Welcome |baaaack| to CS439H!

"It was the scariest day of the year. Friday the 13th, on Halloween."

Stress

- 439H is **not an easy class**
  - Lots of new material
  - Unfamiliar programming environments
  - Fast, often relentless pace
- Struggling in this course is normal
  - There will be times you won't know the answer or solution
  - This is expected - we want everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overwhelmed or spending more time on this class than you think you should be, **please *reach out*** to Dr. Gheith or the TAs
  - We can help out as far as the class goes
  - We can provide other resources if we are not able to help

Mental health resources available at UT

Quiz everybody say VMMMMMMM_ON

```
write(
   1,
   feedback,
   n
);
```

How was the quiz?

A. easy
B. mostly fine
C. mostly fine, but not enough time
D. too hard, but finished mostly in time
E. too hard and not enough time
F. too hard regardless of time

```
fork();
```

How is p6 going?

A. that’s a thing?
B. Cloned the project.
C. Looked through the starter code.
D. Started planning/writing code
E. Done with at least one part of the project
F. Successfully segfaulted t0 but still failing a couple test cases
G. Any% p6 speedrun glitched

Interrupts, more VM, and forks

Interrupts

- What are interrupts?

Interrupts

- What are interrupts?
  - Some sort of exceptional situation occurs during execution, or some hardware device is ready/needs attention/etc
  - The process will **interrupt the current instruction stream** and start executing things based on an **interrupt handler**
  - When the interrupt is handled, we resume the previous execution context
  - We already have two interrupts in our code
    - pit.h/pit.cc: timer interrupts
    - sys.h/sys.cc: syscalls! (int $48)
  - We are getting a third interrupt in p6
    - vmm.h/vmm.cc: page faults

Interrupts

- How are interrupts implemented?
  - Some sort of situation occurs that would cause an interrupt to be signaled
    - Timer: the timer hits its next tick
    - Syscall: user process invokes the interrupt from software
    - Page fault: some virtual address fails translation for some reason
  - This signal reaches the processor and the PC is changed based on the interrupt handler read from the IDT
    - **Interrupt Descriptor Table**: a table that sets up a bunch of pointers for the handlers of different interrupt types

Interrupts

- How are interrupts implemented?
  - Which instructions get to commit and which are flushed?
    - For interrupts that happen based on instruction execution (e.g. page faults, syscalls), execution rolls back to right before that instruction (for a page fault) or right after (syscall)
      - Page fault will rerun the instruction, syscall will not (for practical purposes)
    - For interrupts that happen asynchronously (e.g. timer interrupts), execution pauses at some arbitrary point
    - Uses the same hardware mechanisms that we have for rolling back mispredicted branches or other failed speculative execution

Interrupts

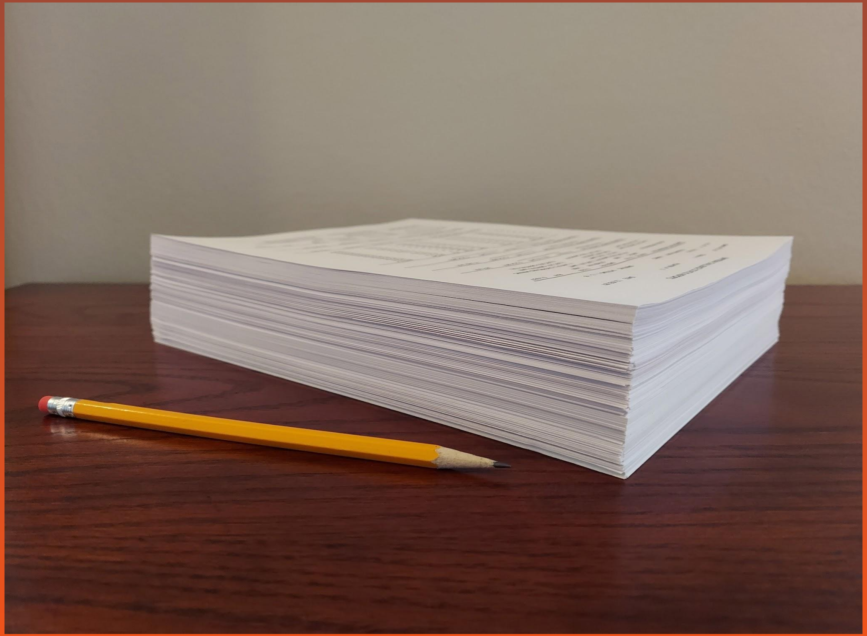
- What happens before the handler starts running?
  - The processor saves necessary state before switching context like this
    - Instruction pointer - need to know where to return to!
    - On our architecture, we also save the stack pointer as well as some segmentation registers
      - If we came from user mode, we switch onto a kernel stack
    - On our architecture, this state is **pushed to the stack**
      - `uint32_t *frame`
  - For some interrupts (e.g. page faults) where the processor knows something about *why* the interrupt was triggered, we get extra state
    - On our architecture, we have a status code of what type of memory access triggered the page fault (read vs write, user vs kernel, etc.) as well as the attempted virtual address, pushed to the stack

Interrupts

- How do we get back to where we were before the interrupt?
  - The special instruction `iret` reads the pushed state from the stack and jumps back to normal execution
  - Returns to whichever instruction should execute next, based on what was committed or not
    - For a page fault, this will automatically retry the memory access
    - For a syscall, this returns to right after the user triggered the syscall
  - We can abuse `iret` to do one very helpful thing
    - You've looked at `switchToUser`, right?

Demand paging

- What is demand paging?



Demand paging

- What is demand paging?
  - One trick we can pull off with virtual memory: track when someone uses a page
  - **Demand paging** is only putting stuff in memory and allocating space **when the process uses it**
    - So, the process won't actually be using physical memory/reading from disk at the start
  - We can lie to the process and act like all its memory is ready, when in reality nothing is allocated
  - When the process tries to access a page we haven't set up, we get notified! (page faults)
    - Then we can load the appropriate data and allocate space only when necessary
  - Especially for large regions of memory, this is much more efficient if we only plan to use sparse parts of it
  - Very easy to mess up! What happens if you don't serve the page properly?
    - Infinite loop, we get stuck page faulting forever

Fork

- What is fork?

Fork

- What is fork?
  - One process duplicates itself into two (almost) identical copies
  - The copies run **independently** afterwards
  - How can we leverage the mechanisms we have to implement this?

Fork

- What is fork?
  - One process duplicates itself into two (almost) identical copies
  - The copies run **independently** afterwards
  - How can we leverage the mechanisms we have to implement this?
    - Duplicate the actual data in physical memory
    - Make a new page directory + page tables setup that uses the copy of the data instead of the original version
    - Can we be more efficient?

Fork

- What is fork?
  - One process duplicates itself into two (almost) identical copies
  - The copies run **independently** afterwards
  - How can we leverage the mechanisms we have to implement this?
    - Duplicate the actual data in physical memory
    - Make a new page directory + page tables setup that uses the copy of the data instead of the original version
    - Can we be more efficient?
      - Copy-on-write forking
      - vfork

Fork

```
printf("*** hello\n");

int x = fork();

if (x < 0) printf("*** fork failed\n");

else if (x == 0) printf("*** child\n")

else printf("*** parent\n");
```

Fork

```
printf("*** hello\n");

int x = fork();

if (x < 0) printf("*** fork failed\n");

else if (x == 0) printf("*** child\n")

else printf("*** parent\n");
```

Output:
*** hello
*** child
*** parent

OR

*** hello
*** parent
*** child

(or, if forking fails for some reason)

*** hello
*** fork failed

Fork

- Why fork?
  - Not just useful to clone yourself, but also to run new processes
  - Example: `sh` wants to run `ls` without stopping its own execution
  - `fork(); if child execl("ls"); else do other stuff;`
  - The standard way of running other processes on POSIX-based systems
- Other ways?
  - Windows: `CreateProcess`
  - Very complicated, have to pass in lots of parameters specifying the process
  - Easier to just inherit process details from the current process (e.g. permissions)

Questions?~

```
*** Don't panic
***
***
***              oooo$$$$$$$$$$$$oooo
***          oo$$$$$$$$$$$$$$$$$$$$$$$$o
***       oo$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$o         o$   $$ o$
***   o $ oo        o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$o       $$ $$ $$o$
***oo $ $ "$      o$$$$$$$$$    $$$$$$$$$$$$$    $$$$$$$$$o       $$$o$$o$
***"$$$$$$o$     o$$$$$$$$$      $$$$$$$$$$$      $$$$$$$$$$o    $$$$$$$$
***  $$$$$$$    $$$$$$$$$$$      $$$$$$$$$$$      $$$$$$$$$$$$$$$$$$$$$$$
***  $$$$$$$$$$$$$$$$$$$$$$$    $$$$$$$$$$$$$    $$$$$$$$$$$$$$  """$$$
***   "$$$""""$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     "$$$
***    $$$   o$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     "$$$o
***   o$$"   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$       $$$o
***   $$$    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$" "$$$$$$ooooo$$$$o
***  o$$$oooo$$$$$  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$   o$$$$$$$$$$$$$$$$$
***  $$$$$$$$"$$$$   $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$     $$$$""""""""""""
*** """"       $$$$    "$$$$$$$$$$$$$$$$$$$$$$$$$$$$"      o$$$
***           "$$$o     """$$$$$$$$$$$$$$$$$$"$$"         $$$
***             $$$o          "$$""$$$$$$""""           o$$$
***              $$$$o                                o$$$"
***               "$$$$o      o$$$$$$o"$$$$o        o$$$$
***                 "$$$$$oo     ""$$$$o$$$$$o   o$$$$""
***                    ""$$$$$oooo  "$$$o$$$$$$$$$"""
***                       ""$$$$$$$oo $$$$$$$$$$
***                               """"$$$$$$$$$$$
***                                   $$$$$$$$$$$$
***                                    $$$$$$$$$$"
***                                     "$$$""""
***
```