Linux Internals
Day 1- Morning
A Tour of Userspace

Joel Isaacson
Ascender Technologies Ltd

Today's Lectures

- Today we will have two sessions
  - Morning 10:00-12:30
  - Afternoon 14:30-17:00
- I have tried to retain some flexibility and interactivity.
- If you have any requests about the next lecture, please speak to me.

Misconceptions of Unix Users

- “Linux is just Unix spelled differently”
- Users that are used to Unix systems will be surprised that Linux can be quite different from the older Unix systems.
- Linux is a more modern version of Unix, based on the Posix standard.
- It is simply more advanced and better than the commercial Unixes.

Linux Strong Simple Abstractions

- Linux tries to use a minimal number of concepts and extends their semantics to provide s synergistic effect.
- Many concepts from non-Posix operating systems such as Plan9 have been incorporated into Linux.

Linux Strong Simple Abstractions

- Files
- Processes
- Memory Spaces
- IPC

Files

- The hierarchical file systems is fundamental to Unix-Posix systems.
- The well know open-read-write-close paradigm is used.
- We shall see that Linux has developed file abstractions far more sophisticated than early Unix systems.

Processes

- Processes contain the basic states of computation.
- Linux has a very efficient process model.
- Linux does not have an independent LWP (Light Weight Process – threading) model.
- Threads are processes.

Memory Spaces

- Memory spaces provides the glue that bind Files and Processes.
- They are usually associated with a process and contain memory areas that can map:
    - Files
    - Shared Memory
    - Device Memory

Memory Spaces Continued

- A memory area might have:
  - COW (Copy On Write – Private) semantics.
  - Shared memory semantics

IPC
Inter-Process Communication

- Linux's IPC is based on the BSD socket API.
- It works both within one computer (Unix domain sockets) and between computer (usually via TCP/IP)

Files: /proc and /sys file systems

- The /proc file system was pioneered in Bell Labs influential but not very popular Plan9 operating system.
- The /proc and /sys file system has eliminated thousands of application specific API's.

File Creation

- Files are created via the standard Posix API's
  - creat
  - open
  - mkdir
  - link

Fork/Clone

- The only way to create a process is through the clone (fork) system call.
- The default is to create a process that inherits the parent's processes memory spaces through COW semantics.
  - A thread is created via the clone system call with the CLONE_VM argument.
  - Threads are no more efficient than processes.

exec

- The exec system call is used to create a new memory space.
- The initial memory space right after the exec system call has four memory areas:
    - Text
    - Data
    - Bss
    - Stack

Real Programmers Don't Use

- Threads
- Asynchronous I/O
- Signals
- Semaphores
- Real-time Scheduling
- Unbridled Ioctl's
- Massive Kernel Code Dumping

Threads

- There are very few standard multi-threaded programs under Linux.
- Linux now has very good threaded support.
- It is not impossible to write a multi-threaded nontrivial program. It is just very difficult.
- There are alternatives that for most project will give better results in less time.

Asynchronous I/O

- New to Linux 2.6.
- Not well supported.

Signals

- Signals suffer from the same problems that threads have.
- They can't easily be used for reliable inter-process communications.
- They basically create an interrupt programing model with a need for masking interrupts (signals).

Semaphores

- Semaphores are difficult to use properly.
- Forget to use one and your data gets scrambled.
- Use too many and you risk deadlock.

Real-Time Scheduling

- Posix.4 adds real-time, absolute, fixed priority scheduling.
- The *sched_setscheduler* system call can be used to set either *fifo* or *round robin* scheduling policy.

Real-Time Scheduling

- The problem is that:
- The real-time scheduler doesn't work well with the standard (SCHED_OTHER) scheduler.
- Deadlock and starvation can occur.
- The standard scheduler does a very good job.

Unbridled Ioctl's

- Adding functionality via many poorly thought out ioctl calls creates a system that is hard to use and very unportable.
- It is much better to use higher level abstraction such as the /proc or /sys file system.
- Ascii is great. Down with binary.

Massive Kernel Code Dumping

- Just taking a mass of C code from another system and inserting it into the Linux kernel is asking for trouble.
- Some difficulties are:
  - No virtual memory or process memory space
  - Small stack (4k or 8k)
  - No standard C library
  - Locking, Preemption etc. etc

Linux vs uClinux

- No fork (or clone) only vfork.
- No brk or sbrk, Only mmap (and only with MAP_ANONYMOUS) can be used to allocate memory (not contiguous).
- Fixed size stack.
- No memory managment.

Examples

- /proc filesystem
- Static Linking
- Dynamic Linking
- Light weight processes