CSE Section 451

Corrections

- You don't need to resize your hash tables.
- To clarify part 3, you only need to implement the functions in the enum described in the .h. Not MEAN, which is sort of mentioned in the write-up.
- To clarify part 1, you do not need to submit tests showing the bugs in the code.

Project 0

- How is it going?
- What's happening here?

  `key == ht[loc]->key`
- Bad! Comparing pointers.
  - Need a “deep equals”
  - Must pass in an equality function
  - Worry about it for this project
- ~~Also, don’t forget about resizing!~~
- ~~And memory management~~
- ~~And memory leaks~~
- And edge cases
- And fun

Part 3. What not to do.

If you do this:

```
if (mathfunctype_t == ADD) {
  ret = add(...);
} else if …
```

You're doing it wrong. Please do not do this.

Project 0 Questions?

Always feel free to ask questions.
Concept question are what we want you to learn.
C problems are what we want to help you get past.

How is the class going?

Shells

- Primary Responsibilities:
  - Parse user commands
  - Execute commands / programs
  - Manage input and output streams
  - Job control

The Unix Shell

- Internal commands
  - Built-in commands. Executes routines in the shell.
  - Manages state of the shell.
- External commands
  - Everything else like cp, ls, ln, cat, etc.

  How can you tell the difference? External commands follow fork/exec.

Other capabilities

- Redirect standard input / output / error

  # ./parser < logfile > outfile 2> errfile
- Command pipelines

  # ps –ef | grep java | awk ‘{print $2}’
- Background execution

  # time make > make.out &

  # jobs

  [1]+ Running time make > make.out &

The CSE451 shell

- Print out prompt
- Accept input
- Parse input
- If built-in command
  - do it directly
- Else spawn new process
  - Launch specified program
  - Wait for it to finish
- Repeat

```
CSE451Shell% /bin/date
Fri Jan 16 00:05:39 PST 2004
CSE451Shell% pwd
/root
CSE451Shell% cd /
CSE451Shell% pwd
/
CSE451Shell% exit
```

Fork and Wait

```
short pid;
if ((pid = fork()) == 0) {
/* some code is here for the child to execute */
    exit(n);
} else {
    int ReturnCode;
    while(pid != wait(&ReturnCode))
    ;
    /* the child has terminated with ReturnCode as its
    return code */
}
```

What is fork returning? What is wait really doing?

Mommy, where do processes go when they die?

- What happens to a Process Control Block when a process completes?
  - What happens to its data? What if the data is needed?
- When a process completes, it goes into a ZOMBIE state.
  - It's PCB isn't reclaimed or cleaned until the parent can check its status using wait() (or waitpid() or wait id())

Wait

```
pid_t wait(int *status);

pid_t waitpid(pid_t pid, int *status, int options);

int waitid(idtype_t idtype, id_t id, siginfo_t
*infop, int options);
```

Exec

```
int pid;
if ((pid = fork()) == 0) {
/* we'll soon discuss what might take place before exec
is called */
    execl("/home/twd/bin/primes", "primes", "300", 0);
    exit(1);
}
/* parent continues here */
while(pid != wait(0))
;
```

What happens to the process in exec? What is exit() called with an error?

Exec

- The process is complete overwritten (code, data, everything) and loaded in its place is the binary of the program.
- Why is exit an error then?
  - Because the line of code should no longer exist if exec behaves correctly. If exec ever returns, it is because it failed.

Fork/Exec

- Fork is THE way in Linux to create processes.
- Fork is commonly followed by exec.
  - Does that fire any red flags? Why are we copying a process only to destroy it? Isn't that horribly inefficient?

    Fork/exec have both been optimized to not be terrible, particularly in the case they are used together. More on that when we talk about virtual memory.