Carnegie Mellon University Qatar

15-348: Embedded Systems Multitasking

Fall 2022

Ryan Riley
(based on original slides by Saquib Razak)

1

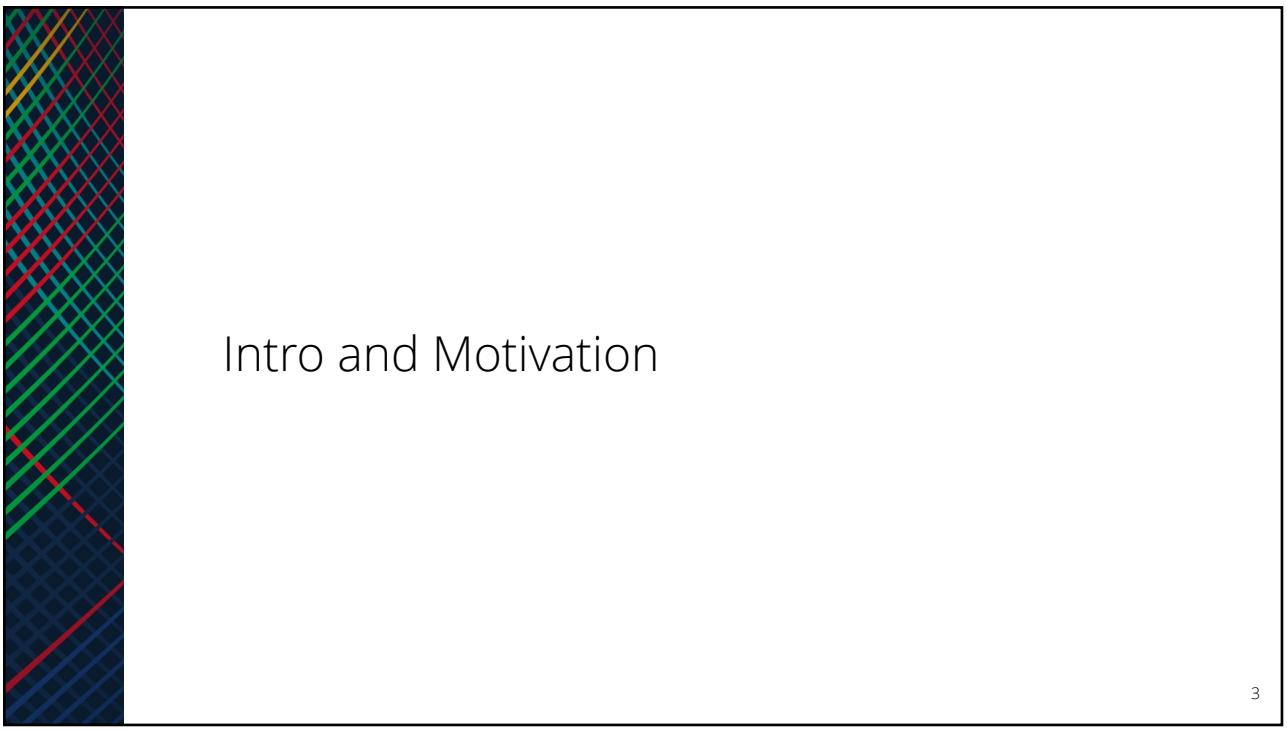
1

Where are we?

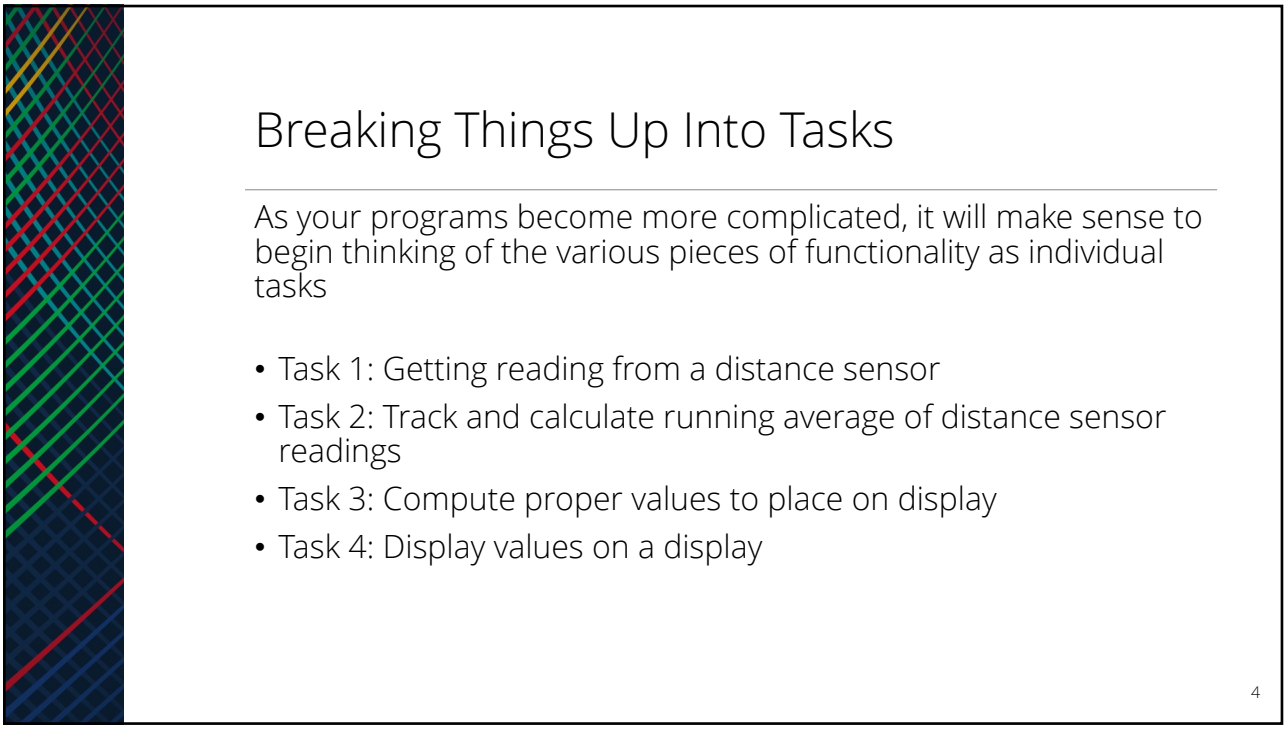
- Recently
  - Control
- Today
  - Multitasking, both cooperative and preemptive

2

2


Intro and Motivation
3

3


Breaking Things Up Into Tasks
As your programs become more complicated, it will make sense to begin thinking of the various pieces of functionality as individual tasks
• Task 1: Getting reading from a distance sensor
• Task 2: Track and calculate running average of distance sensor readings
• Task 3: Compute proper values to place on display
• Task 4: Display values on a display
4

4

What is a Task?

- An *active flow of control* within a computation
- Multiple tasks can be active concurrently
  - But not running concurrently with multiple processors
- Tasks ”share” the processor either cooperatively (nicely taking turns) or preemptively (being forced to take turns)

5

5

Task Terminology from OS

- Process
  - Unique memory, registers, and PC value
- Thread
  - Unique registers and PC value
  - Memory shared with other threads
- On a microcontroller, we are usually talking about threads

6

6

Non-Preemptive Scheduling

- A task is not suspended during execution
- Task runs to completion, or voluntarily stops, before next task starts
- Round robin or prioritized

7

7

Preemptive Scheduling

- Tasks each act as if they are the only thing that needs to happen on the processors
- Forcefully pre-empted by other tasks
  - Need for a *context switch* to switch from one task to the other
- Context switch
  - Save the state of the current task
  - Load the state of the new task
  - Switch to the new task

8

8

Preemptive vs Non-Preemptive Tasks

- Non-preemptive task
  - Task is allowed to run to completion, even if other tasks want service
  - Example: the ISR routines with high priority interrupts
- Preemptive task
  - Task is suspended if some other task wants service, then restarted later
  - Example: main program loop from previous lectures (preempted by ISRs)
- Most real systems are a combination of both
  - ISRs pre-empt non-ISR tasks, but highest priority interrupt is non-preemptable
  - Main tasks are preemptable

9

9

Bookkeeping

- Typically, there is a pool of tasks
- Each task m can be running or suspended
- There is a place in memory to save state for each task m (the PCB)
  - PCB = "Process Control Block"
  - In our case threads, but we'll use PCB name for consistency with OS descriptions
- Each PCB entry contains relevant information and state about a task

10

10

Basic, Cooperative Scheduling

11

11

Cooperating

- Tasks can be written to voluntarily *yield* control to other tasks
- No need for a context-switch, because tasks can save their own state
- You need to be able to think of your programs this way during design

- Simplest Example

```
while(1)
{
    do_task0();
    do_task1();
    do_task2();
}
```

12

12

Popular Approach in HW3 (Keypad)

A variety in tasks in their own helper functions

1. Debouncing read
2. One step of keypad state machine

- Each task executes every run of the main loop
  - Each task does some work, then returns
  - Each tries to avoid *blocking*
    - We used state machines to achieve this

13

13

Towards a Generalized Cooperative Approach

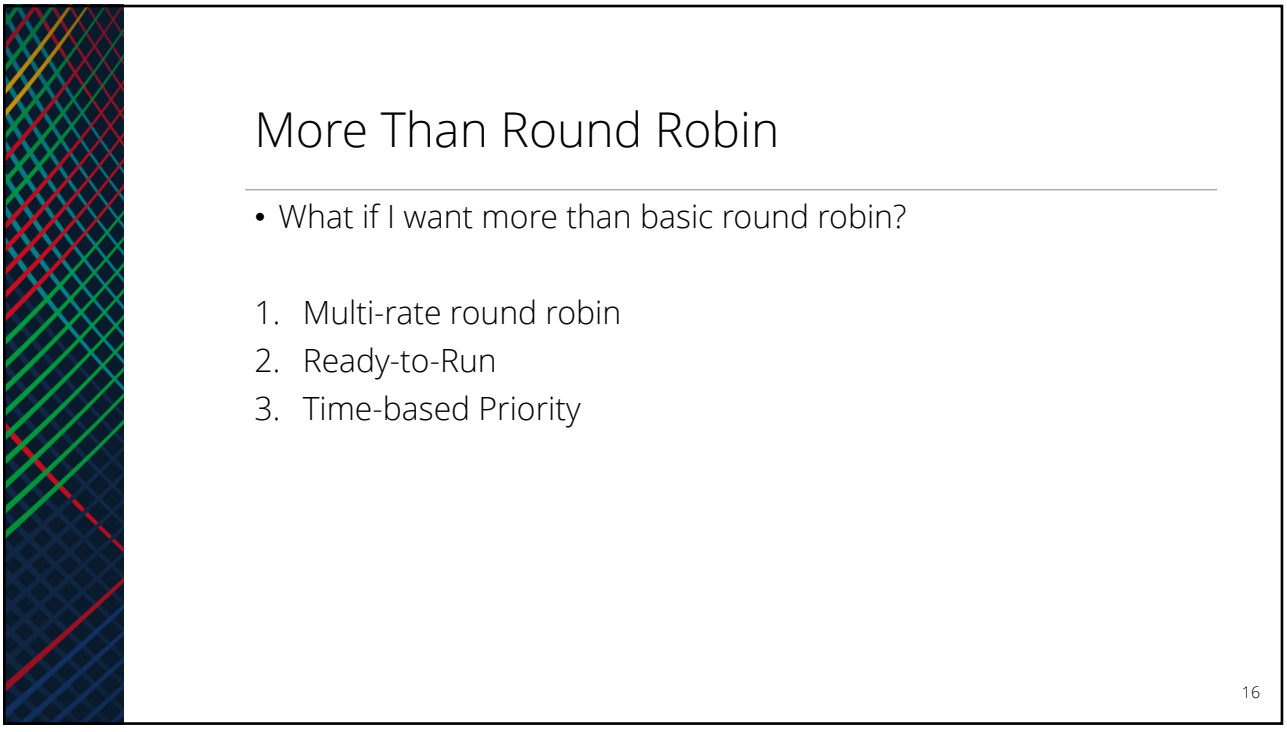
```
#include <stdio.h>
void task0(void) { ... }
void task1(void) { ... }
void task2(void) { ... }
#define NTASKS 3 // NTASKS is number of defined tasks
typedef void (*pt2Function)(void); // pointer to function
pt2Function PCB[NTASKS] = {&task0, &task1, &task2}; // Array of function pointers
int main(void) {
          int i;
          // cooperative multitasking executive -- round robin
          while(1) {
                    for (i = 0; i < NTASKS; i++){
                              PCB[i](); // execute task i
                    }
          }
}
```

14

14


More Complicated Cooperative Scheduling
15

15


More Than Round Robin
• What if I want more than basic round robin?
1. Multi-rate round robin
2. Ready-to-Run
3. Time-based Priority
16

16

Multi-Rate Round Robin

- I want some tasks to run more frequently than others
- For each PCB keep:
  - Pointer to task to be executed
  - Period (number of times main loop is executed for each time task is executed)
    i.e., execute this task every k*th* time through main loop.
  - Current count – counts down from Period to zero, when zero execute task

17

17

Multi-Rate Round Robin PCB

```
typedef void (*pt2Function)(void);
struct PCB_struct
{
      pt2Function Taskptr; // pointer to task code
      uint8 Period; // execute every kth time
      uint8 TimeLeft; // starts at k, counts down
      uint8 ReadyToRun; // flag used later
};
PCB_struct PCB[NTASKS]; // array of PCBs
```

18

18

Multi-Rate Round Robin Code

```
int main(void){
          // init PCB values
          PCB[0].Taskptr = &task0; PCB[0].Period = 1; PCB[0].TimeLeft = 1;
          PCB[1].Taskptr = &task1; PCB[1].Period = 2; PCB[1].TimeLeft = 1;
          PCB[2].Taskptr = &task2; PCB[2].Period = 4; PCB[2].TimeLeft = 1;
          // cooperative multitasking executive -- multirate
          while(1) {
                    for (i = 0; i < NTASKS; i++){
                              if (--PCB[i].TimeLeft == 0) // decrement and check time
                              {
                                        PCB[i].Taskptr(); // execute task
                                        PCB[i].TimeLeft = PCB[i].Period;
                              }
                    }
          }
}
```

19

19

Ready-to-Run

- Task doesn't run unless someone else has flagged it as ready to run
- When it does run, it runs to completion for some purpose

20

20

Ready-to-Run Examples

- Time-Based events
  - SysTick ISR: Increments a tick counter
  - Main Loop: When tick counter reaches a certain value, run a specific task
  - You've done this
- Too Long ISR
  - ISRs should be short, but sometimes process their data can take a long time
  - A skinny ISR: Grabs available data, sets a processing tasks "ready to run"
  - Task: Actually process the data
  - I did this with my WiFi program

21

21

Ready-to-Run Code

```
while(1)
{
	for (i = 0; i < NTASKS; i++)
	{
		if (PCB[i].ReadyToRun)
		{
			PCB[i].ReadyToRun = 0; // OK, it's going to run
			PCB[i].Taskptr();
			break; // exit loop and start again at task 0
			// Why do we break here?
		}
	}
}
```

22

22

Time-Based Priority

- Similar to multi-rate round robin, but processes can be scheduled invidually
- Processes can be scheduled to run at specific, upcoming times, or just run periodically

23

23

Time-Based Priority Code

```
struct PCB_struct{
        pt2Function Taskptr; // pointer to task code
        uint8 Period; // Time between runs
        uint8 NextTime; // next time this task should run
};
while(1) {
        for (i = 0; i < NTASKS; i++){
                if (PCB[i].NextTime < timer_ticks){
                        PCB[i].NextTime += PCB[i].Period;
                        PCB[i].Taskptr();
                        break; // exit loop start again at task 0
                }
        }
}
```

24

24

Non-Preemptive Take-Aways

- Tasks must be written to assume that other tasks exist
  - Each time they run, they do some work and yield (or run to completion if needed)
- Latency is very important
  - Tasks shouldn't take too long
- A basic, round-robin approach with an extra ready-to-run priority will get you far
- You need to consider situations like, "what if all the tasks are needed at the same time?"

25

25

Preemptive Scheduling

26

26

Cooperation Sounds Like a lot of Work…

- What if I don't want to think about cooperation?
- I just want to write every task as if it is in the only thing on the system
- Preemption is for you!
- Note: It probably isn't for you. Coding this is complicated and error-prone. Just cooperate, you slacker!

27

27

Preemptive Tasking

- Executing task is suspended if some other task wants service
- Suspended task is restarted at a later time
- Example: an ISR preempts the main program loop

28

28

Hardware Support for Preemption

- Interrupts are the simplest types of preemption
  - Save main program state on stack
  - Execute interrupt
  - Restore main program state from stack
  - Return to main program

29

29

General Approach to Preemptive Scheduling

- N tasks are running concurrently
- Have a place to store N copies of program state
- Have a scheduler that saves and restores state:
  - Stop task K
  - Save state in the "task K save area"
  - Restore state from the "task J save area"
  - Restart task J from wherever it left off
- Simplest approach is to have K stacks – one per task – and save states there

30

30

When to Preempt?

- Hardware approach
  - Hardware determines something of higher priority needs to execute
  - Works OK for ISRs preempting ordinary code
  - But, tricky for ISRs preempting other ISRs!
- Software approach
  - Software sees which task is highest priority, and starts running it
  - But… what lets the software, that does the checking, run?
  - Simple approach – run the scheduler on a periodic timer
  - You need to interrupt tasks, not run to completion

31

31

The Context Switch

- So, your scheduler is running in a Timer ISR and wants to switch between Task A and Task B
  - How do you do it?
- You need a context switch to make this work
  - "Context" = all data required to define the state of the current task
  - Simplest example: all register values
  - Might include other things in more complex computing environments (for example, file I/O status)

32

32

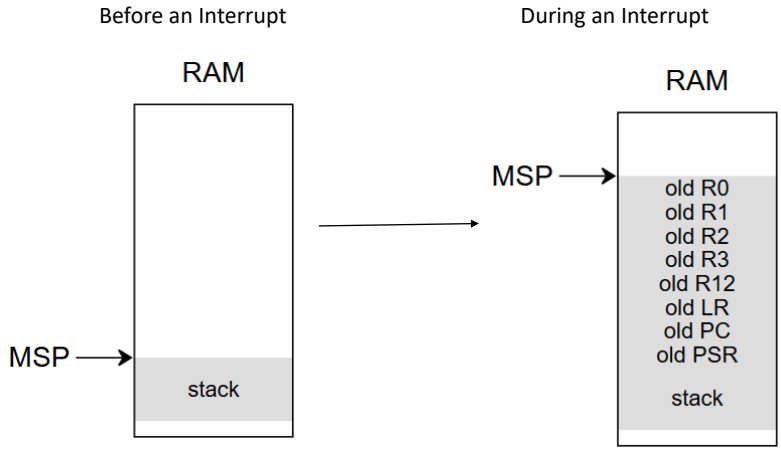
Example Context Switch

- Save everything that matters from old task
  - PC
  - Registers
  - Condition Codes
  - SP
  - Return Address
- Restore everything that matters before restarting new task
  - PC
  - Registers
  - Condition Codes
  - SP
  - Return Address

33

33

Recall…



34

34

High-Level Sketch of Preemption

- Use timer interrupt to do task switching
  - Assume that no other ISRs are active
  - Assume every task has a separate stack space
- Each Timer interrupt triggers a context switch to next round-robin task

35

35

Timer ISR For Preemption

1. Acknowledge Timer interrupt
2. Look at static **`current_task`** variable to figure out which task is running
3. Push unsaved registers onto stack
4. Save **`SP`** into **`PCB[current_task]`**
5. Pick new **`current_task`** based on scheduling policy
6. Restore **`SP`** from **`PCB[current_task]`**
7. Pop registers from stack
8. Execute return from interrupt
   - Restarts with new task where it left off

36

36