Operating Systems
ECE344

Lab assignment 2: System calls

Ding Yuan

Overview of the lab assignment

- So far, we have been entirely in kernel-mode
- Tasks: implementing system calls so we have user-level processes!
  - fork() – most challenging
  - execve() --- challenging
  - waitpid() --- challenging
  - getpid(), _exit(), a printing syscall --- straightforward
- Demo:
  - p /testbin/add

Challenges

- *This is a very challenging lab (much harder than lab 1)*
  - Transition between user/kernel mode
    - How to pass parameters to kernel? return value to user?
    - How to initialize user address space?
    - How to save/restore execution context of user-process?
  - How to fork()?
  - How to waitpid()?

Things to watch-out

- Concurrency
  - *Need to protect shared data-structure with lock!*
  - *Depends on lab 1*
- Moving data across user-kernel boundary is non-trivial
  - Need to read lots of code to understand it
- Debugging
  - cs161-gdb cannot debug user-level code
    - And at the beginning, printf() doesn't work either!
  - Debugging data-race can be a nightmare

Advices

- **Start early!**
- Read code
  - Follow the guidance of the handout
    - Answer those questions first
- Think, think, think
  - and then start coding
  - *Because, again, debugging this lab can be a nightmare*
    - So you want to minimize your debugging time!