# MP1 Walkthrough

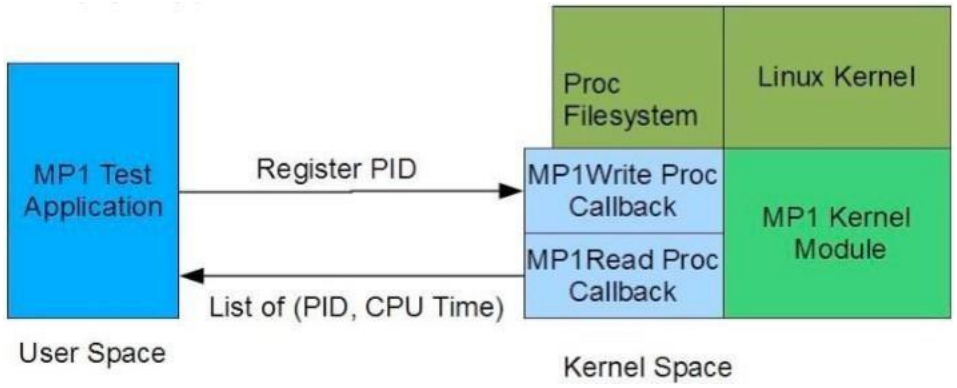
2/12

# Get Starter Code

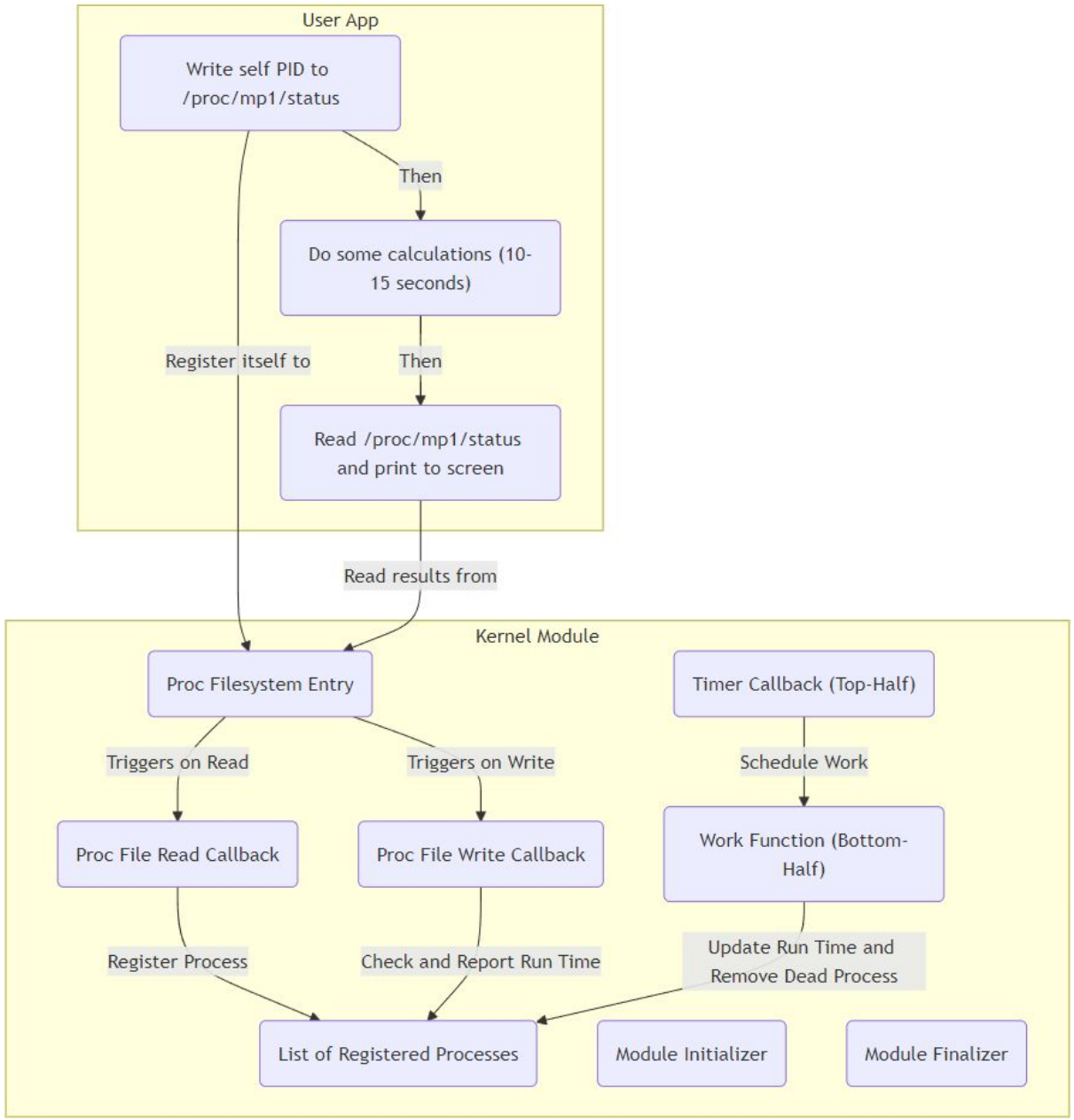
- https://classroom.github.com/a/mn6KbuEb
- Find your name and click (don’t click on other’s name!)
- Due **Feb. 28th at 11:59 PM CT**
- P.S. Don’t forget to submit your MP0! (Due **tonight 11:59PM CT**)

# 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



# Overview

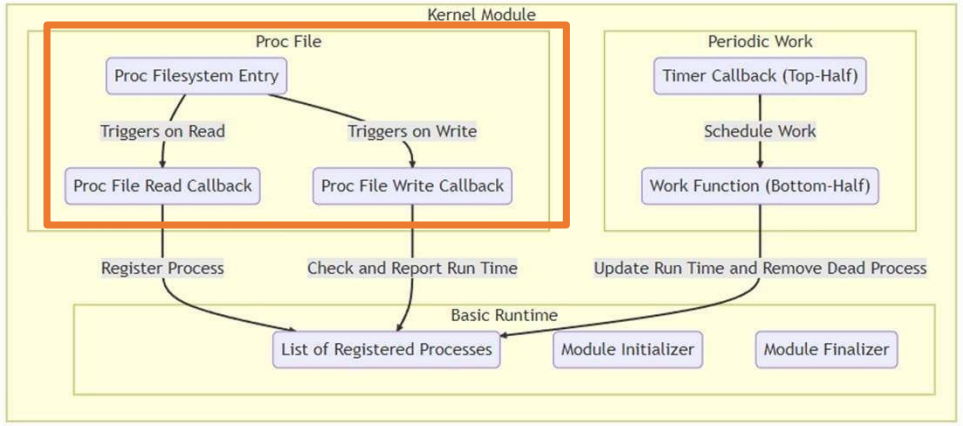

User App
Write self PID to /proc/mp1/status
Then
Do some calculations (10-15 seconds)
Register itself to
Then
Read /proc/mp1/status and print to screen
Read results from
Kernel Module
Proc Filesystem Entry
Timer Callback (Top-Half)
Triggers on Read
Triggers on Write
Schedule Work
Proc File Read Callback
Proc File Write Callback
Work Function (Bottom-Half)
Register Process
Check and Report Run Time
Update Run Time and Remove Dead Process
List of Registered Processes
Module Initializer
Module Finalizer

# Proc Filesystem Entry

- 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:

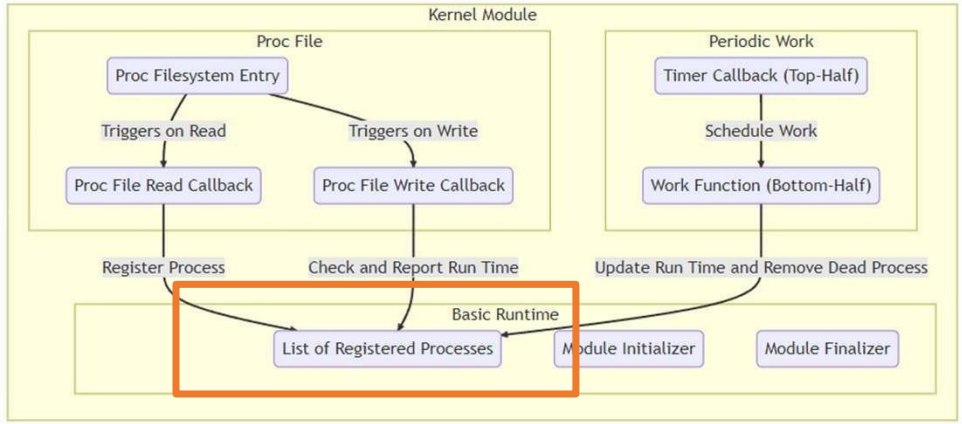
    <PID1>:[space]<CPU time of PID1(decimal)>\n

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



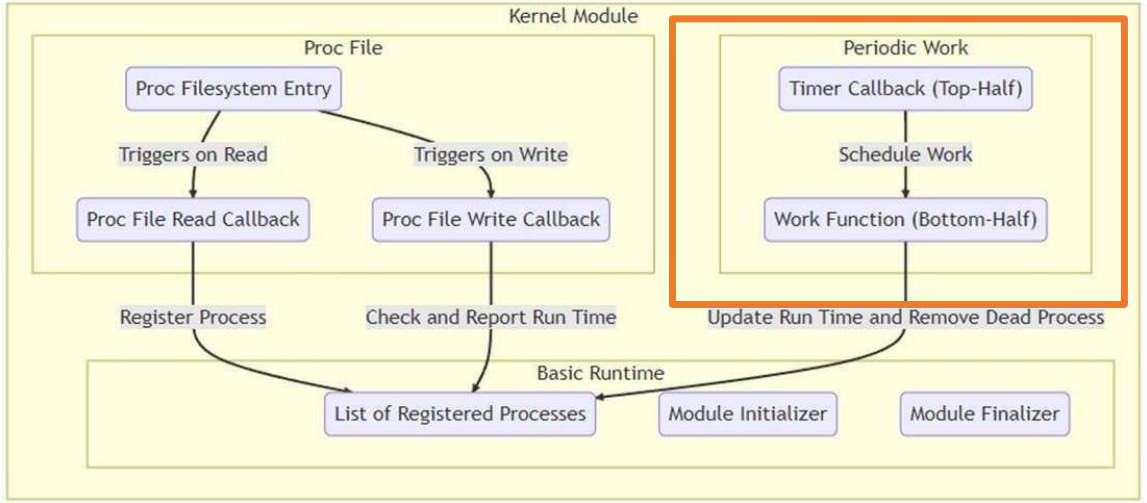
# Store States

- 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



# Periodic work: timer

- Use a kernel timer to perform a task after a preset timeout
  - APIs in <linux/timer.h>
- Setup timer
  - `timer_setup(timer, callback, flags)`
- Setup timeout
  - Timeout is represented in jiffy in kernel. Jiffy can be converted between regular time units (s, ms, etc.)
  - `mod_timer(timer, expires)`
- Challenge: timer only fire once



# Context: Interrupt Handler

## Two Halves

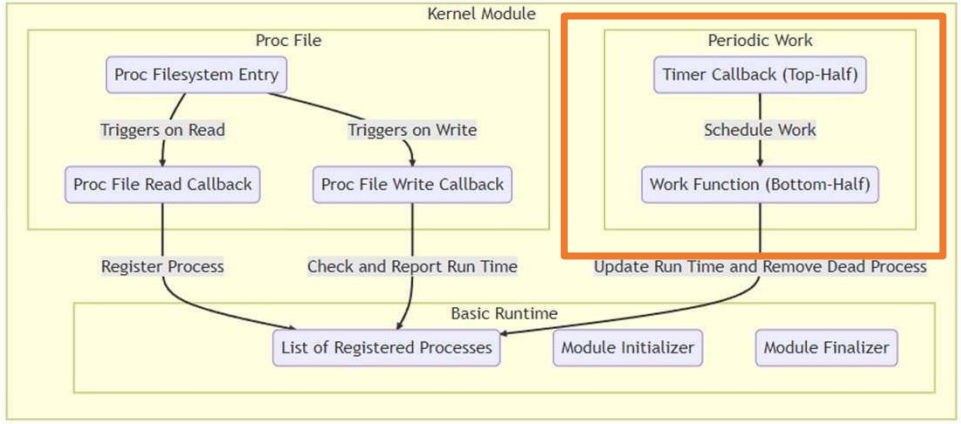
- Interrupts have the highest priority (compared to user and kernel threads) and therefore run first, i.e. the “top half”
  - Must be minimal
  - Cannot block
- All other processing related work must be deferred, the “Bottom Half”
  - Globally serialized

## Work Queues

- Work deferred to its own thread
  - Does not run in interrupt concept
- Can be scheduled together with other threads according to priorities set by a scheduling policy
- Associated with its thread control block and hence can be block (and save context)
  - `DECLARE_WORK();`
  - `INIT_WORK();`
  - `schedule_work();`

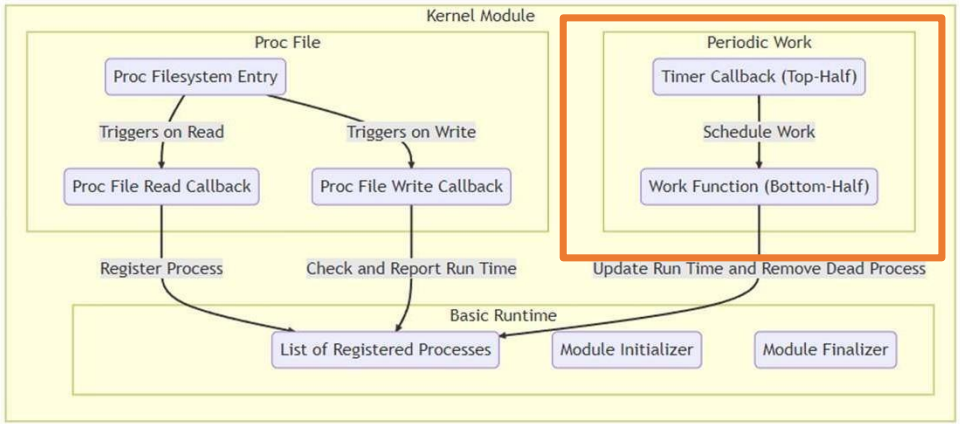
# Periodic work: two-halves

- Not put all work in timer handler
  - Why?
  - Registered list can be long. Better not to block other timers
- Use a two-halves approach
- Use kernel work queue
  - allow kernel functions to be 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)
- Submission
  - Push your code to your GitHub repo before ddl