Build a Minimal Operating System in ARM Cortex-M3

Mini-arm-os

- A minimal multi-tasking OS kernel for ARM from scratch
- From simple to deep, mini-arm-os is a good tutorial to get involved in self-build operating system.
- Including Hello World, Context Switch, Multi-Tasking, Timer Interrupt, Preemptive and Thread.

Let's see how easy an OS could be...

Helloworld

Startup

15 lines (12 sloc) | 0.238 kB

```
1  #include <stdint.h>
2
3  extern void main(void);
4  void reset_handler(void)
5  {
6          /* jump to C entry point */
7          main();
8  }
9
10 __attribute((section(".isr_vector")))
11 uint32_t *isr_vectors[] = {
12         0,
13         (uint32_t *) reset_handler,     /* code entry
14 };
```

Main

```
1  #include <stdint.h>
2  #include "reg.h"
3
4  #define USART_FLAG_TXE  ((uint16_t) 0x0080)
5
6  int puts(const char *str)
7  {
8          while (*str) {
9                  while (!(*(USART2_SR) & USART_FLAG_TXE));
10                 *(USART2_DR) = *str++ & 0xFF;
11         }
12         return 0;
13 }
14
15 void main(void)
16 {
17         *(RCC_APB2ENR) |= (uint32_t) (0x00000001 | 0x00000004);
18         *(RCC_APB1ENR) |= (uint32_t) (0x00020000);
19
20         /* USART2 Configuration */
21         *(GPIOA_CRL) = 0x00004B00;
22         *(GPIOA_CRH) = 0x44444444;
23
24         *(USART2_CR1) = 0x0000000C;
25         *(USART2_CR1) |= 0x2000;
26
27         puts("Hello World!\n");
28
29         while (1);
30 }
```

From Hello World to Multi-thread

Language	files	blank	comment	code
C	2	9	2	33
make	1	5	0	18
C/C++ Header	1	5	3	15
SUM:	4	19	5	66

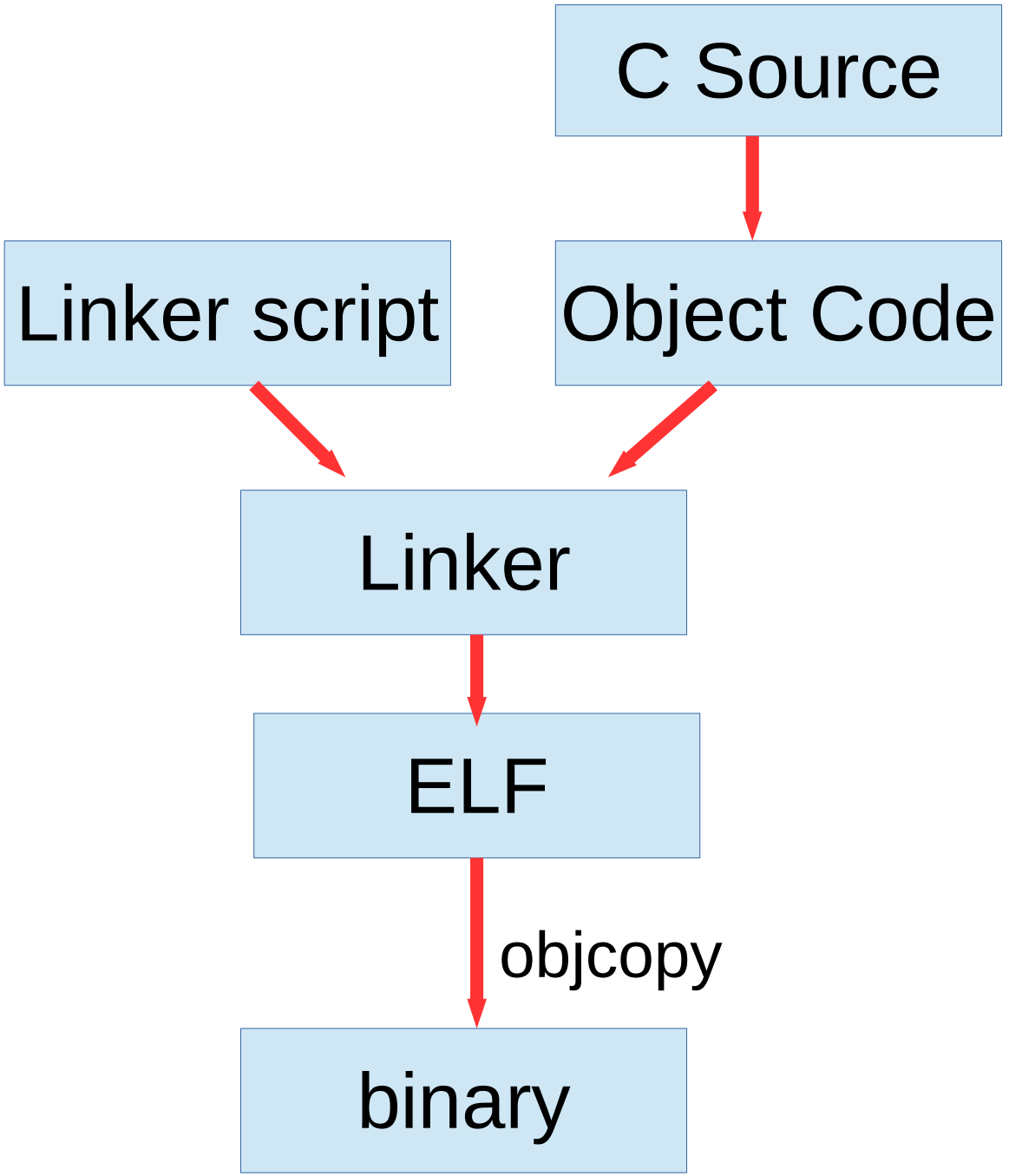
Less than 70 lines!!

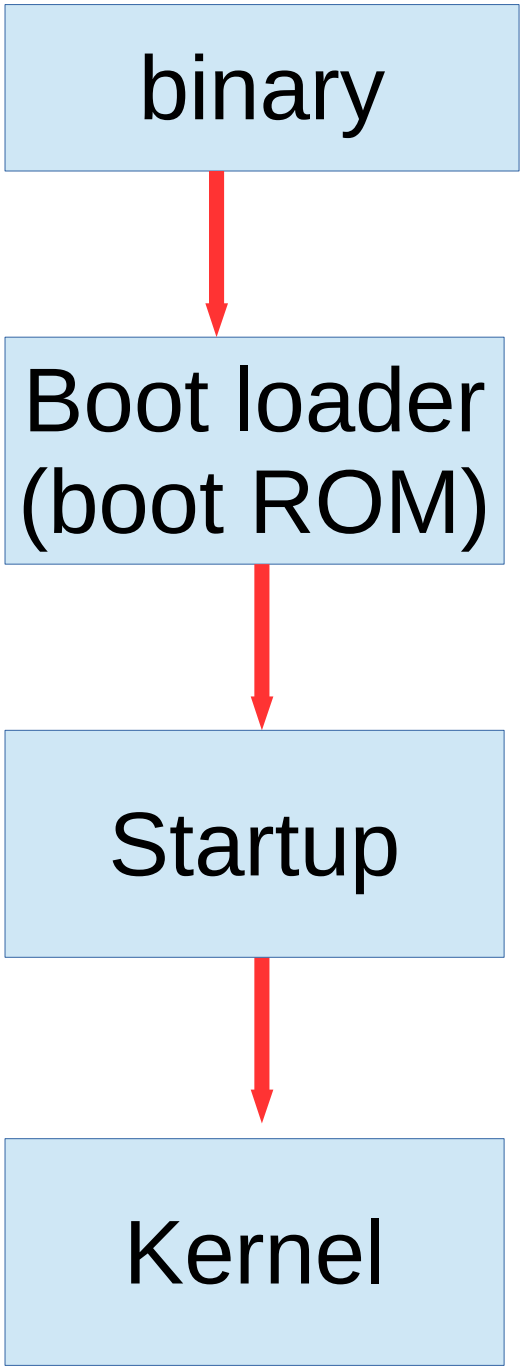
Even a multi-threading:

Language	files	blank	comment	code
C	4	84	59	339
C/C++ Header	4	15	5	61
make	1	3	0	21
SUM:	9	102	64	421

Also less than 500 lines

Helloworld


C Source
Linker script
Object Code
Linker
ELF
objcopy
binary


binary
Boot loader
(boot ROM)
Startup
Kernel

Linker script

- Describe how the sections in the input files should be mapped into the output file, and to control the memory layout of the output file
- We bind entry point address in hardware specific address
- So boot loader can know where to enter the startup

```
ENTRY(reset_handler)

MEMORY
{
  FLASH (rx) :
    ORIGIN = 0x00000000,
    LENGTH = 128K
}

SECTIONS
{
  .text :
  {
      KEEP(*(.isr_vector))
      *(.text)
  } >FLASH
}
```

Interrupt vector table

- A table of interrupt vectors (IVT) that associates an interrupt handler with an interrupt request in a machine specific way.
- Each entry of the IVT is the address of an interrupt service routine (ISR).
- In ARM, It contains the entry point, initial stack pointer and different kinds of exception handlers

```
__attribute((section(".isr_vector")))
uint32_t *isr_vectors[] = {
    (uint32_t *) &_estack,          /* stack pointer */
    (uint32_t *) reset_handler,   /* code entry point */
    (uint32_t *) nmi_handler,     /* NMI handler */
    (uint32_t *) hardfault_handler   /* hard fault handler */
};
```

Hello world

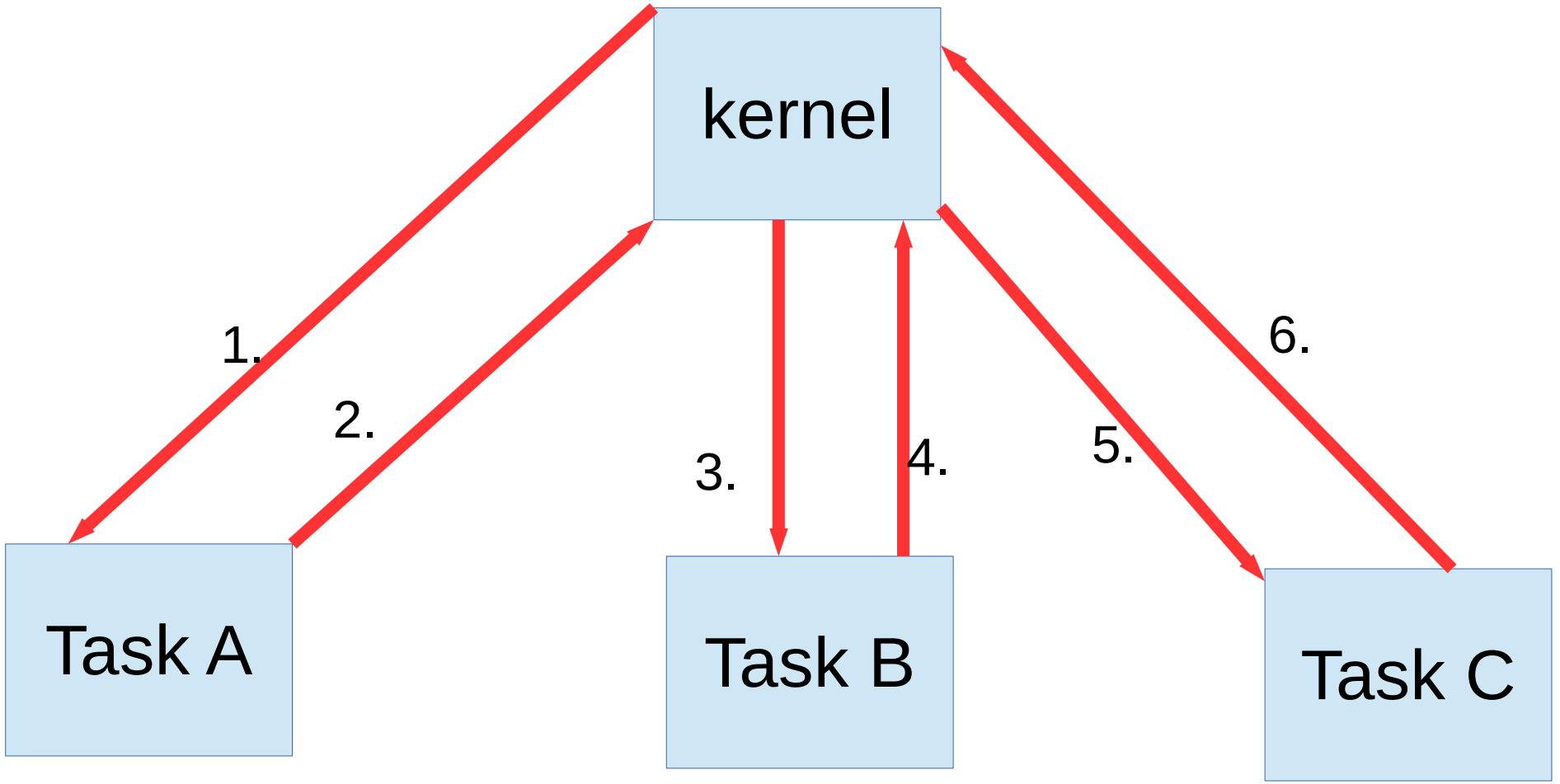
Now we've set memory mapping and had an entry point

So now you can:

- Initial everything your own system and hardware need.

- And do what you want in your kernel!!

Multi-Tasking



From Kernel to Task : activate

From Task to Kernel : SVC

privilege levels

- Unprivileged :
  - has limited access to some instructions (MRS, etc.)
  - cannot access the system timer, NVIC, or system control block
  - might have restricted access to memory or peripherals.
- Privileged :
  - Full access

Processor mode

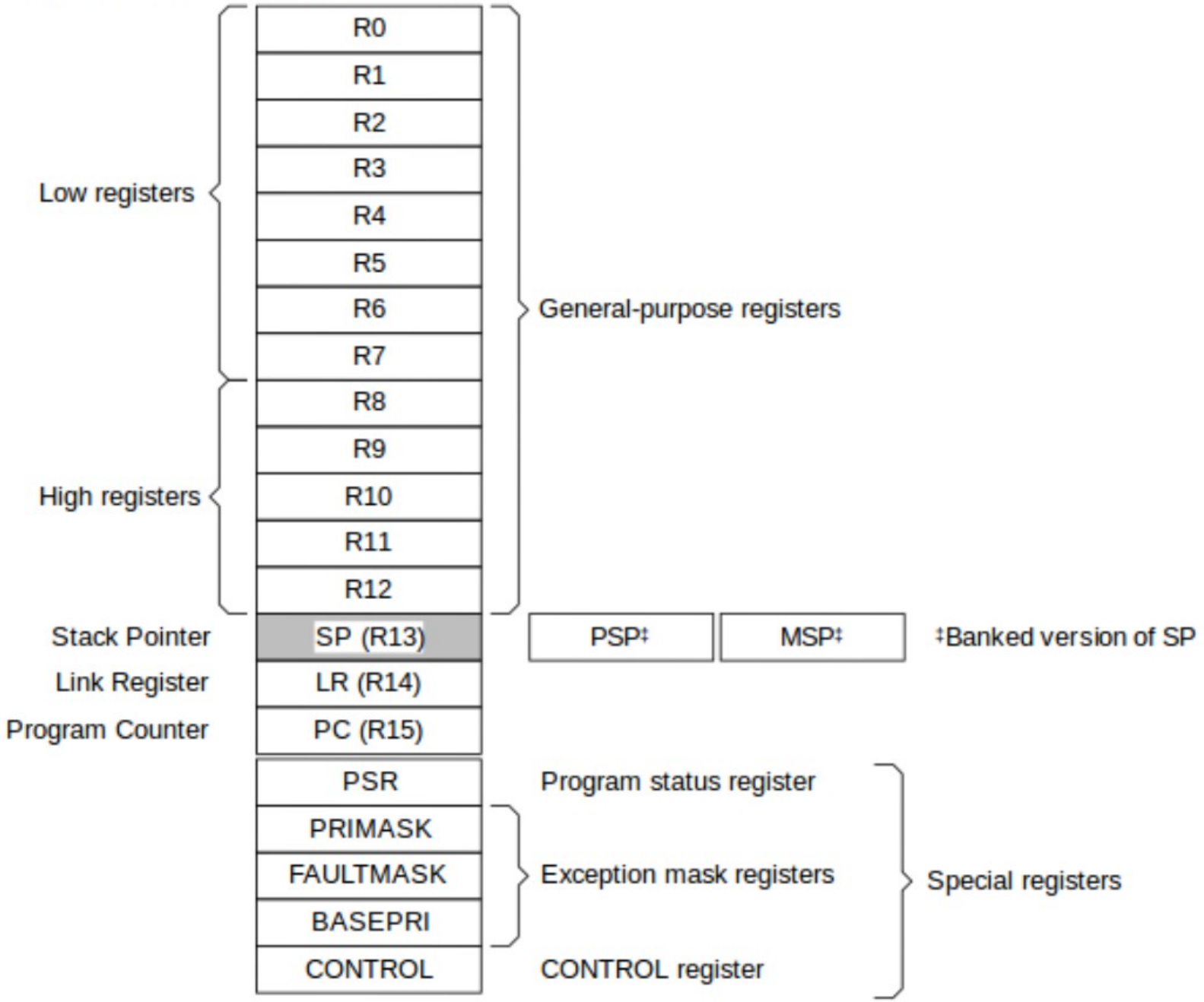
- Thread mode
  - The processor enters Thread mode when it comes out of reset
  - Used to execute application software
  - Privilege or Unprivileged
- Handler mode
  - Used to handle exceptions
  - Only enter when exception happen
  - Privilege

Stack pointer

- MSP ： Main Stack Pointer
  - When program start, It's the reset value
- PSP ： Process Stack Pointer
  - Each task can have it's own stack pointer, it's useful for multi-threading

Core Register

The processor core registers are:

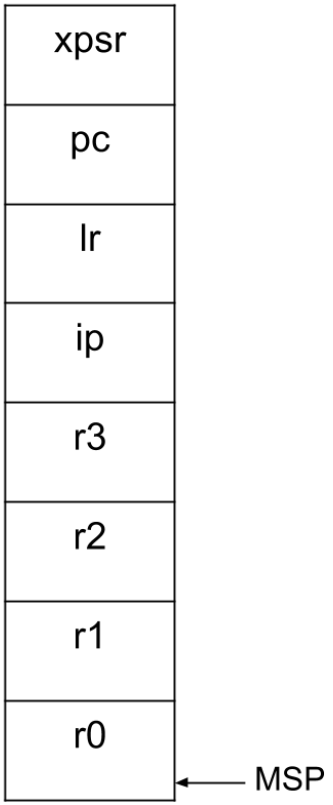


Context-switch

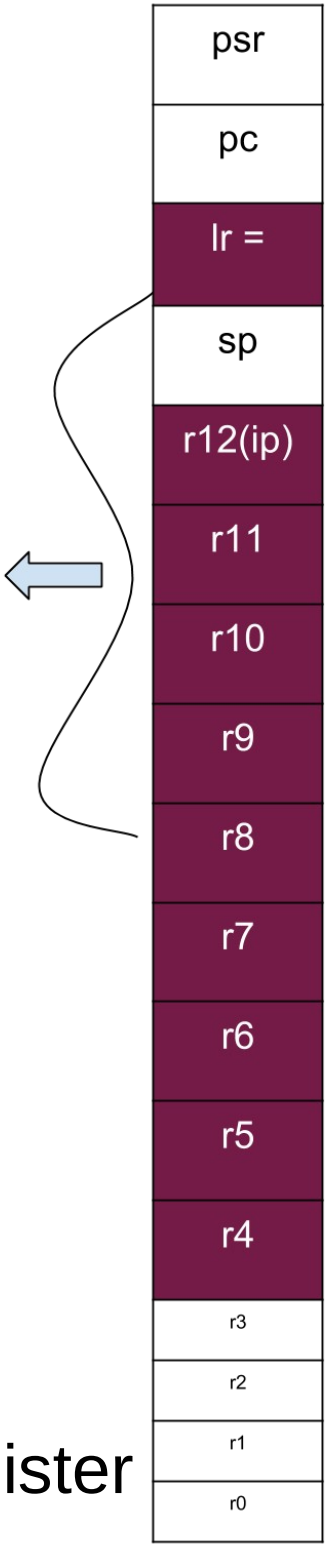
kernel

activate

Task



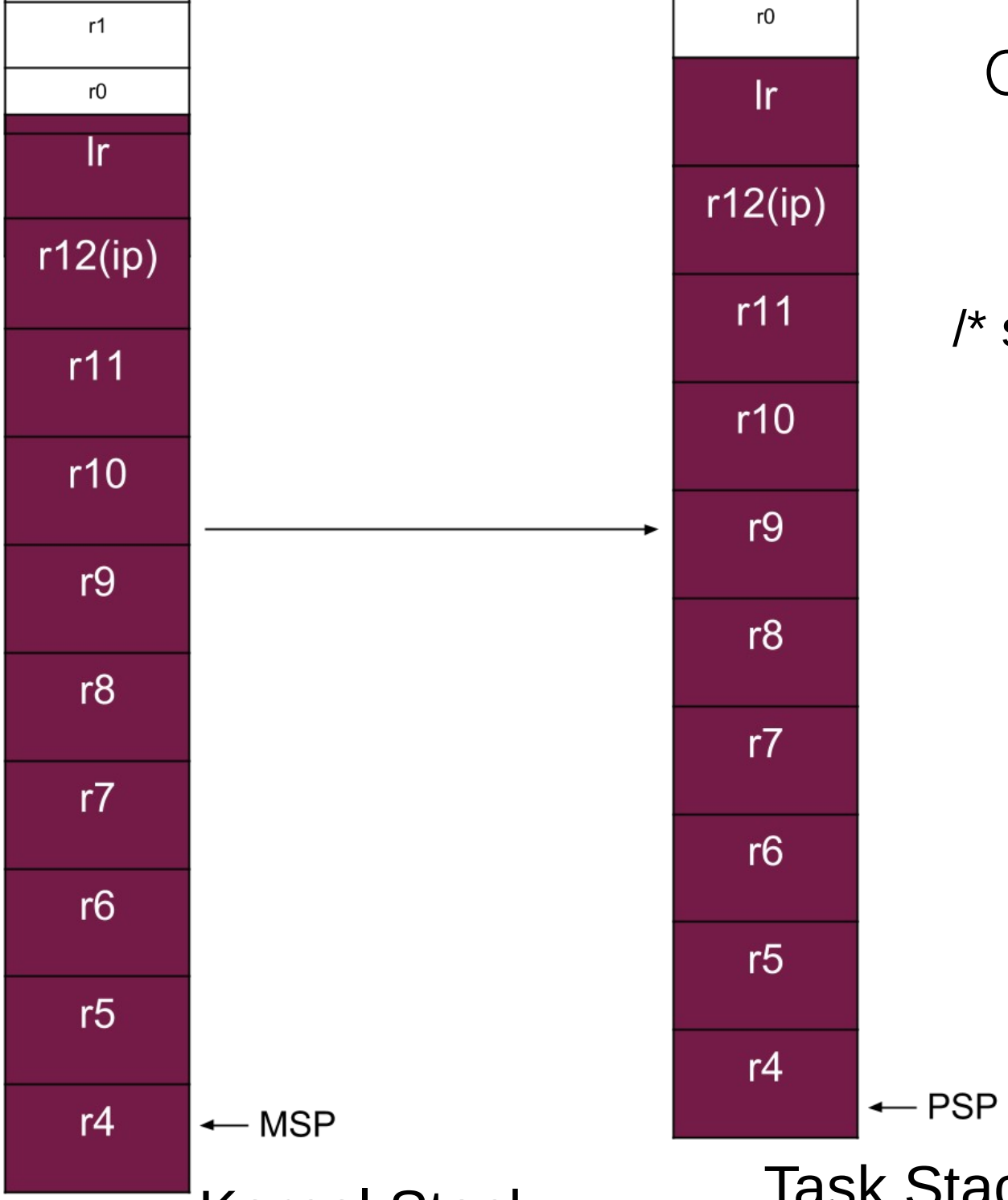
Kernel Stack



Core register

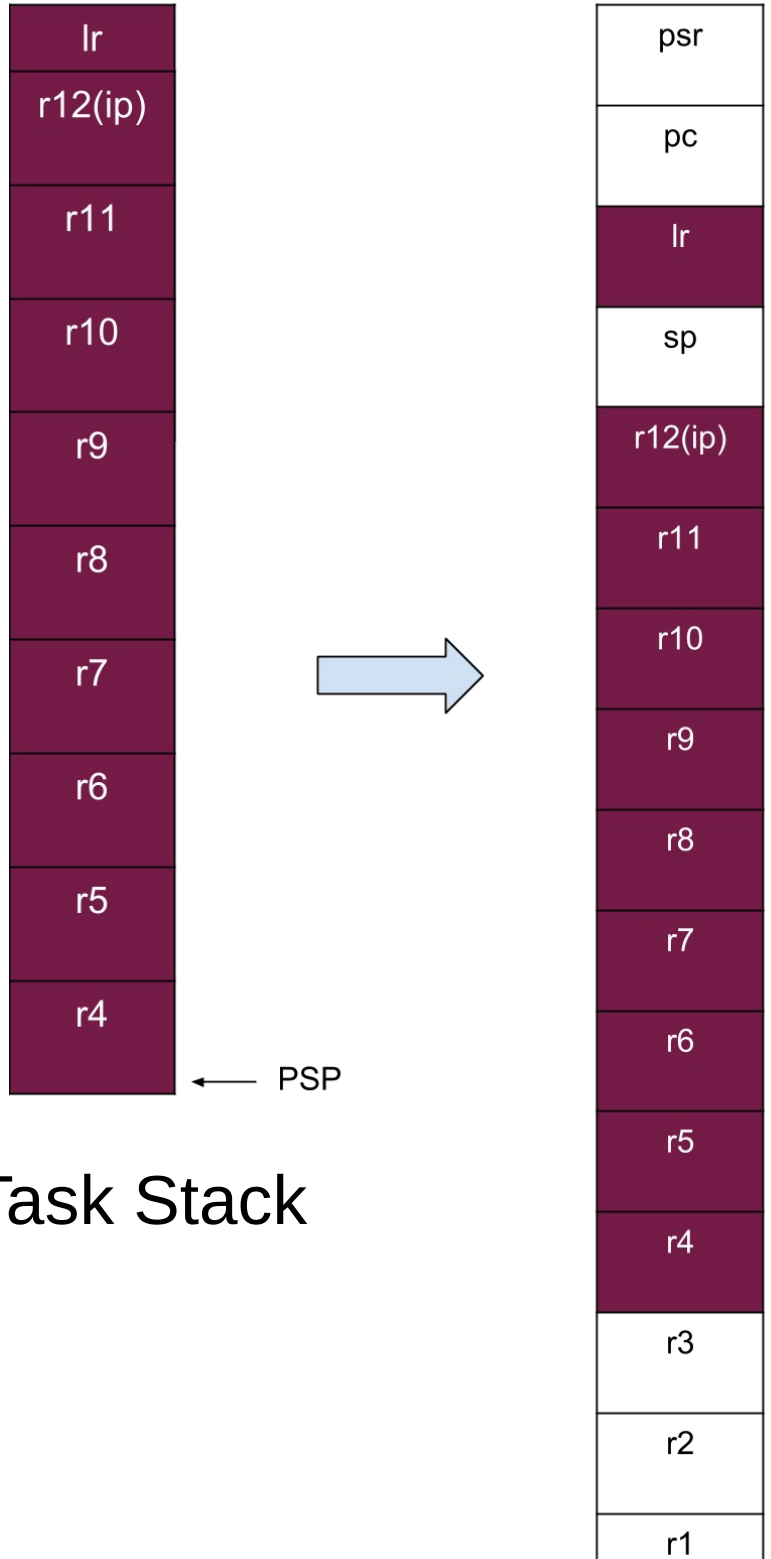
```
activate:
    /* save kernel state */
    mrs ip, psr
    push {r4, r5, r6, r7, r8, r9, r10, r11, ip, lr}
```

Context-switch




```
/* switch to process stack */
    msr psp, r0
    mov r0, #3
    msr control, r0
    isb
```

Context-switch



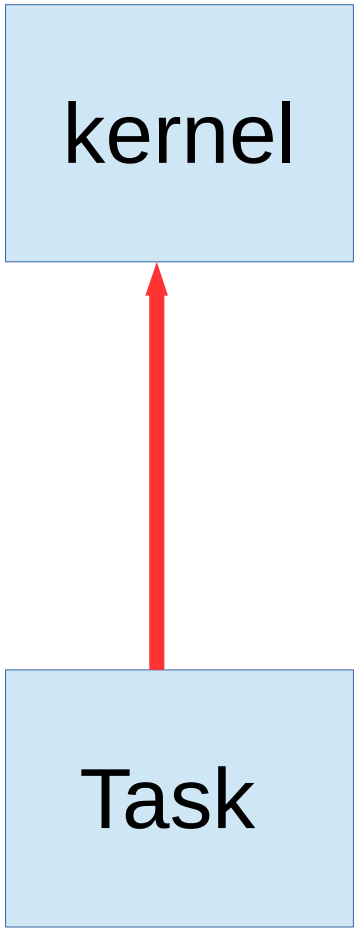
Task Stack

Core register

```
/* load user state */
  pop {r4, r5, r6, r7, r8, r9, r10, r11, lr}

/* jump to user task */
  bx lr
```

Context-switch by exception



- SVCall:
  - A supervisor call (SVC), applications can use SVC instructions to access OS kernel functions and device drivers.
- SysTick
  - A SysTick exception is an exception the system timer generates when it reaches zero.

SysTick Control and Status
SysTick Reload Value
SysTick Current Value
SysTick Calibration Value

Activate vs Exception

They are both use as context-switch in mini-arm-os

What's different?

```
svc_handler:
systick_handler:
    /* save user state */
    mrs r0, psp
    stmdb r0!, {r4, r5, r6, r7, r8, r9, r10, r11, lr}

    /* load kernel state */
    pop {r4, r5, r6, r7, r8, r9, r10, r11, ip, lr}
    msr psr_nzcvq, ip

    bx lr
```

```
activate:
  /* save kernel state */
  mrs ip, psr
  push {r4, r5, r6, r7, r8, r9, r10, r11, ip, lr}
  /* switch to process stack */
  msr psp, r0
  mov r0, #3
  msr control, r0
  isb
  /* load user state */
  pop {r4, r5, r6, r7, r8, r9, r10, r11, lr}
svc_handler:
systick_handler:
  /* jump to user task */
  bx lr
    /* save user state */
    mrs r0, psp
    stmdb r0!, {r4, r5, r6, r7, r8, r9, r10, r11, lr}

    /* load kernel state */
    pop {r4, r5, r6, r7, r8, r9, r10, r11, ip, lr}
    msr psr_nzcvq, ip

    bx lr
```

Exception entry and return

- In Activate, we save context and change to PSP in ourself
- But in exception, cortex-M3 provide specific exception handling:
  - Exception entry
  - Exception return

Exception entry and return

- Exception entry
  - Save xPSR, PC, LR, R12(ip), R3, R2, R1, R0 automatically
  - Change to handler mode and MSP
  - Happen in Thread mode, or exception has higher priority
- Exception return
  - Tell system that exception is complete
  - According to which special address (EXC_RETURN) was set to PC, system will do switch to specific mode

Exception retrun

EXC_RETURN	Description
0xFFFFFFF1	Return to Handler mode. Exception return gets state from the main stack. Execution uses MSP after return.
0xFFFFFFF9	Return to Thread mode. Exception Return get state from the main stack. Execution uses MSP after return.
0xFFFFFFFD	Return to Thread mode. Exception return gets state from the process stack. Execution uses PSP after return.

```
svc_handler:
systick_handler:
    /* save user state */
    mrs r0, psp
    stmdb r0!, {r4, r5, r6, r7, r8, r9, r10, r11, lr}

    /* load kernel state */
    pop {r4, r5, r6, r7, r8, r9, r10, r11, ip, lr}
    msr psr_nzcvq, ip

    bx lr
```

Handler mode and MSP now (automatically)

Save previous state (state before exception)

The state we saved in activate is now pop out

```
activate:
    /* save kernel state */
    mrs ip, psr
    push {r4, r5, r6, r7, r8, r9, r10, r11, ip, lr}

    /* switch to process stack */
    msr psp, r0
    mov r0, #3
    msr control, r0
    isb

    /* load user state */
    pop {r4, r5, r6, r7, r8, r9, r10, r11, lr}

    /* jump to user task */
    bx lr
```

Mini-arm-os

Hello world

Context switch

Multi-Thread

?

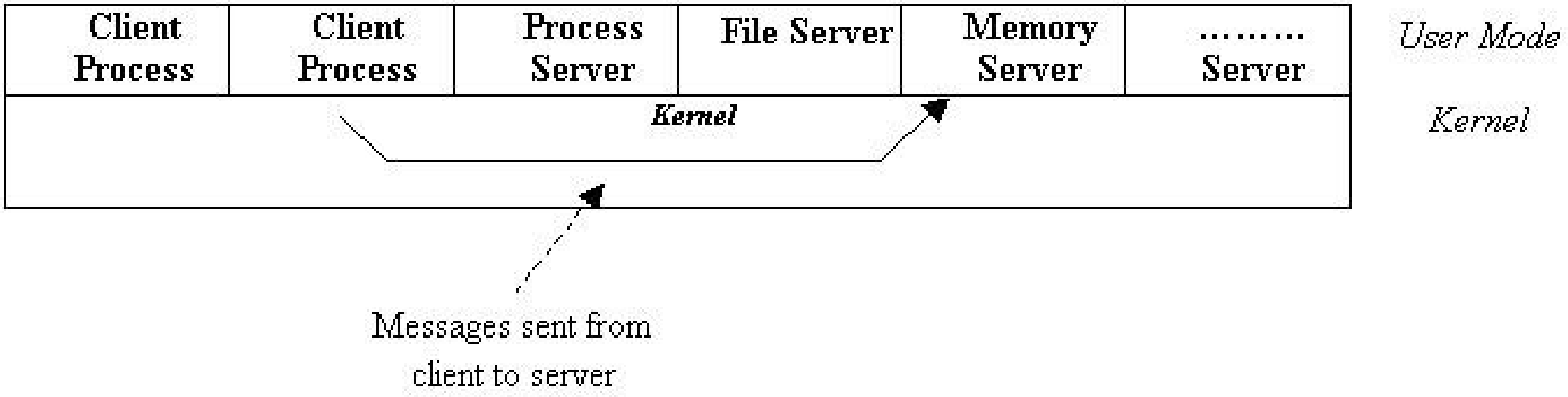
Rtenv+

rtenv+

- rtenv is a small Real-time operating system (RTOS) based on Cortex-M3, used for education
- All source files are written by NCKU students
- Its context-switch mechanism is similar to mini-arm-os, but make more progress with PendSV
- Able to run on real hardware

  (STM32F429i-discovery)
- Able to write user own application like FreeRTOS

microkernel



Rtenv+

- Scheduling :
  - Multi-tasking with priority + round-robin
- Task :
  - ready queue, event queue and three basic states : ready - running – blocked
- Communication :
  - Pipe, block, message queue, register file
- File system :
  - Romfs, read/write by block

Better context-switch PendSV

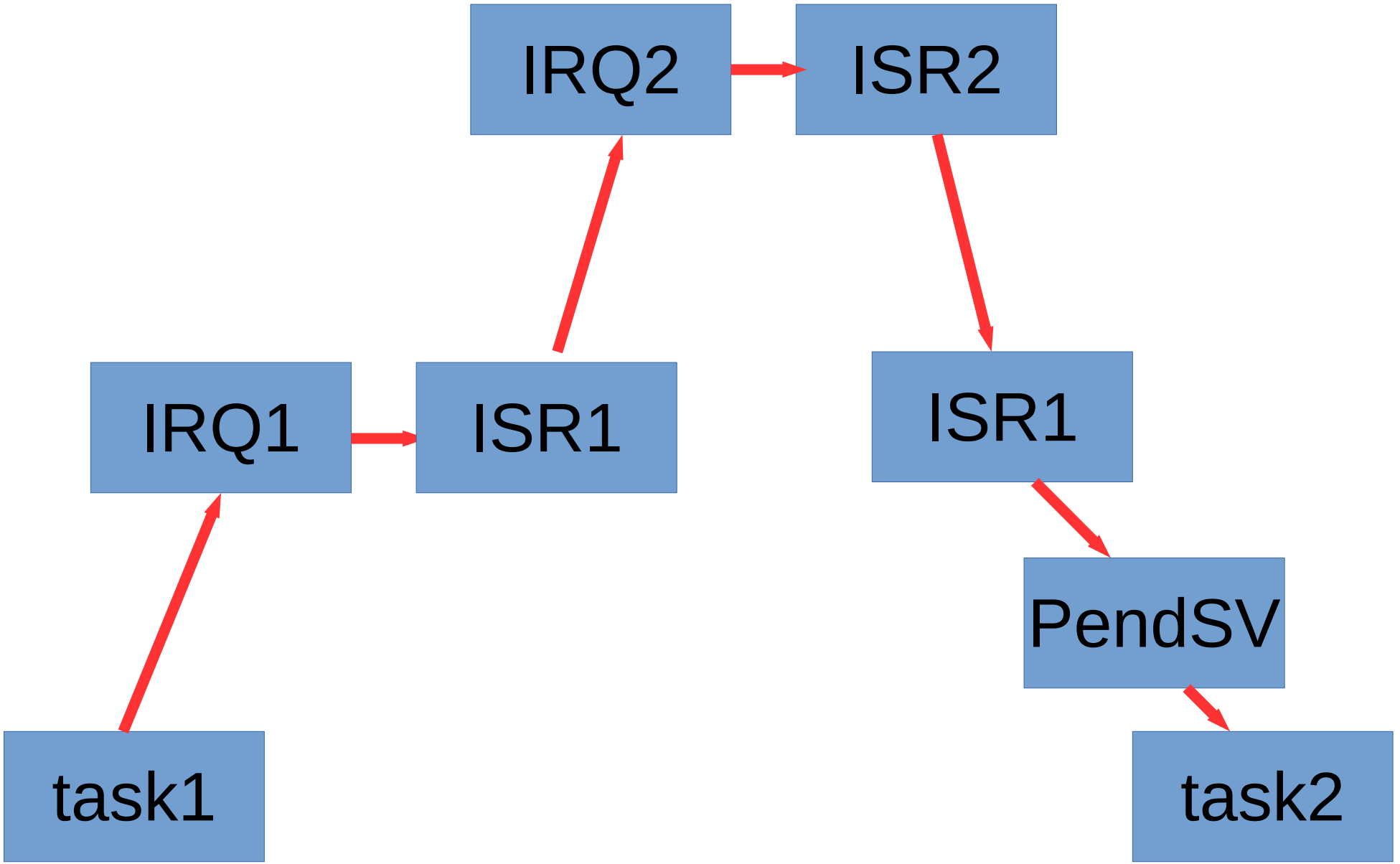
- Pended System Call (PendSV) is an interrupt-driven request for system-level service.
- In an OS environment, use PendSV for context switching when no other exception is active.
- Lowest priority

PendSV

- Without PendSV, context-switch usually happened in Systick or SVCall :

When one exception preempts another, the exceptions are nested, and something wrong might be happened

PendSV



build a new feature

- PSE51 :
  - Minimal Real-time System Profile IEEE Std 1003.13 'PSE51'
  - This profile is intended for embedded systems, with a single multi-threaded process, no file system, no user and group support and only selected options from IEEE Std 1003.1b-1993.

Building Pthread

- Understand POSIX standard for each Pthread API
- Understand what system you are building in
- Use posixtestsuit to make sure your Pthread's behavior is correct
- Trial and error

Now rtenv+ finished...

- pthread_create
- pthread_cancel/exit
- pthread_attr_* (not all)
- Signal.h

Still working~~~

GDB helps build an OS

- Trace register by layout reg
- 'x' to see value in address
- Dprintf : combines a breakpoint with formatted printing
- Backtrace (bt) is useless for exception
- Break, commands and end

Conclusion

- From simple to deep, everyone can Build Your Own Operating System

Reference

- IVT : https://en.wikipedia.org/wiki/Interrupt_vector_table
- Linker scripthttps://sourceware.org/binutils/docs/ld/Scripts.html
- Processor mode and privilege levels for software execution

  http://infocenter.arm.com/help/index.jsp?topic=%2Fcom.arm.doc.dui0552a%2FCHDIGFCA.html
- Core Register

  http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/CHDBIBGJ.html
- Exception entry and return
  http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/Babefdjc.html
- Exception type

  http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/Babefdjc.html

Reference

- Mini-arm-os

  https://github.com/jserv/mini-arm-os
- Rtenv+

  http://wiki.csie.ncku.edu.tw/embedded/rtenv
- PSE51

  http://www.opengroup.org/austin/papers/wp-apis.txt
- POSIX

  http://pubs.opengroup.org/onlinepubs/9699919799/