Kernels I

ENCE361 Embedded Systems 1

Course Coordinator: Ciaran Moore (ciaran.moore@Canterbury.ac.nz)

Lecturer: Le Yang (le.yang@canterbury.ac.nz)

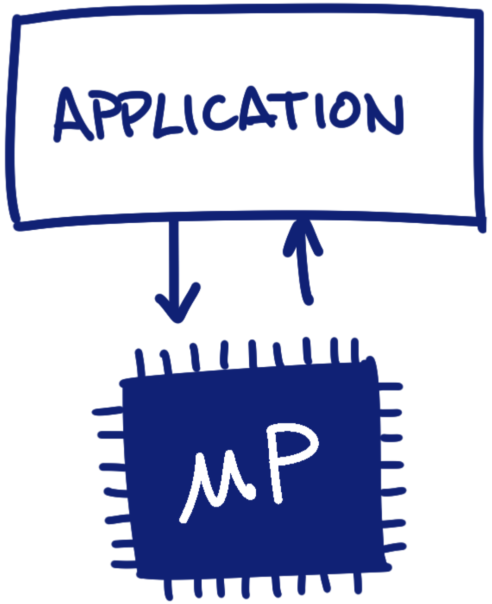
Department of Electrical and Computer Engineering

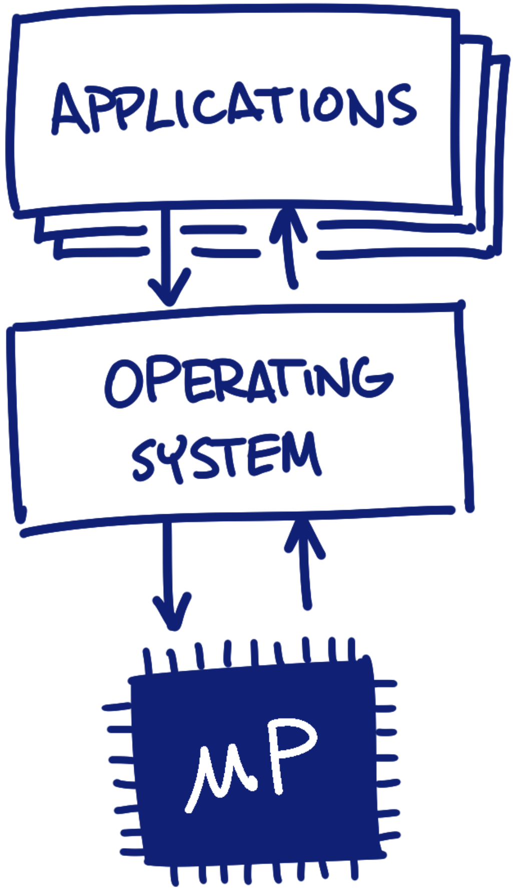
Where we're going today

- **Basics of kernels**
- Round-robin, time-triggered and interrupt-triggered schedulers
- Design example
- Homework

Kernel Basics (1)

- Embedded software vs. most other software





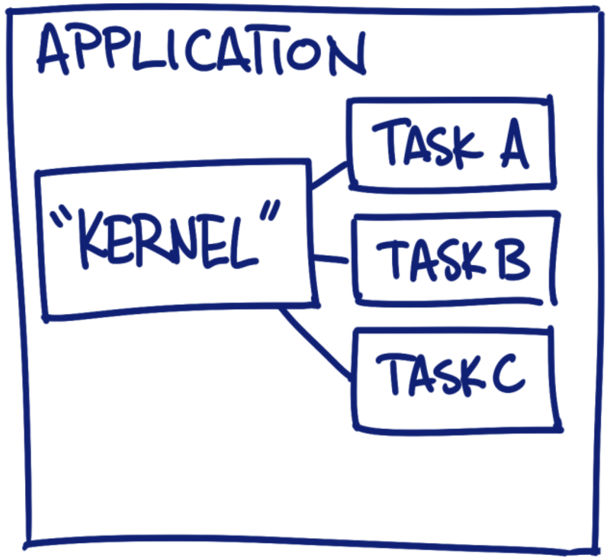
OS manages and controls hardware and software

Kernel Basics (2)

- Complex embedded software

```
void main (void)
{
        while (true)
        {
         // Do everything!
```

```
        }
}
```



- What is a kernel?
  - Main component of an OS
  - Bridge between applications and actual processing done at hardware
  - Responsibility: managing system resource

Functions of Kernels

- What does a kernel do?
  - Schedule background tasks → **scheduler**
    - All processing is achieved in a timely manner
    - All tasks are completed reliably

    - May ensure that higher priority tasks are completed first
    - May allow higher priority tasks to preempt lower priority tasks

  - May provide the ability for tasks to communicate (inter-task communication)

  - May provide the ability for resources (e.g., memory blocks, peripherals, etc.) to be shared

Where we’re going today

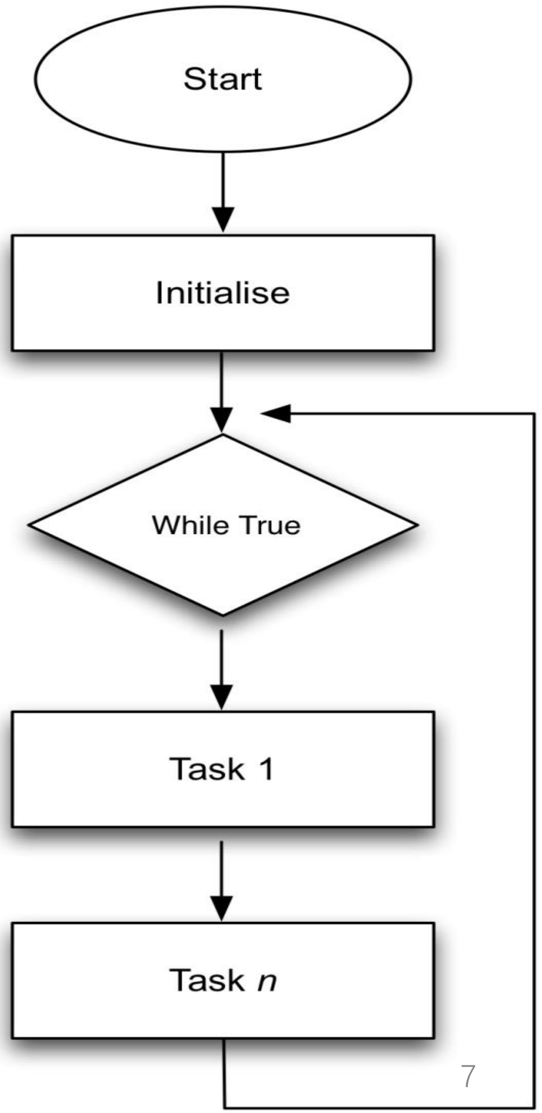
- Basics of kernels
- **Round-robin, time-triggered and interrupt-triggered schedulers**
- Design example
- Homework

Round-Robin Scheduler (1)

- Round-robin scheduler (c.f., polling)
  - Free-running cyclic

- Advantages ☺
  - Simple to implement
  - Same worst-case response time for every task

- Disadvantages ☹
  - Same worst-case response time for every task
    - All tasks have the same priority level
  - Tasks cannot be very long
    - Time slicing may be used to split a task into smaller chunks



Round-Robin Scheduler (2)

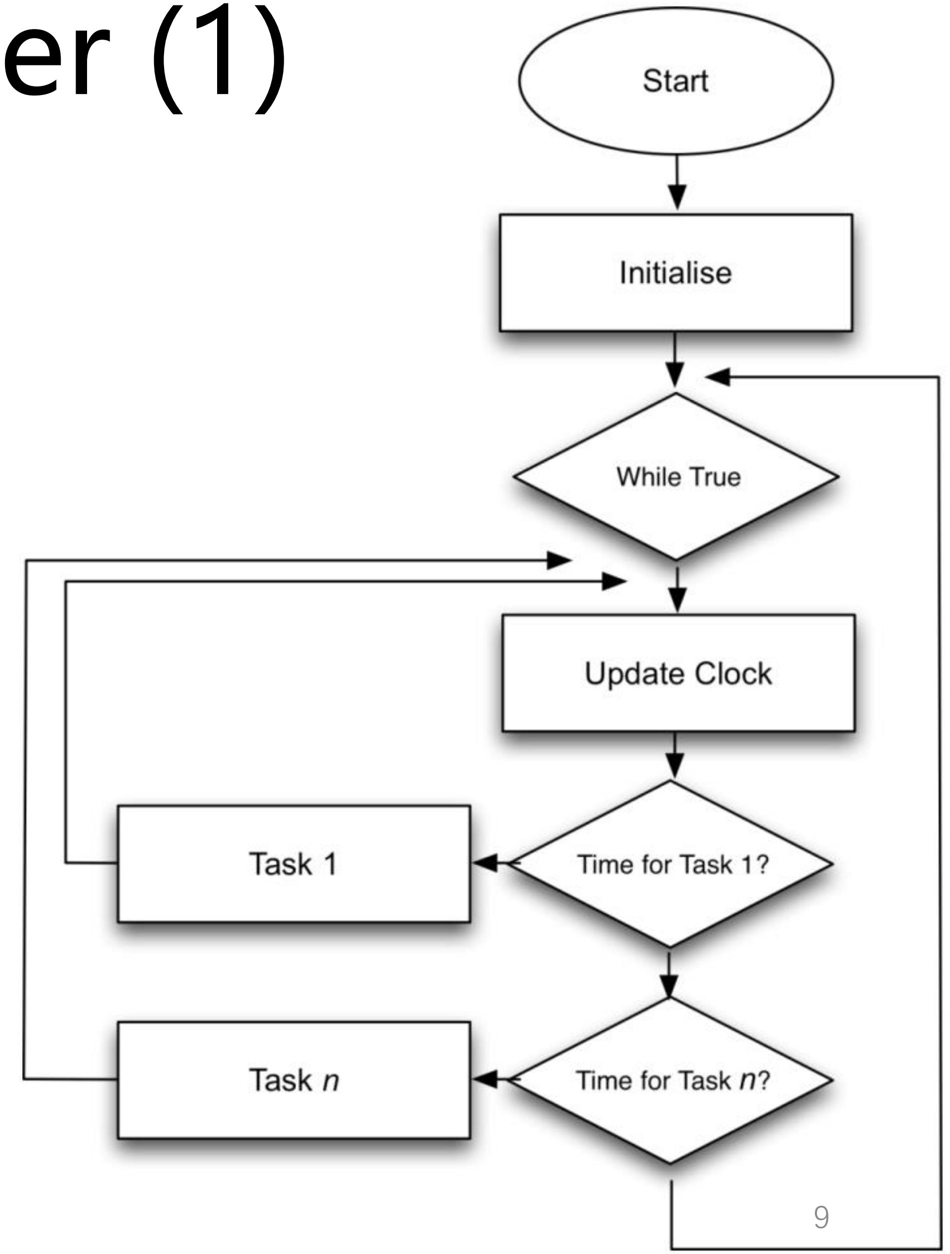
```
void main (void)
{
  while (true)
  {
    if (/* I/O device A needs service */)
    {
      // Handle data to or from Device A
    }
    if (/* I/O device B needs service */)
    {
      // Handle data to or from Device B
    }
    . . . .

    if (/* I/O device Z needs service */)
    {
      // Handle data to or from Device Z
    }
  }
}
```


From Figure 5.1 on 116, D. E. Simon, An Embedded Software Primer, Addison-Wesley

Time-Triggered Scheduler (1)

- Time-triggered scheduler
  - 'Time-division multiplexing' of MCU
- Advantages ☺
  - Perform tasks at controlled times
  - Limited task prioritization
- Disadvantages ☹
  - Worst-case response time depends on scheduled task run times
  - Tasks cannot be very long

    Time slicing may be used to split a task into smaller chunks

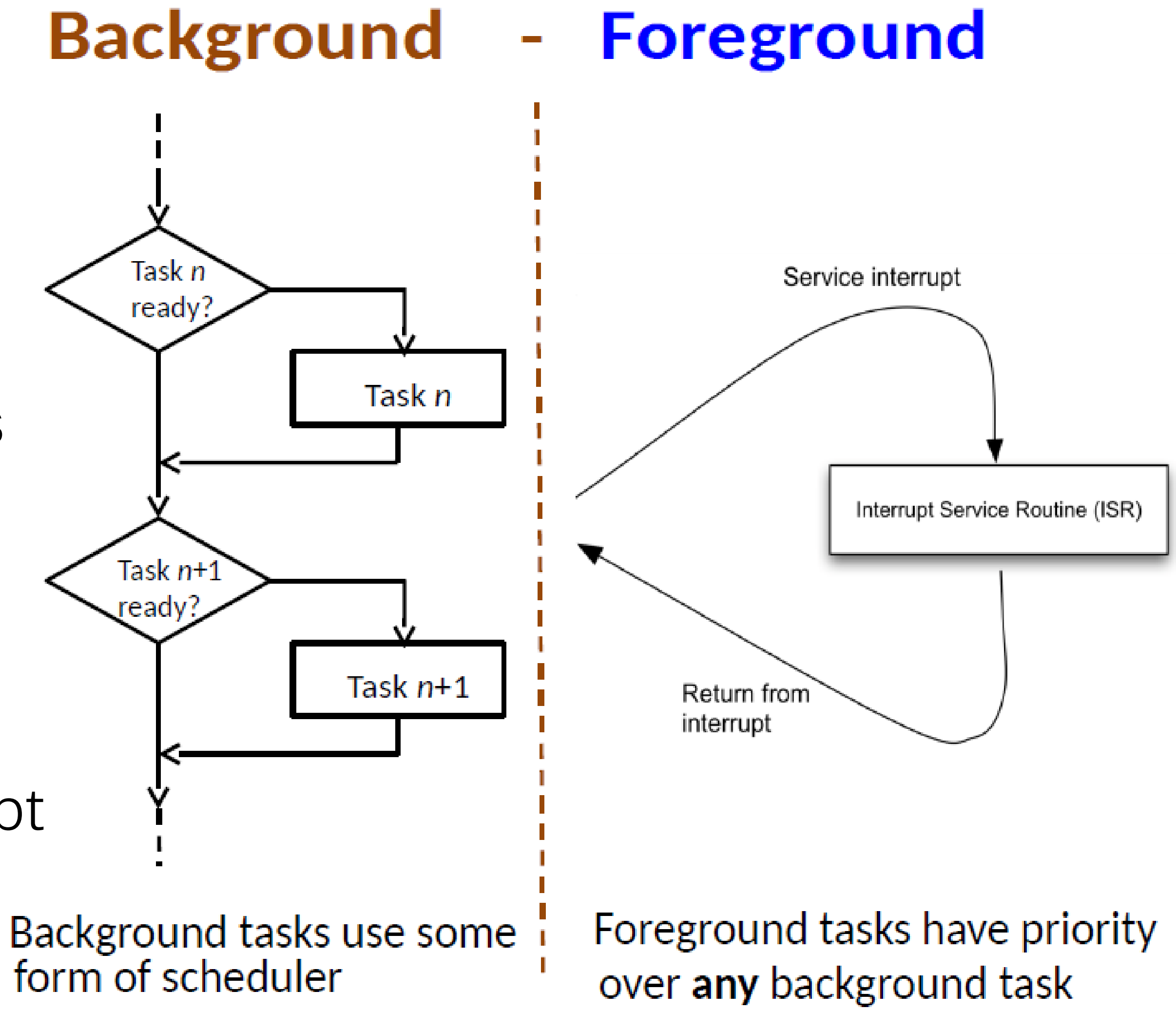


Time-Triggered Scheduler (2)

```
void timerInterruptHandler(void) {
         for (uint8_t i = 0; i < N_TASKS; i++) {
               if (scheduled_task[i].delay == 0) {
                   scheduled_task[i].ready = true;
                   scheduled_task[i].delay = scheduled_task[i].period; // Reset task
                } else {
                   scheduled_task[i].delay -= 1;
               }}}                                                   // Please, do not code like this …

void main (void) {
        while (true) {
             for (uint8_t i = 0; i < N_TASKS; i++) {
                   if  (scheduled_task[i].ready) {
                       // Call task handler
                        . . .
                       scheduled_task[i].ready = false;
                  }
         }}}                                                         // Please, do not code like this …
```

Interrupt-Triggered Scheduler (1)

- Interrupt-triggered scheduler
  - Foreground/background modeling
- Advantages ☺
  - Fast response to asynchronous events
  - Interrupt prioritization
- Disadvantages ☹
  - Worst-case response time of background tasks depends on interrupt arrival rates
  - Background tasks cannot take ‘very long’



Interrupt-Triggered Scheduler (2)

```
volatile boolean flagDeviceA = false; . . . ; flagDeviceZ = false;
void deviceAIntHandler (void) { /* Handle Device A I/O, set flag */ }
. . . .
void deviceZIntHandler (void) { /* Handle Device Z I/O, set flag */ }
void main (void)
{
. . . .
        while (true)
        {
            if (flagDeviceA)
              {
                  // Process data to or from Device A and reset flag
              }
            . . . .
            if (flagDeviceZ)
              {
                 // Process data to or from Device Z and reset flag
              }
}}
```

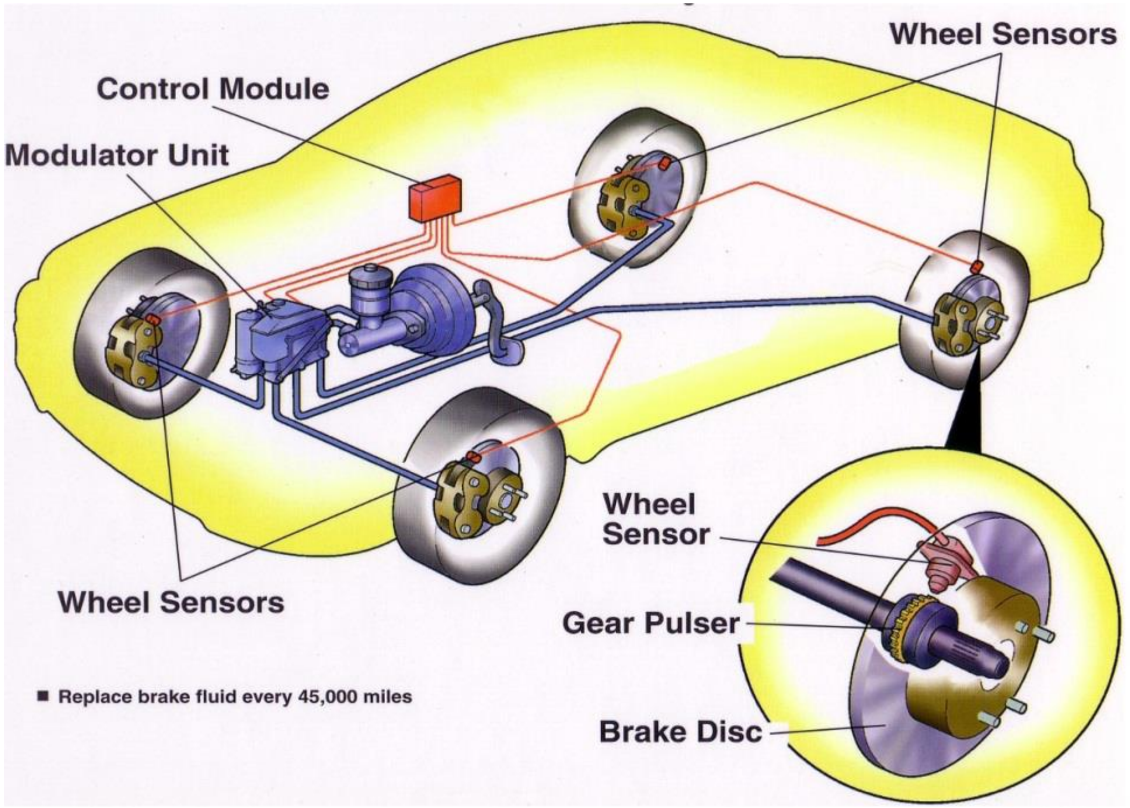

From Figure 5.4 on 120, D. E. Simon, An Embedded Software Primer, Addison-Wesley

Where we’re going today

- Basics of kernels
- Round-robin, time-triggered and interrupt-triggered schedulers
- **Design example**
- Homework

Anti-Lock Braking (ABS) System

- Accurately monitor speeds of four wheels to detect potential skid
- Interrupts for real-time clock and wheel pulses
- Pulse rate up to 1320 pulses/s (200 km/h)
  - 1320*4 = 5280 pulses/s
- Wheel speed evaluated every 5 pulses
  - 264 speed estimates/s for each wheel



Also see Lecture 12

Tasks

Task	Function	Required frequency	Execution time
Real-time clock (RTC)	ISR_SysTick ()	10^5 / s	2.5 μs
Wheel pulses	ISR_Wheels ()	5280/ s	20 μs
Task A: FLwheel speed	speedFL ()	264 / s	30 μs
Task B: FRwheel speed	speedFR ()	264 / s	30 μs
Task C: RLwheel speed	speedRL ()	264 / s	30 μs
Task D: RRwheel speed	speedRR ()	264 / s	30 μs
Task E: Monitor relative speeds	monitorWheels ()	200 / s	250 μs
Task F: Monitor driver actions	monitorDriver ()	50 / s	1.5 ms
Task G: Activate ABS	activateABS ()	Infrequent	5 ms
Task H: Display status	displayABS ()	20 / s	7.5 ms

Round-Robin Kernel-based Design

```
void main (void)
{
   // Initialise
   . . . .

   // Start round-robin kernel
   while (1)
   {
     speedFL ();                          // 30 µs,  264/s. True frequency < 106/s (insufficient!)
     speedFR ();
     speedRL ();
     speedRR ();
     if (monitorWheels ())          // 250 µs,  200/s. True frequency < 106/s (insufficient!)
        activateABS ();
     monitorDriver ();                   // 1.5 ms,  50/s. True frequency < 106/s (not necessary)
     displayABS ();                       // 7.5ms,   20/s. True frequency < 106/s (not necessary)
   }
}
```


From Table 2-8 on Page 103 in

Time-Triggered Kernel-based Design

```
void main (void)
{
// Initialise
. . . .
// Start time-triggered kernel
    while (1)
   {
        if (flagFL) { speedFL (); flagFL = 0; }                  // 30 µs,  264/s = True frequency
        if (flagFR) { speedFR (); flagFR = 0; }
        if (flagRL) { speedRL (); flagRL = 0; }
        if (flagRR) { speedRR (); flagRR = 0; }
        if (monitorWheels ())                                     // 250 µs,  200/s = True frequency
           activateABS ();
        if (monitorDriverFlag) { monitorDriver (); monitorDriverFlag = 0;}       // 1.5 ms,  50/s = True frequency
        if (displayABSFlag)     {  displayABS ();   displayABSFlag = 0;}         // 7.5ms,   20/s = True frequency
    }
}
```


From Table 2-8 on Page 103 in

Homework

1. What is the major difference between the time-triggered and interrupt-triggered schedulers?

2. What determines the resolution required for the real-time clock in the implementation of the ABS braking system?

3. In the time-triggered version of the ABS system, what is the maximum amount of time that could be spent on executing a chunk of the monitorDriver and displayABS tasks without causing the deadlines for wheel speed processing to be missed?