MICROS-2

Timothy Borunov (Most slides by Shriram Raja)
Ph.D. Student, Computer Science

With materials from slides by Dr. Sasan Golchin

MICROS-1

- Setup GDT
- Detect memory to see which regions are free (already done in MEMOS-2)
- Finally, assign work to the threads and call the scheduler to run the first task

MICROS-2

- Setup GDT
- Detect memory to see which regions are free (already done in MEMOS-2)
- Finally, assign work to the threads and call the scheduler to run the first task
- Enable support for preemption of threads by the scheduler to support more complex scheduling policies

Enable Preemption by Scheduler

Control should switch to the scheduler after a predetermined amount of time elapses irrespective of the running task

- System should have the capability to interrupt the running task - Interrupts
- System should have a notion of time - Programmable Interrupt Timer (PIT)

Interrupt Support

- Software Interrupts: Exceptions (non-maskable) and User-defined interrupts called by INT (maskable)
- Hardware Interrupts:
  - generated by devices external to the processor
  - E.g., timer (IRQ0), keyboard (IRQ1)
  - Maskable
  - Each core has only one interrupt line; hence we need multiplexing hardware - PIC
- Interrupts are disabled/enabled by cli and sti instructions that affect EFLAGS.IF

Interrupt Descriptor Table

Gate Descriptor (32-bit):

63 48	47	46 45	44	43 40	39 32
Offset 31 16	**P**	**DPL** 1 0	0	**Gate Type** 3 0	Reserved

31 16	15 0
Segment Selector 15 0	**Offset** 15 0

- Offset: address of the Interrupt Service Routine
- Segment Selector: pointer to valid section in the GDT
- Gate Type: Task, Interrupt or Trap Gate
- DPL: Privilege Level
- P: Present Bit; should be 1 for a valid IDT entry

Interrupt Service Routine

- Depending on the type of interrupt, there may or may not be an error code
- ISR returns using iret instead of ret
- This pops everything from the stack except the error code. So, if your interrupt has an error code, that should be popped manually.

```
my_isr0:
    # retrieve the error code (if any)
    # pass some parameters to C if needed
    # call the handler code in C
    call my_handler_in_c
    # return from the ISR
    iret
```

Testing IDT

- Cause an exception: E.g., Divide-By-Zero
  - After initializing IDT, perform a division by zero and see if the ISR kicks in
- INT instruction:
  - Using inline assembly, issue an INT instruction in your code after IDT is setup

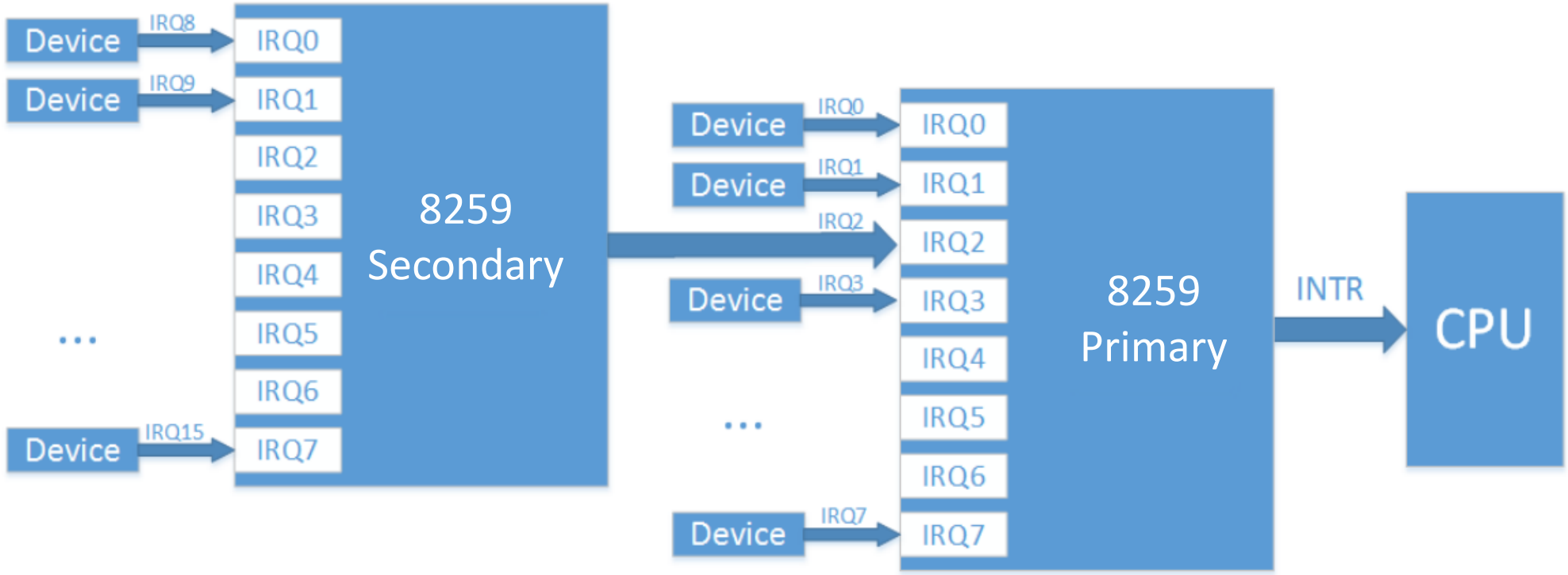
Debugging with gdb

```
$ qemu-system-i386 -S -s -kernel fifos.elf -vnc :1
```

```
$ gdb fifos.elf
…
(gdb) target remote localhost:1234
```

- Execution does not continue until the continue command is given from gdb
- Some salient features:
  - Single step through the code
  - Set breakpoints in the code, and watchpoints, which are triggered when an address/variable changes
  - Print variable and register values

Programmable Interrupt Controller



Programmable Interrupt Timer

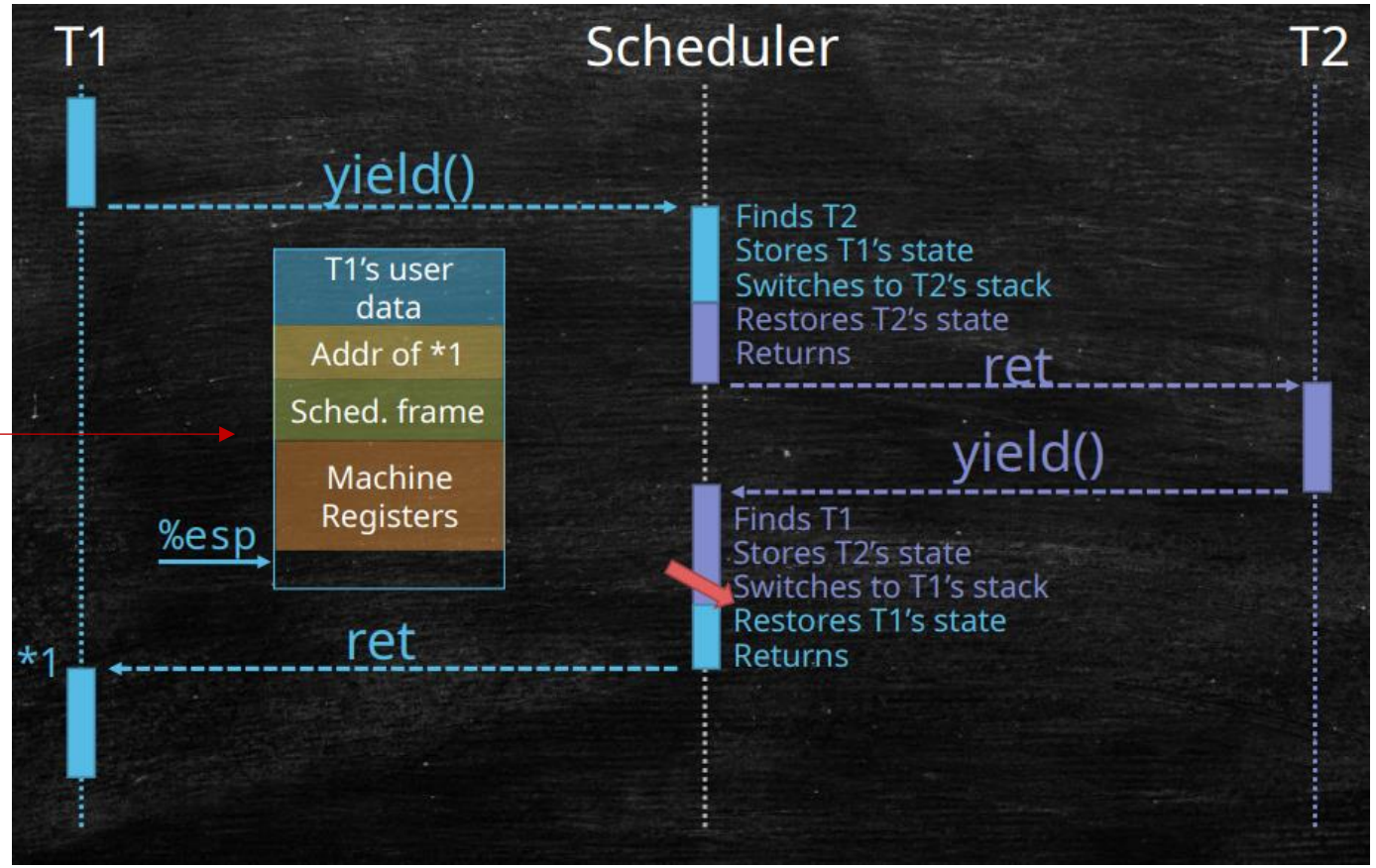
- A peripheral device that can be programmed using I/O ports
  - Base frequency of ~1.19MHz that can be decreased by a single prescaler (a circuit that performs integer division on a clock frequency)
- It is an interval timer with 3 channels
  - While the prescaler affects all channels, each channel has its own frequency divider. So, they can run with different speeds.
  - Channel 0 is usually used as the system timer and generates the “IRQ-0” upon some event that depends on the mode it’s set up to operate in (e.g., when its counter reaches zero in mode 2 – rate generator)
  - Channel 1 and 2 are not really used anymore.

Points to Remember

- Disable interrupts before PIC/PIT setup
- Map IRQs to interrupt numbers beyond 31, i.e., the interrupt number that CPU uses to look up IDT = IRQ number + Offset
  - Why? Remember the first 32 interrupt numbers are reserved for CPU exceptions
- Send EOI signal to the right PIC depending on the interrupt number

Example (T1 to T2 and back to T1)

Setup a dummy stack when creating a thread to make context switch easy



Notes on Clarifying Implementation of Rate Monotonic Scheduling

- Use a timeslice of 1000 microseconds.
- Assume no system/interrupt overhead for the timing
- You do not need to check for overload/admission control test (We might add this as extra credit if you do)
- For now, create your own set of threads with budgets and periods that fit into an RMS scheme and test it.
- Whenever interrupt is triggered, print out budget+period of current thread and if you are switching to new thread. As long as the information is clear and readable to see that RMS works, format is up to you.

References

1. IDT - Overview, Tutorial, Packed structs
2. Interrupt Service Routines
3. Programmable Interrupt Timer
4. Programmable Interrupt Controller
5. gdb - with QEMU, documentation
6. Inline Assembly - OSDev, gcc
7. Felix Cloutier - LGDT/LIDT, PUSHA, CLI, STI