《高级操作系统》实验1

PolyOS AIoT安装和内核编译与安装

实验内容

- 任务一：PolyOS AIoT操作系统的安装
- 任务二： PolyOS AIoT内核编译与安装
- 任务三：内核模块编程

PolyOS AIoT简介

- PolyOS AIoT是什么
  - 面向RISC-V体系架构和基于yocto构建的AIoT嵌入式场景Linux系统。
  - 是一个开源免费的Linux发行版系统/平台。
  - 和开源爱好者共同构建一个开放、多元和架构包容的软件生态体系。

聚元PolyOS
RISC-V based AIoT Operating System

- PolyOS AIoT的发展历程
  - 2022年8月25日，软件所宣布 PolyOS AIoT 开源，PolyOS AIoT 开源社区正式上线。

PolyOS AIoT特点

- PolyOS AIoT特点

  聚元PolyOS AIoT 特点如下：

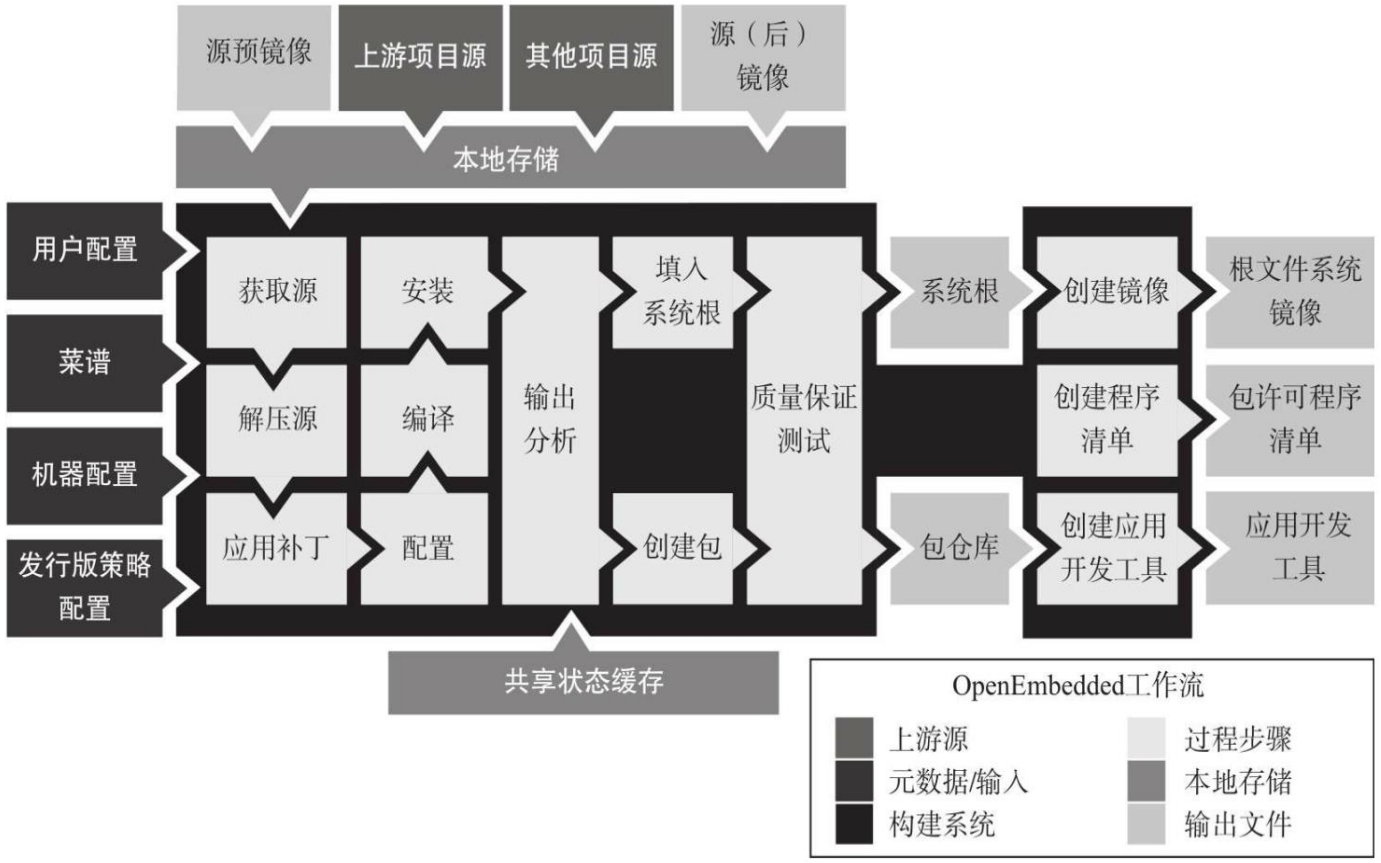
  - 以yocto metadata层为粒度进行定制和组合
  - 基于yocto kas配置管理工具
  - 预配置了openembedded、meta-riscv等软件生态层
  - 通过yml声明硬件描述、应用场景以及打包规则
  - 实现一站式快速构建场景化智能应用镜像

PolyOS AIoT构建基础

- PolyOS AIoT **构建工作流**

构建的工作流由以下几个组成：

- 用户配置
- 元数据层
- 源文件：上游版本、本地项目
- 构建系统：bitbake控制下的进程
- 包提取
- 镜像：由工作流生成的镜像。

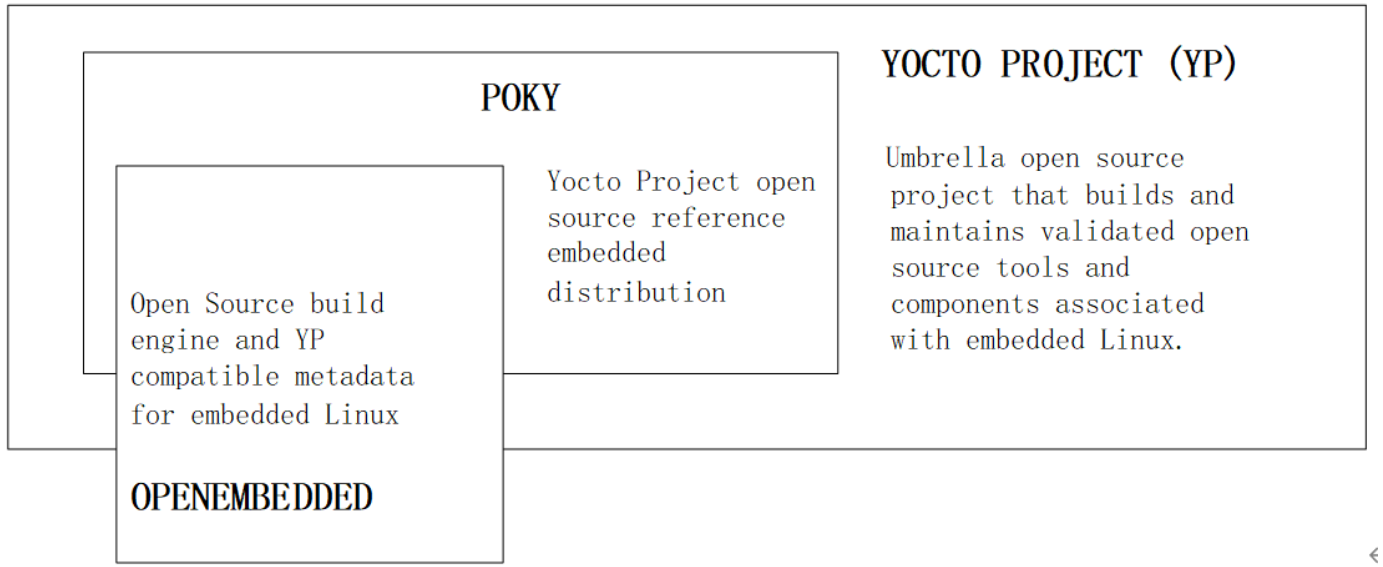


PolyOS AIoT构建基础

- PolyOS AIoT**构建基础**

Yocto主要由以下几部分组成：

1. BitBake：
   Yocto 的构建系统引擎
2. OpenEmbedded-Core ：
   所需的基本配方、相关文件和类
3. Poky ：
   基于各种架构生成文件系统镜像



PolyOS AIoT构建基础

- PolyOS AIoT **元数据**

元数据分为配置文件和配方。

配方

- BitBake用来设置变量
- 包含诸如版本、许可证和上游源等字段
- 使用提取、编译以及其它构建过程
- 以.bb或者.bbappend文件扩展名结尾

配置文件

- 全局变量定义、用户定义变量和硬件配置信息
- 以 .conf 文件扩展名结尾

PolyOS AIoT构建基础

- PolyOS AIoT 元数据

配方例子

```
DESCRIPTION = "Simple helloworld application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
SRC_URI = "file://helloworld.c"
S = "${WORKDIR}"
do_compile() {
        ${CC} ${LDFLAGS} helloworld.c -o helloworld
}
do_install() {
        install -d ${D}${bindir}
        install -m 0755 helloworld ${D}${bindir}
}.
```

PolyOS AIoT 层

- PolyOS AIoT 中的层

PolyOS AIoT采用了 Yocto 项目中的层模型。

层的特点：

- 用于嵌入式和物联网 Linux 创建的开发模型。
- 层模型同时支持协作和定制。
- 包含相关指令集的存储库。
- 将不同的信息隔离成层，有助于简化定制和重用。

PolyOS AIoT 层

- oscourse 层

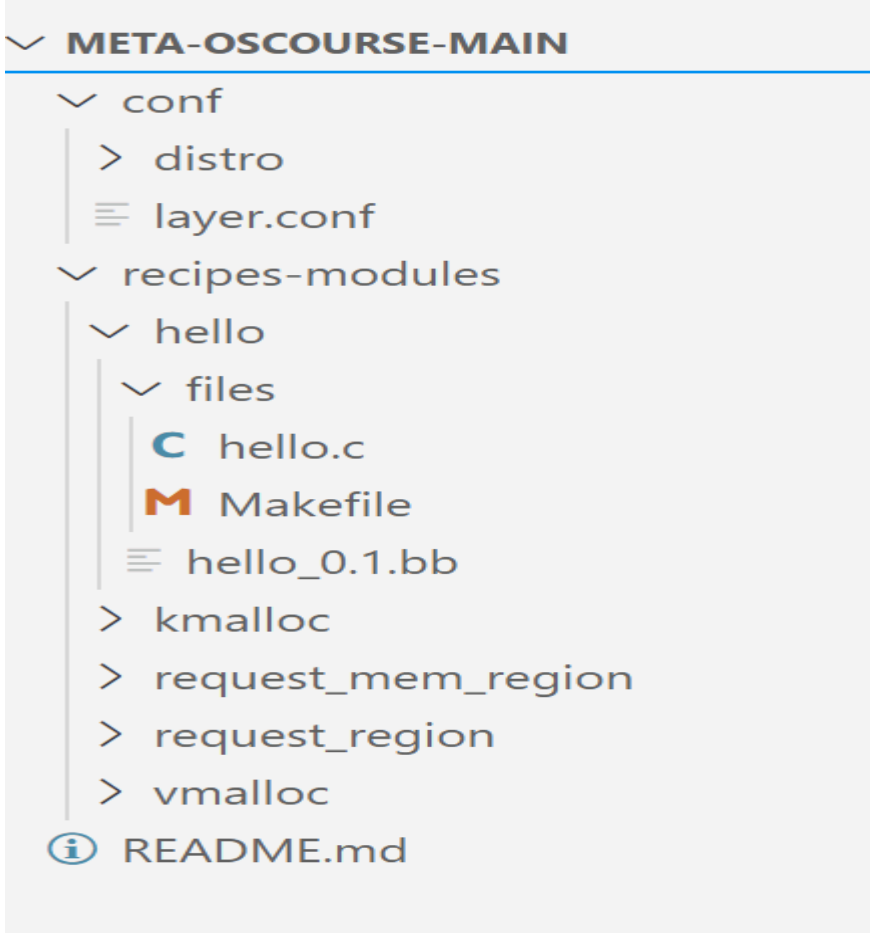
  conf/layer.conf：代表该层的配置文件

  recipes-modules:文件夹下是实验的相关作业

  hello 文件夹 是 第一个实验：

  有配方文件、源文件和Makefile ；

  加载卸载模块，看到内核打印信息。



PolyOS AIoT 层

conf/layer.conf内容

```
BBPATH .= ":${LAYERDIR}"

BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "meta-oscourse"
BBFILE_PATTERN_meta-oscourse = "^${LAYERDIR}/"
BBFILE_PRIORITY_meta-oscourse = "8"
LAYERVERSION_meta-oscourse = "1"
LAYERDEPENDS_meta-oscourse = "core"
LAYERSERIES_COMPAT_meta-oscourse = "honister"
MACHINE_ESSENTIAL_EXTRA_RRECOMMENDS += "hello "
KERNEL_MODULE_AUTOLOAD += "hello "
.
```

PolyOS AIoT 编译构建

- PolyOS AIoT**编译构建**

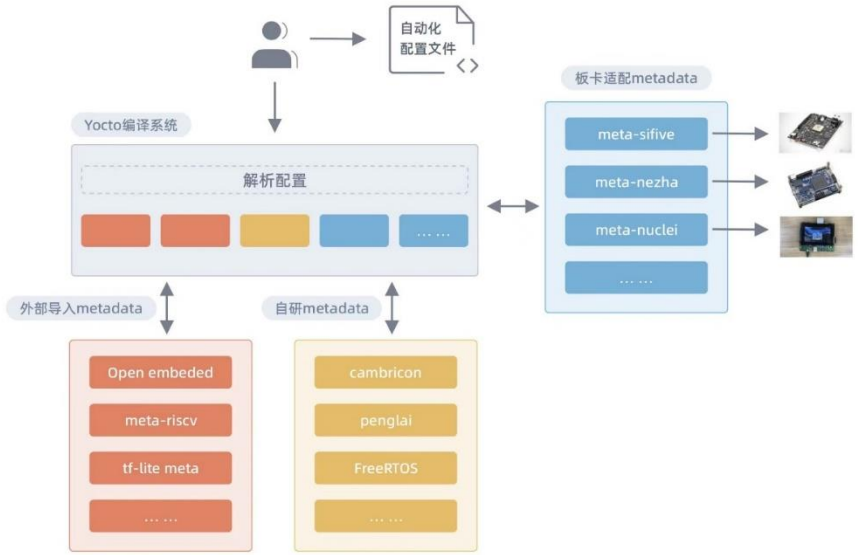
聚元PolyOS AIoT的总体编译架构如右图，

解析配置的层有：

各种板卡适配（meta-xiangshan等）层、

外部导入的层（meta-riscv）、

软件生态层（meta-oscourse）等。



PolyOS AIoT 编译构建

- PolyOS AIoT**编译构建**

build_portal是编译构建PolyOS AIoT入口，借助KAS的实现原理，完成基于bitbake编译的项目。

KAS的特点：

- （1）通过yml中的url和refspec信息，自动下载需要的meta-layer层

- （2）根据yml中的machine值，创建默认的bitbake环境配置

PolyOS AIoT 内核

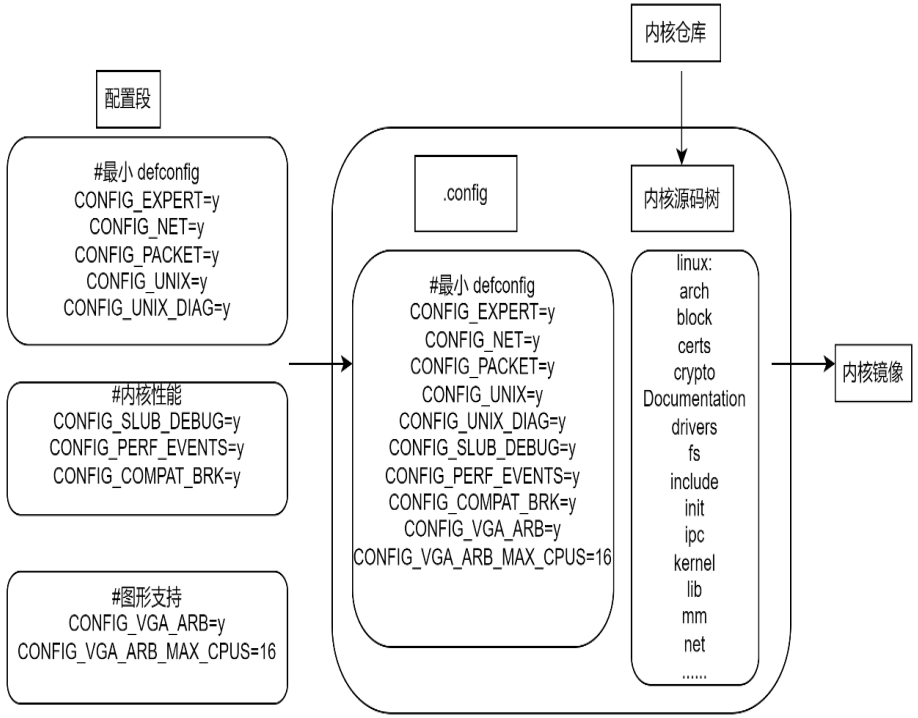
- PolyOS AIoT**内核**

  内核镜像由内核源码和配置片段组成的配置组成。

  内核的源码通过SRC_URI变量进行设定，获取内核源码的地址、分支和协议。

  内核元数据分为3种文件类型：

- 补丁
- 配置
- 描述文件，后缀为.scc



PolyOS AIoT 内核

- 内核特有的任务

do_kernel_configme: 将所有内核配置片段组装并合并为一个合并的配置

do_configure:使用defconfig文件配置解压缩的内核

do_kernel_configcheck: 验证do_config任务生成的配置

do_compile_kernelmodules：构建内核模块

do_bundle_initramfs：将Initramfs映像和内核组合在一起以形成单个映像

任务一：编译和运行PolyOS AIoT

- 基本步骤
  - 1. 编译环境搭建
  - 2. 编译PolyOS AIoT
  - 3. 运行PolyOS AIoT

任务一：编译环境搭建

所需软件：

1. 安装KAS

```
#  cd  existing_repo
#  git clone https://github.com/siemens/kas.git
#  cd kas
#  sudo pip3 install .
```

官方参考文档：https://isrc.iscas.ac.cn/riscv-raios/docs/kas/build_portal_started.html

任务一：编译环境搭建

2. 软件包的安装

在Ubuntu 20.04环境下，需要安转的软件包

```
#  sudo apt install chrpath diffstat zstd  device-tree-compiler lz4
```

在openEuler 22.03环境下，需要安转的软件包

```
#  sudo yum install python3-newt g++ patch  rpcgen chrpath  diffstat zstd  lz4
#  sudo ln -s /usr/bin/python3.9 /usr/bin/python
```

任务一：编译PolyOS AIoT

下载构建PolyOS AIoT入口和编译PolyOS AIoT

```
#  cd  existing_repo
#  git clone https://gitee.com/riscv-raios/build_portal.git
#  cd build_portal
#   PATH=${PATH}:~/.local/bin kas build  common-oscourse-qemuriscv64.yml
```

等待编译结束

任务一：运行PolyOS AIoT

前面的流程编译结束后

```
#  kas shell common-qemuriscv64-core-image-minimal.yml  -c 'runqemu nographic'
```

runqemu实际调用：

qemu-system-riscv64　　（可以自己装下）

后面主要参数：

```
-smp   4                  //设置虚拟CPU数目
-m     256               //内存大小，默认256MB
-machine                 //选择的模拟计算机
-device                  // guest上总线挂载的外部设备
-drive                   //   镜像文件的位置（这指文件系统）
 -bios                   //指定BIOS位置
 -kernel                 //内核的位置
```

任务一：运行PolyOS AIoT

qemu启动界面

```
OpenSBI v0.9
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : riscv-virtio,qemu
Platform Features         : timer,mfdeleg
Platform HART Count       : 4
Firmware Base             : 0x80000000
Firmware Size             : 124 KB
Runtime SBI Version       : 0.2
```

ISCAS

任务一：运行PolyOS AIoT

```
[  OK  ] Finished Permit User Sessions.
[  OK  ] Started Getty on tty1.
[  OK  ] Started Serial Getty on ttyS0.
[  OK  ] Reached target Login Prompts.
[  OK  ] Started User Login Management.
[  OK  ] Reached target Multi-User System.
         Starting Record Runlevel Change in UTMP...
[  OK  ] Finished Record Runlevel Change in UTMP.

PolyOS PolyOS.1.0 qemuriscv64 ttyS0

qemuriscv64 login:
```

输入 root ，进入文件系统

任务一：运行PolyOS AIoT

进入文件系统，输入常用的命令

```
root@qemuriscv64:~# uname -a
Linux qemuriscv64 5.15.6-yocto-standard #1 SMP PREEMPT Wed Dec 1 13:21:37 UTC
 2021 riscv64 GNU/Linux
root@qemuriscv64:~# date
Fri Nov 19 17:33:00 UTC 2021
root@qemuriscv64:~#
```

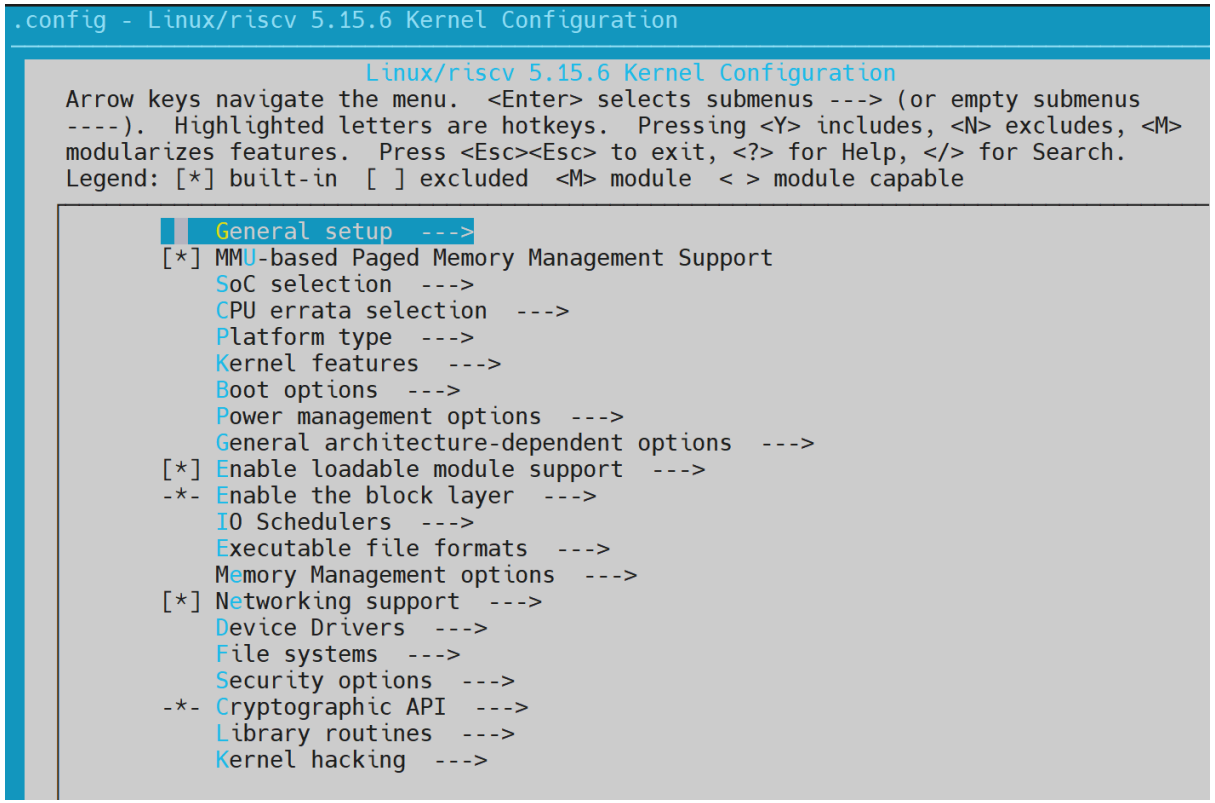
任务二：PolyOS AIoT内核编译与安装

- 任务描述
  - 熟悉PolyOS AIoT模块编译。
  - 熟悉PolyOS AIoT内核配置和编译。
- 审核要求
  - 正确编译内核和模块并完成安装。

任务二：PolyOS AIoT内核编译与安装

- 使用menuconfig工具配置编译选项

```
$ kas shell common-oscourse-qemuriscv64.yml -c 'bitbake  linux-yocto -c menuconfig'
```



Y 入内核

N 不入内核

M 以模块形式编译

任务二：PolyOS AIoT内核编译与安装

- 编译和安装内核

```
# kas shell common-oscourse-qemuriscv64.yml -c 'bitbake  linux-yocto'
```

- 编译和安装模块

```
# kas shell common-oscourse-qemuriscv64.yml -c 'bitbake  hello'
```

任务三：内核模块编程

- 任务描述
  - 编写内核模块，功能是打印“hello , world!” 字符串。
  - 编写对应 Makefile 文件。
  - 手动加载内核模块，查看加载内容。
  - 手动卸载上述内核模块。
- 审核要求
  - 正确编写满足功能的源文件，正确编译。
  - 正常加载、卸载内核模块，且内核模块功能满足任务所述。
  - 提交相关源码与运行截图。

任务三：内核模块编程

- 1. 内核模块基本结构

```
#include<linux/module.h>                    //包含了对模块的结构定义以及模块的版本控制

static __init module_init(void){            //加载模块
      。。。。。。
}

static __exit module_exit(void){            //卸载模块
      。。。。。。
}
module_init(module_init);
module_exit(module_exit);
MODULE_LICENSE("GPL");                  //声明GPL版权
```

内核模块中，使用的是 printk()函数，而非 printf()。

任务三：内核模块编程

- 2. Makefile 文件
  - 内核模块的编译还需要 Makefile 文件
  - 注意：M是大写，不是makefile

```
obj-m += hello.o

SRC := $(shell pwd)

all:
	$(MAKE) -C $(KERNEL_SRC) M=$(SRC)

modules_install:
	$(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install

clean:
	$(MAKE) -C $(KERNEL_SRC) M=$(SRC) clean
```

任务三：内核模块编程

- 3. 配方文件
  - SRC_URI存放源文件和Makefile
  - RPROVIDES:${PN}中值为kernel-module加上模块名

```
LICENSE = "CLOSED"
SRC_URI = "file://hello.c \
                   file://Makefile \
                    "

inherit module

S = "${WORKDIR}"

# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.
RPROVIDES:${PN} += "kernel-module-hello"
```

任务三：内核模块编程

- 内核模块常用操作
  - 查看内核模块：lsmod
    - 如：lsmod | grep hello
  - 若模块未加载，加载内核模块：insmod
    - cd /lib/modules/5.15.6-yocto-standard/extra //模块位置
    - insmod hello.ko
  - 若模块已经加载，卸载内核模块：rmmod
    - 如：rmmod hello.ko

  - 加载/卸载内核模块后，查看模块打印信息：
    - dmesg | tail -n 2
      tail -n <行数> 显示文件的尾部 n 行内容

作业

- 任务一：成功安装PolyOS AIoT
- 任务二：PolyOS AIoT中内核配置的修改和内核编译安装
- 任务三：编写第一个内核模块，并编译、加载、卸载内核模块。通过查看内核消息了解操作是否成功。

作业提交：
每一个任务关键操作截图，所有任务结果保存到一个Word文件中。
内核模块以自己的学号命名。

交作业截止日期：10月8日之前。

本节完