MP1Walkthrough

Gabriella Xue
02/10

Credits: Siyuan Chai, Jinghao Jia

AI Policy

- Feel free to use AI to understand Linux code, however, don't have it do your assignment for you!

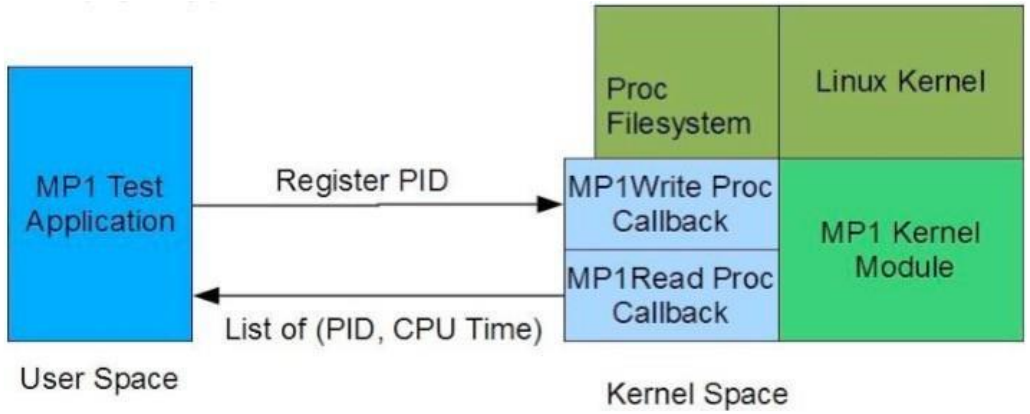
- (Each MP) 20% of the students will be randomly selected to explain their submitted code pieces during OH
  - Fail to explain properly will impact your grade.
  - Totally random: even you got picked for MP1, you can still be picked by subsequent MPs.
  - Midterms/final also include MP related questions.

Get Starter Code

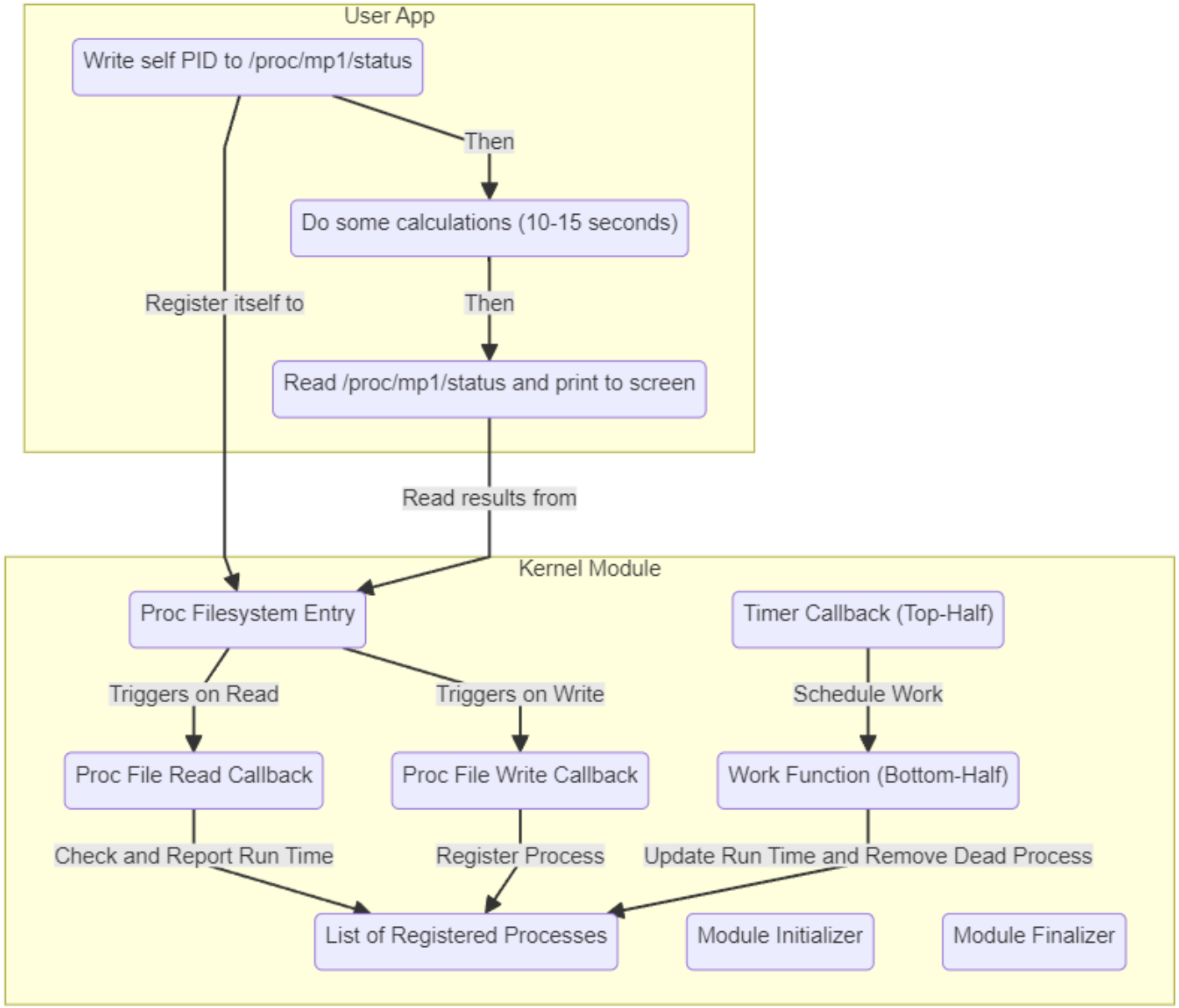
- https://classroom.github.com/a/JnHSoo3j
- Find your name and click (don’t click on other’s name!)
- Due Feb. 25th at 11:59 PM CT

Problem Description

- Write a kernel module that measures the user space CPU time of processes registered within the kernel module
- Register processes using PID through the Proc Filesystem
- Kernel module updates the user space CPU time of each registered process periodically
- Print the userspace CPU time of each registered process

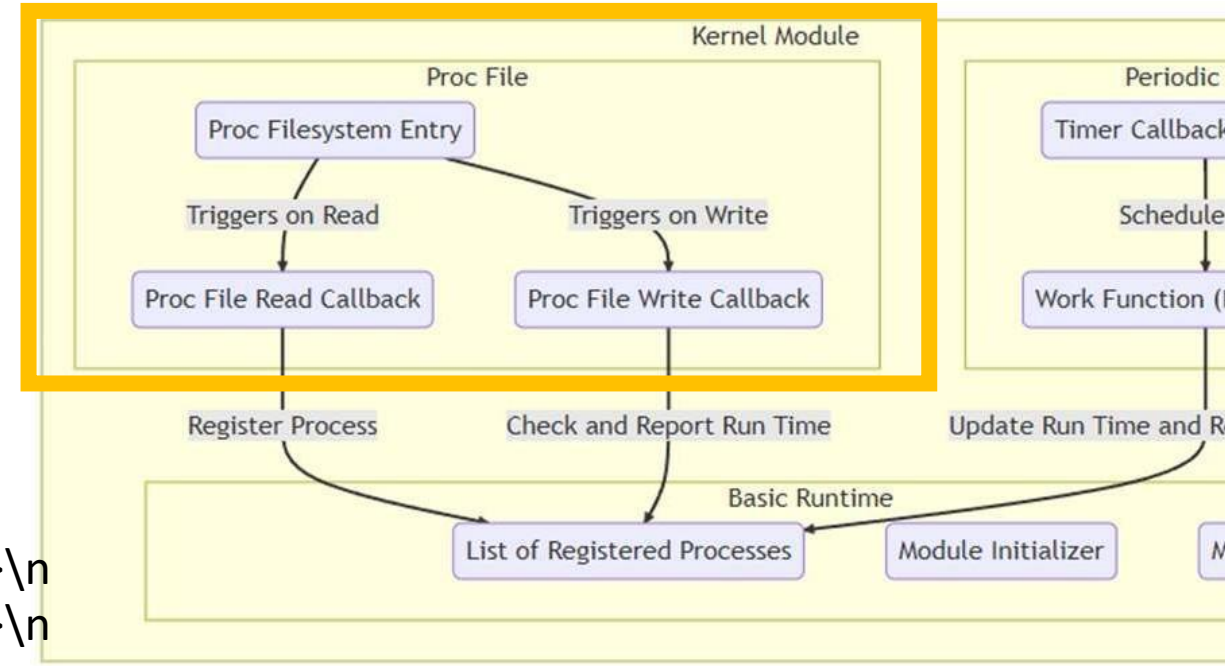


Overview


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)
Check and Report Run Time
Register Process
Update Run Time and Remove Dead Process
List of Registered Processes
Module Initializer
Module Finalizer

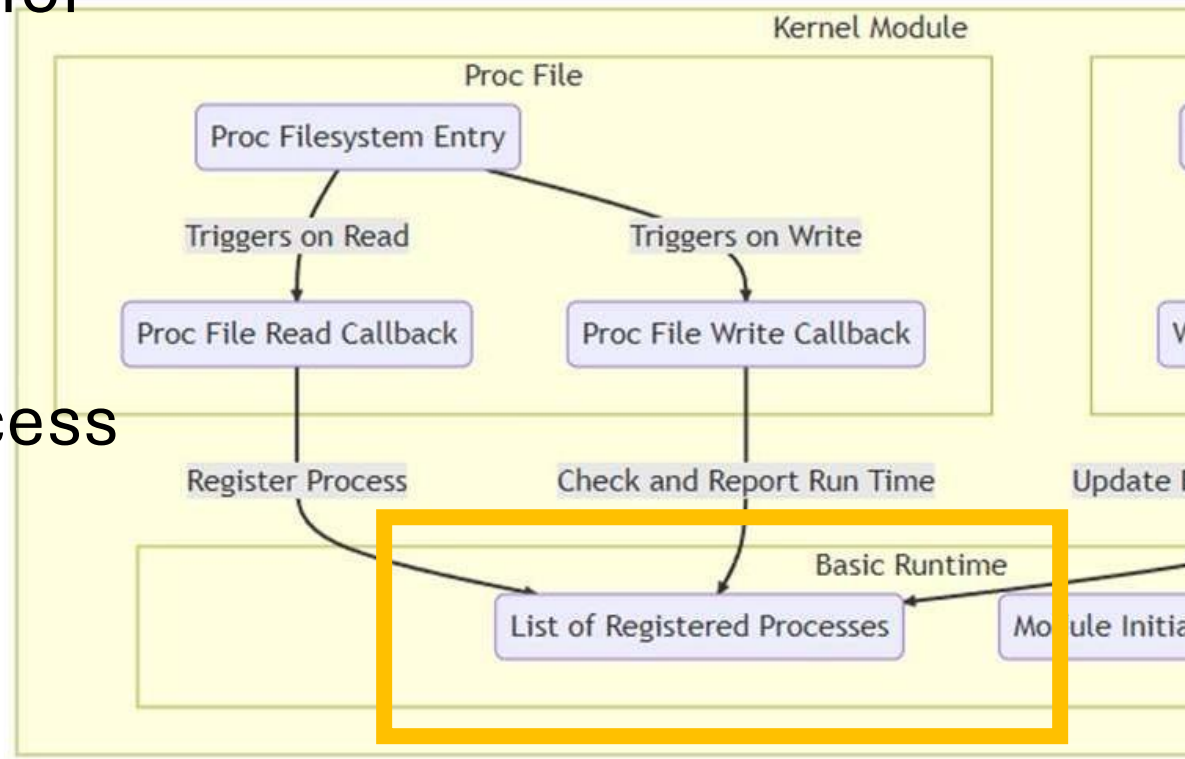
Proc Filesystem Entry

- Not regular files, does not store data on disk
- 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

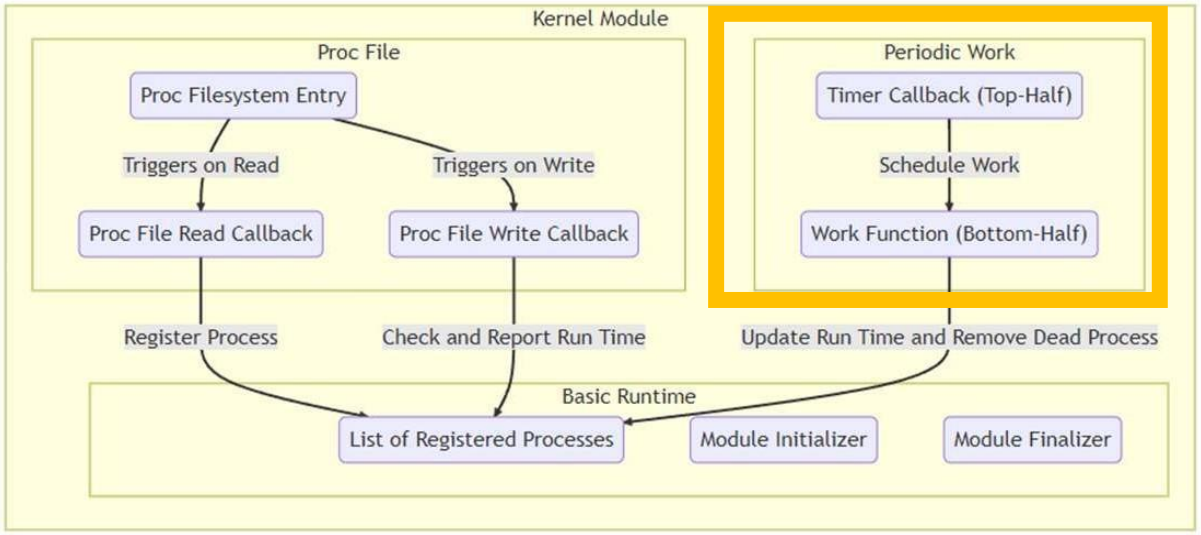


Store States

- 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 mutex



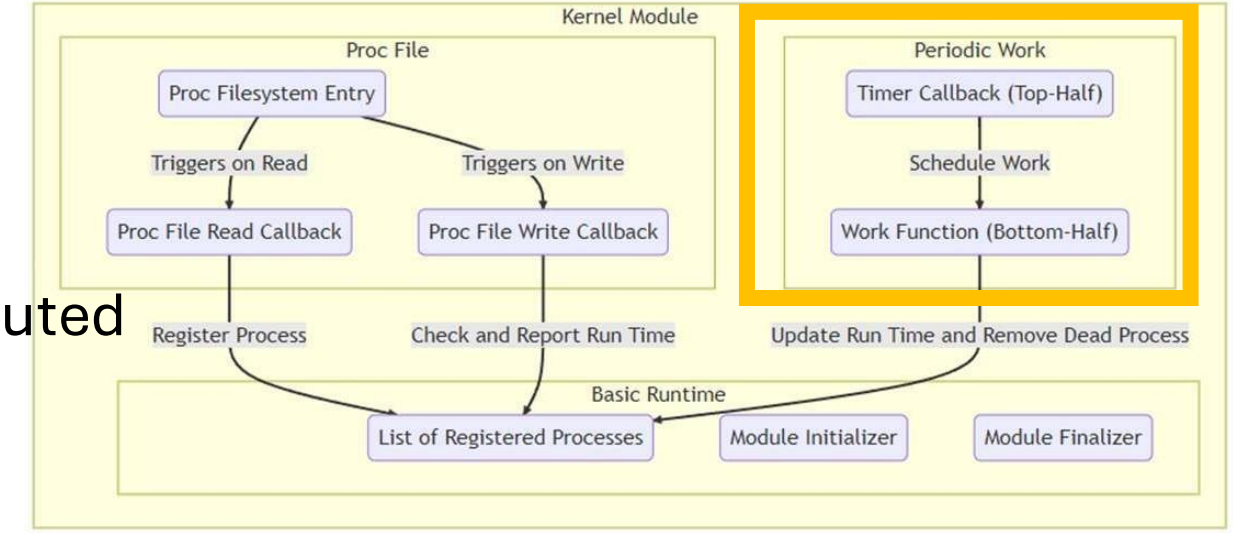
Periodic work: timer



- 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

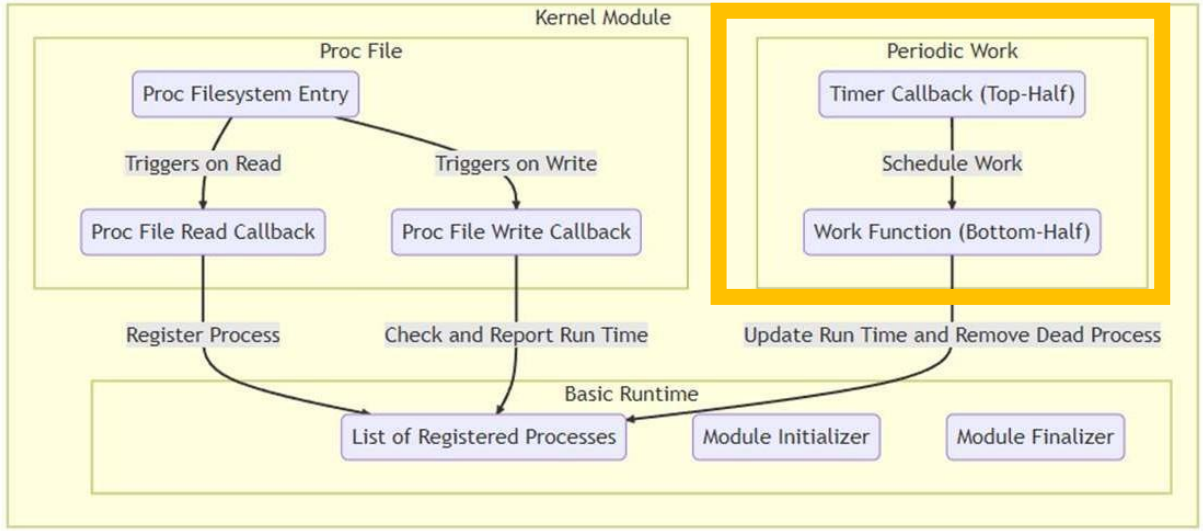
Periodic work: two-halves

- Not put all work in timer handler
  - Why?
  - Registered list can be long. Better not to not 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)



Passing Data

- Passing data between kernel space and user space
  - E.g. ssize_t proc_read(struct file *file, char user *buf, size_t size, loff_t *loff)
  - but here is a user space 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

Run Your Code (Demo)

1. Update makefile
2. Update module_author line
3. Run makefile
4. Start qemu script
5. Insert kernel module (insmod)
6. Run user program to register process
7. listing current processes and user times
8. Remove kernel module (rmmod)

Note: step 3-8 is for testing your code

Debug and Submission

- 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
- Use gdb
  - A bit tricky to load the symbol table for kernel module. You can come to office hour for extra help
- Submission
  - Push your code to your GitHub repo before ddl
  - Making incremental submissions well before the final deadline