CS614: Linux Kernel Programming

Linux Source Study: clone, exec

Debadatta Mishra, CSE, IIT Kanpur

Clone: Implementation in Linux kernel

- Syscall handler for clone should provide flexible sharing. Implementation?
  - Syscall Handler $\longrightarrow$ Kernel clone $\longrightarrow$ Copy process
  - Depending on flags, different subsystems are copied or shared
- Depending on the usage, the saved user state is required to be changed. Why? How implemented?
  - For pthreads, the SP and RIP need to be changed
  - Change the register states during CPU thread copy
- Changes to the kernel space of newly created execution context required. Why? How implemented?
  - Child can not return in the same path, returns through a special stub

Exec: Implementation in Linux kernel

- When should the self destruction of address space take place? What are the design choices?
  - Can not destroy until validity is checked; validity check not complete until the binary/arguments are examined
  - Duplicated processing vs. working with a fresh (discardable) space
  - There would be a point of no return, delayed is better
- How does the kernel parse the binary (and deduce entry address)? What about command line arguments?
  - Basic binary parsing for ELF (and other types) e.g., load_elf_binary ( )
  - Command line arguments are placed in the stack

Code walkthrough (Quiz-3)

- Perform code study to find and explain working of the following features. Need to write a brief explanation of the working giving exact code locations (function names and line numbers)
    - Q1: Mention the exact functions where the temporary (discardable) “mm” is allocated and where the calling process “mm” is replaced with the newly built “mm”?
    - Q2: Explain the code for temporary (discardable) “mm” allocation and initialization? Specifically, explain how much stack is allocated initially (and why) and how is it resized? Which flag is used to suggest if the stack is allocated in its final form?
    - Q3: Explain with reference to kernel code the working of “exec” for a child process created using “vfork”. Where does parent wait? How is it scheduled again? What happens to the shared memory address space?