1. Make a new LKM file. You can name it anything you want. Your LKM must have the following:
   a. include the necessary header files
   b. set the module license
   c. take in one run time parameter which will be the pid of a process
   d. create your modules initialization and closing functions for now just put a printk statement in both of them to show your module works.

   You should load your module using `modprobe` and make sure it works as you expect it to. Unload it when you are done testing it.

2. Create a separate very simple program that just runs an infinite loop and compile it.

3. We will need to get the `task_struct` for the process we want to print out the `vm_areas` of. To get this add the following lines to the start of your initialization function:

   ```c
   struct task_struct *ts;
   ts = pid_task(find_vpid((pid_t) pid), PIDTYPE_PID);
   ```

   Note: `pid` is the run time parameter that you passed into your LKM. Change the second line to match what you named your parameter. Before you go on I encourage you to look at the code and understand what it is doing. In particular, why do we cast `pid` to the type `pid_t`, what does `find_vpid()` do, and what does `pid_task()` do?

4. Compile your LKM, you should receive a bunch of compiler errors. Why? Find out what’s missing and add it in.

5. Once your LKM compiles it is time to test it again. Start your program that runs an infinite loop. Then search for the PID of your program.

   ```bash
   $ ps auxw | grep <your program name>
   EX: $ ps auxw | grep a.out
   ```

   Load your LKM using `modprobe`. Pass the PID of your infinite loop program to your module. Since we are printing no information from the `task_struct` your LKM won’t be doing anything interesting yet but it should run. Unload it so we can make it do something interesting.
6. Before we can do anything with our task_struct we need to acquire a lock on it to make sure it doesn’t disappear while we are using it. We can do this by using a semaphore that is built into the mm_struct of the task_struct. Add the lines

```c
down_read(&ts->mm->mmap_sem);
up_read(&ts->mm->mmap_sem);
```

Note: The down_read function will decrement a value from the mmap_sem semaphore which indicates that something (our LKM) is using the mm_struct and so nothing else should modify it while we are using it.

The up_read function increments a value from the mmap_sem semaphore which indicates that we are done using the mm_struct and we don't care about it anymore.

Any code placed between the down_read and the up_read functions is called a critical section because our LKM holds a lock on the mm_struct for all code between those functions. The rest of the code you add in should be placed in this critical section.

7. Now we can make our LKM do some thing interesting with the task_struct. To start lets print out the name of the process.

```c
printk(KERN_INFO "name of process = %s\n", ts->comm);
```

Try compiling and loading your module after you add that line in. Pass it the PID of your infinite loop process. Your LKM should now print out the name of your infinite loop process. Awesome! Unload your LKM.

8. Inside the task_struct there is an instance of an mm_struct called mm. (line 1255)

http://lxr.linux.no/linux+v3.7.7/include/linux/sched.h#L1190

Within this mm_struct there is a variable of type unsigned long called start_code. This holds the address of the start of the code section. (line 350)

http://lxr.linux.no/linux+v3.7.7/include/linux/mm_types.h#L311

Your mission, should you choose to accept it, is to print out the address of the start and end of the code section as well as the start and end of the data section.

Here's a hint:

```c
printk(KERN_INFO "mm->start_code = 0x%08x\n", ....fill this part in.....);
```

Congratulations you are finished!