University of BRISTOL

Computer Systems B

COMS20012

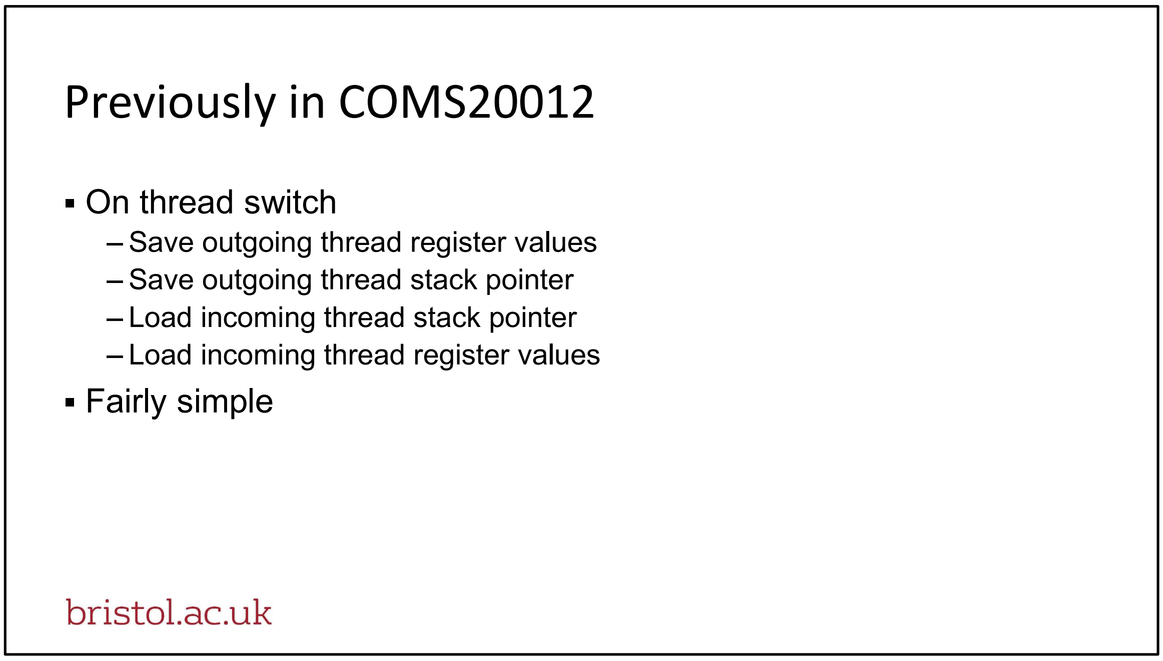
Introduction to Operating Systems and Security

bristol.ac.uk

University of BRISTOL

Thread Switching

bristol.ac.uk



Previously in COMS20012

- On thread switch
  - Save outgoing thread register values
  - Save outgoing thread stack pointer
  - Load incoming thread stack pointer
  - Load incoming thread register values
- Fairly simple

bristol.ac.uk

Implementing thread switch in OS/161 1/2

```
switchframe_switch:
        addi sp, sp, -40

        sw   ra, 36(sp)
        sw   gp, 32(sp)
        sw   s8, 28(sp)
        sw   s6, 24(sp)
        sw   s5, 20(sp)
        sw   s4, 16(sp)
        sw   s3, 12(sp)
        sw   s2, 8(sp)
        sw   s1, 4(sp)
        sw   s0, 0(sp)

        sw   sp, 0(a0)
```

bristol.ac.uk

Implementing thread switch in OS/161 1/2

```
switchframe_switch:
        addi sp, sp, -40        }   Allocate stack space for 10 registers

        sw   ra, 36(sp)
        sw   gp, 32(sp)
        sw   s8, 28(sp)
        sw   s6, 24(sp)
        sw   s5, 20(sp)
        sw   s4, 16(sp)
        sw   s3, 12(sp)
        sw   s2, 8(sp)
        sw   s1, 4(sp)
        sw   s0, 0(sp)

        sw   sp, 0(a0)
```

bristol.ac.uk

Implementing thread switch in OS/161 1/2

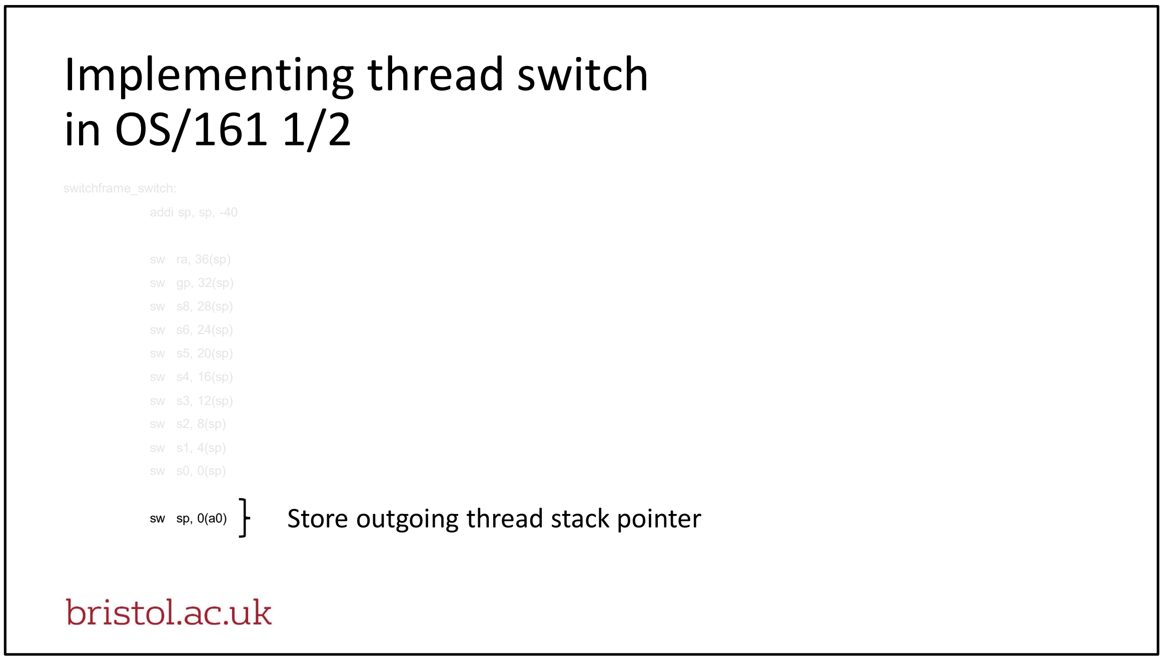
```
switchframe_switch:
            addi sp, sp, -40

            sw   ra, 36(sp)
            sw   gp, 32(sp)
            sw   s8, 28(sp)
            sw   s6, 24(sp)
            sw   s5, 20(sp)
            sw   s4, 16(sp)
            sw   s3, 12(sp)
            sw   s2, 8(sp)
            sw   s1, 4(sp)
            sw   s0, 0(sp)

            sw   sp, 0(a0)
```

Save the registers value to the stack

bristol.ac.uk



Implementing thread switch in OS/161 1/2

```
switchframe_switch:
        addi sp, sp, -40

        sw   ra, 36(sp)
        sw   gp, 32(sp)
        sw   s8, 28(sp)
        sw   s6, 24(sp)
        sw   s5, 20(sp)
        sw   s4, 16(sp)
        sw   s3, 12(sp)
        sw   s2, 8(sp)
        sw   s1, 4(sp)
        sw   s0, 0(sp)

        sw   sp, 0(a0)   }   Store outgoing thread stack pointer
```

bristol.ac.uk

Implementing thread switch in OS/161 2/2

```
        lw   sp, 0(a1)
        nop
        lw   s0, 0(sp)
         lw   s1, 4(sp)
         lw   s2, 8(sp)
         lw   s3, 12(sp)
         lw   s4, 16(sp)
         lw   s5, 20(sp)
         lw   s6, 24(sp)
        lw   s8, 28(sp)
        lw   gp, 32(sp)
         lw   ra, 36(sp)
        nop
        j ra
        addi sp, sp, 40
.end switchframe_switch
```

bristol.ac.uk

Implementing thread switch in OS/161 2/2

```
        lw   sp, 0(a1)  }   Load incoming thread stack pointer
        nop
        lw   s0, 0(sp)
         lw   s1, 4(sp)
         lw   s2, 8(sp)
         lw   s3, 12(sp)
         lw   s4, 16(sp)
         lw   s5, 20(sp)
         lw   s6, 24(sp)
        lw   s8, 28(sp)
        lw   gp, 32(sp)
         lw   ra, 36(sp)
        nop
        j ra
        addi sp, sp, 40
.end switchframe_switch
```

bristol.ac.uk

Implementing thread switch in OS/161 2/2

```
        lw   sp, 0(a1)
        nop
        lw   s0, 0(sp)
         lw   s1, 4(sp)
         lw   s2, 8(sp)
         lw   s3, 12(sp)
         lw   s4, 16(sp)
         lw   s5, 20(sp)
         lw   s6, 24(sp)
        lw   s8, 28(sp)
        lw   gp, 32(sp)
         lw   ra, 36(sp)
        nop
        j ra
        addi sp, sp, 40
.end switchframe_switch
```

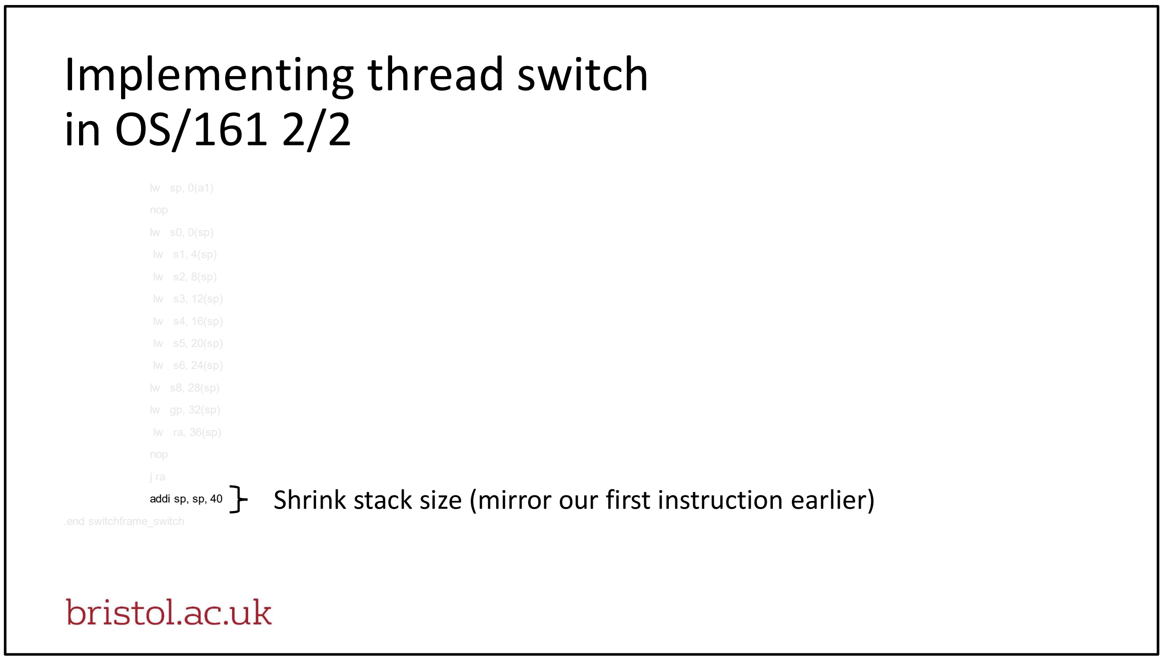
Load incoming thread register values

bristol.ac.uk

Implementing thread switch in OS/161 2/2

```
        lw   sp, 0(a1)
        nop
        lw   s0, 0(sp)
         lw   s1, 4(sp)
         lw   s2, 8(sp)
         lw   s3, 12(sp)
         lw   s4, 16(sp)
         lw   s5, 20(sp)
         lw   s6, 24(sp)
        lw   s8, 28(sp)
        lw   gp, 32(sp)
         lw   ra, 36(sp)
        nop
        j ra              }     Return
        addi sp, sp, 40
.end switchframe_switch
```

bristol.ac.uk



Implementing thread switch in OS/161 2/2

```
    lw   sp, 0(a1)
    nop
    lw   s0, 0(sp)
     lw   s1, 4(sp)
     lw   s2, 8(sp)
     lw   s3, 12(sp)
     lw   s4, 16(sp)
     lw   s5, 20(sp)
     lw   s6, 24(sp)
    lw   s8, 28(sp)
    lw   gp, 32(sp)
     lw   ra, 36(sp)
    nop
    j ra
    addi sp, sp, 40 }   Shrink stack size (mirror our first instruction earlier)
.end switchframe_switch
```

bristol.ac.uk

Implementing thread switch in OS/161 2/2

```
        lw   sp, 0(a1)
        nop
        lw   s0, 0(sp)
         lw   s1, 4(sp)
         lw   s2, 8(sp)
         lw   s3, 12(sp)
         lw   s4, 16(sp)
         lw   s5, 20(sp)
         lw   s6, 24(sp)
        lw   s8, 28(sp)
        lw   gp, 32(sp)
         lw   ra, 36(sp)
        nop
        j ra
        addi sp, sp, 40
.end switchframe_switch
```

bristol.ac.uk

Wait a minute!

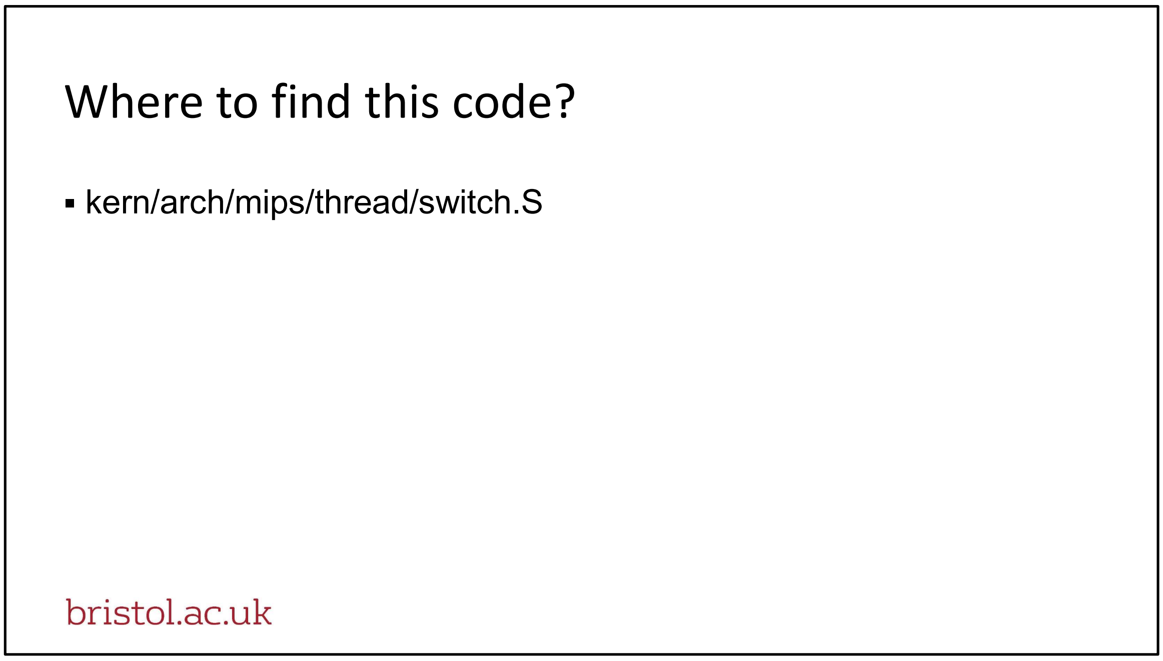
- Something happens after the return?!
- … and what about those “nop”s?!

bristol.ac.uk

MIPS Delay Slot Instruction

- Instruction executed without the effect of the previous instruction
- Present on load, jump, branch etc.
- You can use nop if the effect is necessary to carry the next instruction
- … or do something useful

bristol.ac.uk



Where to find this code?

- kern/arch/mips/thread/switch.S

bristol.ac.uk

Thread switch on x86

```
.globl swtch
swtch:
            movl 4(%esp), %eax
            movl 8(%esp), %edx

            pushl %ebp
            pushl %ebx
            pushl %esi
            pushl %edi

            movl %esp, (%eax)
            movl %edx, %esp

            popl %edi
             popl %esi
             popl %ebx
            popl %ebp
            ret
```

bristol.ac.uk

Thread switch on x86

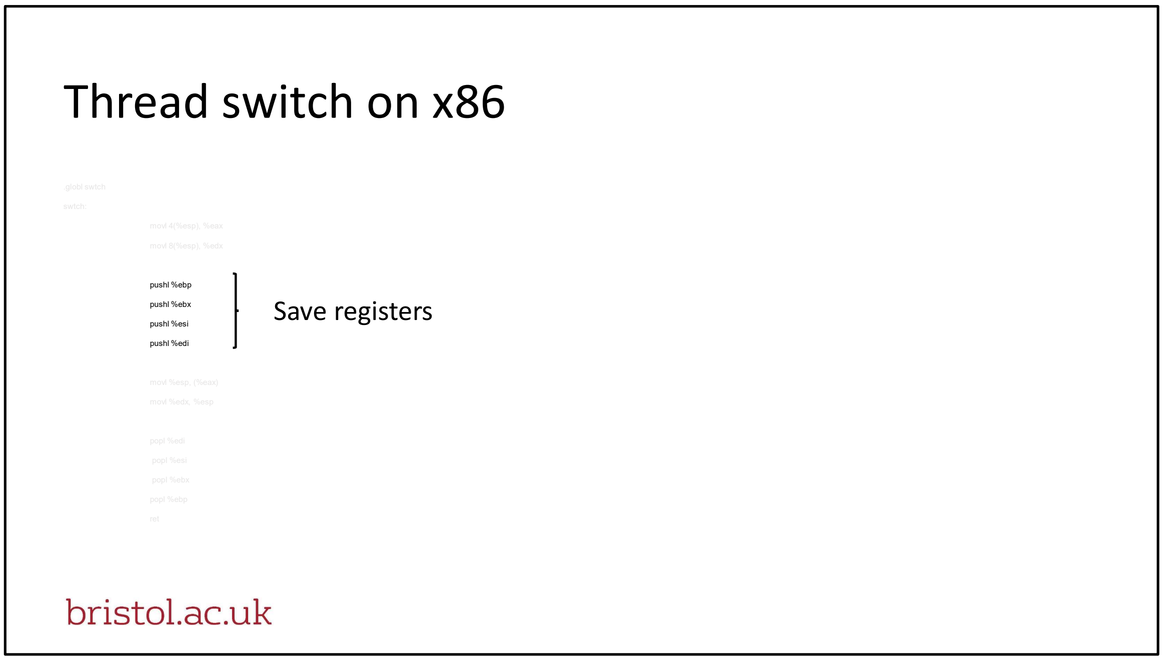
```
.globl swtch
swtch:
          movl 4(%esp), %eax   } Expand stack
          movl 8(%esp), %edx   }

          pushl %ebp
          pushl %ebx
          pushl %esi
          pushl %edi

          movl %esp, (%eax)
          movl %edx, %esp

          popl %edi
           popl %esi
           popl %ebx
          popl %ebp
          ret
```

bristol.ac.uk



Thread switch on x86

```
.globl swtch
swtch:
            movl 4(%esp), %eax
            movl 8(%esp), %edx

            pushl %ebp
            pushl %ebx
            pushl %esi
            pushl %edi

            movl %esp, (%eax)
            movl %edx, %esp

            popl %edi
             popl %esi
             popl %ebx
            popl %ebp
            ret
```

Save registers

bristol.ac.uk

Thread switch on x86

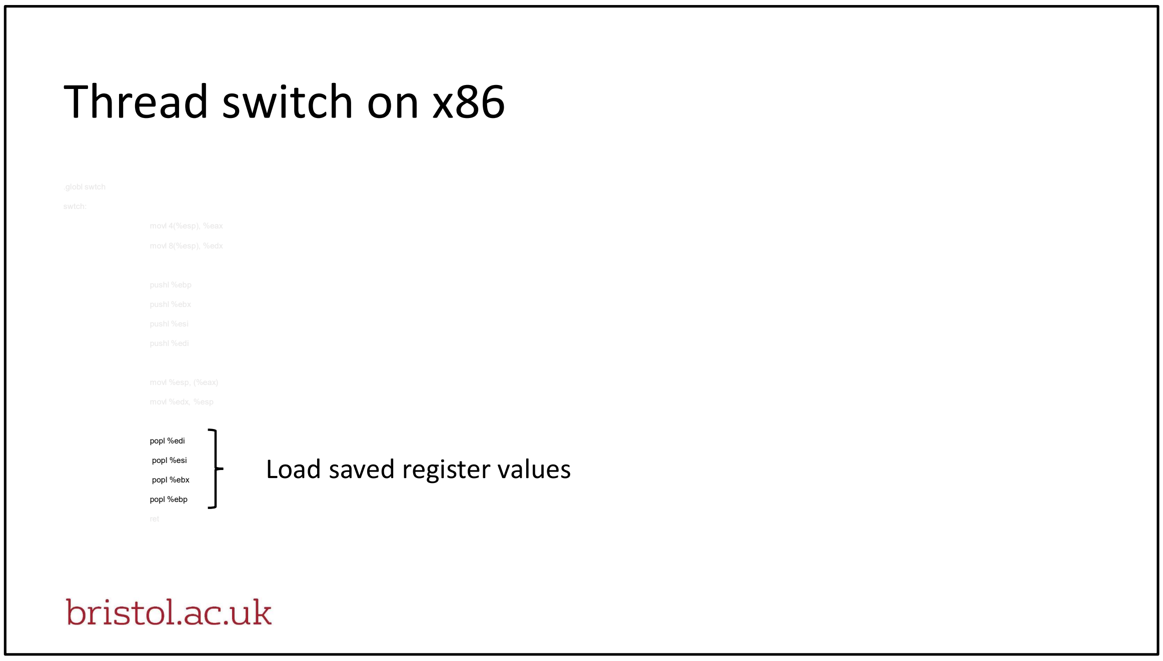
```
.globl swtch
swtch:
            movl 4(%esp), %eax
            movl 8(%esp), %edx

            pushl %ebp
            pushl %ebx
            pushl %esi
            pushl %edi

            movl %esp, (%eax)  }  Switch stack
            movl %edx,  %esp

            popl %edi
             popl %esi
             popl %ebx
            popl %ebp
            ret
```

bristol.ac.uk



Thread switch on x86

```
.globl swtch
swtch:
            movl 4(%esp), %eax
            movl 8(%esp), %edx

            pushl %ebp
            pushl %ebx
            pushl %esi
            pushl %edi

            movl %esp, (%eax)
            movl %edx, %esp

            popl %edi
             popl %esi
             popl %ebx
            popl %ebp
            ret
```

Load saved register values

bristol.ac.uk

Thread switch on x86

```
.globl swtch
swtch:
          movl 4(%esp), %eax
          movl 8(%esp), %edx

          pushl %ebp
          pushl %ebx
          pushl %esi
          pushl %edi

          movl %esp, (%eax)
          movl %edx,  %esp

          popl %edi
           popl %esi
           popl %ebx
          popl %ebp
          ret       }     Return
```

bristol.ac.uk

Thread switch on x86

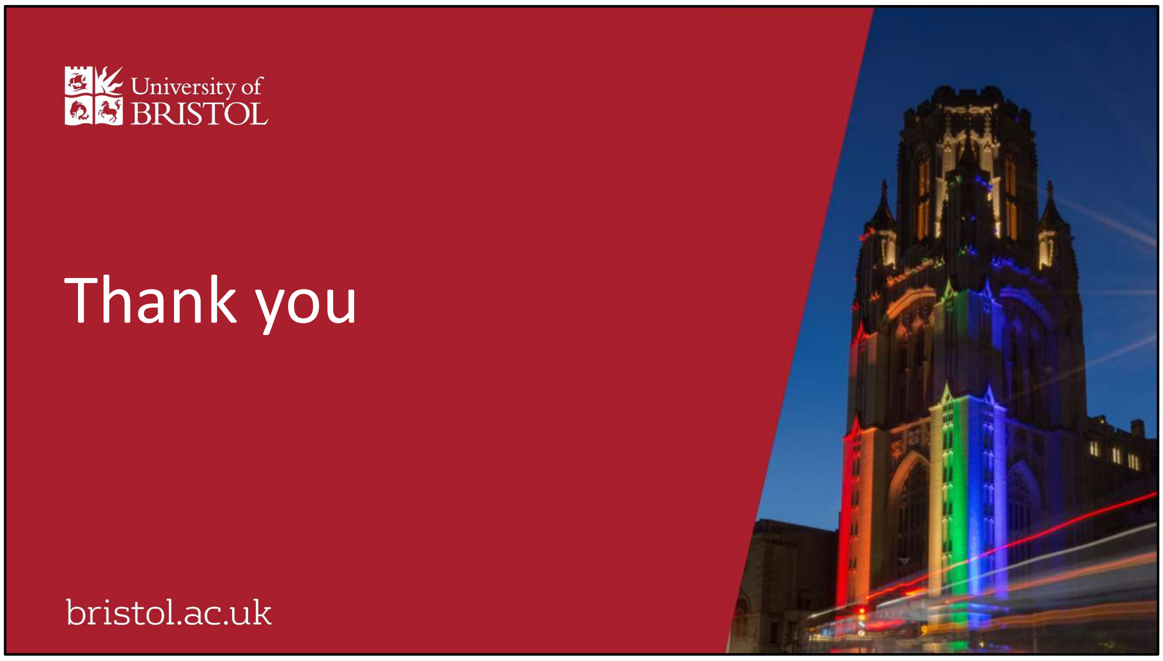
- Different implementation
- High-level logic exactly the same
- Will be the same on all architectures

bristol.ac.uk

What causes context switches?

- Thread switch is a "type" of context switch (see next video)
- Running thread **yield**
  - Voluntarily let another thread run
- Running thread call **thread_exit**
  - Voluntarily stops
- Running thread **blocks**, via a call to **wchan_sleep**
  - We saw that in the semaphore video! (Week 5 Video 6)
- Running thread is **pre-empted**
  - Running thread involuntarily stopped by an **interrupt** (more in the next video)
  - Could be hardware events "**trap**" (Week 5 video 3)
  - … or the timer used to implement the **scheduler** (more on this next week)

bristol.ac.uk



University of BRISTOL

Thank you

bristol.ac.uk