Welcome bbaacckk to CS439H!

No quiz everybody say !

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

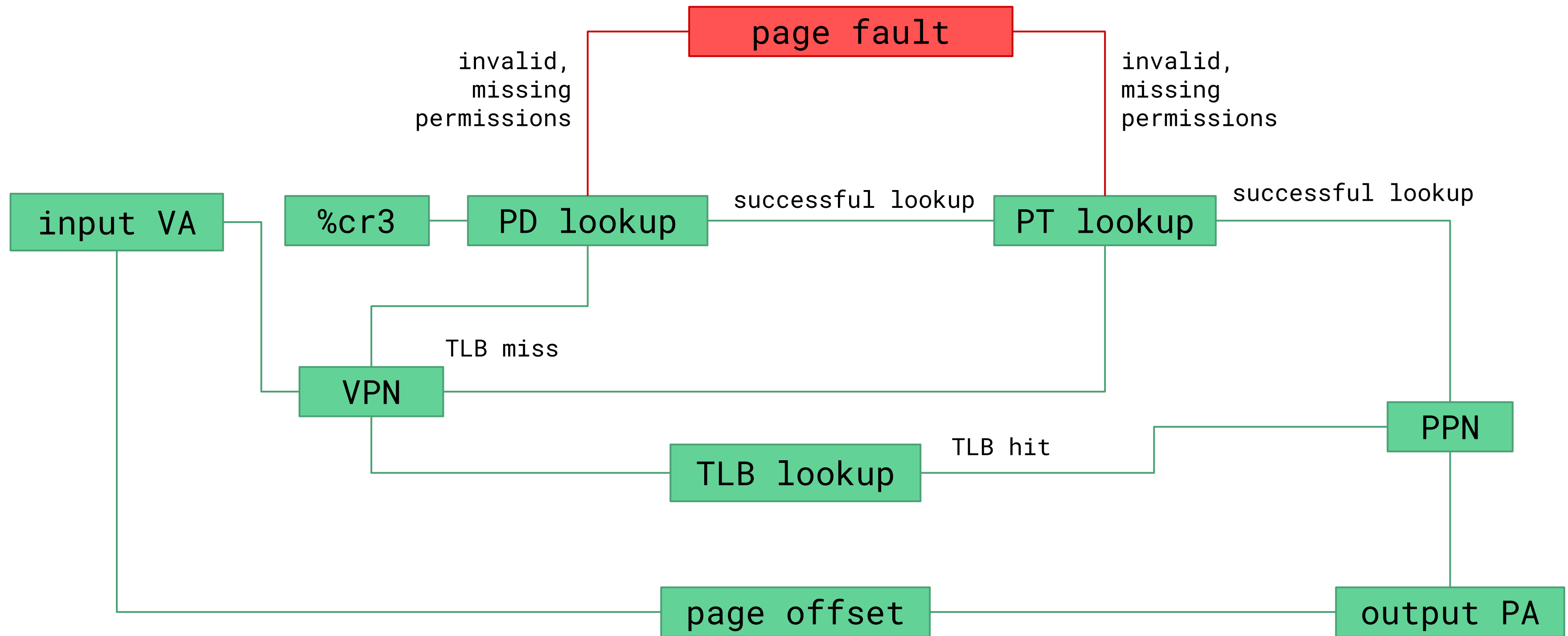
Mental health resources available at UT

```
for (int i = 0; i <
NUM_STUDENTS; i++) {
  int id = fork;
  if (id > 0)
    get_feedback(i);
  else if (id == 0)
    join();
}
```

How is p7 going?

A. that’s a thing?
B. I’ve heard/talked about it
C. Cloned the project.
D. Looked through the starter code.
E. Started planning/writing code
F. Done with at least one part of the project
G. Done with the whole project but still failing a couple test cases
H. Fully preempting

Question 1



Question 2

Large page sizes

- Flatter tree (less depth)
- Faster lookup times

Segmentation

- Less metadata

Larger virtual address spaces

- Can access more memory

Tiny page sizes

- Reduce fragmentation

Paging

- Less fragmentation

Smaller virtual address spaces

- Faster lookup times

Question 3

Solution: reference counting!

- We didn't allow you to change argument types because a one line change from Node* to Shared<Node> as the argument type would work
- Pass an Atomic<int> (by reference or pointer, it should be stored on the heap) counting the number of references to node
- Increment **when the go call is scheduled** (when node is captured into the lambda; incrementing inside func() is too late) and decrement after func()
- Free node when the counter hits 0

```
void decrement(Node* node, Atomic<int>* ref_count){
  if(ref_count->add_fetch(-1) == 0){
    delete node;
  }
}

void func(Node* node, int i, Atomic<int>* ref_count) {
  if (i < 0) return;

  ref_count->add_fetch(1);
  go([node, i] { func(node, i - 1); decrement(node, ref_count); });
  if (random_bool()) {
     ref_count->add_fetch(1);
     go([node, i] { func(node, i - 1); decrement(node, ref_count); })
  };
}

void kernelMain() {
  // ... do work ...
  Node* node = // ... get node from the filesystem ...
  Atomic<int>* ref_count = new Atomic<int>(1);
  func(node, random(), ref_count);
  // ... do more work ...
  decrement(node, ref_count);
}
```

Question 4

Two things:

1. Move program to FS
2. Program Rebooting

Move to FS:

Load following into a separate portion of the FS: Stack,Page Directories and Page Table, and registers (that include the IP and SP). Move it back to user program)

Program Rebooting:

We restore everything that was once saved, and switch to that process.

Note that there needs to be some agreed upon saving/restoring convention.

P7

Context Switching (Turing)

- Remember coroutines?
- Consider how you'd implement `yield()` as a system call
    - Need a mechanism for **saving the state of a process**
    - and a mechanism for **restoring the state of a process**
    - Then you can save the state, and schedule something to switch back into the process at a later point
- In p6, we just used `switchToUser()` to "restore" the PC/stack pointer in fork
- What else do you have to save & restore now?

yield() - Optional, but we really recommend it

- `void yield()` (Syscall number 998)
  - Suspends the current process and resumes it at some future point
  - (by putting something on the event queue to resume it)

- Start here - it's the simplest context switching operation
- Add yield calls to p6 t0's spin loops, and you should pass it with QEMU_SMP=1 (one core)
- Implementing this lets you test your context switching, making preemption much easier to debug (because you'll know the context switching works)
- Once you have yield, `sem_down` is simple to implement

Preemption

- `apitHandler()`
  - Called every time the PIT triggers a timer interrupt (approximately once every ms)
  - By default, just increments `Jiffies`
  - We can make it preempt the current process (if you're in user-mode) and switch contexts
  - This looks a bit like `yield()`... (but not exactly)

join()

- `int join()`
  - **Blocks the calling process** until the *most recently created* child exits
  - Returns the exit code from the child
    - Exit code is the argument passed into `exit`
    - 139 if the child terminated because of an unhandled page fault
  - Returns -1 if the process has no remaining children
- Each process maintains a LIFO stack of children
- When a process `forks`, the child is added to the parent's stack of children

Semaphores, again

- `unsigned int sem(unsigned int n)`
  - Creates a semaphore initialized to a count of `n`, and returns a corresponding number to refer to the semaphore
- `void up(unsigned int sem)`
- `void down(unsigned int sem)`
  - Performs the corresponding operation on the corresponding semaphore
  - For this project's tests, `sem` **must be a value returned from the `sem` syscall**
  - `down` will **block the user process** until `up` is called (like normal semaphores)

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$$$$$$$$$"""
***                            ""$$$$$$$oo $$$$$$$$$$
***                                    """"$$$$$$$$$$$
***                                        $$$$$$$$$$$$
***                                         $$$$$$$$$$"
***                                          "$$$""""
```