Lab0

实验准备

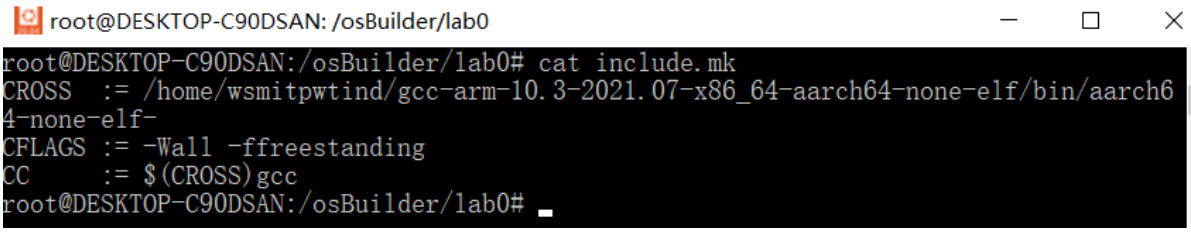
- 树莓派3B裸板
- USB-TTL模块
- win10电脑
- Ubuntu-20.04 LTS
- 树莓派Raspi OS启动盘
- Putty通信软件

实验流程

1. 进入root用户，编写实验代码
2. 使用GCC交叉编译器进行编译，得到内核镜像文件kernel8.img
3. 更改树莓派Raspi OS启动盘中配置文件
4. kernel8.img替换Raspi OS启动盘内核镜像文件
5. 将储存卡插回树莓派裸板，启动树莓派
6. 使用USB-TTL模块，连接电脑与树莓派对应管脚
7. 启动Putty，设置波特率为115200，与USB-TTL模块匹配
8. 在Putty中与树莓派通信进行实验

结果展示

配置交叉编译器路径如图所示，由于环境在Ubuntu20.04，使用x86_64，并进行相关配置



主函数编写如下：

![image-20220524211415573](C:\Users\zhangke\AppData\Roaming\Typora\typora-user-images\image-20220524211415573.png

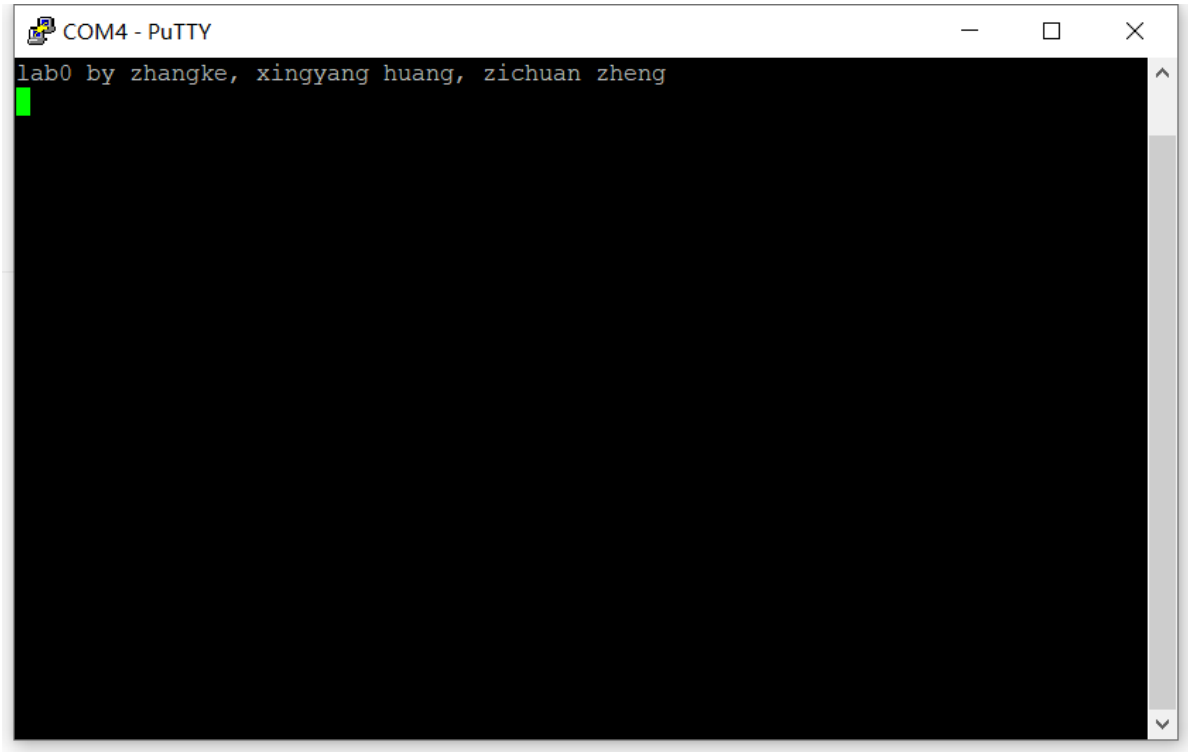
```c
#include <printf.h>
#include <drivers/include/uart.h>

void main() {
    uart_init();
    while (1) {
        char c = uart_getc();
        uart_send(c);
    }
}
```

功能逻辑为读入串口输入，并原样发回。成功编译结果如下：

```
root@DESKTOP-C90DSAN: /osBuilder/lab0
root@DESKTOP-C90DSAN:/osBuilder/lab0# ls
Makefile  drivers  include.mk  kernel.elf  kernel8.img
boot      include  init        kernel.lds  lib
root@DESKTOP-C90DSAN:/osBuilder/lab0# make
make --directory=boot
make[1]: Entering directory '/osBuilder/lab0/boot'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/osBuilder/lab0/boot'
make --directory=drivers
make[1]: Entering directory '/osBuilder/lab0/drivers'
/home/wsmitpwtind/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf/bin/aarch64-none-elf
-gcc -Wall -ffreestanding -I../include -I./include -c mbox.c
make[1]: Leaving directory '/osBuilder/lab0/drivers'
make --directory=init
make[1]: Entering directory '/osBuilder/lab0/init'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/osBuilder/lab0/init'
make --directory=lib
make[1]: Entering directory '/osBuilder/lab0/lib'
/home/wsmitpwtind/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf/bin/aarch64-none-elf
-gcc -Wall -ffreestanding -I./ -I../ -I../include/ -I../drivers/include -c printf.c
make[1]: Leaving directory '/osBuilder/lab0/lib'
/home/wsmitpwtind/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf/bin/aarch64-none-elf
-ld -o kernel.elf -e _start -Tkernel.lds boot/*.o init/*.o drivers/*.o lib/*.o
/home/wsmitpwtind/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf/bin/aarch64-none-elf
-objcopy kernel.elf -O binary kernel8.img
root@DESKTOP-C90DSAN:/osBuilder/lab0#
```

进行实验时的物理配件图如下：

串口通信结果如下，发现可以回传收到的字符：



Lab1

实验内容

- 编写printf函数

核心难点

本节中的架构，与课设中lab1部分区别不大，相同部分不再赘述。与lab1的不同之处，主要在于：

1. 映射到串口输出
2. 换行符处理

串口输出映射

与课设中的主要不同，是此处将printf函数的出口重定向到uart输出，核心代码位于lib/printf.c下。

```
static void myoutput(void *arg, char *s, int l)
{
    int i;

    // special termination call
    if ((l==1) && (s[0] == '\0')) return;

    for (i=0; i< l; i++) {
        uart_send(s[i]);
    }
}

void printf(char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    lp_Print(myoutput, 0, fmt, ap);
```

```
    va_end(ap);
}
```

虽然uart库中给出了uart相关的string等方法，此处为了保持与课设代码的一致性，统一选用了uart_send函数。

换行符处理

笔者团队的putty所在环境为window，而编译环境为linux，与课设中的纯linux环境不同，故而对于换行符的处理非常棘手。

系统	换行符
linux	/n，也叫LF
windows	/r/n，也叫CRLF

此外，课设中的printf函数不具备相应的换行符处理功能，只会原样输出/n。对于纯linux环境来说，不会产生异常。但笔者此处需要将/n替换为/r/n，以适应对window端的putty发送需求。

将原有lib/print.c下的lp_Print函数中，字符串处理部分，从：

```
while ( (*fmt != '\0') && (*fmt != '%')) {
                fmt++;
                }
```

替换为：

```
char r[2];
r[0] = '\r';
r[1] = '\0';
while ( (*fmt != '\0') && (*fmt != '%')) {
                if (*fmt == '\n') {
                        OUTPUT(arg,fmtStart,fmt - fmtStart);
                        fmtStart = fmt;
                        fmt++;
                        OUTPUT(arg,r,1);
                } else {
                        fmt++;
                }
            }
```

在读到/n时，提前输出，且额外输出/r。

结果展示

修改init/main.c为如下：

```
#include <printf.h>
#include <drivers/include/uart.h>

void main() {
    uart_init();
    printf("System started!\n");
```

```
    printf("This is lab1 for kezhang, xingyang huang, zichuan zheng\n");
    int test = 3;
    printf("Bock1<>\n %d %d",test ,test);
    while (1) {
        char c = uart_getc();
        printf("%c",c);
    }
}
```

测试输入字符串为"\nzhangke\nhuangxingyang\nzhengzichuan"，表现正确。