需要在线下载相关内容，所以需要保证网络环境的稳定
   (b) 安装目录最好不要带有空格
   (c) 安装支持工具时选择安装所有支持

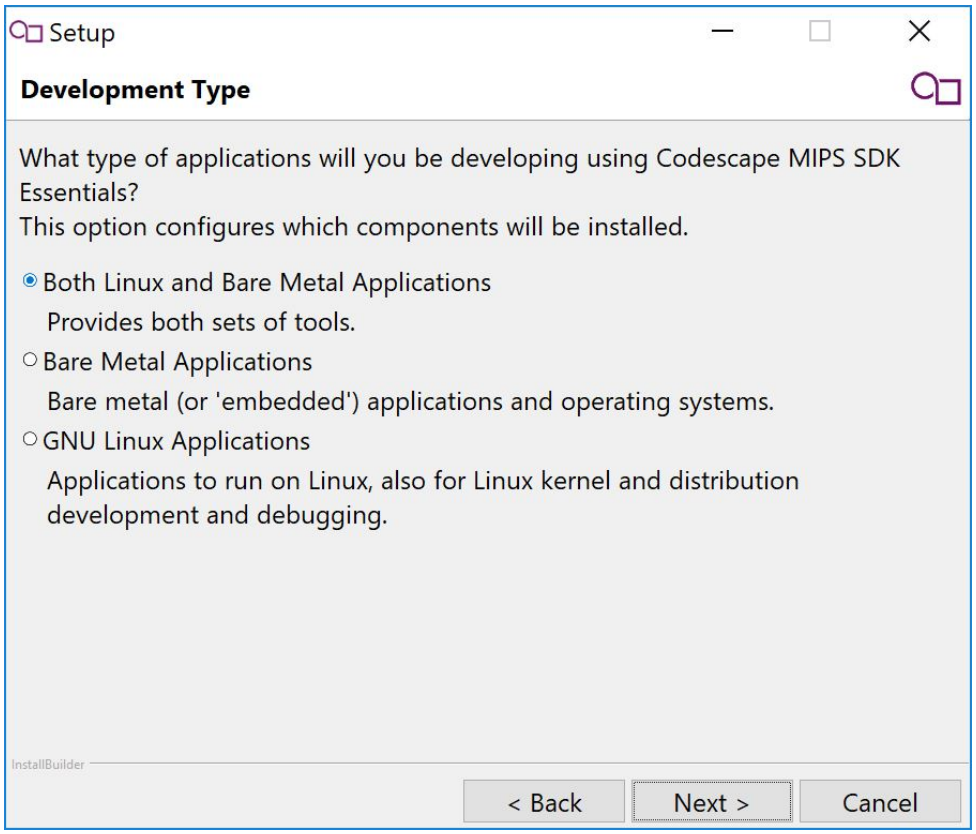


图 1.1: 安装相关支持包

(d) 在选择处理器平台时选择第一项 (Classic) 即可 (不选择任何平台可能导致工具链无法安装)

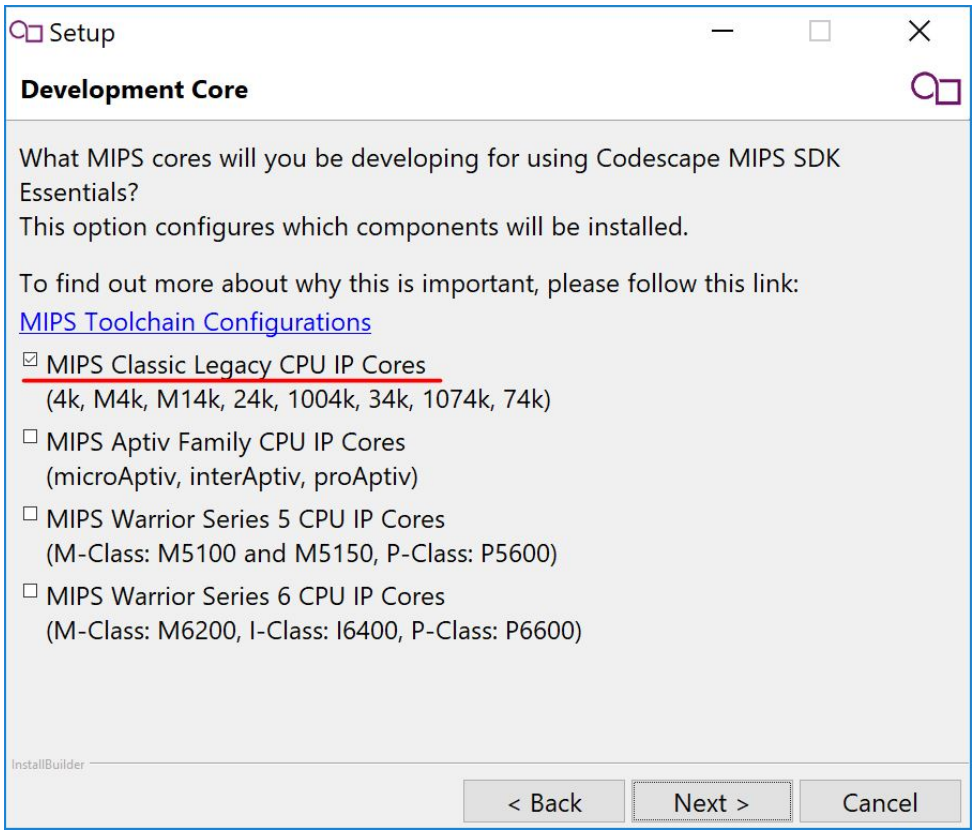


图 1.2: 选择处理器平台

3. 将交叉编译工具所在的目录添加到系统的 PATH 路径中

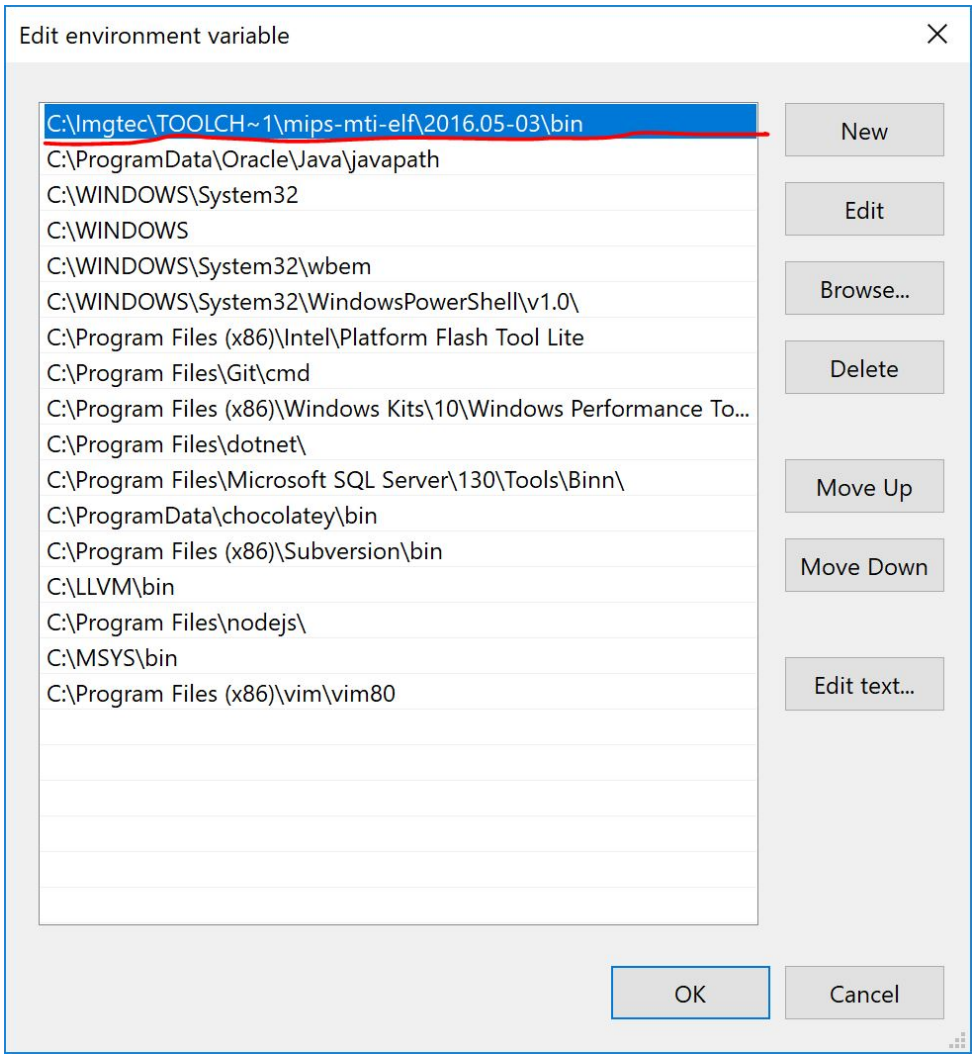


图 1.3: 添加到系统 PATH

4. 在 Windows 下需要额外安装 make 工具与相关工具，推荐使用 MIPS-SDK 自带的 MSYS 工具包

   (a) 找到 MIPS-SDK 自带的 MSYS 工具包位置，一般位于安装目录/Internals/msys/bin

   (b) 将上述目录添加到系统 PATH 路径中

5. 重启或重新登录 Windows，以更新 PATH 路径

6. 在 cmd 中使用命令 make -v 查看版本型号，MIPS-SDK V1.4 提供的 MSYS 中的 make 版本如图 1.4所示

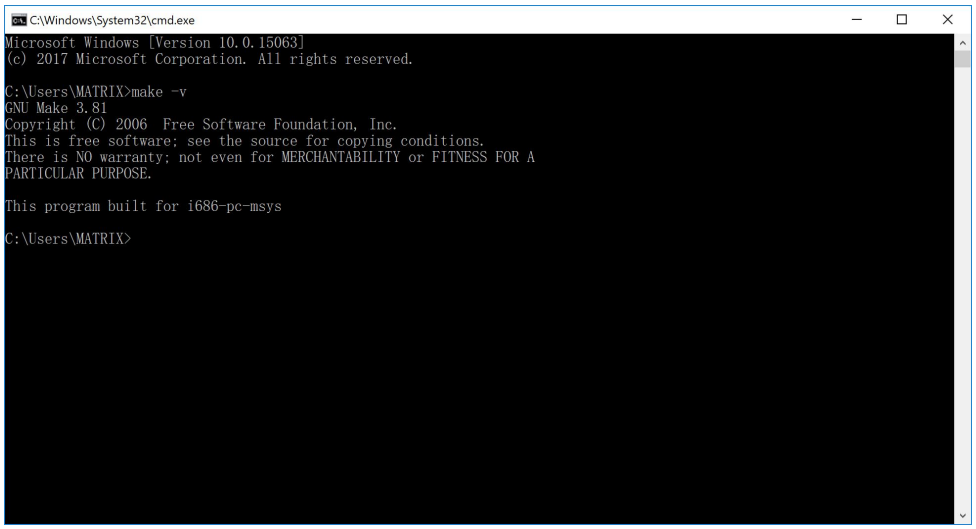


图 1.4: 查看 make 工具是否可用

7. 如果以上步骤可以看到 make 版本型号，说明 MSYS 工具包中的 make 工具已经可以使用

Linux 安装

1. 系统版本要求
   - 在 Ubuntu 16.04 LTS 版本上使用 MIPS SDK V1.4 测试编译功能通过
   - MIPS SDK 官方支持 CentOS
   - 其他版本的 Linux 发行版暂时没有测试
2. 下载交叉编译工具链MIPS SDK
3. 安装 MIPS SDK，请注意以下要点
   (a) 因为安装需要在线下载相关内容，所以需要保证网络环境的稳定
   (b) 安装目录最好不要带有空格
   (c) 安装支持工具时选择安装所有支持，如图 1.5

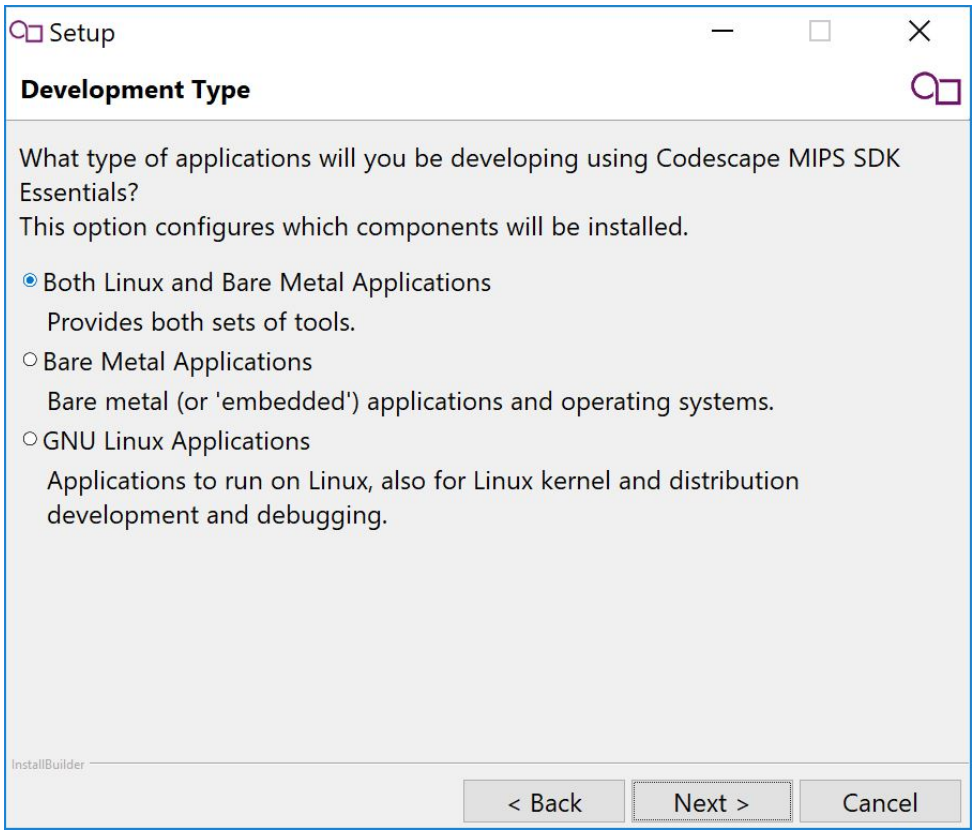


图 1.5: 安装相关支持包

(d) 在选择处理器平台时选择第一项 (Classic) 即可 (不选择任何平台可能导致工具链无法安装)，如图 1.6

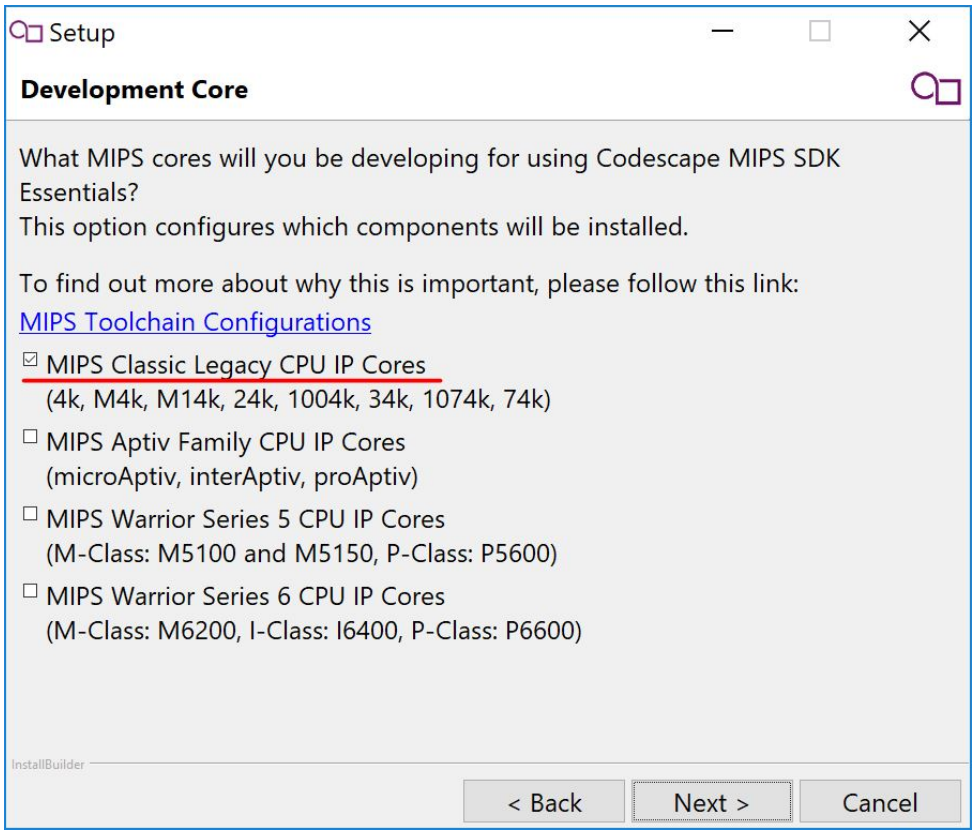


图 1.6: 选择处理器平台

Mac 安装

1. 很遗憾在 Mac 的 OS X 系统下没有对应版本的交叉编译工具链的支持，但是我们可以提供一些其他方法来帮助您完成环境配置
   - 但是如果您的 Mac 电脑还有足够的空间与性能安装一个虚拟机，则请您耐心读完接下来的说明
   - 如果您的 Mac 电脑无法安装虚拟机，您可以在到达培训地点后使用我们提供的机器，我们会提供统一的操作系统与配置好的软件环境供您使用
2. 请您在 Mac 中安装一个 Ubuntu 16.04 LTS 的虚拟机 (官方也支持 Cent OS，但是由于条件有限我们暂时没有验证过可行性，仅验证过 Ubuntu 16.04 LTS 上软件环境的正确性)
3. 然后请您参照节 1.2.2的对应指导进行安装
4. 如果您在虚拟机中使用工具表现不佳，可以在到达培训地点后使用我们提供的机器，会有统一的操作系统和预装的软件环境

1.3 编译一个简单工程

1.3.1 获取代码

获取工程代码有两种方式：

1. 工程代码托管在 Github 上，可以在 https://github.com/ZJUNIX/ZJUNIX-Exp/releases 下载完整工程 Source coee (zip)，其中每个实验的工程都以文件夹区分

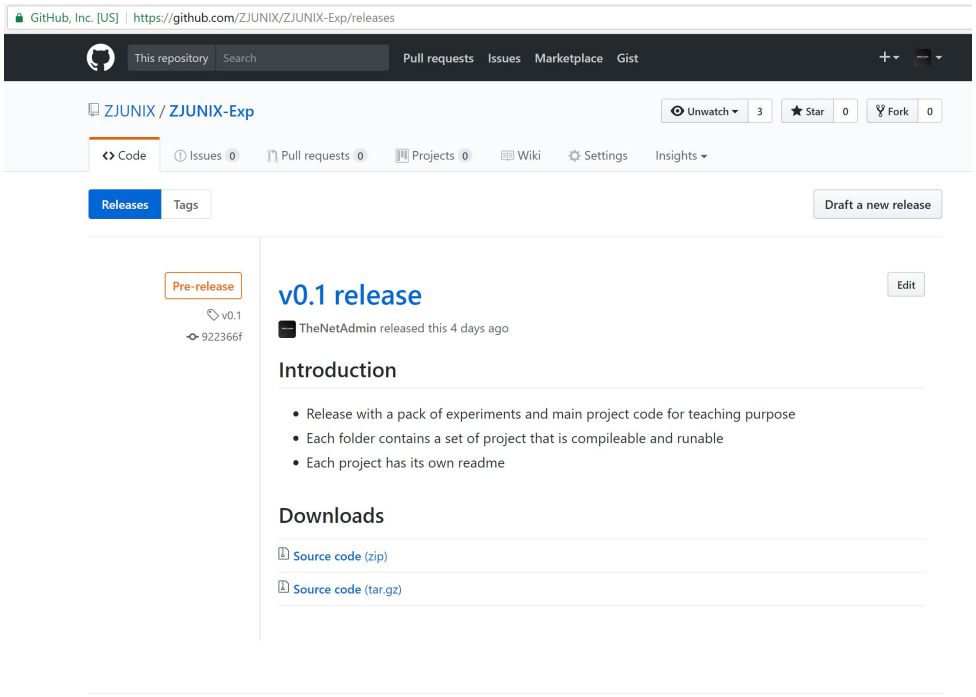


图 1.7: Github 下载工程

2. 从浙大云盘/2017 面向系统能力培训研讨会/培训资料/代码/ZJUNIX 内核处可以下载到相同的工程代码

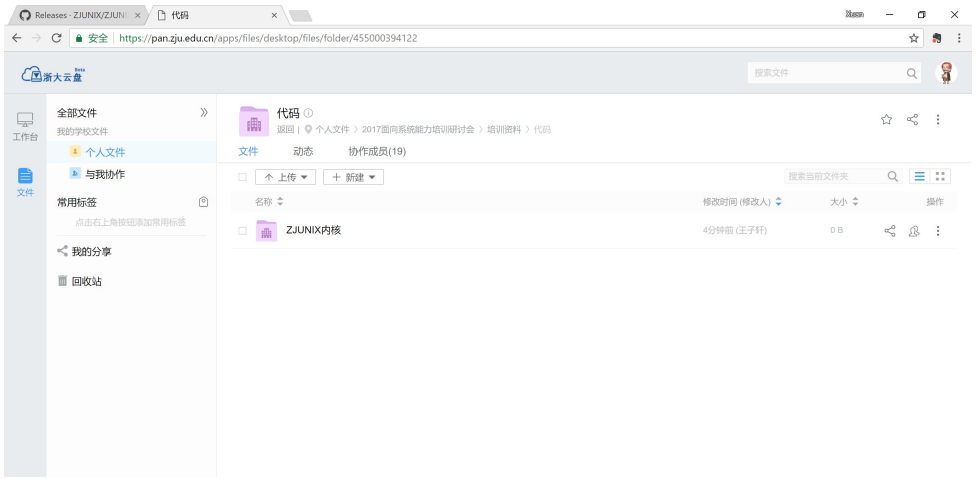


图 1.8: 浙大云盘下载的工程

1.3.2 执行第一次编译

编译前的配置

1. 设置交叉编译工具路径

- 在程序根目录下的 config/tools.conf 中修改 TOOLCHAIN_DIR 为编译器二进制文件所在目录（需要包含工具名称前缀）

```
# toolchain location, with tool filename prefix
export TOOLCHAIN_DIR := "C:/Imgtec/Toolchains/mips-mti-elf/2016.05-03/bin/mips-mti-elf-"
```

图 1.9: 设置交叉编译工具路径

2. 设置 make 工具路径

- 如果使用的是 Windows 系统，则修改 config/tools.conf 中的 MAKE 为 make 可执行文件所在绝对路径，参考项 4，即使已经添加了 PATH 也需要指定绝对路径

```
# explicitly set under Windows to prevent path problems
export MAKE := "C:/MSYS/bin/make.exe"
```

图 1.10: 指定 make 工具路径

- 如果使用的是 Linux 系统，则注释或删除该行

开始编译

进入 exp0 工程目录，在该工程根目录运行 make 命令，即可进行编译，结果如图 1.11

```
PS C:\CODE\ZJUNIX-Exp\exp0> make
Building ZJUNIX kernel for mips32
        LD -Map kernel.map -o kernek.elf
c:\Imgtec\Toolchains\mips-mti-elf\2016.05-03\bin\mips-mti-elf-ld.exe: warning: cannot find entry symbol exception; defaulting to 0000000080000000
        OC -S -O binary kernel.elf kernel.bin
Target kernel.bin build finished.
```

图 1.11: 编译结果

在编译时会产生一个 warning，如代码 1.1所示

代码 1.1: 编译 Warnning

```
warning: cannot find entry symbol exception; defaulting to 0000000080000000
```

这是因为我们在 Makefile 中指定了入口代码标签为 exception，如果修改了 Makefile 中的 ENTRANCE 值，则会产生不同的入口标签 warning

这个 warning 暂时可以不用理会。在默认情况下，如果代码中含有 main() 函数，则编译器会自动为其生成一个特殊的标签，并将这个标签作为代码入口标签。但是在内核中不存在 main() 函数，所以我们在内核开发时，需要手动指定一个代码入口标签，告诉编译器我们的代码从什么位置开始执行

1.4 综合使用编译工具

1.4.1 make 工具使用

命令	说明
make	生成内核镜像 kernel.bin
make all	生成内核镜像 kernel.bin
make disassembly	生成反汇编文件 kernel.txt
make install INSTALL_DIR=…	生成内核镜像 kernel.bin 并将其复制到指定目录
make clean	清除所有编译产生的文件
make distclean	清除中间文件，仅保留目标文件与反汇编文件

表 1.2: make 命令列表

请分别尝试以上各个命令，查看其效果

1.4.2 反汇编

在工程根目录下，使用 make disassembly 进行反汇编，可以得到的结果如下

代码 1.2: 反汇编结果

```
kernel.elf:     file format elf32-tradlittlemips


Disassembly of section .text:

80000000 <private_func>:
80000000:   27bdfff8    addiu   sp,sp,-8
80000004:   afbe0004    sw  s8,4(sp)
80000008:   03a0f025    move    s8,sp
8000000c:   03c0e825    move    sp,s8
80000010:   8fbe0004    lw  s8,4(sp)
80000014:   27bd0008    addiu   sp,sp,8
80000018:   03e00008    jr  ra
8000001c:   00000000    nop

80000020 <public_func>:
80000020:   27bdfff8    addiu   sp,sp,-8
80000024:   afbe0004    sw  s8,4(sp)
80000028:   03a0f025    move    s8,sp
8000002c:   03c0e825    move    sp,s8
80000030:   8fbe0004    lw  s8,4(sp)
80000034:   27bd0008    addiu   sp,sp,8
80000038:   03e00008    jr  ra
8000003c:   00000000    nop
```

由于这两个函数都没有做任何事情，所以这两个函数中仅做了入口的入栈，出口的出栈以及函数返回的操作

代码 1.3: 反汇编结果分析

```
80000000 <private_func>:
# 入口的入栈
80000000:   27bdfff8    addiu   sp,sp,-8
80000004:   afbe0004    sw  s8,4(sp)
80000008:   03a0f025    move    s8,sp
# 出口的出栈
8000000c:   03c0e825    move    sp,s8
80000010:   8fbe0004    lw  s8,4(sp)
80000014:   27bd0008    addiu   sp,sp,8
# 函数返回
80000018:   03e00008    jr  ra
8000001c:   00000000    nop
```

1.4.3 内联汇编

请在工程的 kernel/init.c 中加入如代码 1.4所示的函数

代码 1.4: 内联汇编使用

```
unsigned int write_status(unsigned int val){
    unsigned int ret;
    asm volatile(
        "mfc0 %0, $12\n\t"
        "mtc0 %1, $13\n\t"
        :"=r"(ret)
        :"r"(val));
    return ret;
}
```

然后依次执行 make、make disassembly 即可得到反汇编结果，如代码 1.5

代码 1.5: 反汇编结果分析

```
80000000 <write_status>:
# 入栈
80000000:   27bdfff0    addiu   sp,sp,-16
80000004:   afbe000c    sw  s8,12(sp)
80000008:   03a0f025    move    s8,sp
8000000c:   afc40010    sw  a0,16(s8)
80000010:   8fc20010    lw  v0,16(s8)
# 内嵌汇编
80000014:   40026000    mfc0    v0,c0_status
80000018:   40826800    mtc0    v0,c0_cause
# 出栈
8000001c:   afc20000    sw  v0,0(s8)
```

```
80000020:   8fc20000    lw  v0,0(s8)
80000024:   03c0e825    move    sp,s8
80000028:   8fbe000c    lw  s8,12(sp)
8000002c:   27bd0010    addiu   sp,sp,16
# 函数返回
80000030:   03e00008    jr  ra
80000034:   00000000    nop
```