

Lab5 实验报告

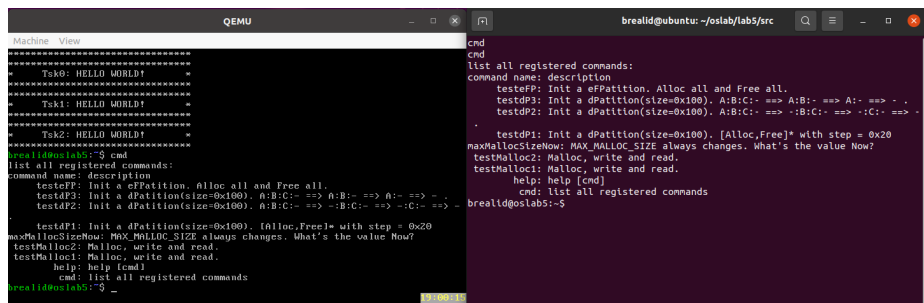
PB21000033 赵奕

本次报告省略了大部分说明，仅保留文档要求的思考题与运行结果

思考题

1. 在上下文切换的现场维护中，pushf 和 popf 对应，pusha 和 popa 对应，call 和 ret 对应，但是为什么 CTS SW 函数中只有 ret 而没有 call 呢？
> 这段代码本身是作为实现上下文切换的代码片段嵌入的，所以不需要使用 call 指令调用
2. 谈一谈你对 stack_init 函数的理解。
> stack_init() 函数用于初始化栈空间。
> unsigned long **stk: stk 为一个二级指针，通过 stk 可以修改指向栈顶的指针的值。
> void (*task)(void): task 为一个指向函数的指针，此处指向进程的任务函数。
> 这段代码向下移动堆栈指针，在每个遍历的位置上设置不同的值来初始化堆栈，而这些值对应寄存器的初始值。
3. myTCB 结构体定义中的 stack[STACK_SIZE] 的作用是什么？BspContextBase[STACK_SIZE] 的作用又是什么？
> 在 myTCB 结构体的定义中，stack[STACK_SIZE] 数组用于为每个任务分配栈空间。每个任务都需要一个栈来存储其执行上下文，包括寄存器的值、局部变量和函数调用信息。stack 数组用于保留这个空间。
> BspContextBase[STACK_SIZE] 数组的作用是为启动多任务调度模式准备一个基本的上下文。在任务切换时，需要保存当前任务的执行上下文，并加载下一个任务的执行上下文。BspContextBase 数组用于保存 BSP 的上下文信息，包括寄存器的值、堆栈指针等。由于 BSP 的上下文信息在任务切换时不会改变，所以只需要保存一次即可，而每个任务的上下文信息在任务切换时会改变，所以需要为每个任务分配一个栈空间。
4. prevTSK StackPtr 是一级指针还是二级指针？为什么？
> prevTSK StackPtr 是一个二级指针，它指向保存前一个任务的栈指针的地址。
> 使用这个二级指针，我们可以修改指针所指向的地址，以便在任务切换时将下一个任务的栈指针地址赋予它，从而可以在切换回该任务时正确恢复栈的状态。> 如果使用一级指针，那么我们只能修改指针所指向的值，而不能修改指针本身的值，这样就无法在任务切换时正确恢复栈的状态。

运行结果



```
Machine View
=====
Task0: HELLO WORLD!
=====
Task1: HELLO WORLD!
=====
Task2: HELLO WORLD!
=====
breall@oslab5:~$ cd
list all registered commands:
command name: description
testcPP: Init a cFPartition. Alloc all and free all.
testdP3: Init a dFPartition(size=0x100). A:B:C:- ==> A:B:- ==> A:- ==> - .
testdP2: Init a dFPartition(size=0x100). A:B:C:- ==> -B:C:- ==> -C:- ==> -
testdP1: Init a dFPartition(size=0x100). [alloc,free]* with step = 0x20
maxMallocSizeNow: MAX_MALLOC_SIZE always changes. What's the value Now?
testMalloc2: Malloc, write and read.
testMalloc1: Malloc, write and read.
help: help [cmd]
cmd: list all registered commands
breall@oslab5:~$
```

Figure 1: 运行结果