

# 《操作系统》

## 实验九报告

# 目录

|                     |    |
|---------------------|----|
| 1 实验代码分析 .....      | 3  |
| 1.1 实验目的 .....      | 3  |
| 1.2 代码结构 .....      | 3  |
| 1.3 shell 实现 .....  | 4  |
| 1.4 接收输入 .....      | 4  |
| 1.5 shell 处理 .....  | 5  |
| 1.6 main 函数修改 ..... | 5  |
| 1.7 项目构建 .....      | 6  |
| 1.8 执行结果 .....      | 7  |
| 2 实验任务 .....        | 9  |
| 2.1 作业 1 .....      | 9  |
| 2.1.1 总函数 .....     | 9  |
| 2.1.2 HNU .....     | 9  |
| 2.1.3 help .....    | 10 |
| 2.1.4 echo .....    | 10 |
| 2.1.5 错误输出 .....    | 11 |
| 2.1.6 top .....     | 11 |
| 2.1.7 tick .....    | 11 |

# 1 实验代码分析

## 1.1 实验目的

实现 shell。

## 1.2 代码结构



对应 lab8 的结构来看，新增了 prt\_shell.h 头文件，对先前 print.c、prt\_exc.c 修改以连接中断，修改 prt\_task.c 和 prt\_tick.c 实现展示功能，新增 shell 文件夹，src/shell/shmsg.c 文件处理 shell。

### 1.3 shell 实现

新建 src/include/prt\_shell.h 头文件，定义了两个宏和结构体 ShellCB。

### 1.4 接收输入

QEMU 的 virt 机器默认没有键盘作为输入设备，但当我们执行 QEMU 使用 **-nographic** 参数（**disable graphical output and redirect serial I/Os to console**）时 QEMU 会将串口重定向到控制台，因此我们可以使用 UART 作为输入设备。

向 src/bsp/print.c 中的 PRT\_UartInit 添加初始化代码，使其支持接收数据中断。同时增加定义了用于串口接收的信号量 sem\_uart\_rx。

```

39 #include <stdarg.h>
38 #include "prt_typedef.h"
37
36 #define UART_0_REG_BASE 0x09000000 // pl011 设备寄存器地址
35 // 寄存器及其位定义参见：https://developer.arm.com/documentation/ddi0183/g/programmers-model/summary-of-registers
34 #define DW_UART_THR 0x00 // UARTDR(Data Register) 寄存器
33 #define DW_UART_FR 0x18 // UARTFR(Flag Register) 寄存器
32 #define DW_UART_LCR_HR 0x2c // UARTLCR_H(Line Control Register) 寄存器
31 #define DW_XFIFO_NOT_FULL 0x020 // 发送缓冲区满置位
30 #define DW_FIFO_ENABLE 0x10 // 启用发送和接收FIFO
29
28 #define UART_BUSY_TIMEOUT 1000000
27 #define OS_MAX_SHOW_LEN 0x200
26
25
24 #define UART_REG_READ(addr) (*(volatile U32 *)(((uintptr_t)addr))) // 读设备寄存器
23 #define UART_REG_WRITE(value, addr) (*(volatile U32 *)(((uintptr_t)addr)) = (U32)value) // 写设备寄存器
22
21 U32 PRT_UartInit(void)
20 {
19     U32 result = 0;
18     U32 reg_base = UART_0_REG_BASE;
17     // LCR寄存器：https://developer.arm.com/documentation/ddi0183/g/programmers-model/register-descriptions/line-control-register--uartlcr-h?lang=en
16     result = UART_REG_READ((unsigned long)(reg_base + DW_UART_LCR_HR));
15     UART_REG_WRITE(result | DW_FIFO_ENABLE, (unsigned long)(reg_base + DW_UART_LCR_HR)); // 启用 FIFO
14
13     return OS_OK;
12 }
11 // 读 reg_base + offset 寄存器的值。uartno 参数未使用
10 S32 uart_reg_read(S32 uartno, U32 offset, U32 *val)
9 {
8     S32 ret;
7     U32 reg_base = UART_0_REG_BASE;
6
5
4     *val = UART_REG_READ((unsigned long)(reg_base + offset));
3     return OS_OK;
2 }
1

```

在原先的 print.c 的基础上增加代码，并覆盖原先的 PRT\_UartInit(void)函数。

src/bsp/print.c 中实现 OsUartRxHandle() 处理接收中断。

src/bsp/prt\_exc.c 中 OsHwiHandleActive() 链接 OsUartRxHandle()。增加中断号为 33 时的情况。

```
switch(irqNum){
    case 30:
        OsTickDispatcher();
        // PRT_Printf(".");
        break;
    case 33:
        OsUartRxHandle();
    default:
        break;
}
```

src/kernel/task/prt\_task.c 中加入函数 OsDisplayTasksInfo(void)，显示当前正在运行或挂起的任务的信息。

src/kernel/tick/prt\_tick.c 中加入函数 OsDisplayCurTick(void)，输出 PRT\_TickGetCount()。

## 1.5 shell 处理

新建 src/shell/shmsg.c 文件。

ShellTask 函数是 Shell 任务的主要执行体。它在一个无限循环中等待从 UART 接收数据（通过信号量 sem\_uart\_rx），并将接收到的字符存储在一个缓冲区（shellCB->shellBuf）中。然后，它将这些字符回显到控制台，并检查是否收到了一个换行符（\r）。如果收到了换行符，它将检查用户输入的命令并执行相应的操作（例如，显示任务信息或当前时钟滴答数）。

ShellTaskInit 用于初始化并启动 Shell 任务。它首先创建一个 TskInitParam 结构体实例 task1，并设置任务的入口函数、优先级、栈大小和参数。它将 shellCB 作为参数传递给任务。然后调用 PRT\_TaskCreate 来创建任务，并检查返回值以确保任务创建成功。如果成功，调用 PRT\_TaskResume 来启动。ShellTask 作为任务的入口函数。

## 1.6 main 函数修改

把所有设置、初始化全部打开，调用 Shell 初始化函数，打开任务调度。

```

1  #include "prt_typedef.h"
2  #include "prt_tick.h"
3  #include "prt_task.h"
4  #include "prt_sem.h"
5  #include "prt_shell.h"
6  extern U32 PRT_Printf(const char *format, ...);
7  extern void PRT_UartInit(void);
8  extern void CoreTimerInit(void);
9  extern U32 OsActivate(void);
10 extern U32 OsTskInit(void);
11 extern U32 OsSemInit(void);
12 extern U32 OsHwiInit(void);
13 extern U32 ShellTaskInit(ShellCB *shellCB);
14
15 static SemHandle sem_sync;
16 extern ShellCB g_shellCB;
17
18
19 S32 main(void)
20 {
21     // 初始化GIC
22     OsHwiInit();
23     // 启用Timer
24     CoreTimerInit();
25     // 任务模块初始化
26     OsTskInit();
27     OsSemInit(); // 参见demos/ascend310b/config/prt_config.c 系统初始化注册表
28
29     PRT_UartInit();
30
31
32     PRT_Printf("----- \n ");
33     PRT_Printf("***** \n");
34     PRT_Printf("---- \n");
35     PRT_Printf("***** \n ");
36     PRT_Printf("----- \n");
37     PRT_Printf("***** \n ");
38     PRT_Printf("---- \n");
39     PRT_Printf("***** \n");
40
41     PRT_Printf("ctr-a h: print help of qemu emulator. ctr-a x: quit emulator.\n\n");
42
43     U32 ret = ShellTaskInit(&g_shellCB);
44     if(ret){
45         return ret;
46     }
47
48     OsActivate();
49     return 0;
50 }

```

## 1.7 项目构建

总 src/CMakeLists.txt:

```

19 add_subdirectory(bsp)
20 add_subdirectory(kernel) # 增加 kernel 子目录
21 add_subdirectory(shell)
22
23 list(APPEND OBJS ${TARGET_OBJECTS:bsp} ${TARGET_OBJECTS:tick} ${TARGET_OBJECTS:task} ${TARGET_OBJECTS:sched}
24      ${TARGET_OBJECTS:sem} ${TARGET_OBJECTS:shell}) # 增加 ${TARGET_OBJECTS:tick}
25      目标
26 add_executable(${APP} main.c ${OBJS})

```

src/shell/CMakeLists.txt:

```

1  get(SRCS shmsg.c )
2  add_library(shell OBJECT ${SRCS}) # OBJECT类型只编译生成.o目标文件，但不实际链接成库

```

修改脚本中的命令，加入-nographic 参数。

```
1 #sh runMiniEuler.sh 直接运行
1 #sh runMiniEuler.sh -s 启动后在入口处暂停等待调试
2
3 echo qemu-system-aarch64 -machine virt,gic-version=2 -m 1024M -cpu cortex-a53 -nographic -kernel build/miniEuler -nographic -s $
4
5 qemu-system-aarch64 -machine virt,gic-version=2 -m 1024M -cpu cortex-a53 -nographic -kernel build/miniEuler -nographic -s $1
```

## 1.8 执行结果

本实验逻辑为：

新增了 prt\_shell.h 文件，定义宏和结构体。

对先前 print.c、prt\_exc.c 修改以连接中断。

修改 prt\_task.c 和 prt\_tick.c 实现展示功能。

新增 shell 文件夹，src/shell/shmsg.c 文件处理 shell。

需要对 main 函数进行修改。

```
guorulling@guorulling-virtual-machine: /lab9$ sh makeMiniEuler.sh
mkdir: cannot create directory 'build': File exists
-- The C compiler identification is GNU 11.2.1
-- The ASM compiler identification is GNU
-- Found assembler: /home/guorulling/aarch64-none-elf/gcc-arm-11.2-2022.02-x86_64-aarch64-none-elf/bin/aarch64-none-elf-gcc
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /home/guorulling/aarch64-none-elf/gcc-arm-11.2-2022.02-x86_64-aarch64-none-elf/bin/aarch64-none-elf-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/guorulling/lab9/build
[ 5%] Building C object bsp/CMakeFiles/shell.dir/shmsg.c.obj
[ 5%] Built target shell
Scanning dependencies of target bsp
[ 10%] Building ASM object bsp/CMakeFiles/bsp.dir/start.S.obj
[ 15%] Building ASM object bsp/CMakeFiles/bsp.dir/prt_reset_vector.S.obj
[ 20%] Building C object bsp/CMakeFiles/bsp.dir/print.c.obj
[ 25%] Building C object bsp/CMakeFiles/bsp.dir/vsnprintf.S.c.obj
[ 30%] Building C object bsp/CMakeFiles/bsp.dir/prt_exc.c.obj
[ 35%] Building ASM object bsp/CMakeFiles/bsp.dir/prt_vector.S.obj
[ 40%] Building C object bsp/CMakeFiles/bsp.dir/hwi_init.c.obj
[ 45%] Building C object bsp/CMakeFiles/bsp.dir/timer.c.obj
[ 50%] Building ASM object bsp/CMakeFiles/bsp.dir/cache_asm.S.obj
[ 55%] Building C object bsp/CMakeFiles/bsp.dir/mmu.c.obj
[ 55%] Built target bsp
[ 60%] Building C object kernel/tick/CMakeFiles/tick.dir/prt_tick.c.obj
[ 60%] Built target tick
[ 65%] Building C object kernel/task/CMakeFiles/task.dir/prt_task.c.obj
[ 70%] Building C object kernel/task/CMakeFiles/task.dir/prt_sys.c.obj
[ 75%] Building C object kernel/task/CMakeFiles/task.dir/prt_task_init.c.obj
[ 75%] Built target task
[ 80%] Building C object kernel/sched/CMakeFiles/sched.dir/prt_sched_single.c.obj
[ 80%] Built target sched
[ 85%] Building C object kernel/sen/CMakeFiles/sen.dir/prt_sen_init.c.obj
[ 90%] Building C object kernel/sen/CMakeFiles/sen.dir/prt_sen.c.obj
[ 90%] Built target sen
[ 95%] Building C object CMakeFiles/miniEuler.dir/main.c.obj
[100%] Linking C executable miniEuler
[100%] Built target miniEuler
```



# HNO I Love You

```
miniEuler # help
help - To show the meaning of the instruction.
top  - To display task information.
tick - To display clock.
echo - To give you an echo.
HNU  - I Love You.
```

```
miniEuler # QEMU: Terminated
```



## 2 实验任务

### 2.1 作业 1

题目：实现一条有用的 shell 指令。

全部实现指令：help, HNU, echo, top, tick, 当前指令错误。总效果见执行效果部分。

注意：只要是输出的，都需要输出前加\n,否则会覆盖掉当前行。

#### 2.1.1 总函数

```
if (ch == '\r'){
    // PRT_Printf("\n");
    if(cmd[0]=='t' && cmd[1]=='o' && cmd[2]=='p'){
        OsDisplayTasksInfo();
    } else if(cmd[0]=='t' && cmd[1]=='i' && cmd[2]=='c' && cmd[3]=='k'){
        OsDisplayCurTick();
    } else if(cmd[0]=='h' && cmd[1]=='e' && cmd[2]=='l' && cmd[3]=='p'){
        help();
    } else if(cmd[0]=='e' && cmd[1]=='c' && cmd[2]=='h' && cmd[3]=='o'){
        echo(cmd);
    } else if(cmd[0]=='H' && cmd[1]=='N' && cmd[2]=='U'){
        HNU();
    }
    else
    {
        PRT_Printf("\nCommand is not found.");
    }
    break;
}
```

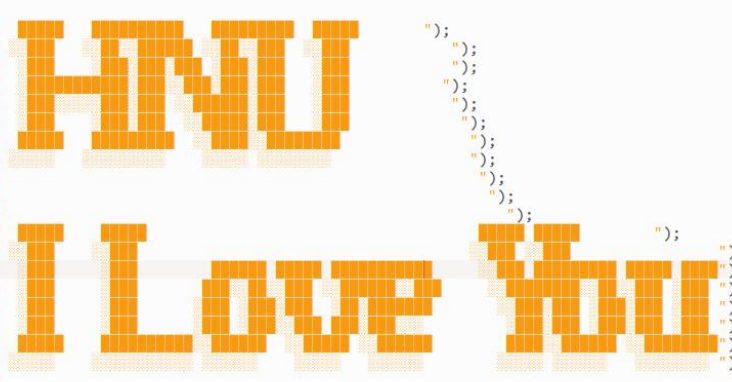
#### 2.1.2 HNU



```

29 void HNU()
30 {
31     PRT_Printf("\n");
32     PRT_Printf("\n");
33     PRT_Printf("\n");
34     PRT_Printf("\n");
35     PRT_Printf("\n");
36     PRT_Printf("\n");
37     PRT_Printf("\n");
38     PRT_Printf("\n");
39     PRT_Printf("\n");
40     PRT_Printf("\n");
41     PRT_Printf("\n");
42     PRT_Printf("\n");
43     PRT_Printf("\n");
44     PRT_Printf("\n");
45     PRT_Printf("\n");
46     PRT_Printf("\n");
47     PRT_Printf("\n");
48     PRT_Printf("\n");
49     PRT_Printf("\n");
50
51 }

```



### 2.1.3 help

```

miniEuler # help
help - To show the meaning of ihe instruction.
top  - To display task information.
tick - To display clock.
echo - To give you an echo.
HNU  - I Love You.

```

```

13 void help()
14 {
15     PRT_Printf("\n help - To show the meaning of ihe instruction.");
16     PRT_Printf("\n top  - To display task information.");
17     PRT_Printf("\n tick - To display clock.");
18     PRT_Printf("\n echo - To give you an echo.");
19     PRT_Printf("\n HNU  - I Love You.");
20 }

```

### 2.1.4 echo

```

miniEuler # echo hello
hello
miniEuler # echo !?@#
!?!@#
miniEuler # echo "help"
"help"

```

```

21 void echo(char * str)
22 {
23     PRT_Printf("\n");
24     for(int i = 5; i < SHELL_SHOW_MAX_LEN; i++)
25     {
26         PRT_Printf("%c", str[i]);
27     }
28 }

```

### 2.1.5 错误输出

```
miniEuler # something
Command is not found.
```

```
else
{
    PRT_Printf("\nCommand is not found.");
}
```

### 2.1.6 top

```
miniEuler # top
PID          Priority      Stack Size
1             63           4096
0             9           4096
Total 2 tasks
```

```
16 OS_SEC_TEXT void OsDisplayTasksInfo(void)
17 {
18     struct TagTskCb *taskCb = NULL;
19     U32 cnt = 0;
20
21     PRT_Printf("\nPID\tPriority\tStack Size\n");
22     // 遍历g_runQueue队列, 查找优先级最高的任务
23     LIST_FOR_EACH(taskCb, &g_runQueue, struct TagTskCb, pendList) {
24         cnt++;
25         PRT_Printf("%d\t%d\t%d\n", taskCb->taskId, taskCb->priority, taskCb->stackSize);
26     }
27     PRT_Printf("Total %d tasks", cnt);
28
29 }
```

### 2.1.7 tick

```
miniEuler # tick
Current Tick: 6737
miniEuler # tick
Current Tick: 8631
```

```
32 OS_SEC_TEXT void OsDisplayCurTick(void)
33 {
34     PRT_Printf("\nCurrent Tick: %d", PRT_TickGetCount());
35 }
```