

題目一

將檔案分類放好後，我開始改寫 Makefile

因為把這個資料夾獨立拉出來，所以把之前 include 的上層 common 內容直接貼過來

```
CROSS_COMPILE = riscv64-unknown-elf-
CFLAGS = -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g
QEMU = qemu-system-riscv32
QFLAGS = -nographic -smp 1 -machine virt -bios none
GDB = riscv64-unknown-elf-gdb
CC = ${CROSS_COMPILE}gcc
OBJCOPY = ${CROSS_COMPILE}objcopy
OBJDUMP = ${CROSS_COMPILE}objdump
```

把最後的編譯目標修改為 objs 資料夾底下的.o 檔

```
OBJS = \
    ./objs/start.o \
    ./objs/mem.o \
    ./objs/entry.o \
    ./objs/kernel.o \
    ./objs/uart.o \
    ./objs/printf.o \
    ./objs/page.o \
    ./objs/sched.o \
```

分別設定各個資料夾下的目標與依賴關係

```
os.elf: ${OBJS}
    ${CC} ${CFLAGS} -T os.ld -o os.elf $^
    ${OBJCOPY} -O binary os.elf os.bin

./objs/%.o : ./startup/%.s
    $(CC) $(CFLAGS) -c -o $@ $<

./objs/%.o : ./asm/%.s
    $(CC) $(CFLAGS) -c -o $@ $<

./objs/%.o : ./kernel/%.c
    $(CC) $(CFLAGS) -c -o $@ $<

./objs/%.o : ./lib/%.c
    $(CC) $(CFLAGS) -c -o $@ $<
```

將 `objs/*.o` 加到 `clean` 偽目標中

```
clean:
```

```
rm -rf *.o *.bin *.elf objs/*.o
```

編譯看看

```
lyc1ih@DESKTOP-CR5MUFU MINGW64 ~/Desktop/github/Operating-System-Engineering/第三週作業/題目一/03-contextswitch (main)
$ make
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/start.o startup/start.s
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/mem.o asm/mem.s
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/entry.o asm/entry.s
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/kernel.o kernel/kernel.c
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/uart.o kernel/uart.c
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/printf.o lib/printf.c
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/page.o kernel/page.c
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -c -o objs/sched.o kernel/sched.c
riscv64-unknown-elf-gcc -nostdlib -fno-builtin -march=rv32ima -mabi=ilp32 -g -T os.ld -o os.elf objs/start.o objs/mem.o objs/entry.o objs/kernel.o objs/uart.o objs/printf.o objs/page.o objs/sched.o
riscv64-unknown-elf-objcopy -O binary os.elf os.bin
```

成功

測試看看是否能運作



[illegible]

順利運行

題目二

1. 請說明 `beqz t6, 1f` 指令會做何事?

我檢查了教材中的各個檔案，最後找到一份名為 `as.pdf` 的文件，這是一份介紹組譯器指令的說明書

FreedomStudio-2020-06-3-win64 > SiFive > riscv64-unknown-elf-gcc-8.3.0-2020.04.1 > share > doc				
				
porting.html	2022/9/22 下午 09:16	檔案資料夾		
stabs	2022/9/22 下午 09:16	檔案資料夾		
PDF annotate	2020/7/13 下午 10:05	Microsoft Edge P...	200 KB	
 PDF as	2020/7/13 下午 05:53	Microsoft Edge P...	1,393 KB	
PDF bfd	2020/7/13 下午 10:05	Microsoft Edge P...	609 KB	
PDF binutils	2020/7/13 下午 05:53	Microsoft Edge P...	431 KB	

在裡面我找到了以下敘述

Local Labels

Local labels are different from local symbols. Local labels help compilers and programmers use names temporarily. They create symbols which are guaranteed to be unique over the entire scope of the input source code and which can be referred to by a simple notation. To define a local label, write a label of the form `'N:'` (where `N` represents any non-negative integer). To refer to the most recent previous definition of that label write `'Nb'`, using the same number as when you defined the label. To refer to the next definition of a local label, write `'Nf'`. The `'b'` stands for "backwards" and the `'f'` stands for "forwards".

There is no restriction on how you can use these labels, and you can reuse them too. So that it is possible to repeatedly define the same local label (using the same number `'N'`), although you can only refer to the most recently defined local label of that number (for a backwards reference) or the next definition of a specific local label for a forward reference. It is also worth noting that the first 10 local labels (`'0:'. . . '9:'`) are implemented in a slightly more efficient manner than the others.

Here is an example:

```
1:      branch 1f
2:      branch 1b
1:      branch 2f
2:      branch 1b
```

Which is the equivalent of:

```
label_1: branch label_3
label_2: branch label_1
label_3: branch label_4
label_4: branch label_3
```

簡單來說，這是一個可以在 `label` 之間跳躍的指令，數字代表 `label` 名，`<f>` 代表 `forward`，`` 代表 `back`，所以 `beqz t6, 1f` 的意思就是判斷如果 `t6` 為 0，則往前跳到下一個名為 `1:` 的標籤

```
switch_to:
  csrrw t6, mscratch, t6    # swap t6 and mscratch
  beqz t6, 1f                # Notice: previous task may be NULL
  reg_save t6                # save context of prev task

  # Save the actual t6 register, which we swapped into
  # mscratch
  mv t5, t6                 # t5 points to the context of current task
  csrr t6, mscratch         # read t6 back from mscratch
  sw t6, 120(t5)            # save t6 with t5 as base

  1:                          # switch mscratch to point to the context of the next task
```

2. 請修改 entry.S 裡的 switch_to 函數，修改成 kernel start 後 switch 到第一個 Task 用
並在 entry.S 裡增加一個 sys_switch 函數，主要讓 kernel start 以後 contextSwitch 用

```
# void sys_switch(struct context *old, struct context *new);  
# a0: pointer to the context of old task  
# a1: pointer to the context of the new task
```

為了能模擬從 kernel 跳到 task，以及 task 跳到另一個 task 的狀況，我製作了兩個 task

```
//為兩個 task 產生 context，和他們各自的 stack
```

```
struct context ctx_task1;  
struct context ctx_task2;  
uint8_t task1_stack[STACK_SIZE];  
uint8_t task2_stack[STACK_SIZE];
```

```
//第一個 task
```

```
void user_task1(void)  
{  
    uart_puts("Task 1: Running...\n");  
    uart_puts("                |...\n");  
    uart_puts("                |...\n");  
    uart_puts("                |...\n");  
    task_delay(6000);  
    sys_switch(&ctx_task1, &ctx_task2);  
}
```

```
//第二個 task
```

```
void user_task2(void)  
{  
    uart_puts("Task 2: Running.....|.....\n");  
    uart_puts("                ...|\n");  
    uart_puts("                ...|\n");  
    uart_puts("                ...|\n");  
    task_delay(6000);  
    sys_switch(&ctx_task2, &ctx_task1);  
}
```

```
//將兩個 task 初始化
```

```
void schedule_init()  
{  
    ctx_task1.sp = (reg_t) &task1_stack[STACK_SIZE-1];  
    ctx_task1.ra = (reg_t) user_task1;  
    ctx_task2.sp = (reg_t) &task2_stack[STACK_SIZE-1];  
    ctx_task2.ra = (reg_t) user_task2;  
}
```

接著修改組合語言

```
#-----從 kernel 跳到 task 的程式-----
.globl switch_to
.align 4
switch_to:                # a0 代表第一個參數，是指向第一個 task 的 context 的指針
    mv t6, a0             #把 a0 寫入 t6，用來當作載入第一個 task 的 context 的基址
    reg_restore t6        #利用 t6 當基址載入第一個 task 的 context
    ret                   #用 ra 覆蓋 PC，跳到 task 的起始位置開始執行

#-----從 task 跳到 task 的程式-----
.globl sys_switch
.align 4
sys_switch:
    csrrw t6, mscratch, t6 #將 t6 與 mscratch 做交換，暫時存放 t6 的值
    mv t6, a0              #把 a0 參數寫入 t6
    reg_save t6            #利用 t6 中的地址儲當前的暫存器內容到自己的 context
    mv t5, t6              #因為 t5 已經存過了，複寫沒關係，把 t6 中的地址寫到 t5 中
    csrr t6, mscratch      #把 t6 跟 mscratch 交換前的值換回來
    sw t6, 120(t5)         #利用 t5 中的地址儲存換回來的 t6 到自己 context 的 t6 位置

    #以上完成存 context 的動作，下一階段開始載入目標 task 的 context

    # a1 代表第二個參數，是指向下一個 task 的 context 的指針
    mv t6, a1              #把 a1 寫入 t6
    reg_restore t6         #用 t6 中的地址載入下一個 task 的 context
    ret                   #用 ra 覆蓋 PC，跳到下一個 task 的起始位置開始執行
.end
```

將初始化及測試的 task 帶入 os 中執行

```
printf("\n\n\n---- schedule_init ----\n");

schedule_init();
printf("\n\n\n---- schedule start ----\n\n\n");
schedule();
```

測試成功

```
free 8001c007 -> 8001c00a
free 8001c00a -> 8001c00c
free 8001c00c -> 8001c013
free 8001c013 -> 8001c01b
```

```
----- malloc test success -----
```

```
----- schedule_init -----
```

```
----- schedule start -----
```

```
Task 1: Running...|
                  |...
                  |...
                  |...
Task 2: Running.....|.....
                  |...
                  |...
                  |...
Task 1: Running...|
                  |...
                  |...
                  |...
Task 2: Running.....|.....
                  |...
                  |...
                  |...
Task 1: Running...|
                  |...
                  |...
                  |...
Task 2: Running.....|.....
                  |...
```