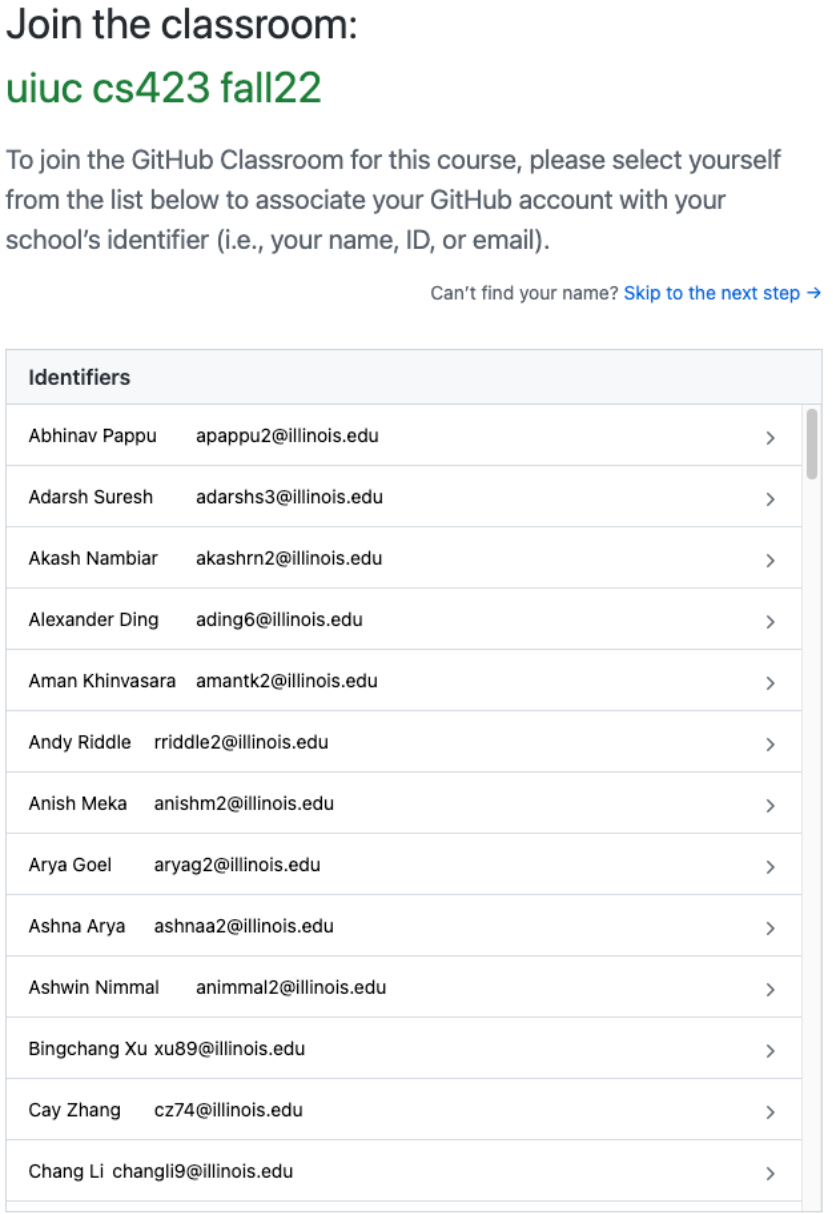
MP1 Walkthrough

9/13

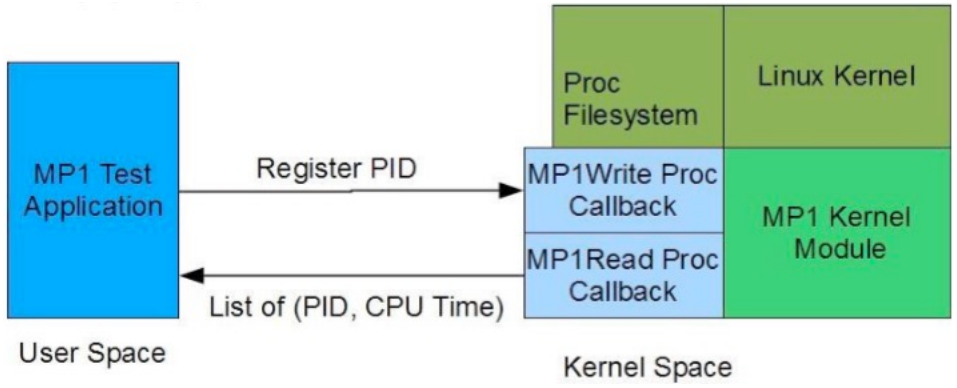
Get Starter Code

- https://classroom.github.com/a/gFqT4asl
- Find your name and click (don't click on other's name!)
- P.S. Don't forget to submit your MP0!



Problem Description

- Write a kernel module that measures the Userspace CPU Time of processes registered within the kernel module
- Register processes using PID through the Proc Filesystem
- Kernel module updates the userspace CPU time of each registered process every 5s
- Print the userspace CPU time of each registered process



Proc Filesystem

- Not regular files, does not store data in binary format
- Can be read/write as regular files
- Create an entry (e.g. `/proc/mp1/status`) in the proc filesystem
  - `proc_mkdir()`
  - `proc_create()`
- Register a process:
  - `echo “pid” > /proc/mp1/status`
  - Use `fprintf()`, etc.
- Get userspace CPU time:
  - `cat /proc/mp1/status`
  - Use `fgets()`, etc.
  - Should print in the following format:

    <PID1>:[space]<CPU time of PID1(decimal)>\n
    <PID2>:[space]<CPU time of PID2(decimal)>\n
    (end)

Store States

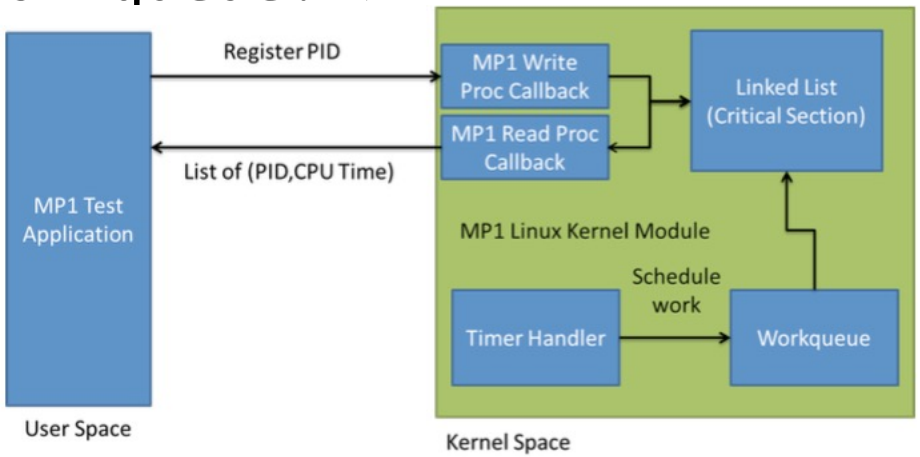
- Implement read and write callback for the proc entry
  - `proc_read()`
  - `proc_write()`
- Use kernel linked list to store the information of every registered process
  - APIs in `<linux/list.h>`
- Need to consider concurrency for linked list operations
  - E.g. using a lock

Update States

- Use a kernel timer to perform a task after a preset timeout
  - APIs in `<linux/timer.h>`
- Setup timer
  - `timer_setup(timer, callback, flags)`
  - `callback` will be called after timeout fires
- Setup timeout
  - Timeout is represented in Jiffy in kernel. Jiffy can be converted between regular time units (s, ms, etc.)
  - `mod_timer(timer, expires)`
  - `expires` is an absolute time (unit in Jiffy)
- Kernel timer is one-shot

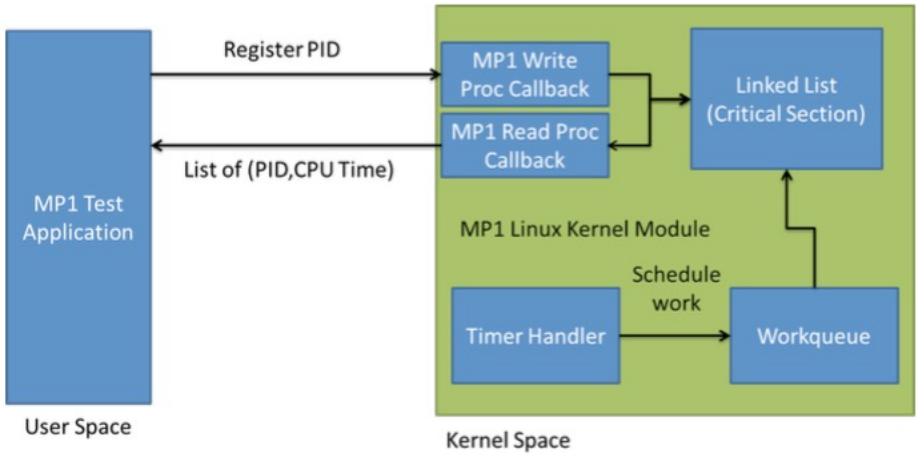
Work Queue

- We are not going to put all the update work in the `callback`!
- Use a two-halves approach
- Use kernel work queue
  - allow kernel functions to be activated (much like deferrable functions) and later executed by special kernel threads
  - APIs in `<linux/workqueue.h>`



Work Queue

- Schedule a function to be run in a work queue
  - `queue_work(work_queue, work)`
  - `callback` only calls `queue_work()` (Top-Half)
  - `work` is where we are going to do the actual updates (Bottom-Half)



Other Things

- Access data in userspace
  - E.g. `ssize_t proc_read(struct file *file, char __user *buf, size_t size, loff_t *loff)`
  - `buf` here is a userspace address and can't be dereferenced directly in kernel space
  - Use `copy_from_user()` to copy to a kernel buffer
  - Same for `copy_to_user()`
- Free/deallocate any memory/objects before exiting the kernel module
  - Dynamic allocated memory using `kmalloc()` must be freed using `kfree()`
  - Objects such as timer/work_queue must be destroyed
  - Proc FS entry must be removed

Other Things

- Debug
  - Use `printk()` to print to the kernel log
    - View the kernel log using `dmesg` (e.g. `dmesg | less`)
    - Works on any platform
    - Sufficient for MP1 (from my experience)
  - Use `gdb`
    - Only works for those who use `qemu`
    - A bit tricky to load the symbol table for kernel module. You can ask Jinghao/Siyuan how to do that
- Submission
  - Push your code to your GitHub repo before ddl

Demo