Name	NetID
Juanli Shen	Jshen35
Funing Xu	fxu12
Wei Yang	weiyang4
Yunsheng Wei	wei29

1. Design
   1.1. The core data structure is a linked list that stores the PID and CPU time of registered processes.
   1.2. The Proc file of the module is the I/O interface between the module in kernel space and the application processes in user space. By writing the PID into the Proc file, an application process can register itself into the kernel linked list. Similarly, by reading the Proc file, someone can check the PID and CPU time from the kernel linked list.
   1.3. Timer interrupts are used to update the records in the kernel linked list. The timer handler will defer the updating work into a workqueue.

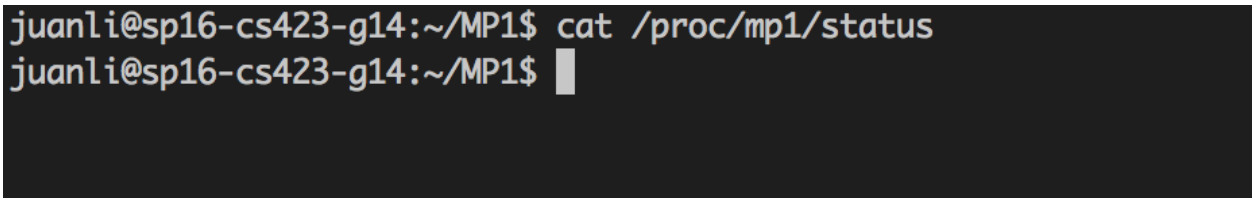
2. Implementation
   2.1. The Proc file utilizes the sequential file to simplify the open() and read() callbacks, eliminating the worry about page breaks.
   2.2. The data written by a user is restricted to a specified size in kernel, although normal PID won't excess this size.
   2.3. Since kernel timer is one-shot, the timer handler should restart the timer to make it periodic.
   2.4. When the timer expires, the timer handler will create an updating work and insert it into a workqueue immediately. The work(s) in the workqueue will be done later to update the CPU time of each registered process in the kernel linked list. If a process no longer exists, the corresponding item in the list will be deleted.
   2.5. Every time the kernel linked list is accessed (read/written), a spin lock is required.

3. How to run
   1) "make"
      Compile the code.
   2) "sudo insmod mp1.ko"
      Install the module.
   3) "./userapp & ./userapp &"
      Start two processes that will register themselves via Proc FS.
   4) "cat /proc/mp1/status"
      Check the CPU time of these two processes.
   5) "kill [PID]"
      Kill one of the processes.
   6) "cat /proc/mp1/status"
      There is only one registered process now.
   7) "kill [PID]"
      Kill the remaining process

8) “cat /proc/mp1/status”
   No registered process now.
9) “sudo rmmod mp1”
   Remove the module.

4. Screenshots of the output after running the command “cat /proc/mp1/status” at different stages

   1) The “status” file is empty before we start userapp(s), because no process has been registered yet.



   2) After starting two userapps (17675 and 17676), we can read the PID and the corresponding CPU time from the “status” file.

3) After several seconds, we run the “cat” command again, and we can see the values have been updated.

```
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
juanli@sp16-cs423-g14:~/MP1$ ./userapp & ./userapp &
[1] 17675
[2] 17676
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
17675: 90
17676: 90
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
17675: 404
17676: 403
juanli@sp16-cs423-g14:~/MP1$
```

4) We kill one of the running processes so that it can be deregistered. As a result, we can no longer see the entry of the killed process in the status file, while the one of the active process keeps being updated.

```
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
juanli@sp16-cs423-g14:~/MP1$ ./userapp & ./userapp &
[1] 17675
[2] 17676
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
17675: 90
17676: 90
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
17675: 404
17676: 403
juanli@sp16-cs423-g14:~/MP1$ kill 17675
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
17675: 1657
17676: 1656
[1]-  Terminated              ./userapp
juanli@sp16-cs423-g14:~/MP1$ cat /proc/mp1/status
17676: 2276
juanli@sp16-cs423-g14:~/MP1$
```