实验二 Hello, miniEuler

计科2302 王任杰 202308010212

print函数是学习几乎任何一种软件开发语言时最先学习使用的函数，同时该函数也是最基本和原始的程序调试手段，但该函数的实现却并不简单。本实验的目的在于理解操作系统与硬件的接口方法，并实现一个可打印字符的函数（非系统调用），用于后续的调试和开发。

了解virt机器

操作系统介于硬件和应用程序之间，向下管理硬件资源，向上提供应用编程接口。设计并实现操作系统需要熟悉底层硬件的组成及其操作方法。

本系列实验都会在QEMU模拟器上完成，首先来了解一下模拟的机器信息。可以通过下列两种方法：

查看QEMU关于 virt的描述 ， 或者查看QEMU的源码，如github上的 virt.h 和 virt.c。virt.c中可见如下有关内存映射的内容。

static const MemMapEntry base\_memmap[] = {

/\* Space up to 0x8000000 is reserved for a boot ROM \*/

[VIRT\_FLASH] = { 0, 0x08000000 },

[VIRT\_CPUPERIPHS] = { 0x08000000, 0x00020000 },

/\* GIC distributor and CPU interfaces sit inside the CPU peripheral space \*/

[VIRT\_GIC\_DIST] = { 0x08000000, 0x00010000 },

[VIRT\_GIC\_CPU] = { 0x08010000, 0x00010000 },

[VIRT\_GIC\_V2M] = { 0x08020000, 0x00001000 },

[VIRT\_GIC\_HYP] = { 0x08030000, 0x00010000 },

[VIRT\_GIC\_VCPU] = { 0x08040000, 0x00010000 },

/\* The space in between here is reserved for GICv3 CPU/vCPU/HYP \*/

[VIRT\_GIC\_ITS] = { 0x08080000, 0x00020000 },

/\* This redistributor space allows up to 2\*64kB\*123 CPUs \*/

[VIRT\_GIC\_REDIST] = { 0x080A0000, 0x00F60000 },

[VIRT\_UART] = { 0x09000000, 0x00001000 },

[VIRT\_RTC] = { 0x09010000, 0x00001000 },

[VIRT\_FW\_CFG] = { 0x09020000, 0x00000018 },

[VIRT\_GPIO] = { 0x09030000, 0x00001000 },

[VIRT\_SECURE\_UART] = { 0x09040000, 0x00001000 },

[VIRT\_SMMU] = { 0x09050000, 0x00020000 },

[VIRT\_PCDIMM\_ACPI] = { 0x09070000, MEMORY\_HOTPLUG\_IO\_LEN },

[VIRT\_ACPI\_GED] = { 0x09080000, ACPI\_GED\_EVT\_SEL\_LEN },

[VIRT\_NVDIMM\_ACPI] = { 0x09090000, NVDIMM\_ACPI\_IO\_LEN},

[VIRT\_PVTIME] = { 0x090a0000, 0x00010000 },

[VIRT\_SECURE\_GPIO] = { 0x090b0000, 0x00001000 },

[VIRT\_MMIO] = { 0x0a000000, 0x00000200 },

/\* ...repeating for a total of NUM\_VIRTIO\_TRANSPORTS, each of that size \*/

[VIRT\_PLATFORM\_BUS] = { 0x0c000000, 0x02000000 },

[VIRT\_SECURE\_MEM] = { 0x0e000000, 0x01000000 },

[VIRT\_PCIE\_MMIO] = { 0x10000000, 0x2eff0000 },

[VIRT\_PCIE\_PIO] = { 0x3eff0000, 0x00010000 },

[VIRT\_PCIE\_ECAM] = { 0x3f000000, 0x01000000 },

/\* Actual RAM size depends on initial RAM and device memory settings \*/

[VIRT\_MEM] = { GiB, LEGACY\_RAMLIMIT\_BYTES },

};

通过QEMU导出设备树

安装设备树格式转换工具

Mac下安装

brew install dtc

Linux下安装

apt-get install device-tree-compiler

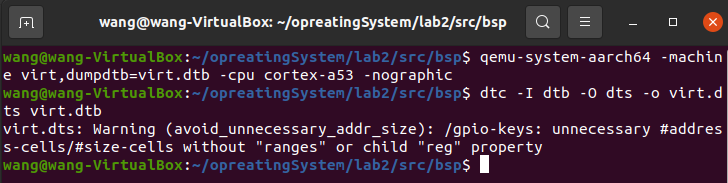
通过QEMU导出设备树并转成可读格式

qemu-system-aarch64 -machine virt,dumpdtb=virt.dtb -cpu cortex-a53 -nographic

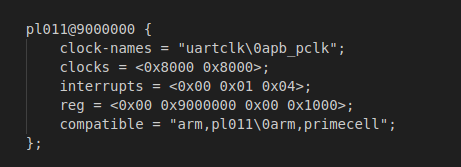
dtc -I dtb -O dts -o virt.dts virt.dtb

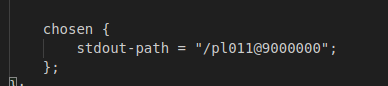
备注

-machine virt 指明机器类型为virt，这是QEMU仿真的虚拟机器。



virt.dtb转换后生成的virt.dts中可找到如下内容





由上可以看出，virt机器包含有pl011的设备，该设备的寄存器在0x9000000开始处。pl011实际上是一个UART设备，即串口。可以看到virt选择使用pl011作为标准输出，这是因为与PC不同，大部分嵌入式系统默认情况下并不包含VGA设备。

实现 PRT\_Printf 函数

本系列实验每个实验均依赖前序相关实验，因此可拷贝 lab1 目录并重命名为 lab2 ，在 lab2 目录中再操作（后续实验照此操作）。

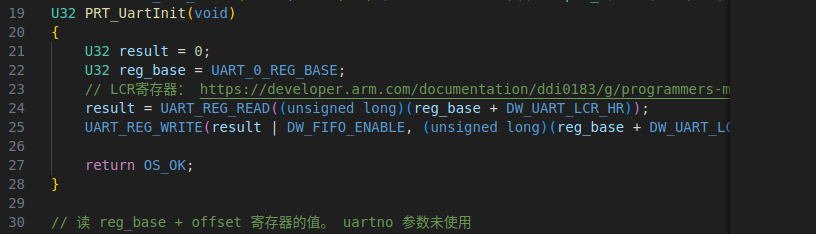
新建 src/bsp/print.c 文件，完成如下部分。

宏定义

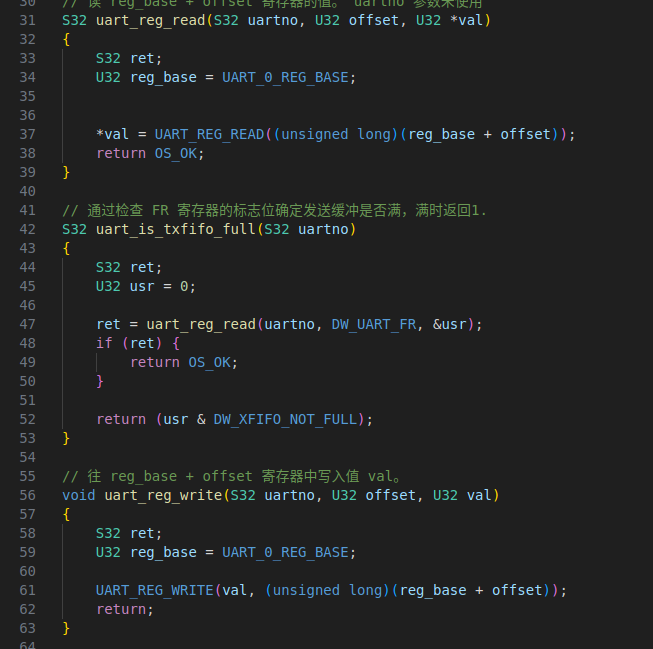
在 print.c 中包含所需头文件，并定义后续将会用到的宏

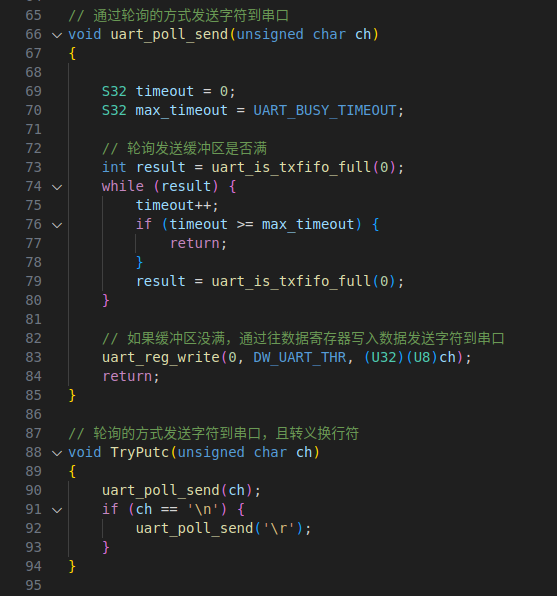


串口的初始化



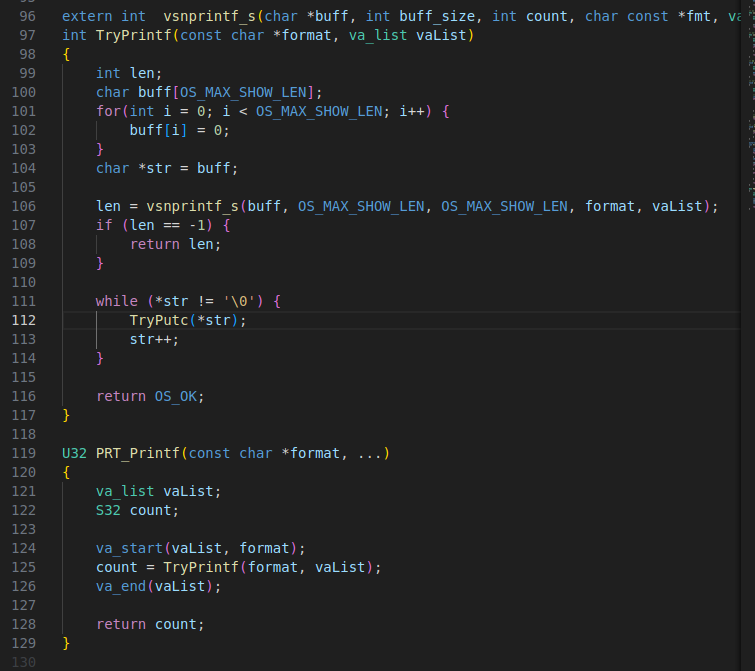
往串口发送字符





上面的代码很简单，就是通过轮询的方式向 PL011 的数据寄存器 DR 写入数据即可实现往串口发送字符，实现字符输出。

支持格式化输出

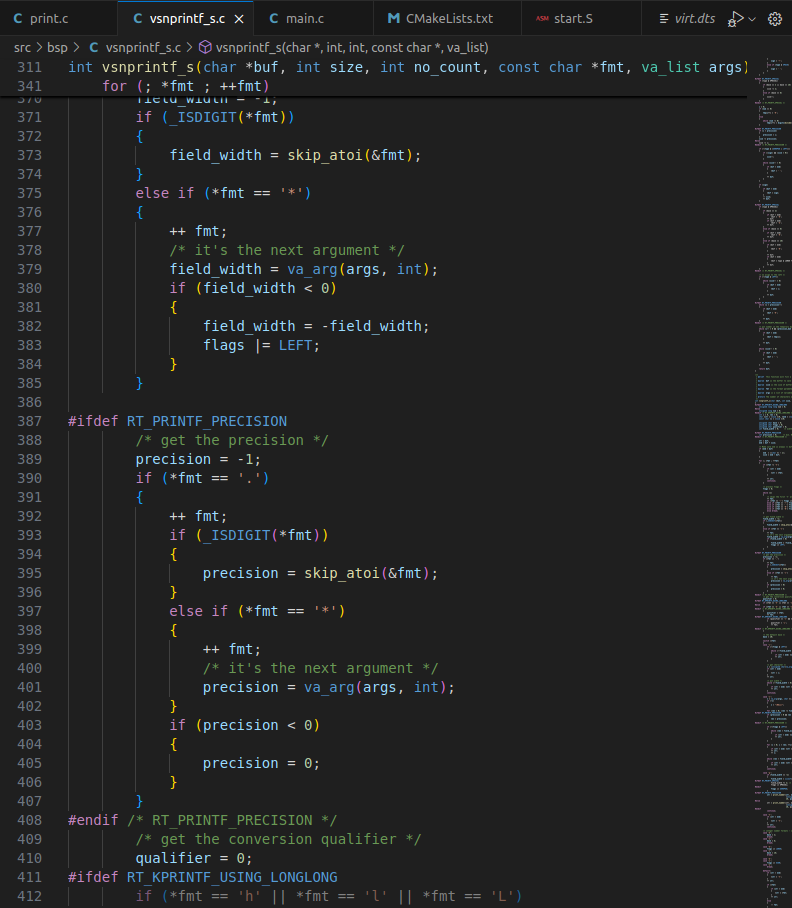


为了实现与 C 语言中 printf 函数类似的格式化功能，我们要用到可变参数列表 va\_list 。而真正实现格式化控制转换的函数是 vsnprintf\_s 函数。

实现 vsnprintf\_s 函数

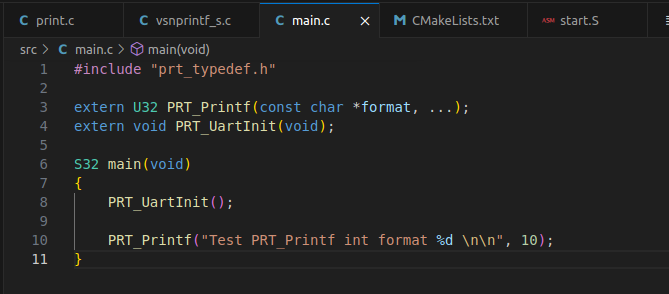
新建 src/bsp/vsnprintf\_s.c 实现 vsnprintf\_s 函数

vsnprintf\_s 函数的主要作用是依据格式控制符将可变参数列表转换成字符列表写入缓冲区。UniProton 提供了 libboundscheck 库，其中实现了 vsnprintf\_s 函数，作为可选作业你可以试着使用 libboundscheck 库中的 vsnprintf\_s 函数。简单起见，我们从另一个国产实时操作系统 RT-Thread 的 kservice.c 中引入了一个实现并进行了简单修改。



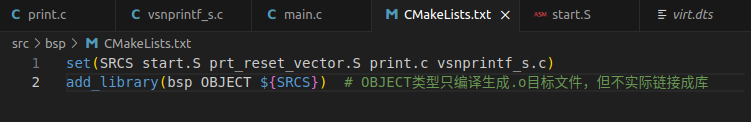
调用 PRT\_Printf 函数

main.c 修改为调用 PRT\_Printf 函数输出信息。



将新增文件纳入构建系统

修改 src/bsp/CMakeLists.txt 文件加入新增文件 print.c 和 vsnprintf\_s.c

s 启用 FPU

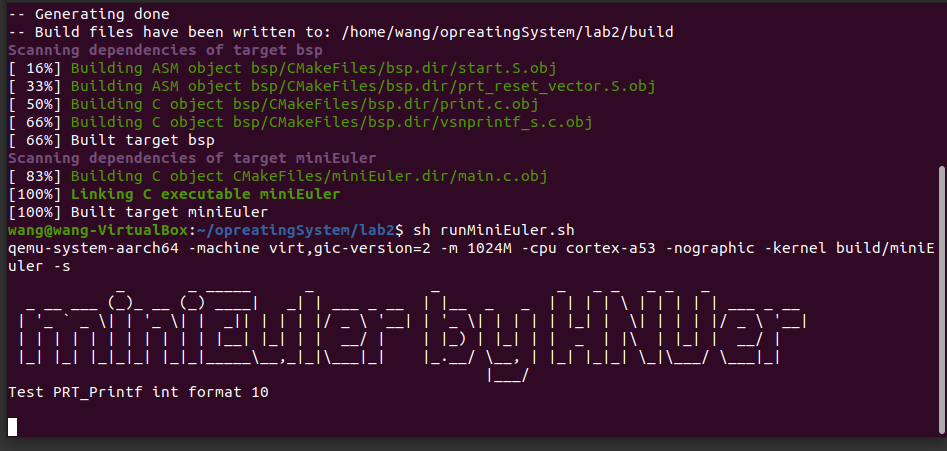
构建项目并执行发现程序没有任何输出。 需启用 FPU (src/bsp/start.S)。



**Hello, miniEuler**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab2/index.html#id11)

再次构建项目并执行，发现已可正常输出。至此，我们获得了一个基本的输出和调试手段，如我们可以在系统崩溃时调用 PRT\_Printf 函数进行输出。

我们可以利用 PRT\_Printf 函数来打印一个文本 banner 让我们写的 OS 显得专业一点😁。 [manytools.org](https://manytools.org/hacker-tools/ascii-banner/) 可以创建 ascii banner，选择你喜欢的样式和文字（下面选择的是 Standard 样式），然后在 main.c 的 main 函数中调用 PRT\_Printf 输出。

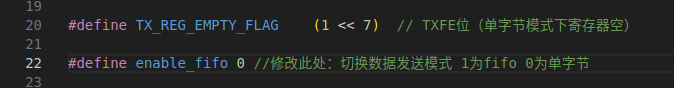


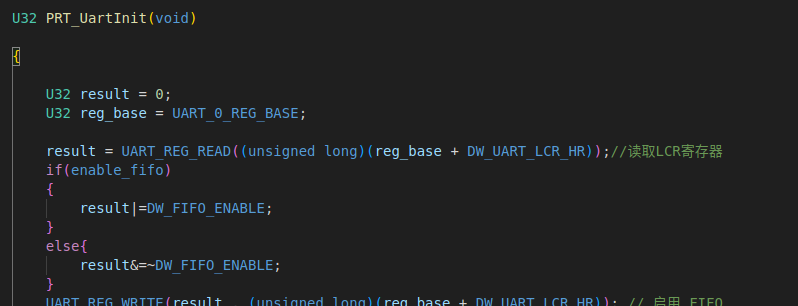
**作业1**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab2/index.html#id13)

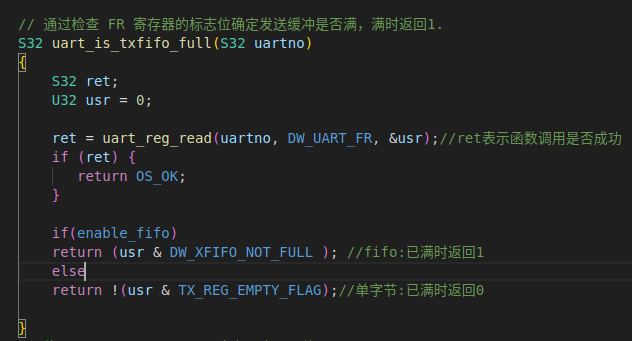
不启用 fifo，通过检测 UARTFR 寄存器的 TXFE 位来发送数据。

如果 ​**​FIFO 被禁用​**​，UART 会退化为 ​**​单字节传输模式​**​（即每个字节单独发送/接收），数据通过 ​**​数据寄存器（UARTDR）​**​ 直接传输

现在禁用FIFO换成检测UARTFR寄存器的TXEF位来进行判断。TXEF位是第8位，当处于空闲状态时，该位为1，有数据发送进寄存器时，该位为0







**作业2（可选）**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab2/index.html#id14)

采用 UniProton 提供的 libboundscheck 库实现 vsnprintf\_s 函数。