MP3: Virtual Memory Page Fault Profiler

Peizhe Liu

CS 423: Operating System Design

Fall 2025

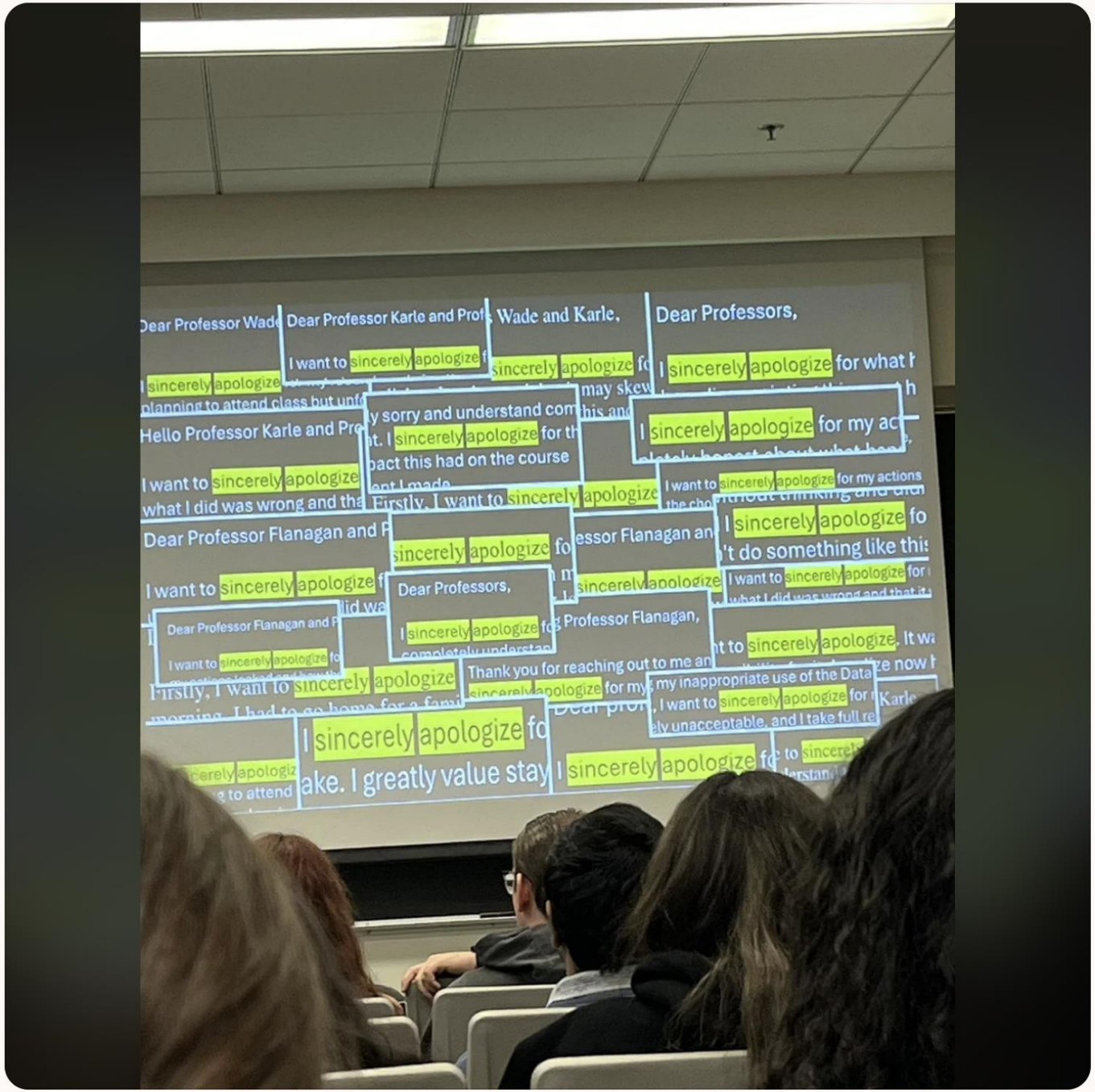
Important Dates

- **MP3 is released today.**
- **The due date is 11/20.**
- **MP3 will likely cost you several days.**
- **NO LATE SUBMISSION. We found that GitHub allows it, but we won’t.**

Important Reminders

- **You must complete this MP individually.**
- **Copy & Paste from whatever AI is a severe academic integrity violation. (Student Code 1-402)**

everybody apologizing for cheating with chatgpt



Goals

- Understand Linux paging management.
- Develop a kernel PF profiler tool.
- Work with your profile tool to analyze PF and its effect on CPU usage.
- Utilize more kernel APIs, like char device, vmalloc, paging.

What You Need

- Your MP0 environment.
- VSCode+clangd setup (strongly recommended).
- Instructions on the course website.
  - https://github.com/cs423-uiuc/cs423-uiuc.github.io/blob/master/fall25/mps/MP3.md
- Accept your assignment on GitHub classroom and start right away.
  - https://classroom.github.com/a/V5g52O1A

Major and Minor PF

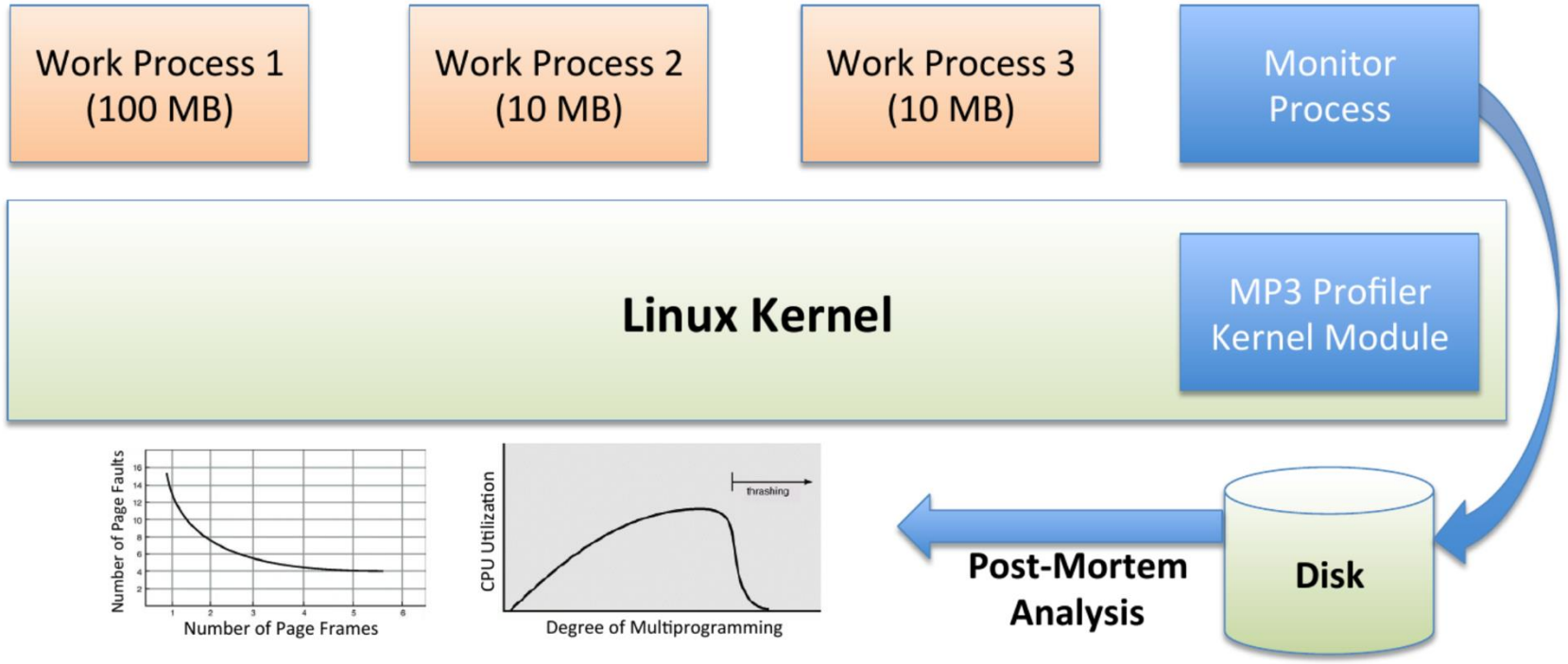
- What is a Page Fault?
- What is a Minor Page Fault?
- What is a Major Page Fault?
- How do various PFs affect CPU usage?

MP3 Profiler

- Can profile multiple programs.
- Provides: jiffies, minor PF count, major PF count, and CPU usage data.
- Read out the data using a monitor.
- Analyze the PF and how it affects CPU usage.

MP3 Profiler

- Can profile multiple work programs (provided).
- Captures: jiffies, minor PF count, major PF count, and CPU usage data.
- Save the statistics using a monitor program (provided).



Register, De-Register, and Process List

- **Register: worker registers itself with its PID via proc write.** `R <PID>`
- **De-register: worker has finished and de-registers itself via proc write.** `U <PID>`
- **MP1-style process list via proc read.**

Profiling PF

- Use a delayed workqueue
- Profile at 20 Hz (we provide a function):

```
struct kbuf_sample {
    unsigned long jiffies;
    unsigned long minfault;
    unsigned long majfault;
    unsigned long cpu;
};
```

- However, these statistics are only available in kernel (read via procfs?).
- We want to avoid overheads crossing the kernel boundary.

Buffer Sharing

- We can create and populate a buffer in kernel.
- Map the buffer back to user.
- So, the buffer is shared between kernel and user.

Creating the Buffer

- **Must be at least 512KB (we force you to use `vmalloc()`).**
- **Must not be swapped to hard disk (reserved bit).**
- **Kernel populates it, while monitor program reads it.**

Populating the Buffer

- We make the buffer a queue.
- Must contains 12000 samples.
- For each sample:

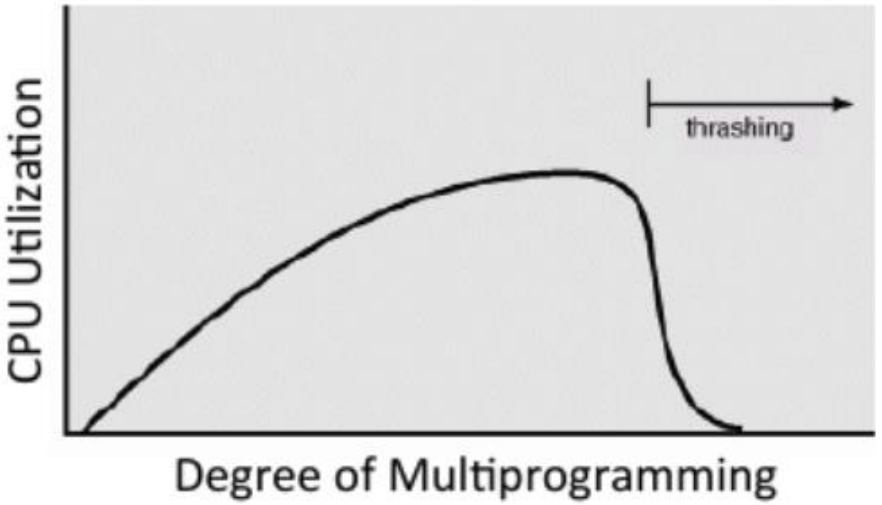
```
struct kbuf_sample {
    unsigned long jiffies;
    unsigned long minfault;
    unsigned long majfault;
    unsigned long cpu;
};
```

Mapping the Buffer

- Implement `mmap`
  - procfs, which everyone is very familiar with, does not support it!
- Implement a character device.
- Remap buffer pages to user page table.
  - Does kernel still have access?
  - Be careful that pages are not continuous!

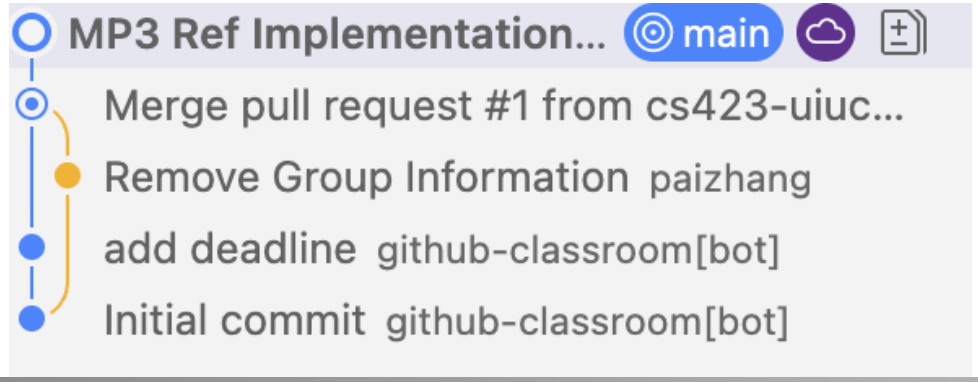
Case Study

- We provide the monitor and work program.
- You will be required to do some case studies.
- CPU Utilization: $U_T = \frac{cpu\ time_T}{wall\ time} = \frac{stime_T + utime_T}{jiffies}$



Pro Tips

- **Good tools can be very helpful.**
- **Try conventional commits.**
- **Source code is your best teacher.**
- **Start right away!**




```
413     module_init(mp3_init);
414     module_exit(mp3_exit);
415
```

Total: ~400 lines