The (weights) of questions are relative and may not necessarily add up to 100.

Read the project description (https://fxlin.github.io/p1-kernel/) before proceeding

Read the "implementation notes" that is downloadable from the assignment page

diff is required. If you need help on diff, seek help early on

Part B: Fix a broken kernel for QEMU

This assignment refers to the code of "exp6/". It expects students to work with QEMU, even for students who worked with Rpi3 hardware in previous assignments (whom I expect to be more comfortable with QEMU)

Problem symptom

Build and run the given code on **the Rpi3 hardware**, it works as expected:

```
kernel boots ...
Kernel process started. EL 1
User process
123451234512345abcdeabcdeabcdeabcd12345123451234eabcdeabcdeabcdeab51234512345123
4cdeabcdeabcdeabcde51234512345123abcdeabcdeabcdeabc451234512345123deabcdeabcdeabc
dea45123451234512bcdeabcdeabcdeabcd345123451234512eabcdeabcdeabcdeab345123451234
51cdeabcdeabcdeabcde234512345123451abcdeabcdeabcdeabc23451234512345deabcdea
```

Now build and run it **on QEMU**. What have you observed?

```
VNC server running on 127.0.0.1:5900
kernel boots ...
Kernel process started. EL 1
User process
12345123451234512345123451234512345123451234512345123451234512345123451234512345123451
23451234512345123451234512345123451234512345123451234512345123451234512345123451234512
34512345123451234512345123451234512345123451234512345123451234512345123451234512345123
45123451234512345123451234512345123451234512345123451234512345123451234512345123451234
51234512345123451234512345123451234512345123451234512345123451234512345123451234512345
12345123451234512345123451234512345123451234512345123451234512345123451234512345123451
23451234512345123451234512345123451234512345123451234512345123451234512345123451234512
34512345123451234512345123451234512345123451234512345123451234512345123451234512345123
45123451234512345123451234512345123
```

Problem cause

Why is our kernel not preempting user tasks on QEMU?

The (weights) of questions are relative and may not necessarily add up to 100.

The kernel's preemptive scheduling is driven by timer interrupts. Turns out that the given kernel source code only includes a driver (`timer.c`) for Rpi3's **"system timer"** (which we briefly mentioned in experiment 3 when introducing interrupts). However, QEMU's emulation of system timer is incomplete – it cannot generate interrupts. QEMU emulates **the ARM generic timer**, which we have been using since exp3.

The ARM generic timer is present on both the Rpi3 hardware and QEMU.

Fix it!

We will make the kernel (which implements user-level process support) work with the ARM generic timer. If successful, our kernel should be able to preempt user tasks on QEMU; it will behave as usual as running on the Rpi3 hardware.

Roadmap & challenges

The first step is to port the driver for the generic timer from our previous kernel versions to this kernel version. This is easy -- mostly copy & paste, update some macros, etc.

The challenge is the memory mapping for IRQ registers of the Arm generic timer. In previous experiments, this was not a problem because MMU was off and our kernel uses physical addresses. In this experiment, we turned on MMU so the kernel must establish memory mapping for any registers before accessing them.

In the exp3 doc (https://fxlin.github.io/p1-kernel/exp3/rpi-os/), search for "Other timers on Rpi3": the IRQ register for the generic timers are above address 0x40000040.

```
// See BCM2836 ARM-local peripherals at
// https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/QA7_rev3.4.pdf

#define TIMER_INT_CTRL_0    (0x40000040)
#define INT_SOURCE_0        (LPBASE+0x60)
```

Unfortunately, our current kernel only maps up to 1GB (0x40000000) physical memory. Additional mapping is needed for these registers.

Now you will have to figure out how to allocate the needed additional pgtables, install the pgtables to the existing pgtable tree, and eventually get everything work.

Good luck!

1. (10) In your own words, describe the cause of the problem – why isn't the given kernel doing preemptive scheduling?

The (weights) of questions are relative and may not necessarily add up to 100.

The problem is that the kernel is not properly switching tasks causing it to only execute the task for *12345* without doing it for *abcde*. The main problem is that QEMU's system timer does not generate interrupts which causes the kernel to not schedule the *abcde* task.

2. (10) In your own words, write a list of items to fix
    1. Port drivers
    2. Map memory for IRQ registers
    3. Setup page tables via allocating and installing pagetables

3. (10) Describe 1-2 most challenging & unexpected issues you have run into.

    One of the most challenging that I experience was creating and mapping the second PMD table. I initially thought that the second PMD table was simply another layer in the paging tree hierarchy however, I did not realize that it was actually another page that PMD can map to. This meant that I was essentially trying to a create a PTE page as opposed to a second PMD as suggested from the resources.

4. (40) Code

    [upload a diff file named as ComputingID.diff]