Embedded Linux

Device driver development

Sébastien Bilavarn
Outline

- Ch1 – Introduction to Linux
- Ch2 – Linux kernel overview
- Ch3 – Linux for Embedded Systems
- Ch4 – Embedded Linux distributions
- Ch5 – Case study: Xilinx PowerPC Linux
- Ch5 bis – Case study: Xilinx Zynq-7000 Linux
- Ch6 – Device driver development
Linux kernel overview

- System overview
- Process management
- Memory management
- Virtual file system
### System overview

- **CPU rings, privilege and protection**
  - All platforms that are intended to support operating systems with memory isolation do have some kind of privilege system.
    - On x86 processors, the privilege levels are called “ring 0” through “ring 3”
    - On ARM processors, the privilege levels are called “supervisor mode” and “user mode”
  - These hierarchical protection Domains are mechanisms to protect data and functionality from faults (fault tolerance) and malicious behaviour (computer security).
System overview

CPU rings, privilege and protection

- In x86, privilege levels provide a mechanism whereby the OS and CPU restrict what user-mode programs can do.
- There are four privilege levels, numbered 0 (most privileged) to 3 (least privileged),
- There are three main resources being protected: memory, I/O ports, and the ability to execute certain machine instructions.
- At any given time, an x86 CPU is running in a specific privilege level which determines what code can and cannot do. These privilege levels are often described as protection rings, where the innermost ring is the most privilege.
System overview

- CPU rings, privilege and protection
  - Linux (and Windows) are two operating systems that use supervisor/user-mode
    - Two levels of privilege:
      - "kernel space" vs. "user space" for Linux
      - Ring 0 and ring 3 for Windows
    - /\ These privileges have nothing to do with Linux superuser (root).
  - To perform specialized functions, user-mode code must perform a system call into supervisor mode or even to the kernel space where trusted code of the operating system will perform the needed task and return it back to user space.
System overview

The Linux kernel is one component of a system, which also requires libraries and applications to provide features to end users.

- Applications are in user space
- Drivers are in kernel space
System overview

- System calls
  - The main interface between the kernel and userspace is the set of system calls
  - About ~300 system calls that provides the main kernel services
    - File and device operations, networking operations, inter-process communication, process management, memory mapping, timers, threads, synchronization primitives, etc.
  - This interface is stable over time: new system calls can be added by kernel developers only.
  - This system call interface is wrapped by the C library, and userspace applications usually never make a system call directly but rather use the corresponding C library function
    - E.g. `fread/fwrite` wraps `read/write` system calls
Linux kernel overview

- System overview
- Process management
- Memory management
- Virtual file system
Linux process management

- Process
  - A process is an instance of execution that runs on a processor.
  - All processes running on Linux are managed by the `task_struct` structure (process descriptor)
Linux process management

- Process address space
  - The process memory area consist of these segments
    - Text segment: the area where executable code is stored.
    - Data segment: the data segment consists of these three areas.
      - Data: the area where initialized data such as static variables are stored.
      - BSS: the area where zero-initialized data is stored.
      - Heap: area where malloc() allocates dynamic memory based on the demand.
    - Stack segment: area where local variables, function parameters, and the return address of a function is stored.
**Linux process management**

- **Thread**
  - A thread is an execution unit generated in a single process.
  - It runs in parallel with other threads in the same process. They can share the same resources such as memory, address space, opened files, and so on.
  - The kernel deals with both processes and threads in a similar manner in terms of scheduling.

- In current Linux implementations, a thread is supported with the Portable Operating System Interface for UNIX (POSIX) compliant library (pthread).
Native POSIX Thread Library

A standard version of threads (IEEE POSIX 1003.1c)

Defines a standard Application Programming Interface

pthread API can be grouped into four major groups:

- **Thread management**: routines that work directly on threads - creating, detaching, joining, etc
- **Mutexes**: provide functions for creating, destroying, locking and unlocking mutexes (used for synchronization)
- **Condition variables**: allow threads to synchronize based upon the actual value of data
- **Signal handling**: POSIX supports signals, specific actions can be defined when a process/thread is delivered a signal.
Linux process management

- Thread process scheduling

- Time measurement in Linux
  - Ticks (also called jiffies)
    - In x86-based processor architectures, one tick corresponds to a time period of 10 ms.

- Default scheduling policies
  - Time sharing
    - Linux achieves the effect of an apparent simultaneous execution of multiple processes by switching from one process to another in a short time frame, called time slice (100ms).
    - Each (executable) task is dispatched to a CPU for this short period of time. If the time slice expires, the active task will be preempted in favor of another task.
Thread process scheduling

- Time shared scheduling policies
  - **SCHED_OTHER**
    - Default universal time-sharing scheduler policy used by most threads. These threads must be assigned with a priority of zero. Each thread runs for its time slice, after which the next thread is allowed to run.

  - **SCHED_FIFO**
    - Can be used only with priorities greater than zero: when a SCHED_FIFO process becomes available, it preempts any normal SCHED_OTHER thread.
    - If a SCHED_FIFO process that has a higher priority becomes available, it preempts an existing SCHED_FIFO process if that process has a lower priority. This thread is then kept at the top of the queue for its priority. There is no time slicing.

  - **SCHED_RR**
    - Round Robin Scheduling: time slices are assigned to each process in equal portions and in circular order.
    - Similar to SCHED_FIFO, but with one difference: each SCHED_RR process is allowed to run for a specified time quantum (delay given to a process to execute).
Linux kernel overview

- System overview
- Process management
- Memory management
- Virtual file system
Linux memory management

- Virtual memory management
  - Large Address Spaces
    - The operating system makes the system appear as if it has a larger amount of memory than it actually has.
  - Protection
    - Each process in the system has its own virtual address space. These virtual address spaces are separate from each other, so a process running one application cannot affect another.
    - The physical memory architecture of an operating system is usually hidden to the application and the user.
      - applications do not allocate physical memory, but request a memory map of a certain size at the Linux kernel and in exchange receive a map in virtual memory
      - virtual memory does not necessarily have to be mapped into physical memory, it might be mapped to the swap file on the disk subsystem.
Linux memory management

Virtual memory management

- To make translation between virtual and physical addresses easier, memory are divided into *pages*.
  - Pages are all the same size, usually 4 K bytes in size.
  - Each page is given a unique number, the page frame number (PFN).
    - Virtual page frame number: page number to which a virtual address belongs.
    - Physical page frame number: page number in physical memory of the corresponding physical address (if the data is actually mapped in physical memory).
- In this paged model, an address is composed of two parts; an offset and a page frame number.
  - Bits 11:0 of the address contain the offset and bits 12 and above are the page frame number.
Linux memory management

Virtual memory management

- As there is much less physical memory than virtual memory, a virtual page frame number is not necessarily mapped into physical memory, it might be mapped to the swap file on the disk subsystem.
  - The information about the mapping of a virtual page frame number into physical memory (and the corresponding physical page frame number) is present in the page tables.
  - If a process tries to access a virtual address that is currently in physical memory, the physical address is computed from the physical page frame number (provided in a page table entry corresponding to the virtual page referenced).
  - If a process tries to access a virtual address that is not currently in physical memory, the processor cannot find a page table entry for the virtual page referenced:
    - If the faulting virtual address is invalid this means that the process has attempted to access a virtual address that it should not have. In this case the operating system will terminate it, protecting the other processes in the system from this rogue process.
    - If the faulting virtual address was valid but the page that it refers to is not currently in physical memory, the operating system must bring the appropriate page into physical memory from the image on disk (swap).
Linux memory management

- Virtual memory management
  - Page frame allocation
    - A page is a group of contiguous linear addresses (page frame) in physical memory or virtual memory.
    - When a process requests a certain amount of pages
      - if there are available pages, the Linux kernel can allocate them to the process immediately.
      - Otherwise pages have to be taken from some other process or page cache
  - The Linux kernel has to maintain its free pages efficiently
    - It tries to keep the memory area contiguous
    - avoids memory fragmentation
  - Swapping
    - If the virtual memory manager in Linux realizes that a memory page has been allocated but not used for a significant amount of time, it moves this memory page to swap space.
Linux kernel overview

- System overview
- Process management
- Memory management
- Virtual file system
Virtual file systems

- Linux makes system and kernel information available in user-space through virtual filesystems.

- Virtual file systems allow applications to see directories and files that do not exist on any real storage: they are created on the fly by the kernel.

- The two most important virtual filesystems are
  - `proc` for process-related information
  - `sysfs` for device-related information
Virtual file systems

The `proc` virtual file system

- The proc virtual file system exists since the beginning of Linux
  - It allows
    - the kernel to expose statistics about running processes in the system
    - the user to adjust at runtime various system parameters about process management, memory management, etc.
  - The `proc` file system is used by many standard userspace applications, and they expect it to be mounted in `/proc`
  - Applications such as `ps` or `top` would not work without the `proc` file system
  - Documentation `/filesystems/proc.txt` in the kernel sources
  - `man proc`
- Example
  - `cat /proc/cpuinfo`
Virtual file systems

- The **proc** virtual file system
  - provides details about the files opened by processes, the CPU and memory usage, etc.
    - one directory for each running process in the system
      - `/proc/<pid>`
      - `cat /proc/3840/cmdline`
    - `/proc/interrupts`, `/proc/devices`, `/proc/iomem`, `/proc/ioports` provides general device-related information
    - `/proc/cmdline` shows the kernel command line
    - `/proc/sys` has many files that can be written to adjust kernel parameters
      - They are called sysctl. See `Documentation/sysctl/` in kernel sources.
Virtual file systems

The *sysfs* virtual file system

- The sysfs filesystem is a feature integrated in the 2.6 Linux kernel.
- It allows to represent in userspace the vision that the kernel has of the buses, devices and drivers in the system.
- It is useful for various userspace applications that need to list and query the available hardware.
- All applications using *sysfs* expect it to be mounted in the */sys* directory

Examples

- `ls /sys/`
  
  block bus class dev devices firmware fs kernel module power

- `cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq`