On the edge of the Linux kernel and user space

Pavel Šimerda

pavlix@pavlix.net

The kernel and the user space

Code execution contexts

- User space (process) code
- Kernel code in process context
- Interrupt context (preemption)

Preemption

- Source
  - User space
  - Process context
- Destination
  - Interrupt context

Scheduling

- Source
  - Process context (cooperative)
  - Interrupt context (preemptive)
- Destination
  - Process context (of another process)

System call

- Source
  - User space
- Destination
  - Process context

System call return

- Source
  - Process context
- Destination
  - User space

Virtual memory access

User space

- Backed by physical memory
  - Direct access
- Backed by a file
  - Page fault
- Copy on write
  - Page fault
- Page fault handled by the kernel
  - Scheduler triggered
  - Process resumed

Process context

- User memory access
  - `copy_from_user()`
  - `copy_to_user()`
- Page fault handler triggered
  - Scheduler triggered
  - Process resumed (in kernel mode)

Virtual memory layout (x86)

- Process memory space
  - Partially backed by RAM
- Kernel memory range
  - Low memory fully backed
  - High memory mapped explicitly
- Switching modes
  - Supervisor mode
  - User mode

Threading notes

- Tasks
  - Processes (main threads)
  - Other threads
- Shared resources
  - Virtual memory
  - File descriptors
- Distinct resources
  - Execution stacks
  - Signal handlers
  - Thread-local data

Synchronization

Kernel wait queues

- Wait queue
  - Waiting in process context
  - One or more sleeping waiters
  - Woken up and scheduled to run
- Goals
  - Idle tasks
  - Synchronized access (shared resources)
- Example usage
  - Task (children) management
  - Blocking I/O
  - Futexes

Fast user space mutexes

- Futex feature
  - Fast userspace mutex
  - Waiting and waking syscall
  - Requires shared memory word
  - Generalized mutex (semaphore)
- User space operation
  - Atomic word operations
  - Fast decrement when not waiting
  - Fast increment when not waking
  - Waiting and waking syscalls
- Kernel operation
  - Using wait queue
- Library wrappers
  - Mutexes in pthreads
  - POSIX semaphores

Data exchange

Kernel buffers

- Buffers
  - Preallocated
  - Auditable memory usage
- Simple pipe
  - Kernel buffer
  - `write()`
  - `read()`
  - Signalling lost reader
- Complex mechanisms
  - Message queues
  - Sockets

Shared memory

- Shared memory pages
  - Multiple processes
  - Total optimization
  - Shared via file system
  - Preallocated
- Drawbacks
  - No protection between processes
  - Synchronization needed (in user space)

Bonus: Message passing optimizations

- Temporary shared memory buffers
  - Requires a side channel
- Skipping the kernel buffer
  - Only useful for large payloads
  - Buffers from one process locked
    - `get_user_pages()`
  - Pages remembered by kernel
  - Physical pages mapped to kernel memory
  - Direct copy in the context of the other process
    - `kmap()`
    - `copy_to_user()` and/or `copy_from_user()`

Bonus: Event loop considerations

- Event multiplexing/demultiplexing
  - Kernel `select()`, `poll()`, `epoll()`
  - Event loop library (e.g. libevent)
  - Pipes and sockets are file descriptors (POSIX)
  - Message queues are file descriptors (Linux)
- Mixing file descriptor waiting and mutexes/semaphores
  - Futexes are shared memory based
  - Futex file descriptor idea abandoned
  - Non-blocking calls available, not multiplexed

Das Ende!

Pavel Šimerda

pavlix@pavlix.net

Helping developers and companies in Czech Republic and abroad.